ON THE SECURITY OF MACHINE LEARNING BEYOND THE FEATURE SPACE

VON DER
CARL-FRIEDRICH-GAUSS-FAKULTÄT
DER TECHNISCHEN UNIVERSITÄT CAROLO-WILHELMINA
ZU BRAUNSCHWEIG

ZUR ERLANGUNG DES GRADES EINES
DOKTORS DER NATURWISSENSCHAFTEN (DR. RER. NAT.)

GENEHMIGTE DISSERTATION

VON
ERWIN QUIRING
GEBOREN AM 02.06.1989
IN DUSCHANBE

Eingereicht am:	15.07.2021
Disputation am:	15.10.2021
1. Referent:	Prof. Dr. Konrad Rieck
2. Referent:	Prof. Dr. Lorenzo Cavallaro

2021

PROMOTIONSAUSSCHUSS:

1. Referent:	Prof. Dr. Konrad Rieck Technische Universität Braunschweig
2. Referent:	Prof. Dr. Lorenzo Cavallaro University College London
Vorsitz:	Prof. Dr. Martin Johns Technische Universität Braunschweig

Braunschweig, Oktober 2021

ABSTRACT

Machine learning is increasingly used in security-critical applications, such as malware detection, face recognition, and autonomous driving. However, learning methods are vulnerable to different types of attacks that thwart their secure application. So far, most research has focused on attacks in the feature space of machine learning, that is, the vector space underlying the learning process. Although this has led to a thorough understanding of the possible attack surface, considering the feature space alone ignores the environment machine learning is applied in. Inputs are usually given as real-world objects from a problem space, such as malicious code or PDF files. Hence, an adversary has to consider both the problem and the feature space. This is not trivial, as both spaces have no one-to-one relation in most application areas and feature-space attacks are thus not directly applicable in practice. As a result, a more thorough examination is required to understand the real-world impact of current attacks against machine learning.

In this thesis, we explore the relation between the problem and the feature space regarding the attack surface of learning-based systems. First, we analyze attacks in the problem space that create real objects and that mislead learning methods in the feature space. A framework is developed to examine the challenges, constraints, and search strategies. To gain practical insights, we examine a problem-space attack against source code attribution. An empirical evaluation shows that the generated adversarial examples mislead the attribution in the majority of cases. Second, we analyze the mapping from problem to feature space. Using the example of image scaling, we study attacks that exploit the mapping and that are agnostic to the learning model or training data. After identifying the root cause of these attacks, defenses for prevention are developed and empirically evaluated against adversaries of different strengths.

Furthermore, the feature space also has an inherent connection to the media space of digital watermarking. This space is a vector space in which watermarks are embedded and detected. As adversaries target this process, attacks and defenses have been extensively studied here as well. Linking both spaces allows us to transfer attacks, defenses, and knowledge between machine learning and watermarking. Two case studies empirically demonstrate that defenses from watermarking can mitigate model-extraction attacks and, similarly, that defenses from machine learning can fend off watermarking attacks.

Taken together, this thesis provides a novel view on the security of machine learning beyond the feature space by including the problem space and the media space into the security analysis.

ZUSAMMENFASSUNG

Maschinelles Lernen wird zunehmend in sicherheitskritischen Anwendungen eingesetzt, zum Beispiel im Bereich der Schadsoftware-Erkennung, der Gesichtserkennung und des autonomen Fahrens. Allerdings können Angreifer Lernmethoden selbst gezielt umgehen oder täuschen. Hierbei hat sich bislang ein Großteil der Forschung auf Angriffe im Merkmalsraum von Lernmethoden beschränkt. In diesem Vektorraum findet der Lernprozess statt, sodass der Fokus auf diesen Raum zu einem soliden Verständnis über die Angriffsfläche im maschinellen Lernen geführt hat. Jedoch ist die alleinige Betrachtung des Merkmalsraums nicht ausreichend. In der Regel bestehen die Eingaben im maschinellen Lernen aus realen Objekten aus einem Problemraum, wie beispielsweise schädlichen Programmcode- oder PDF-Dateien. Ein Angreifer muss daher sowohl diesen Problemraum als auch den Merkmalsraum berücksichtigen. Dies ist nicht trivial, da beide Räume häufig keine 1:1-Beziehung aufweisen und somit Angriffe aus dem Merkmalsraum in der Praxis nicht direkt anwendbar sind. Um ein besseres Verständnis über die realen Auswirkungen möglicher Angriffe im maschinellen Lernen zu erlangen, sind daher weiterführende Untersuchungen nötig.

Diese Dissertation untersucht dazu die Beziehung zwischen Problemraum und Merkmalsraum hinsichtlich der Angriffsfläche lernbasierter Systeme. Es werden zuerst Angriffe im Problemraum betrachtet, welche reale Objekte erzeugen und gleichzeitig Lernmethoden im Merkmalsraum täuschen. Die mit dem Angriff verbundenen Herausforderungen, Nebenbedingungen und Suchstrategien werden hierbei systematisch festgehalten. Die dabei gewonnenen Erkenntnisse werden praktisch am Beispiel eines Angriffs gegen Identifikationsmethoden, welche Entwickler basierend auf Programmcode erkennen, eingesetzt. Eine empirische Evaluation zeigt, dass manipulierter Programmcode die korrekte Identifikation in der Mehrheit der Fälle verhindert. Als zweiter Kernpunkt der Analyse wird konkret die Abbildung aus dem Problemraum in den Merkmalsraum betrachtet. Am Beispiel von Bildskalierungen wird ein Angriff untersucht, welcher die Vorverarbeitung in dieser Abbildung gezielt ausnutzt. Der Angriff hängt somit weder vom Lernmodell noch von den Trainingsdaten ab. Eine Analyse der Angriffsursachen führt anschließend zur Entwicklung mehrerer Verteidigungsstrategien, die einen Angriff präventiv verhindern. Diese werden mit verschiedenen Angreifermodellen empirisch überprüft.

Abschließend stellt diese Dissertation eine Abbildung zwischen dem Merkmalsraum aus dem maschinellen Lernen und dem Medienraum digitaler Wasserzeichenverfahren her. Der Medienraum ist hierbei ein

Vektorraum, in dem die Einbettung und Erkennung von Wasserzeichen stattfindet. Auch hier versuchen Angreifer diesen Prozess zu umgehen, weshalb verschiedene Angriffe und Verteidigungen ebenfalls im Kontext von Wasserzeichen erforscht worden sind. Das Herstellen einer Abbildung zwischen Merkmalsraum und Medienraum erlaubt somit den Transfer von Angriffen, Verteidigungen und gewonnenen Erkenntnissen aus beiden Forschungsdisziplinen. Zwei Experimente belegen den Wissenstransfer empirisch. Erstens kann eine Verteidigung aus dem Wasserzeichen-Kontext die Extraktion eines Lernmodells erschweren. Zweitens verhindert eine Verteidigung aus dem maschinellen Lernen erfolgreich Wasserzeichen-Angriffe.

Zusammengefasst stellt diese Dissertation eine neue Sicht auf die Sicherheit von maschinellen Lernen her, indem zusätzlich zum Merkmalsraum auch der Problemraum und der Medienraum in die Sicherheitsanalyse miteinbezogen werden.

PUBLICATIONS

This thesis is based on concepts and results from the following peer-reviewed papers published by the author:

[1] E. Quiring, D. Arp, and K. Rieck. "Forgotten Siblings: Unifying Attacks on Machine Learning and Digital Watermarking." In: *Proc. of IEEE European Symposium on Security and Privacy (EuroS&P)*. 2018.

[2] E. Quiring, A. Maier, and K. Rieck. "Misleading Authorship Attribution of Source Code using Adversarial Learning." In: *Proc. of USENIX Security Symposium*. 2019.

[3] E. Quiring, D. Klein, D. Arp, M. Johns, and K. Rieck. "Adversarial Preprocessing: Understanding and Preventing Image-Scaling Attacks in Machine Learning." In: *Proc. of USENIX Security Symposium*. 2020.

Moreover, this thesis presents previously unpublished concepts. Furthermore, the following papers of the author directly or indirectly contributed to the thesis as well:

[1] E. Quiring and P. Schöttle. "On the Combination of Randomized Thresholds and Non-Parametric Boundaries to Protect Digital Watermarks against Sensitivity Attacks." In: *Proc. of the ACM Workshop on Information Hiding and Multimedia Security (IH&MMSec)*. 2014.

[2] E. Quiring and M. Kirchner. "Fragile Sensor Fingerprint Camera Identification." In: *IEEE International Workshop on Information Forensics and Security (WIFS)*. 2015.

[3] E. Quiring and K. Rieck. "Adversarial Machine Learning Against Digital Watermarking." In: *European Signal Processing Conference (EUSIPCO)*. 2018.

[4] E. Quiring, M. Kirchner, and K. Rieck. "On the Security and Applicability of Fragile Camera Fingerprints." In: *Proc. of European Symposium on Research in Computer Security (ESORICS)*. 2019.

[5] E. Quiring, L. Pirch, M. Reimsbach, D. Arp, and K. Rieck. *Against All Odds: Winning the Defense Challenge in an Evasion Competition with Diversification*. arXiv:2010.09569. 2020.

[6] E. Quiring and K. Rieck. "Backdooring and Poisoning Neural Networks with Image-Scaling Attacks." In: *Deep Learning and Security Workshop (DLS)*. 2020.

[7] D. Arp, E. Quiring, C. Wressnegger, and K. Rieck. "Privacy Threats through Ultrasonic Side Channels on Mobile Devices." In: *Proc. of IEEE European Symposium on Security and Privacy (EuroS&P)*. 2017.

[8] D. Arp, E. Quiring, T. Krueger, S. Dragiev, and K. Rieck. "Privacy-Enhanced Fraud Detection with Bloom Filters." In: *Proc. of Int. Conference on Security and Privacy in Communication Networks (SECURECOMM)*. 2018.

[9] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck. "Dos and Don'ts of Machine Learning in Computer Security." In: *Proc. of USENIX Security Symposium*. 2022 (to appear).

ACKNOWLEDGMENTS

First of all, I would like to especially thank Prof. Dr. Konrad Rieck, who is every PhD student's dream come true—supervisor, mentor, and partner in research. He provided invaluable scientific insights and advice. He always created a positive and inspiring environment for research that also encouraged me to pursue in the most challenging phases of research.

Special thanks also to Prof. Dr. Lorenzo Cavallaro for refereeing the thesis and to Prof. Dr. Martin Johns for chairing the defense committee. I know you have busy schedules, so I am very glad that you took the time for my thesis and defense.

Going back in time, my studies at the University of Münster provided a perfect basis for my PhD. I cannot possibly express enough gratitude to Prof. Dr. Matthias Kirchner, who supervised me during my Master's thesis in Binghamton. This positive experience fostered my decision to start my PhD. I am also very grateful to Prof. Dr. Rainer Böhme and Dr. Pascal Schöttle, who enabled and supported my research activities already during my studies. Almost needless to say, studying is better with friends, so that I am grateful to Dennis Assenmacher and Björn Ross. We learned a lot together—giving each other valuable criticism and support.

Great research is only possible with an incredible team, which is why I am very grateful to everyone at the Institute of System Security. Special thanks go to Dr. Daniel Arp. I could not imagine a better office mate for the PhD. We conducted exciting research projects together and had intriguing discussions about machine learning, research, and life. Thanks to Robert Michael and Dr. Christian Wressnegger, who were there with advice and support in the everyday life at the institute. Katja Barkowsky and Frank Rust have my sincere gratitude for always helping with all the organizational and technical issues at work.

Unfortunately, I cannot name all inspiring people that I met during my PhD, but believe me, I am thankful for all of you.

Finally, I am deeply indebted to my family, especially my parents. Without them, my thesis would never have been. They supported me all the time and made it possible for me to study. I am also grateful to my grandfather Otto, who raised my interest in science very early with his attitude and our discussions.

Most importantly, I would like to thank my wife Christina. You believed in me, encouraged me, supported me from day one unconditionally. You proofread the thesis and helped to organize the defense. I cannot tell in words how grateful I am. No one could ask for a truer companion in life.

CONTENTS

1 INTRODUCTION 1
- 1.1 Motivation 1
- 1.2 Thesis Contributions 3
- 1.3 Structure of Thesis 4

2 BACKGROUND 5
- 2.1 Machine Learning 5
  - 2.1.1 The Learning Task 6
  - 2.1.2 The Mapping 6
  - 2.1.3 Model Training 9
  - 2.1.4 Model Output 13
  - 2.1.5 Putting the Stages Together 13
  - 2.1.6 Classifier Performance 14
- 2.2 Adversarial Machine Learning 16
  - 2.2.1 Threat Model 17
  - 2.2.2 Adversarial Example 20
  - 2.2.3 Model Extraction 26
  - 2.2.4 Hyperparameter Extraction 28
  - 2.2.5 Poisoning Attack 29
  - 2.2.6 Membership Inference 31
- 2.3 Summary 31

3 ATTACK IN THE PROBLEM SPACE 33
- 3.1 Authorship Attribution of Source Code 34
  - 3.1.1 Mapping from Code to Features 34
  - 3.1.2 Multiclass Classification 37
- 3.2 Problem-Feature Space Dilemmas 38
  - 3.2.1 Problem Space ⇝ Feature Space 39
  - 3.2.2 Feature Space ⇝ Problem Space 40
  - 3.2.3 Generalization of Dilemmas 41
- 3.3 Formalization of Problem-Space Attacks 41
  - 3.3.1 Constraints of Problem-Space Attacks 42
  - 3.3.2 Search Strategy of Problem-Space Attacks 46
  - 3.3.3 Choice of Search Strategy 49
  - 3.3.4 Refining the Search Strategy for Side Effects 50
  - 3.3.5 Final Definition of Problem-Space Attacks 50
- 3.4 Attacking Authorship Attribution 52
  - 3.4.1 Threat Model 52
  - 3.4.2 Attack Constraints 53
  - 3.4.3 Available Code Transformations 54
  - 3.4.4 Search Strategy 59
- 3.5 Evaluation 61
  - 3.5.1 Experimental Setup 61
  - 3.5.2 Untargeted Attack 62

3.5.3 Targeted Attack 64
3.5.4 Verification of Attack Constraints 68
3.6 Limitations 69
3.7 Related Work 70
3.8 Chapter Summary 73
4 ATTACK ON THE MAPPING 75
4.1 Image Scaling in Machine Learning 76
4.2 Image-Scaling Attacks 77
4.2.1 Threat Model 77
4.2.2 Attack Strategy 80
4.3 Attack Analysis 81
4.3.1 Scaling as Signal Processing 82
4.3.2 Scaling and Convolution 83
4.3.3 Root-Cause Analysis 84
4.4 Defenses 87
4.4.1 Attacker Model 87
4.4.2 Defense 1: Robust Scaling Algorithms 87
4.4.3 Defense 2: Image Reconstruction 90
4.5 Evaluation 92
4.5.1 Experimental Setup 92
4.5.2 Defense 1: Non-Adaptive Attacks 93
4.5.3 Defense 1: Adaptive Attacks 95
4.5.4 Defense 2: Non-Adaptive Attacks 98
4.5.5 Defense 2: Adaptive Attacks 100
4.6 Discussion 102
4.6.1 Downgrade Attack to Nearest Scaling 102
4.6.2 Limitations 103
4.7 Further Attack Surfaces in the Mapping 104
4.7.1 Attack on the Mapping in the Text Domain . . . 104
4.7.2 Fragile Camera Fingerprints 105
4.8 Related Work 106
4.9 Chapter Summary 107
5 LINKING FEATURE AND MEDIA SPACE 109
5.1 Digital Watermarking 110
5.1.1 Example of a Watermarking Scheme 111
5.1.2 Attacks Against Watermarking 111
5.2 Unifying Adversarial Learning and Watermarking . . . 112
5.2.1 Feature Space vs. Media Space 114
5.2.2 Classifier vs. Watermark Detector 114
5.2.3 Adversarial Example vs. Oracle Attack 116
5.2.4 Model Extraction vs. Watermark Estimation . . 118
5.2.5 Defenses 119
5.2.6 Differences 124
5.3 Case Studies 124
5.3.1 Transfer of Defense: ML → DW 125
5.3.2 Transfer of Defense: DW → ML 128

5.4 Related Work 132
5.5 Summary 132
6 CONCLUSION 135
6.1 Summary of Results 135
6.2 Future Work 136
A ADDITIONAL BACKGROUND INFORMATION 139
B ATTACK IN THE PROBLEM SPACE 141
B.1 Example for Problem-Feature Space Dilemmas 141
B.2 Semantic Gap in Code Attribution 143
B.3 List of Code Transformations 143
B.4 Monte-Carlo Tree Search 146
B.5 List of Developers for Impersonation 149
B.6 Additional Evaluation Results 149
B.7 Additional Information for Related Work 152
C ATTACK ON THE MAPPING 155
C.1 Combining Poisoning with Image-Scaling Attacks . . . 155
C.2 Selective Random Filter 157
C.3 Additional Evaluation Results 158
C.4 Adaptive Attack Against Median Filter 161
D LINKING FEATURE AND MEDIA SPACE 163
D.1 Blind Newton Sensitivity Attack 163
D.2 Kalker’s Attack 165
D.3 Security Margin Construction 166

BIBLIOGRAPHY 167

LIST OF FIGURES

Figure 1.1 Schematic overview of the thesis 4
Figure 2.1 Typical machine-learning pipeline 7
Figure 2.2 Bag-of-words model for source code 8
Figure 2.3 Decision tree with numerical features 10
Figure 2.4 Fully connected neural network 11
Figure 2.5 Attacks along the machine-learning pipeline . 16
Figure 2.6 Threat model: Range of knowledge 18
Figure 2.7 Adversarial examples in the image domain . . 20
Figure 2.8 Adversarial example generation 22
Figure 2.9 Model-extraction attack 26
Figure 2.10 Training-only poisoning attack 30
Figure 2.11 Backdoor attack 30
Figure 3.1 Code sample as running example 35
Figure 3.2 Layout features from code sample 35
Figure 3.3 Lexical features from code sample 36
Figure 3.4 AST for code sample 37
Figure 3.5 Multiclass classification 38
Figure 3.6 Problem-feature space dilemmas 40
Figure 3.7 Attack procedure in $\mathcal{Z}$ and $\mathcal{F}$ 41
Figure 3.8 Visualization of semantic gap 45
Figure 3.9 Comparison of search strategies 48
Figure 3.10 CFG with UDCs for code sample 55
Figure 3.11 AST with DRM for code sample 56
Figure 3.12 Example of a control transformation 57
Figure 3.13 Example of a declaration transformation 57
Figure 3.14 Example of an API transformation 58
Figure 3.15 Monte-Carlo Tree Search (MCTS) 60
Figure 3.16 Changed features by untargeted attack 63
Figure 3.17 Changed lines of code by untargeted attack . . 64
Figure 3.18 Impersonation matrix for Caliskan et al. [36] . 65
Figure 3.19 Changed lines of code by targeted attack . . . 66
Figure 3.20 Changed features by targeted attack 66
Figure 4.1 Example of an image-scaling attack 75
Figure 4.2 Principle of image-scaling attacks 78
Figure 4.3 Applications of image-scaling attacks 79
Figure 4.4 Decomposition of image-scaling attack 81
Figure 4.5 Example of an undersampled signal 82
Figure 4.6 Scaling with convolution 83
Figure 4.7 Visualization of kernel functions for scaling . . 84
Figure 4.8 Scaling with convolution under attack 85
Figure 4.9 Influence of the scaling ratio and kernel size . 86

Figure 4.10 Comparison of bilinear scaling in libraries . . . 89
Figure 4.11 Comparison of scaling algorithms under attack 90
Figure 4.12 Reconstruction defense with median filter . . . 91
Figure 4.13 Examples from reconstruction defense 91
Figure 4.14 Success rate of scaling attacks regarding O1 . . 94
Figure 4.15 User study on scaling attacks regarding O2 . . 95
Figure 4.16 Adaptive attack against area scaling 97
Figure 4.17 PSNR distribution with reconstruction defense 99
Figure 4.18 Run-time performance of defenses 100
Figure 4.19 Success rate of adaptive scaling attacks (O1) . 101
Figure 4.20 User study for adaptive scaling attacks (O2) . 102
Figure 4.21 Text stemming as attack surface 105
Figure 5.1 Attacks against learning and watermarking . . 109
Figure 5.2 Example of a digital watermark 111
Figure 5.3 Unified view on learning and watermarking . 113
Figure 5.4 Defense transfer to watermarking 125
Figure 5.5 Image distortions with 1½-detector 126
Figure 5.6 Defense transfer to machine learning 129
Figure B.1 Example for problem-feature space dilemmas 142
Figure B.2 Impersonation matrix for Abuhamad et al. [2] 149
Figure B.3 Impersonation example with source code . . . 151
Figure C.1 Example of an overlay-poisoning attack 156
Figure C.2 Success rate of scaling attacks (O2, PSNR) . . . 158
Figure C.3 Best images of L_0 attack against area scaling . 159
Figure C.4 Selective source scenario against area scaling . 159
Figure C.5 Evaluation examples from reconstruction . . . 160
Figure C.6 Examples from adaptive scaling attack 160
Figure C.7 User study examples from adaptive attack . . 161
Figure D.1 Blind Newton Sensitivity Attack (BNSA) . . . 164
Figure D.2 Kalker's attack 165

LIST OF TABLES

Table 2.1 Summary of a machine-learning pipeline . . . 14
Table 3.1 Constraints in problem-space attacks 43
Table 3.2 Categorization of search strategies 47
Table 3.3 Constraints in attacks with source code 54
Table 3.4 Implemented families of code transformations 56
Table 3.5 Performance of code attribution methods . . . 62
Table 3.6 Performance of attacks against code attribution 63
Table 3.7 Used transformation families for attack 67
Table 3.8 Study on plausibility of transformed code . . . 69
Table 3.9 List of published problem-space attacks 71

Table 3.10 Detailed overview of problem-space attacks . . 72
Table 3.11 Comparison of code attribution methods . . . 73
Table 4.1 Scaling algorithms in learning frameworks . . 77
Table 4.2 Table of symbols for scaling attacks 78
Table 4.3 Kernel width per algorithm and library 88
Table 4.4 Recovery performance of reconstruction 99
Table 4.5 Implementation of $\nu(p)$ in libraries 103
Table 5.1 Transfer of defense techniques between adversarial learning and digital watermarking . . . 120
Table 5.2 Performance of the 1½-detector 127
Table 5.3 Datasets for transferred stateful defense 129
Table 5.4 Performance of stateful defense 130
Table 5.5 Performance with informed attacker 131
Table B.1 List of code transformations 144
Table B.1 List of code transformations (continued) 145
Table B.2 List of developers for impersonation 149

ACRONYMS

ANSI	American National Standards Institute
API	Application Programming Interface
AST	Abstract Syntax Tree
BNSA	Blind Newton Sensitivity Attack
CFG	Control-Flow Graph
CNN	Convolutional Neural Network
DRM	Declaration-Reference Mapping
FPR	False Positive Rate
GCJ	Google Code Jam
JPEG	Joint Photographic Experts Group
kNN	k-Nearest-Neighbor
LOC	Lines of Code
LSTM	Long Short-Term Memory
MCTS	Monte-Carlo Tree Search
PDF	Portable Document Format
PE	Portable Executable
PSNR	Peak Signal to Noise Ratio
RNN	Recurrent Neural Network
SIFT	Scale-Invariant Feature Transform
SVM	Support Vector Machine
TPR	True Positive Rate
UDC	Use-Define Chain

1

INTRODUCTION

1.1 MOTIVATION

Machine learning is nowadays a key component in computer science and engineering. It has paved the way for breakthroughs in various areas [120], such as the recognition of image content [190, 201] and speech [97], as well as the translation of natural languages [15, 199]. Similarly, machine learning has also become a key enabler in computer security, spawning multiple learning-based security systems, such as for intrusion detection [145, 173], authorship attribution [2, 36], and malware analysis [10, 174, 235]. For example, machine learning has advanced the state-of-the-art in source-code authorship attribution which allows identifying developers based on their coding style [2, 36]. Likewise, several detection systems for malicious software integrate learning methods for analyzing data more effectively [10, 174, 235].

Despite great potential, machine learning itself can introduce a considerable attack surface—at all stages of a learning pipeline. At training time, *poisoning and backdoor attacks* allow an adversary to manipulate the training process to change the classifier's behavior [24, 128]. As an example, consider the recognition of traffic signs, a core component for autonomous driving [122, 234]. With access to the training data, an adversary can easily inject a backdoor into a learning model by adding a trigger, for instance a small symbol, into a few training images of her target class [92]. If pixels are directly used for the learning process, a learning algorithm unnoticeably associates the trigger with the target class. At deployment time, the classifier correctly recognizes traffic signs in the absence of the trigger. Yet, once the trigger is provided, the classifier returns the target label irrespective of the actual traffic sign.

Moreover, a wide range of attacks are also possible after training at deployment time. *Adversarial examples and evasion attacks* mislead the prediction of a trained classifier by modifying the input only [26, 200]. These attacks typically operate in the classifier's *feature space* which is a vector space where the training and the classification are performed. The objective is to find manipulations of the feature vector in this space, so that the target prediction is returned. In the image domain, these attacks have been successfully applied against deep neural networks and they demonstrated that small changes of the feature vector already lead to an arbitrary target prediction [38, 200]. Due to the one-to-one relation between pixels and features, the found modifications can be directly mapped back to images. As the changes are often imperceptible for human beings, these attacks raised questions about the usage of deep learning for image recognition [38, 200].

Although attacks in the feature space provide a good understanding of the vulnerability of learning methods [12, 38, 39], they are only applicable if the feature manipulations can be mapped back to the *problem space* which is the input space to machine learning. This is straightforward with digital images if pixels from the problem space have a direct relation to features. However, most application areas with inputs such as text, source code, or PDF files do *not* have a one-to-one relation between problem and feature space. Thus, there is no inverse mapping and the real-world impact of feature-space attacks on machine learning is rather unclear.

Let us consider source code as example. An adversary might identify a promising feature vector for an attack, but this vector does not necessarily consider the syntax and semantics of source code and thus might not be realizable as code in the problem space. If an attack, for instance, required adding more opening brackets than closing brackets, this would directly violate the syntax. Moreover, the direction from problem to feature space can also introduce a notable hurdle. Due to side effects, the impact of a change in the source code on its feature representation can be hard to predict. Renaming a variable, for example, requires adapting all its usages which can also have an effect on other features.

Most prior work has also overlooked the vulnerability of the mapping from problem to feature space. This mapping typically consists of a preprocessing and the feature-extraction part. Both together are the basis for the learning pipeline, so that the mapping can introduce a considerable attack surface. Image-scaling attacks, for instance, exploit the preprocessing [225]. They slightly modify an input such that the mapping creates an arbitrary output. This enables an adversary to conceal backdoors in training data, so that the trigger only appears after the mapping during the training process [166]. The mapping can also be tricked into returning an arbitrary output, which allows controlling the prediction—similar to adversarial examples [171].

All these examples underline the complexity that the problem space introduces—most of it neglected by prior work. While first pioneering steps have been taken to integrate the problem space into the security analysis [see 162, 169, 236], a thorough analysis of the relation between the problem and the feature space regarding the attack surface has been missing in research so far. In this thesis, we explore their relation from two perspectives. First, we examine adversaries that mislead learning methods in the feature space and that create real objects in the problem space. Second, we take a closer look on the mapping itself.

The preceding insights highlight that we ought to look at more than just the feature space for a comprehensive view on secure learning. In fact, the feature space also has an inherent connection to the media space of digital watermarking. This space is a vector space where the watermarking process is carried out. Similarly to secure learning, watermarking also needs to deal with adversaries who seek to extract or remove the watermark from a signal [60]. Therefore, this field has also examined attacks and defenses in the media space and derived lessons learned. These insights can help to apply machine learning more securely. The other way round, concepts and insights from learning are also relevant to watermarking. In this thesis, we thoroughly explore the relation between the feature space of machine learning and the media space of watermarking. This joint view allows transferring attacks, defenses, and lessons learned.

1.2 THESIS CONTRIBUTIONS

Taken together, this thesis provides a comprehensive analysis of the relation between problem, feature, and media space regarding secure machine learning. In particular, four main contributions are provided.

- *Attacks along the learning pipeline.* For a comprehensive analysis of the different spaces, we study the basic concepts of machine learning along a full learning pipeline before exploring possible attacks at each component of this pipeline. Ultimately, this systematic picture of the adversarial environment allows deploying learning-based systems more securely by considering the different types of attacks that machine learning can reveal.
- *Attack in the problem space.* To obtain an understanding of the real-world implications, we analyze how adversaries can mislead learning methods in the feature space while creating real objects in the problem space. As these attacks are different to feature-space attacks, a framework is developed to capture the unique challenges, constraints, and search strategies. Using the example of source code attribution, we apply this framework and practically learn how real adversarial examples of code can be created. Hence, this thesis does not only examine how an attack in the problem space is realized, but also reveals weaknesses in current state-of-the-art authorship attribution methods.
- *Attack on the mapping.* We further study how an adversary can exploit the mapping from problem to feature space. In the context of preprocessing, in particular image scaling, we analyze how an adversary can control the output of the mapping with imperceptible changes to the input image. This attack is agnostic to the learning model, features, and training data. Moreover, it affects common libraries, such as OpenCV, Pillow, and TensorFlow. Yet, a root-cause analysis of this attack allows us to examine defenses for prevention that do not interfere with the typical pipeline of machine learning libraries.

- *Linking feature and media space.* Finally, we connect learning and watermarking. The feature and the media space are typically vector spaces that are divided into subspaces through decision boundaries. Thus, black-box attacks relying on input-output queries use similar strategies that are transferable. This leads, for instance, to novel black-box methods for adversarial examples or model extraction. Moreover, the similar attack surface allows transferring defenses, which is practically underlined in two case studies. In addition, we study lessons learned from each of the two fields, and how they can serve as guidance for the other.

1.3 STRUCTURE OF THESIS

Figure 1.1 provides an overview of the thesis' structure. To provide a foundation, Chapter 2 presents the learning pipeline and attacks at each component of this pipeline. In Chapter 3, we study the attacks in the problem space. The results against source code attribution and parts of the developed framework are based on a paper that was published at the *USENIX Security Symposium* in 2019 [169]. Chapter 4 examines the mapping from problem to feature space. The methods and results around the scaling attacks are based on a paper that was published at the *USENIX Security Symposium* in 2020 [171]. In Chapter 5, we then study the correspondence between feature space and media space. This chapter is mainly based on a paper published at the *IEEE European Symposium on Security and Privacy (EuroS&P)* in 2018 [168], updated with recent work to account for developments in research since publication. Finally, Chapter 6 provides a summary and directions for future work. Additional insights and results for each main chapter are provided in Appendix A, B, C, and D.

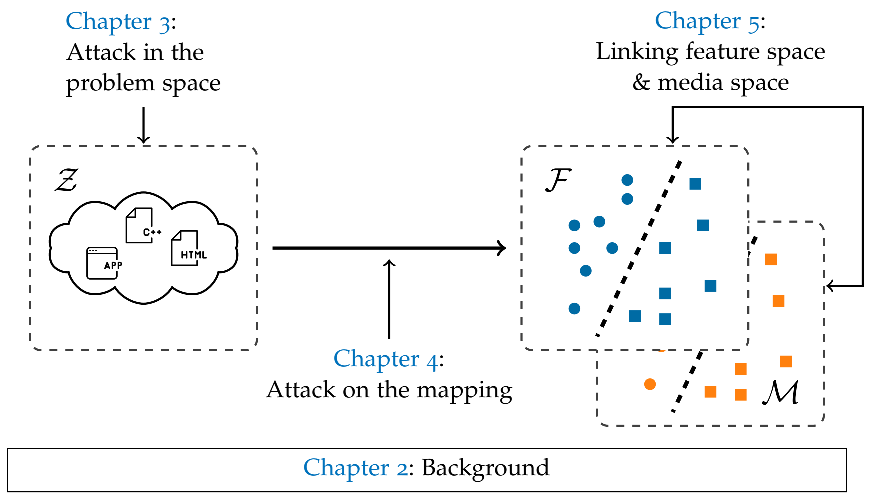


Figure 1.1: A schematic overview of the thesis with references to chapters. The thesis systematically examines the relations between the problem space $\mathcal{Z}$, feature space $\mathcal{F}$, and media space $\mathcal{M}$.

2

BACKGROUND

Machine learning has become the tool of choice in a number of areas. Learning methods are not only applied in classic settings, such as speech [97] and handwriting recognition [188], but increasingly operate at the core of security-critical applications, such as autonomous driving [e.g., 122, 234] and malware detection [e.g., 10, 174, 235]. The success of machine learning methods is rooted in the capability to automatically find patterns and relations within given data sets [see 67, 94]. However, this inference is usually not robust against attacks and thus may be disrupted or deceived by an adversary. In this chapter, we will first examine the basic concepts in machine learning along a typical data pipeline. This allows us then to systematically analyze the threat scenario of machine learning which lays the ground for the remainder of this thesis.

2.1 MACHINE LEARNING

Machine learning means learning from data automatically [1]. To respond to the diverse problems in our world, a vast set of concepts and techniques has been developed. To better understand and categorize this set, we can break down machine learning into three learning paradigms that deal with different assumptions and situations [1].

SUPERVISED LEARNING In this setting, we have a dataset of objects, such as PDF files or programs, and a respective target output assigned to each object. This output can be a numerical value in regression or a class label in *classification* tasks. Based on this labeled dataset, the objective is to automatically learn a prediction function that provides a value or label for future objects. The main contribution is that this function generalizes the data to the inherent patterns and relations instead of just memorizing the dataset. In security-critical applications, for instance, we are interested in classifying programs into goodware or malware in order to detect malicious behavior [e.g., 10]. The learning method extracts the malicious patterns without the need to define or search for these manually.

UNSUPERVISED LEARNING In this setting, we do not have an output information for the given training data. Yet, we can learn patterns and relations from the objects themselves [1]. For instance, clustering methods allow us to assign the input data to groups. In the context of malware detection, we may be interested in grouping programs to malware families to unveil similar malware samples [e.g., 16, 174]. Note that variations are possible. In semi-supervised learning, supervision information are available for some of the data [42]. This setting thus lies between supervised and unsupervised learning.

REINFORCEMENT LEARNING Finally, we have situations where we may not have output information, but obtain feedback for each action of a learning method. This allows improving the method by iteratively asking for feedback. For instance, the paradigm was effectively used in gaming with AlphaGo [187].

We focus on classification methods that...

We focus on classification in this thesis, which is of particular relevance in many security applications, such as malware detection [9], intrusion detection [173] or code authorship attribution [36]. Note that the discussed concepts in this thesis are also relevant to other learning paradigms, as we work around the problem and the feature space.

2.1.1 *The Learning Task*

...take real objects from a problem space $\mathcal{Z}$...

...and predict their respective class.

To begin with, we have to define the actual learning task. Formally, we have a *problem space* $\mathcal{Z}$ (or input space) that contains the objects $z \in \mathcal{Z}$ of a particular application. Each object in $\mathcal{Z}$ is associated with a class label $y \in \mathcal{Y}$, where $\mathcal{Y}$ denotes the space of labels. For example, $\mathcal{Z}$ may represent all possible PDF files. Each file may have two possible classes: $\mathcal{Y} = \{\mathit{Benign}, \mathit{Malicious}\}$. In authorship attribution, $\mathcal{Z}$ can represent all possible source codes and $\mathcal{Y}$ a set of authors that can be assigned to source codes. The overall objective is to learn a classification function

$$c_f : \mathcal{Z} \longrightarrow \mathcal{Y} \tag{2.1}$$

This prediction is learned with training objects, but...

that assigns $z \in \mathcal{Z}$ to a class label $y \in \mathcal{Y}$. The learning is based on a labeled training dataset that contains pairs of objects and labels:

$$\mathbb{D} = \{(z_i, y_i) \mid i = 1, \ldots, N\}\ , \tag{2.2}$$

where N is the number of available objects. For the subsequent threat analysis in Section 2.2, it is helpful to divide the learning process of the function c_f into different stages along a pipeline, as Figure 2.1 highlights. In the following, we will examine each stage in more detail.

2.1.2 *The Mapping*

...to obtain a suitable format for learning and predicting,...

The direct application of machine learning to problem-space objects in $\mathcal{Z}$ is usually not possible, as a learning algorithm cannot typically process objects as such [105]. Hence, the first stage for applying machine learning is the overall mapping of a problem-space object z to a suitable format. In particular, this stage consists of two steps that are discussed in the following.

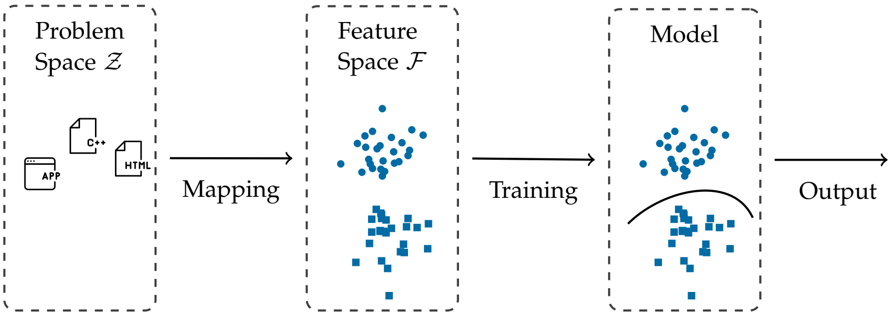


Figure 2.1: Overview of a typical machine-learning pipeline with three stages: the mapping, training, and output stage.

PREPROCESSING As a first step, an application-specific preprocessing of objects in $\mathcal{Z}$ can be necessary. This depends on the requirements of the subsequent learning stages. In the case of images, many learning algorithms require a fixed-size input, where the chosen input dimensions are often small to reduce the computational complexity. For instance, the deep neural networks for object recognition VGG19 [190] and Inception-v3 [201] expect inputs of 224×224 and 299×299 pixels, respectively. They are only applicable if images are scaled to these dimensions. As images typically do not match the input dimension of learning models, image scaling is a mandatory preprocessing step in most learning-based systems operating on images. In the case of source code, expanding macros is another example for preprocessing. As further outlined in Chapter 3, this can be necessary to remove artifacts in the dataset [11]. Otherwise, a learning method might use these artifacts as simple shortcuts instead of learning the authors' actual programming style.

...each object from $\mathcal{Z}$ is first preprocessed...

We formally express this preprocessing step as a function

$$\rho : \mathcal{Z} \longrightarrow \mathcal{Z} \tag{2.3}$$

that maps each object $z \in \mathcal{Z}$ to its processed representation in $\mathcal{Z}$.

FEATURE MAPPING The second step consists in the extraction of suitable features that capture the characteristics of the objects from $\mathcal{Z}$. As learning methods typically operate on vectorial data [29], we need a mapping from $\mathcal{Z}$ to a vector space using the extracted features. Formally, this mapping can be expressed as

...and then mapped to a feature vector...

$$\phi : \mathcal{Z} \longrightarrow \mathcal{F} = \mathbb{R}^d \quad \text{with } d \in \mathbb{N}^+ \tag{2.4}$$

where $\mathcal{F}$ represents the *feature space*, a d-dimensional vector space describing properties of the extracted features. Different techniques can be applied for constructing this map, which may include the computation of specific metrics as well as generic embeddings of features and their relations [67]. An example is the TF-IDF weighting, which adapts the map to account for the frequency of each feature

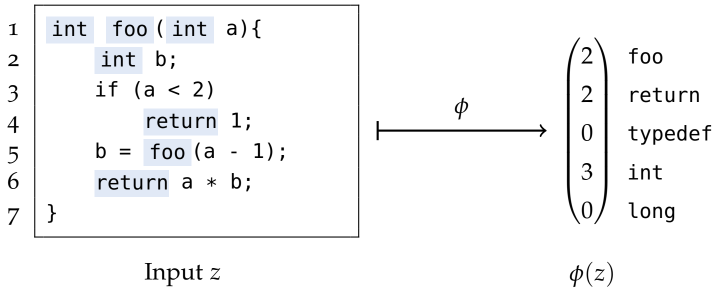
...in a feature space $\mathcal{F}$.

in the dataset [86]. Note that feature spaces in machine learning can also be constructed implicitly, for example using non-linear maps and kernel functions [67, 182]. Yet, an equivalent representation is often possible through vectors.

As an example for a feature mapping, let us consider the application domain of authorship attribution. We divide source code $z \in \mathcal{Z}$ into individual words. The frequency of occurrence of each word is a single feature and associated with a particular dimension in $\mathcal{F}$. This leads to the following feature vector,

$$\boldsymbol{x} = \phi(z) = (\phi_1(z), \phi_2(z), \ldots, \phi_d(z)) \in \mathcal{F}\,, \tag{2.5}$$

where $\phi_i(z)$ is the feature value for the i-th word. This representation is also called bag-of-words model [86]. The example in Figure 2.2 illustrates this mapping from source code to a feature vector. The first dimension is associated with the word `foo`. As it occurs twice in z, $\phi_1(\texttt{foo})$ is 2. The value is zero for a particular dimension if the word is not present in z. Note that Chapter 3 provides more details on possible features for source code authorship attribution.



```
int foo(int a){
    int b;
    if (a < 2)
        return 1;
    b = foo(a - 1);
    return a * b;
}
```

Figure 2.2: Bag-of-words model as example for a feature mapping from source code to a vector space.

Finally, we have a special case if ϕ is the identity function. For digital images, for instance, pixels from the problem space can have a one-to-one relation to the features. In this case, ϕ is also invertible, that is, a digital image can be created from any computed vector in $\mathcal{F}$ within the dynamic range. This is in contrast to various security domains, such as malware detection or authorship attribution, where ϕ is not invertible. This introduces a non-trivial hurdle for constructing attacks, as we will examine in more detail in Chapter 3 when studying the problem-feature space dilemmas.

SUMMARY Taken together, the first stage is the mapping $\phi \circ \rho$ from problem space $\mathcal{Z}$ to feature space $\mathcal{F}$:

$$\phi \circ \rho : \mathcal{Z} \longrightarrow \mathcal{F} \tag{2.6}$$

It consists of two steps: the preprocessing ρ and feature mapping ϕ. Figure 2.1 exemplifies the outcome of this process, where objects from $\mathcal{Z}$ are mapped to a feature vector in $\mathcal{F}$.

2.1.3 *Model Training*

The training stage then creates...

Equipped with a mapping, we can compute the feature vector for all objects in the training dataset $\mathbb{D}$. This allows the actual learning process in the next stage where a learning algorithm is used to infer functional dependencies from the training data for classification. These dependencies are described in a learning model with *model parameters* θ that parameterize a discriminant function g. Given a feature vector $\boldsymbol{x} \in \mathcal{F}$, the function $g(\boldsymbol{x})$ returns scores for all classes:

...a learning model in $\mathcal{F}$, that captures the inferred relations.

$$g : \mathcal{F} \longrightarrow \mathbb{R}^{|\mathcal{Y}|} . \tag{2.7}$$

The score for a respective class is given by $g_i(\boldsymbol{x})$. This setting has different advantages: First, we can examine all top-ranked classes. Second, we can interpret the returned scores to determine the confidence for each class. Depending on the model, the confidence can also be interpreted as probability, as we will see in the next section.

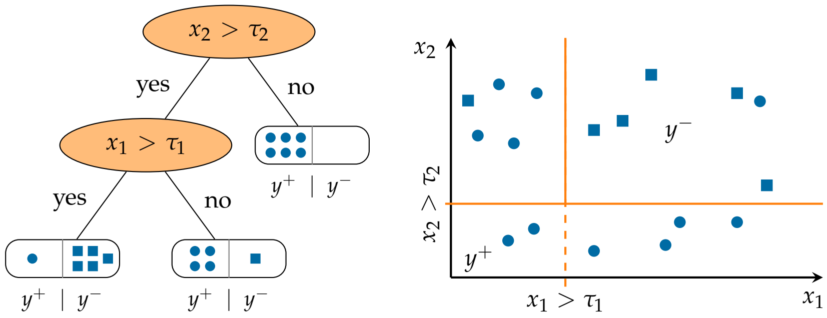
To this end, different learning algorithms can be applied,...

Different learning algorithms can be used to construct the classifier g, as for example, a support vector machine (SVM) [30, 213], a random forest [32], and a neural network [29, 120]. It is important to note that there is no universal learning algorithm that outperforms any other in general, which is also known as No Free Lunch Theorem [224]. As a result, we need to consider different algorithms in practice to respect the nature of the problem and data [67].

Each learning algorithm has different implications on the model and thus on the resulting attack surface. The differentiability of the discriminant function g, for example, plays an important role for possible attacks that are discussed in the subsequent chapters. Therefore, we examine a decision tree and a neural network as examples for learning algorithms in the following. While a decision tree is not differentiable, a neural network is. Moreover, both principles are commonly applied in security-related application areas [e.g., 2, 36, 101, 145]. Both together thus provide a good intuition on learning algorithms in an adversarial environment.

...such as a decision tree...

DECISION TREE We start with a decision tree [67]. Figure 2.3a illustrates the principle of a decision tree with numerical features. Each internal node is associated with a splitting function that takes a particular feature $\boldsymbol{x}_i$ as input and determines the subsequent path to a child node. In our example, the node is basically an if-else-statement. As depicted by Figure 2.3a, each leaf contains the number of training samples of each class that end up there. Class probabilities g_y can be estimated as fraction of training samples of the same class y in the respective leaf. In Figure 2.3a, for instance, the lower-left leaf has one training sample of class y^+ and five samples of y^-. Hence, we obtain the probabilities $g(\boldsymbol{x}) = (1/6,\ 5/6)^T$ for any $\boldsymbol{x}$ ending in this leaf. At deployment time, given a previously unseen feature vector $\boldsymbol{x}$,



(a) Decision Tree (b) Recursive partitioning in $\mathcal{F}$

Figure 2.3: Decision tree with numerical features. The left figure shows the tree structure with the number of training samples of each class in the leaves. The right figure shows the representation in $\mathcal{F}$.

the classification is obtained by traversing the tree from the root to a leaf node, and returning the class information associated with the particular leaf. In the previous example, if an input $\boldsymbol{x}$ ends up in the lower-left leaf, y^- can be assumed as most likely class for $\boldsymbol{x}$.

Geometrically, each splitting function in the internal nodes recursively partitions the feature space $\mathcal{F}$ into disjunct subspaces, as shown by Figure 2.3b. The finally retrieved subspaces are associated with a leaf node and thus represent the characteristics of a particular class. Due to this recursive partitioning, the underlying function g is not differentiable. In Chapter 5, we will take a closer look on this partitioning, when devising a new defense against attacks that aim at recovering the decision tree by querying it.

Decision trees are a basic element of more advanced learning algorithms, such as random forest [32] and gradient boosted trees [47]. In both cases, multiple decision trees are learned as ensemble. The final prediction is obtained by aggregating the prediction of each tree. From an adversary's perspective, this aggregation can complicate an attack, as any change in the input z may affect multiple trees at the same time in possibly contradicting ways. Nevertheless, an attack is still possible as we will examine in Chapter 3 with an authorship attribution method that uses a random forest.

...or a neural network.

FEEDFORWARD NEURAL NETWORK Another popular learning algorithm is a feedforward neural network [89]. In the following, we focus on a fully connected network that also illustrates the principle for other architectures, such as a convolutional neural network (CNN) [119].

Neurons are the building blocks of neural networks. The left part of Figure 2.4 illustrates a single neuron. It is essentially a linear function followed by a nonlinear activation function. For the linear part, we denote a vector of feature weights by $\boldsymbol{w} = (w_1, \ldots, w_d) \in \mathbb{R}^d$ and a bias term by $b \in \mathbb{R}$. The nonlinear activation function is denoted by ν. The latter part is necessary, as a composition of multiple lin-

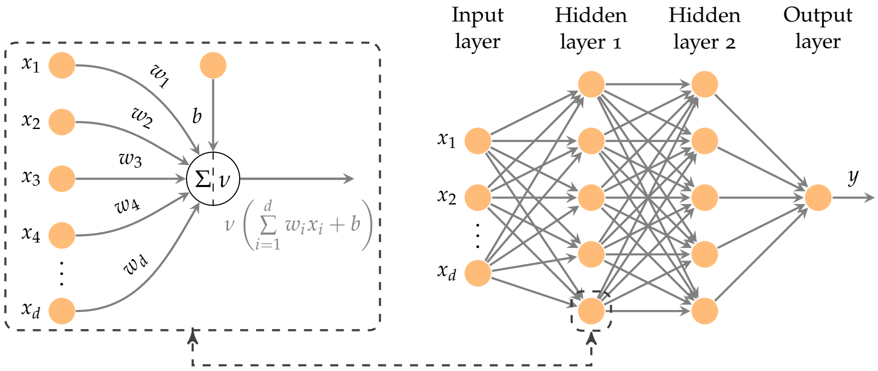


Figure 2.4: Overview of a neural network. The left figure shows a single neuron that is used together with more neurons in a fully connected neural network (right figure).

ear functions as neural network would only create a linear function again [89], preventing us from learning nonlinear relations. As a result, the computed function of a neuron is given as

$$\nu\left(\boldsymbol{w}^T \cdot \boldsymbol{x} + b\right) \ . \tag{2.8}$$

Different activation functions can be used, such as a logistic sigmoid function, the hyperbolic tangent function tanh or a rectified linear unit (ReLU) [see 89].

One or more neurons form a single layer, as depicted by Figure 2.4. The computation from Equation 2.8 for all neurons in a respective layer can be summarized by a single matrix multiplication if we stack all weight vectors together row-wise. To this end, we use a single matrix $W \in \mathbb{R}^{k \times n}$, where k denotes the number of neurons at the current layer, and n is the number of features. We also stack all bias terms together into a vector $\boldsymbol{b} \in \mathbb{R}^k$. As a consequence, the first layer can be expressed as

$$\boldsymbol{h}^{(1)} = \nu^{(1)}\left(W^{(1)} \cdot \boldsymbol{x} + \boldsymbol{b}^{(1)}\right) \ . \tag{2.9}$$

Note that the activation function ν operates element-wise, so that we obtain an output vector $\boldsymbol{h}^{(1)} \in \mathbb{R}^k$. Each element is the output of a respective neuron. The output $\boldsymbol{h}^{(1)}$ is then passed to the next layer, and the process is repeated for each layer until the last layer l:

$$\boldsymbol{h}^{(2)} = \nu^{(2)}\left(W^{(2)} \cdot \boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right) \tag{2.10}$$

$$\dots$$

$$\boldsymbol{h}^{(l)} = \nu^{(l)}\left(W^{(l)} \cdot \boldsymbol{h}^{(l-1)} + \boldsymbol{b}^{(l)}\right) \ . \tag{2.11}$$

The last layer returns the overall output $g(\boldsymbol{x})$. In the case of two classes, a single output neuron can be used (see Figure 2.4). With multiple classes, we can use a neuron for each class to output its score.

Often, the output of the last layer is converted into probability values for each class by applying a softmax function [see 89].

As Equations 2.9, 2.10, and 2.11 underline, the layers are connected in a chain. The information flows from the first to the last layer, hence the name feedforward network [89]. This network is a composition of multiple differentiable functions, so that the underlying discriminant function g is differentiable. Thus, it is possible to compute the gradient with respect to the input in order to find the direction where an input must be changed to move it towards a target class. As we will see in Section 2.2.2 and Chapter 3, this is a key element for gradient-based attacks.

Extensions in the design of neural networks have enabled breakthroughs in various areas, such as image and speech recognition or natural language processing [120]. While a fully connected network treats pixels being close or far away to each other equally, a CNN considers the spatial structure of the input [119]. It is thus particularly suitable for images, videos, audio, and speech [119, 120]. In Chapter 4, a CNN [190] is therefore used to evaluate preprocessing attacks on images. Another notable type of network is a recurrent neural network (RNN). It allows sequential inputs, such as the subsequent words in text or speech. The RNN internally keeps a state from previous inputs to predict the current outcome. Different architectures exist, such as long short-term memory (LSTM) networks. They have been also successfully applied in authorship attribution, as we will explore in Chapter 3. For more information on the extensions, the reader is referred to LeCun et al. [120].

Hyperparameters control the learning & model structure...

MODEL PARAMETERS AND HYPERPARAMETERS In theory, we can have random forests with an arbitrary number of decision trees. Likewise, neural networks can have an arbitrary number of layers. The number of trees or layers cannot be learned by the learning algorithm and have to be set in advance. These parameters belong to the group of *hyperparameters* that determine the structure of the learning model and the learning process of its parameters θ in general [29].

Note the difference between model parameters θ, such as the neuron weights $W^{(l)}$ in neural networks, and hyperparameters, such as the number of layers l. The learning algorithm automatically determines θ from the training data. On the contrary, hyperparameters are set in advance and are not directly learned within the learning algorithms.

...and are also part of the training stage.

Different algorithms exist to determine suitable hyperparameters. With grid search, for instance, a model is trained and evaluated for multiple combinations of different hyperparameters. The best combination is selected. As an example, let us assume we have one hyperparameter: the total number of layers $l = \{1, 2, 3\}$ that a neural network will have. In this example, the training dataset $\mathbb{D}$ is divided into a validation $\mathbb{D}_V$ and smaller training dataset $\mathbb{D}_T$. For each possible value of l, the learning algorithm finds optimal model parameters θ on $\mathbb{D}_T$ and its performance is evaluated on $\mathbb{D}_V$. The value l from $\{1,2,3\}$ with the highest performance on $\mathbb{D}_V$ is chosen as hyperparameter to train the final model on $\mathbb{D}$.

SUMMARY The training stage provides us with a learning model θ and its discriminant function g. This function allows us to obtain predictions in form of confidence values for each class with a given feature vector as input. As we in this thesis are primarily interested in the attack surface of an existing classifier, this section focuses on the output from the learning process and does not examine the training itself. Multiple crucial steps have to be considered to train a learning model, such as defining a loss function to find suitable model parameters θ or testing various hyperparameters of learning models. The reader is referred to Duda et al. [67] for a general introduction, and to Arp et al. [11] for a security-research-oriented discussion on avoiding pitfalls during this stage.

2.1.4 *Model Output*

In the last stage, class scores or class labels are returned.

In the last stage, the model is used and we have different options as output. This will have a direct impact on possible attacks, as discussed in Section 2.2.1. To begin with, the discriminant function g that returns scores for all classes can be directly used. Variations, such as the top-k scores or only the highest score of all classes, are also possible as model output. Alternatively, only the final classification can be returned by computing the *decision function* f:

$$f: \mathcal{F} \longrightarrow \mathcal{Y},\ \boldsymbol{x} \mapsto \operatorname*{arg\,max}_{y \in \mathcal{Y}} g_y(\boldsymbol{x}) \,. \tag{2.12}$$

Geometrically, f creates a decision boundary in the feature space $\mathcal{F}$, that separates the different classes from each other. This is illustrated in Figure 2.1 with two classes. In general, with k classes $y^{(k)} \in \mathcal{Y}$, the decision boundary separates $\mathcal{F}$ into k subspaces:

$$\mathcal{F} = \left\{\boldsymbol{x} \in \mathcal{F} | f(\boldsymbol{x}) \ = y^{(1)}\right\} \cup \dots \cup \left\{\boldsymbol{x} \in \mathcal{F} | f(\boldsymbol{x}) \ = y^{(k)}\right\}. \tag{2.13}$$

2.1.5 *Putting the Stages Together*

The successive stages form a learning pipeline where...

We are now ready to deploy the learning-based system by putting the different stages together. Remember that the overall objective in classification is to learn the function c_f from Equation 2.1 that returns a class label $y \in \mathcal{Y}$ for an input object $z \in \mathcal{Z}$. This chapter shows

Stage	**Function**	**Description**
Mapping	$\rho : \mathcal{Z} \longrightarrow \mathcal{Z}$	Mapping of problem-space object $z \in \mathcal{Z}$ to preprocessed format.
	$\phi : \mathcal{Z} \longrightarrow \mathcal{F}$	Mapping of (preprocessed) problem-space object to feature space
Training	$g : \mathcal{F} \longrightarrow \mathbb{R}^{\lvert\mathcal{Y}\rvert}$	Learning of a discriminant function g to separate all classes in a feature space $\mathcal{F}$.
Output	$f : \mathcal{F} \longrightarrow \mathcal{Y}$ or $g : \mathcal{F} \longrightarrow \mathbb{R}^{\lvert\mathcal{Y}\rvert}$	Usage of a decision function f or the discriminant function g as classification output.

Table 2.1: Summary of a machine-learning pipeline.

that this function can be represented as a composition of multiple functions:

$$c_f(z) = (f \circ \underbrace{\phi \circ \rho}_{\text{Thesis}})(z)\ . \tag{2.14}$$

As outlined before, it is also common to provide scores. In this case, the classifier c_f can be simply adjusted by replacing f with g, so that we obtain $c_g(z) = (g \circ \phi \circ \rho)(z)$.

...the mapping from $\mathcal{Z}$ to $\mathcal{F}$ is evidently an essential part.

Table 2.1 summarizes the resulting machine-learning pipeline. It highlights that the mapping $\phi \circ \rho$ from problem to feature space is of particular relevance for the whole pipeline. This underscores the need for the thorough security analysis of the mapping in this thesis.

2.1.6 *Classifier Performance*

We can measure the resulting classifier performance with...

Finally, we are interested in the system's performance if it is deployed. To this end, a test dataset is used that is distinct from the training dataset. A wide range of performance measures exist that provide different information and are suitable depending on the setting, such as multiclass problems or imbalanced data [see 11, 72]. Two performance measures are relevant in this thesis to evaluate a learning-based system in an adversarial environment. Note that the evaluation of the performance is also a crucial step in the training stage as well. We study the performance evaluation during deployment time, as the main attacks in this thesis are primarily targeting the final system.

...the accuracy...

ACCURACY To begin with, the *accuracy* reports the fraction of correct predictions:

$$\text{Acc} = \frac{1}{N} \sum_{i=1}^{N} [\ y_i = c_f(z_i)\ ], \tag{2.15}$$

where N is the number of tested objects, and $[\cdot]$ denotes the Iverson bracket. This performance measure can be used with two or multiple class labels if the class ratio in the given dataset is balanced.

With a large number of classes, the problem of *class ambiguity* can arise [117, 209]. Different classes can be very similar to each other or even overlap. For instance, the ImageNet ILSVRC 2012 dataset [66, 178] is widely used as benchmark for object recognition. Among 1000 classes, it has the two classes notebook and laptop. Objects of these classes are hard to distinguish even for human beings [209]. Furthermore, an input may have multiple valid labels, but only one label is assigned as true label. ImageNet, for example, has the two distinct classes canoe and paddle. An image showing a canoe in use naturally contains both classes, as a canoe without paddle is a rather unfavorable way of moving. As a result, the correct class for an input can be ambiguous. In this case, evaluating the performance from only one output might be too restrictive. The top-k accuracy is then preferred [178]. A correct prediction is reported if the correct class is one of the k most likely classes from $g(\boldsymbol{x})$. In other words, the classifier can make k predictions for each input, and at least one prediction needs to match the true label for a correct prediction. A value of $k = 5$ is typically used in image classification for the ImageNet dataset [178, 201]. This value is thus also used in Chapter 4 when evaluating scaling attacks with the ImageNet dataset.

...or top-k accuracy.

TRUE- AND FALSE POSITIVE RATE Although the accuracy breaks down the system's performance to a compact single value, we may need a more detailed picture of the performance. In binary classification settings, the *true positive rate* (TPR) and *false positive rate* (FPR) depict the tradeoff between two classes more clearly. In malware classification, for instance, we have two classes where malware is typically the positive class that needs to be detected. The benign sample represents the negative class. A true positive is then a correctly classified malware sample. A false positive occurs if $c_f(z)$ classifies a benign sample as malware. In general, the true positive rate is given as

Yet, the true- and false-positive rate provide...

$$\text{TPR} = \frac{\sum_i^N [\, c_f(z_i) = y^+ \mid y_i = y^+ \,]}{\sum_i^N [\, y_i = y^+ \,]} = \frac{\text{\# true positives}}{\text{\# samples of } y^+} \tag{2.16}$$

while the false positive rate is

$$\text{FPR} = \frac{\sum_i^N [\, c_f(z_i) = y^+ \mid y_i = y^- \,]}{\sum_i^N [\, y_i = y^- \,]} = \frac{\text{\# false positives}}{\text{\# samples of } y^-} \tag{2.17}$$

with $[\cdot]$ as Iverson bracket. Each nominator is conditioned on the true class label, so that we only check samples of a respective class. Ideally, the model has a high TPR and small FPR. Several problems in security especially have a need for a low FPR, such as intrusion detection or malware classification [10, 193]. Otherwise, a high FPR will lead to a large number of false alarms, and users, for example, may not trust a positive detection anymore.

...more insights on the performance.

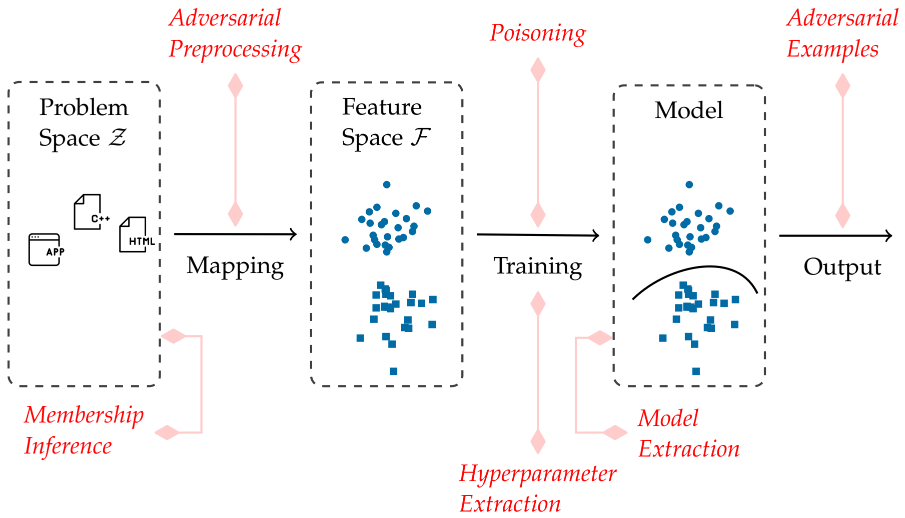

Adversarial Preprocessing
Poisoning
Adversarial Examples
Problem Space $\mathcal{Z}$
APP
C++
HTML
Feature Space $\mathcal{F}$
Model
Mapping
Training
Output
Membership Inference
Hyperparameter Extraction
Model Extraction

Figure 2.5: Attack surface of a typical machine-learning pipeline. Each attack category (in italics) is connected with the targeted component.

Note that the TPR and FPR need to be interpreted with care if we have a strong class imbalance and the base rate of the negative class needs to be considered [see 11]. With a predominant negative class, even a very low FPR can cause surprisingly high numbers of false positives. For example, 99% true positives at 1% false positives seem to provide a good performance. Yet, if we have a malware to benign class ratio of 1:100, this actually leads to 100 false positives for every 99 true positives.

2.2 ADVERSARIAL MACHINE LEARNING

Originally, learning methods have not been designed with security in mind. Many methods are vulnerable to different types of attacks that thwart their successful application. This problem has motivated the research field of *adversarial machine learning* which is concerned with the theory and practice of learning in an adversarial environment [21, 98, 105].

Unfortunately, the learning pipeline can be attacked at any component.

Figure 2.5 highlights that each component in a machine-learning pipeline provides an attack possibility. In this section, we examine the respective attacks at each component, providing a systematic picture of the adversarial environment. This overview is necessary to put the relation between problem space and feature space with its mapping $\phi \circ \rho$ into the context of different attacks. Ultimately, this overview enables us to consider the security of learning-based systems at the whole pipeline in order to apply these systems securely. Before examining each attack, we first define a threat model for the adversary.

2.2.1 *Threat Model*

A threat model provides a common understanding of the adversary's profile. Different taxonomies are used in the literature to characterize threats against learning-based systems [e.g., 21, 156, 159, 236]. The threat model in this thesis is based on the common taxonomy to model an adversary regarding her *knowledge*, *goals*, and *capabilities* [21].

A threat model defines the adversary.

KNOWLEDGE The adversary's strategy substantially depends on the knowledge of the learning-based system. The more the adversary knows about the different components in the learning pipeline, the better she can identify and exploit weak spots. The adversary's knowledge can be characterized by the following four main components of the pipeline: (i) the training data, (ii) the features, including the preprocessing ρ and feature mapping ϕ, (iii) the kind of learning algorithm, and (iv) the parameters of the learning model. Based on the known components, three attack scenarios are typically distinguished [21] and reflect how a learning-based system is deployed in practice:

Varying levels of knowledge are distinguished,...

...commonly defined in three attack scenarios, from...

- *White-Box Attacks.* In this setting, the adversary has a perfect knowledge about the learning-based system, including the training data, the features and the classifier.
- *Gray-Box Attacks.* Here, the adversary only knows a subset of the components. Although different combinations of the components are possible, the attacker is typically expected to know the features and kind of algorithm, but not the model parameters. The original training data are not known, but the attacker is assumed to collect an own surrogate dataset, ideally from the same or similar data distribution. Moreover, she may send queries to the system to augment her data. Overall, this enables her to learn an own surrogate classifier.

  If even the kind of algorithm is unknown, the attacker can rely on the *transferability* property. As learning algorithms infer general statistical patterns and relations from data, it is likely that different algorithms will infer similar relations [26, 158, 236]. As a result, an attack that works on a surrogate classifier based on a different algorithm can transfer to the original classifier [e.g., 236].
- *Black-Box Attacks.* In this case, the adversary has no knowledge about the learning-based system. Yet, she can send queries to the system that acts as *oracle* by returning an output for any input. Different options are possible as model output, as introduced before in Section 2.1.4. The adversary may retrieve the class label y from $c_f(z)$ or class scores from $c_g(z)$. Variations, such as the top-k scores or only the highest score, are also possible.

...white-box with full knowledge...

...over gray-box with limited knowledge...

...to black-box with no knowledge about the system.

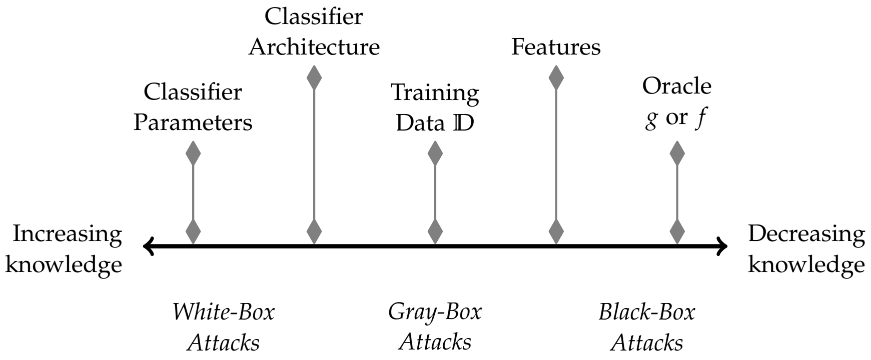


Figure 2.6: Threat model: Range of knowledge in the attack scenarios.

Yet, even in a black-box setting, an attacker may gather...

However, even in a black-box scenario, we need to assume that the adversary can obtain at least an approximation of possible features and training data [21]. For instance, the literature or the application domain itself can provide valuable information about typical features. API calls, for example, are common features for malware detection and are proposed in various publications [10, 140]. With deep neural networks in the computer vision domain, the features are typically the pixels [190, 201]. It is also often mandatory to scale images to apply learning methods. As common open-source libraries have a limited number of scaling options, it is reasonable to assume that an adversary can deduce and exploit the preprocessing function ρ after a few tests (see Chapter 4). Finally, an attack is possible even if the features are only partially reconstructed [e.g., 236], or transformations of the input object in the problem space are rather generic changes without targeting individual features specifically (see Chapter 3).

...some information about the features,...

Regarding the data, an attacker may exploit the same data source. For instance, common datasets for Android malware detection are based on samples from different app market places [10], so that the guessed dataset may approximate the original dataset reasonably well. Moreover, even if the adversary cannot narrow down the training data, she can benefit from additional, external data. For instance, the black-box attack against source code attribution in Chapter 3 also considers such a scenario. The training data are not known, but two external example files of the target developer are available. Both files are not part of the training- or test set and help as external source to obtain template information, such as recurring custom variable names.

...and the data.

Figure 2.6 summarizes the range of knowledge that the adversary typically has in the different attack scenarios [21, 156, 159]. Note that the figure only gives a tendency for the order of knowledge, as an adversary may know, for example, the classifier and features, but not the training data.

GOAL Adversaries can have different goals when working against a learning-based system. We can summarize these goals regarding their impact on *integrity*, *availability*, and *confidentiality* [21, 159]. The goal has a direct implication on the necessary attack procedure.

If the attacker is trying to control the model output without compromising the normal model behavior, her attack aims at the *integrity* of the system [21, 159]. An attack that compromises the normal functionality of the system for legitimate users is against the *availability* [21]. Both attacks can be *targeted* if the output should be a specific class. Otherwise, an attack is *untargeted* if any other class different from the original class should be predicted. Both goals require the adversary to choose, for instance, adversarial examples or poisoning attacks as attack procedure.

An adversary also has various goals...

If the attacker is targeting private information about the model or data by exploiting the query access, her attack aims at the *confidentiality* [21, 159]. For example, an attacker causes a serious privacy violation if she finds out that a person participated at a medical study by being present in the training data [186]. The confidentiality goal can require model extraction or membership inference as attack procedure.

CAPABILITY The capabilities refer to the possible modifications that an adversary performs to achieve her goals. Two aspects define the capabilities [21].

...and capabilities, such as...

First, it is relevant where an adversary has access in a machine-learning pipeline and what she can alter. She might be able to modify the training data or the model. On the contrary, if the model is already learned and thus fixed, she might only be able to modify the input sample. Each option leads to a different possible attack procedure, such as poisoning during training, and adversarial examples or preprocessing attacks against a fixed system.

...her learning pipeline access...

Second, the capabilities are determined by constraints in $\mathcal{Z}$ and $\mathcal{F}$. For instance, any modification of an object z in $\mathcal{Z}$ needs to preserve its functionality. The feature range in $\mathcal{F}$ can also be restricted, such as for 8-bit images with the pixel range $[0, 255]$. As we will further examine in Chapter 3, the capabilities are also determined by the relation between $\mathcal{Z}$ and $\mathcal{F}$. In many security-related applications, there is no one-to-one mapping between $\mathcal{Z}$ and $\mathcal{F}$. It is, for instance, impossible to find a valid source code or PDF file for any feature vector.

...and manipulation constraints.

Equipped with a definition of the threat model, we continue with examining the different attack procedures as shown by Figure 2.5. Adversarial examples, model extraction, and adversarial preprocessing are the main focus of this thesis. The first two attacks are introduced in detail in the following, while adversarial preprocessing is skipped here, as it will be thoroughly introduced in Chapter 4. Finally, in this section, we briefly examine the other attacks: hyperparameter extraction, poisoning, and membership inference. This overview enables us to thoroughly understand the attack surface.

An adversary can perform different types of attacks along the learning pipeline.

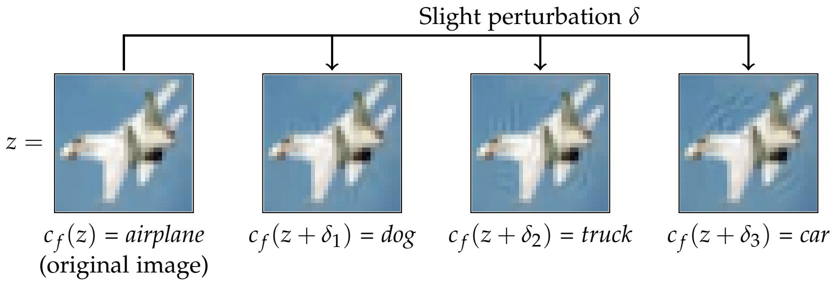


Figure 2.7: Adversarial examples against a deep neural network [38] trained on the CIFAR-10 [113] dataset. Slight, targeted perturbations cause a different prediction, although an airplane is still visible only.

2.2.2 *Adversarial Example*

With adversarial examples, an adversary misleads the prediction at deployment time.

In this attack setting, the adversary's goal is to thwart the prediction of a trained classifier [21, 159]. To this end, she carefully manipulates characteristics of the object z provided to the classifier to change the predicted class. The modified object is commonly denoted as *adversarial example*[1] [21, 159]. This attack impacts the *integrity* of the prediction. The classifier itself is *not* changed. Figure 2.7 shows an example in the image domain where an adversary can control the predicted class with slight, targeted perturbations. This is a substantial threat, considering the fact that the used images are small ($32 \times 32 \times 3$ pixels) and such perturbation would be less perceptible for larger images.

In the security domain, the detection of attacks with machine learning plays a vital role, such as for malware detection [10]. In this context, using adversarial examples to evade this detection is often referred to as *evasion attack* [21, 26, 148]. For example, in the case of spam filtering, the adversary may omit words from spam emails indicative for unsolicited content [131]. A common variant of evasion attacks are *mimicry attacks*, in which the adversary mimics characteristics of a particular class to hinder a correct prediction [70, 195]. Evasion and mimicry attacks have been successfully applied against different learning-based systems, for example in network intrusion detection [e.g., 71, 195], malware detection [e.g., 91, 227, 236] and face recognition [185].

Attacks are possible from white-box to black-box scenarios.

THREAT MODEL Depending on the adversary's *knowledge*, adversarial examples can be created in different settings. Attacks range from black-box [e.g., 45, 63, 132, 158, 220] to white-box settings [e.g., 26, 38, 88, 156, 200].

---

[1] Note that multiple terms are used to refer to the manipulated object. *Attack sample* [26] or *evasive sample* [227] are used in security-related domains. The term *adversarial example* was initially coined by Szegedy et al. [200] for attacks in the image domain with minimal modifications against deep neural networks. Yet, at the time of writing, *adversarial example* also became the established term in security, with any learning algorithm, and for high-confidence attacks (explained in this section) [38, 91, 124, 162].

Regarding the *capabilities*, we need to differentiate between *problem-space* and *feature-space* attacks. The latter attacks only operate in $\mathcal{F}$ and find suitable feature vectors that change the prediction as wanted. Feature-space attacks provide us with an understanding of the vulnerability of learning-based classifiers. They unveil, for instance, the number or amount of features that need to be changed for an adversarial example [21]. Besides, an adversarial example in $\mathcal{F}$ is a necessary condition for an adversarial example in $\mathcal{Z}$ [162]. Yet, as $\mathcal{F}$ and $\mathcal{Z}$ have no one-to-one relation in general, feature-space attacks cannot guarantee that a real-world object z can be created with the discovered feature vector. In this case, problem-space attacks are necessary. They additionally consider realizing an object in $\mathcal{Z}$ while misleading the classifier in $\mathcal{F}$.

The attack operates in the feature space $\mathcal{F}$...

...or in the problem space $\mathcal{Z}$.

In the special case where $\mathcal{F}$ and $\mathcal{Z}$ have a one-to-one relation, the feature-space attack is able to realize $z \in \mathcal{Z}$. A prominent example are digital images where pixels often correspond to features [38]. The pixels in the image z can then be directly changed according to the computed feature vector. Compared to problem-space attacks, this special case has received a lot of attention that led to a good understanding of possible attacks in $\mathcal{F}$ [12, 38, 208]. To also gain more insights on problem-space attacks, we will study them in detail in Chapter 3 as part of the relation between $\mathcal{Z}$ and $\mathcal{F}$.

Images with $\mathcal{F} \cong \mathcal{Z}$ are a special case.

In the following, we study feature-space attacks. They are helpful as background for problem-space attacks, and become relevant when we examine the relation between $\mathcal{F}$ and the media space $\mathcal{M}$ of watermarking in Chapter 5.

OBJECTIVE OF FEATURE-SPACE ATTACKS Given a fixed feature vector $\boldsymbol{x} \in \mathcal{F}$, the adversary wants to find a modification $\boldsymbol{\delta} \in \mathcal{F}$, so that the classifier predicts the adversary's target class $y^* \in \mathcal{Y}$ for $\boldsymbol{x}' = (\boldsymbol{x} + \boldsymbol{\delta})$. Here $\boldsymbol{x}'$ is the adversarial example. Geometrically, the adversary wants to obtain a feature vector $\boldsymbol{x}'$ that lies in the subspace of y^*, as illustrated by Figure 2.8 with two classes. In the case of an untargeted attack, y^* represents any other class than the original, source class y^s, that is, $y^* \neq y^s$. In the case of a targeted attack, y^* is defined as a particular target class y^t. This attack is more powerful than the untargeted variant by explicitly choosing the target. Without loss of generality, we examine the targeted variant in the following. Finding an adversarial example can be described as an optimization problem. In the literature, two formulations are mainly used:

An attack that solely works in the feature space optimizes...

- *High-confidence attack.* The adversary aims at increasing the classifier's confidence in y^t.
- *Minimum-modification attack.* The adversary aims at minimizing the required modifications to obtain y^t.

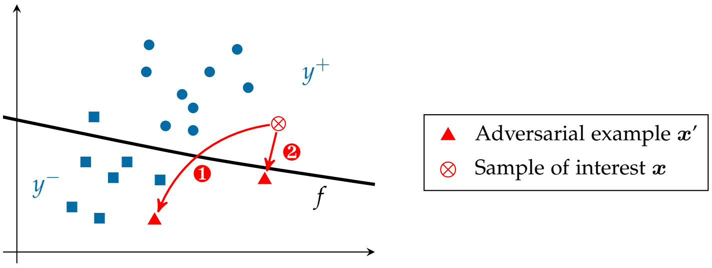

y^+
y^-
f
❶
❷
▲ Adversarial example $\boldsymbol{x}'$
⊗ Sample of interest $\boldsymbol{x}$

Figure 2.8: Goal of adversarial examples in $\mathcal{F}$. The adversary manipulates a sample to change its classification (❶ high-confidence or ❷ minimum-modification).

Figure 2.8 illustrates both versions. The minimum-modification attack just tries to cross the decision boundary to obtain a different prediction while the high-confidence variant increases the confidence with a position that lies more in the target region.

...either the classifier confidence...

For the high-confidence formulation, the following attack objective function can be used to optimize the confidence [21, 38]:

$$\zeta(\boldsymbol{x}') = \max_{k \neq y^t}\{g_k(\boldsymbol{x}')\} - g_{y^t}(\boldsymbol{x}'). \tag{2.18}$$

If $\zeta(\boldsymbol{x}') < 0$, the score for y^t is higher than for any other class, so that the classifier usually predicts y^t. The lower $\zeta(\boldsymbol{x}')$, the higher the confidence in y^t is. The overall attack is then [21]:

$$\begin{aligned} \arg\min_{\boldsymbol{\delta}} \quad & \zeta(\boldsymbol{x}+\boldsymbol{\delta}) & (2.19)\\ \text{s.t.} \quad & \mathcal{D}(\boldsymbol{x}, \boldsymbol{x}+\boldsymbol{\delta}) \leqslant \mathrm{d}_{\max} & (2.20)\\ & (\boldsymbol{x}+\boldsymbol{\delta}) \in [\boldsymbol{x}_{\mathrm{lb}}, \boldsymbol{x}_{\mathrm{ub}}], & (2.21) \end{aligned}$$

where $\mathcal{D}$ is a distance metric and $\mathrm{d}_{\max}$ sets a bound on the maximum modification. In addition, the modifications need to remain within the possible feature range, given by $\boldsymbol{x}_{\mathrm{lb}}$ and $\boldsymbol{x}_{\mathrm{ub}}$. Regarding $\mathcal{D}$, the L_0, L_2 or L_∞ norm are commonly used as distance metrics for feature-space attacks [38, 88, 156]. Appendix A provides a short introduction to these distance metrics for readers who are unfamiliar with them.

The adversary can ensure high-confidence adversarial examples by adding the constraint $\zeta(\boldsymbol{x}') < -\kappa$, where $\kappa \in \mathbb{R}$ is a desired confidence [38]. In this way, the classifier's confidence score for y^t must be larger by at least κ compared to any other class.

...or the required modifications.

In the second formulation, the adversary tries to minimize the required modifications for her adversarial example [e.g., 156, 200]:

$$\zeta(\boldsymbol{x}') = \mathcal{D}(\boldsymbol{x}, \boldsymbol{x}'). \tag{2.22}$$

The attack is then given by

$$\begin{aligned} \arg\min_{\boldsymbol{\delta}} \quad & \zeta(\boldsymbol{x}+\boldsymbol{\delta}) & (2.23)\\ \text{s.t.} \quad & f(\boldsymbol{x}+\boldsymbol{\delta}) = y^t & (2.24)\\ & (\boldsymbol{x}+\boldsymbol{\delta}) \in [\boldsymbol{x}_{\mathrm{lb}},\ \boldsymbol{x}_{\mathrm{ub}}]. & (2.25) \end{aligned}$$

In this way, the adversary minimizes $\boldsymbol{\delta}$ while achieving her target classification.

Note that further objective functions ζ are possible for both formulations [e.g., 26, 38, 88, 200, 208]. Equation 2.18 and Equation 2.22 can also be combined to optimize both formulations together [see 38].

Each formulation has its pros and cons. High-confidence attacks allow us to measure the worst-case vulnerability of a classifier for varying attack strengths $d_{\max}$ [21]. Furthermore, they create stronger adversarial examples by pushing them further away from the decision boundary compared to minimum-modification attacks, as depicted by Figure 2.8. Hence, the attack still succeeds even if the adversarial example or the learning model's decision boundary would be slightly changed. The higher confidence can also increase the transferability chances to the original model if the adversary uses a surrogate model to compute her adversarial example [38]. On the contrary, the minimum-modification attack can unveil the general sensitivity of a classifier to input changes. This attack initially led to the insight by Szegedy et al. [200] that minimal, imperceptible image perturbations can mislead deep neural networks. Moreover, we will see in Chapter 3 that minimizing modifications can be especially relevant for problem-space attacks to fulfill their constraints. Adversarial examples in $\mathcal{Z}$ have to preserve the semantics and remain plausible, which can be easier to realize by minimizing modifications.

White-box adversaries commonly use gradient descent...

FEATURE-SPACE ATTACK STRATEGIES Let us obtain an intuition how adversaries can find a solution to the previous optimization problems. This depends on the adversary's knowledge. In the white-box setting, gradient descent is the most widely adopted method [208]. The adversary computes the gradient of the objective function ζ with respect to the features $\boldsymbol{x}$:

$$\nabla_{\boldsymbol{x}}\, \zeta(\boldsymbol{x}). \tag{2.26}$$

The gradient gives the direction in $\mathcal{F}$ where ζ changes the most. In other words, the gradient shows which features need to be changed and how much they need to be changed to move towards the target classification. Starting from $\boldsymbol{x}_0 = \boldsymbol{x}$, the adversary then basically follows the gradient in multiple steps [208]:

$$\boldsymbol{x}_{i+1} = \texttt{Proj}\left(\boldsymbol{x}_i + \alpha \cdot \texttt{normalize}(\nabla_{\boldsymbol{x}_i}\, \zeta(\boldsymbol{x}_i))\right). \tag{2.27}$$

Essentially, `Proj` and `normalize` are used to control the modifications. In particular, `Proj` projects its input to a smaller domain to restrict the modifications [e.g., 115, 136]. The function `normalize` is responsible for a unit-length step size under the used norm, for instance, by applying a clipping operator `sign` [e.g., 88]. The parameter α controls the step size. The process is repeated until a satisfying position in $\mathcal{F}$ is found. Various feature-space attacks build on this principle [e.g., 26, 38, 88, 115, 136].

...while gray- and black-box adversaries either...

In the gray-box and black-box setting, the adversary is unable to compute the gradient in the previously described way, since she does not have full access to the learning model. Nevertheless, two groups of attacks are possible. The adversary uses either a *direct* or *learning-based* attack. In the first group, she sends queries to the classifier and directly uses its output to construct an adversarial example [e.g., 33, 45, 100]. In Chapter 5, we will examine such attacks in more detail when examining a related attack paradigm in watermarking. In the second group, the adversary uses an own surrogate learning model to create an adversarial example [e.g., 39, 132, 157, 158]. This allows her to use a white-box method again. This group of attacks is based on the *transferability* property [26, 200]. An adversarial example on model A often works on a different model B, despite another learning algorithm or dataset [88, 200, 236]. Hence, an adversarial example that misleads the adversary's surrogate model will probably mislead the original model as well. To obtain a suitable surrogate model, the adversary typically needs to conduct a model-extraction attack that is discussed in the next section.

...directly use input-output queries or...

...transfer adversarial examples from an own surrogate model.

WHITE-BOX DEFENSES The development of defenses against adversarial examples is a vivid research area at the time of writing, so that a comprehensive overview of all defenses is beyond the scope. In the following, we rather focus on the main concepts that are required for this thesis. For a broad overview of defenses, the reader is referred to Papernot et al. [159] and Xu et al. [228]. Publications that devise counter-attacks against a large set of defenses also provide a recommendable overview [see 12, 13, 39, 208]. In the white-box setting, defenses can be broadly categorized into three groups [21, 228]: *model robustness*, *classifier ensembles*, and *detection*.

White-box defenses...

In the first case, the defender learns a more robust model [e.g., 83, 88, 136, 155, 200, 226]. One possibility is to use *adversarial training* [e.g., 88, 200]. In this case, the defender creates adversarial examples herself and adds them to the training dataset together with the ground-truth labels. Ideally, this allows the learning algorithm to consider its own Achilles heel. However, the resulting model is only robust against attack algorithms that were used for training. An adaptive attacker with a different algorithm to create adversarial examples can be successful again [146]. A stronger variant of this defense principle is based on the concept of *robust optimization*. It aims at providing guarantees that the prediction does not change around an input given a certain amount of modification [e.g., 83, 136]. Consequently, the adversary will have to increase the amount of modification. In the case of images, for instance, this ideally leads to clearly visible modifications that would also cause misclassifications by humans [136].

...adjust the model to be more robust,...

The second group of defenses builds on classifier ensembles [e.g., 22, 23, 28, 142, 161, 220]. The prediction is then retrieved from multiple

...use multiple classifiers, or...

classifiers $f^{(i)}$ through an aggregation function E:

$$f(\boldsymbol{x}) = E\left(f^{(1)}(\boldsymbol{x}), f^{(2)}(\boldsymbol{x}), \ldots, f^{(k)}(\boldsymbol{x})\right). \tag{2.28}$$

For instance, multiple classifiers can be learned where each is built from a random subset of the feature set [22, 23, 161, 220].

The third group of defenses detects an adversarial example [e.g., 20, 142, 144, 183, 228]. For instance, an additional detector can be trained with legitimate samples as one class and adversarial examples as another class [144]. A subgroup of detection-based defenses tries to reduce the attack surface by detecting adversarial examples that were created outside the learning data distribution [21, 142, 179]. These *blind-spot* examples can simplify an attack, as it is not necessary to find plausible features of the target class [179]. In Figure 2.8, the lower-right corner represents a blind-spot that the adversary could simply identify by exploring the basis vectors. Blind-spots can be detected, for instance, by enclosing the learning data tightly [e.g., 28, 179]. In Chapter 5, we will thoroughly examine this defense against attacks in the media space of watermarking.

...detect an adversarial example.

BLACK-BOX DEFENSES In the black-box setting, the classifier is outside of the attacker's control. As a result, a stateful defense becomes possible in addition, where the defender seeks to identify sequences of malicious queries [e.g., 46, 168]. This defense exploits the fact that an adversary will typically have to send multiple queries that follow a specific strategy. If the classifier only returns class labels, for example, an adversary may need to use a line search to localize the boundary. This search creates a sequence of queries that differ from benign queries and thus can be detected. In Chapter 5, we will take a further look on this defense strategy where stateful defenses from watermarking are demonstrated to be applicable in adversarial learning.

Queries can also be monitored in the black-box case.

Unfortunately, most defenses are demonstrated to be vulnerable against counter-attacks at the time of writing [e.g., 12, 39, 208]. Yet, it is noteworthy that this iterative cat-and-mouse game led to various, valuable insights on the design and evaluation of defenses against adversarial examples [12, 39, 40, 208].

REMARK ON EVASION ATTACKS Finally, note that an adversary can use evasion techniques that do not directly target machine learning [e.g., 147, 195, 216]. For instance, adversaries can obfuscate the control flow, data location or data usage and so deceive the feature extraction from a static analysis [147]. However, this only allows an untargeted attack in the two-class detection setting. This thesis instead focuses on the general case with adversarial examples to analyze the security of machine learning under attack.

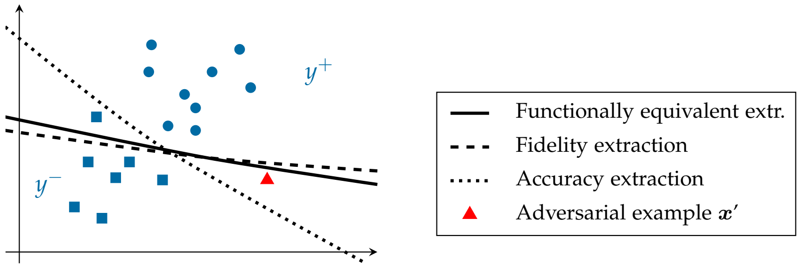


Figure 2.9: Model extraction. Functionally equivalent extraction recovers the decision boundary f exactly. Fidelity extraction creates $\hat{f}$ that matches f on all data points, incl. mistakes. Accuracy extraction creates $\hat{f}$ with perfect accuracy, but is less suitable for adversarial examples (e. g., $\boldsymbol{x}'$ from Figure 2.8 would not be found).

2.2.3 *Model Extraction*

With model extraction, the adversary aims at the learning model...

In this attack setting, the adversary actively probes a learning method and analyzes the returned output to reconstruct the underlying learning model [132, 207]. Geometrically, the attack tries to recover the decision boundary of the learning model (see Figure 2.9). The attack, denoted as *model extraction*, impacts the *confidentiality* of the learning model. In the following, we focus on attacks that aim at the model parameters θ. Methods to recover hyperparameters are discussed separately in the following section as *hyperparameter-extraction* attacks.

THREAT MODEL The adversary operates in a gray-box or black-box scenario. Regarding the *capabilities*, she can send queries to a classifier and observe the respective output. The query can be the object $z \in \mathcal{Z}$ or the feature vector $\boldsymbol{x} \in \mathcal{F}$ if ρ and ϕ are conducted on the user side [207]. The former case requires considering the problem space as well and thus induces more restrictions. For example, $\mathcal{Z}$ is discrete, so that not every feature vector can be queried.

Regarding the *knowledge*, the classifier may provide all scores $g(\cdot)$, the top-k scores, or only the class label y from $f(\cdot)$ [102]. The adversary is typically assumed to initially have a small surrogate training dataset [107]. Yet, depending on the application domain, she may be able to exploit more knowledge. For instance, image classifiers typically make use of the publicly available ImageNet dataset [178, 190, 201]. Hence, the adversary can use this dataset as well, although she may not know which images were used exactly [102, 152]. In the gray-box scenario, the adversary may know the kind of learning algorithm or used hyperparameters [207].

...with the motivation to steal the functionality...

Two *motivations* can be differentiated to conduct model extraction [102]. First, an adversary wants to *steal* the model for her own usage. For instance, she could leverage an online service's resources to train a learning model over a large dataset and then recover the model parameters with a few queries [207]. Second, model extraction can serve as stepping stone for further attacks, such as generating adversarial examples or conducting membership inference attacks. Depending on her motivation, the adversary has different *goals* [102]:

...or to get a model for further attacks.

- *Accuracy extraction.* If she wants to steal a model, the goal is to obtain a surrogate classifier $\hat{f}$ that has a similar or even higher performance on the expected data distribution $\mathcal{X}$ in $\mathcal{F}$. In practice, $\mathcal{X}$ is represented by the test set. Taken together, she maximizes $\Pr_{(\boldsymbol{x},y)\sim\mathcal{X}\times\mathcal{Y}}[\hat{f}(\boldsymbol{x}) = y]$.
- *Fidelity extraction.* With the intention for further attacks, the goal is to obtain a surrogate classifier $\hat{f}$ whose predictions correspond to the predictions of f. Thus, $\hat{f}$ also replicates the mistakes, which is important for the transferability of adversarial examples or membership inference. For instance, the adversary can maximize $\Pr_{\boldsymbol{x}\sim\mathcal{X}}\,[\mathbb{1}(\hat{f}(\boldsymbol{x}) = f(\boldsymbol{x}))]$.
- *Functionally equivalent extraction.* This is the strongest goal where the surrogate classifier behaves as the original classifier on all possible samples, including samples off the actual data distribution. The attacker thus aims at $\hat{f}(\boldsymbol{x}) = f(\boldsymbol{x})\ \forall \boldsymbol{x} \in \mathcal{F}$.

Figure 2.9 geometrically illustrates the model extraction attack with the different goals.

ATTACKS Attacks can be divided into two groups: *direct* and *learning-based* extraction. In the first group, the attacker directly reconstructs the model parameters θ based on her queries [e.g., 102, 132, 207]. As an example, let us assume a linear model $\boldsymbol{w}^T \cdot \boldsymbol{x} + b$ with $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$. The model is $\theta = [\boldsymbol{w}, b]$. Given the output score $g(\boldsymbol{x})$ for an input $\boldsymbol{x}$, the adversary obtains a linear equation $g(\boldsymbol{x}) = \boldsymbol{w}^T \cdot \boldsymbol{x} + b$. By querying $d+1$ linearly independent inputs, the attacker is able to create a system of linear equations and to calculate $\boldsymbol{w}$ and b exactly. Direct extraction can lead to an exact reconstruction of θ [102, 132, 207], as in the previous example. In such case, the functionality or fidelity goal is achieved by design.

Attacks either...

...reconstruct the model directly or...

In the second group, the adversary learns a surrogate learning model [e.g., 41, 102, 107, 152, 158, 207]. This group is primarily used for accuracy or fidelity extraction [102]. In general, the attack is based on the following steps [107]. Starting from an initial surrogate model and dataset, the adversary *repeatedly* (i) creates new samples (or selects samples), (ii) collects their predictions from the original classifier, and (iii) uses the improved dataset to train the surrogate model. Various strategies are explored to guide the queries [e.g., 152, 158]. For instance, the adversary iteratively creates new synthetic samples where her current surrogate model is the most uncertain [158]. In this way, the model can benefit the most from the queried samples and their predictions.

...train a surrogate learning model.

Defenses either change outputs...

DEFENSES Defenses can be divided into two groups. As first group, the defender changes the *model output*. This group, in turn, can be divided into several sub-groups.

- *Output truncation.* Instead of detailed prediction scores for all classes, the model may only return truncated scores [152, 207]. That is, only the top-k scores are returned or the scores are rounded. In the most vigorous case, the model only returns the predicted class label. However, research has demonstrated that this defense sub-group has little impact [152, 207].
- *Randomization.* Second, the model can use randomized outputs near the decision boundary [233]. The intuition is that the adversary tries to recover the decision boundary, so that her attacks will operate around this boundary [e.g., 132, 207]. Zheng et al. [233] create a security margin around the boundary and return randomized responses if a query lies in this margin.
- *Laying a false trail.* Third, the model can intentionally mislead the adversary's computations to obtain a surrogate model [121, 153]. Orekondy et al. [153], for example, modify the output scores in such a way that the attacker's gradient with respect to θ maximally deviates from the correct gradient during training. At the same time, the changed scores still convey the predicted classes correctly, thereby remaining useful for benign queries.

...or analyze the query sequences.

The next group is based on a *stateful defense* as active countermeasure. Instead of analyzing each query separately, the defender tries to identify sequences of malicious queries—similar to the stateful defense against adversarial examples. For instance, a cloud service providing machine learning as a service may monitor incoming queries for patterns indicative of model-extraction attacks. Juuti et al. [107], for example, analyze the distance between queries. If this distance deviates too much from the distribution of benign queries, an attack is detected. In Section 5.3.2, we examine another strategy that observes the closeness of queries to the decision boundary to identify malicious sequences. Note that stateful defenses against adversarial examples and model extraction share the same conceptual idea, but may have to identify different attack sequences that are used for adversarial examples and model extraction.

Finally, adding watermarks to learning models can be seen as an additional group of defense [e.g., 3, 118, 202, 211, 232]. Yet, this can only detect a model theft afterwards and not mitigate or prevent model extraction.

2.2.4 *Hyperparameter Extraction*

In addition to the model parameters θ, an adversary can also reconstruct the used hyperparameters [150, 217]. Remember that possible hyperparameters are the number of decision trees in a random forest or the number of layers in a neural network. For example, Oh et al. [150] present a method to recover neural network attributes. The adversary operates in a black-box threat model with the ability to send queries and observe scores $g(\cdot)$. The training data are also known by the adversary in the experiments. The idea is to learn a new metamodel that predicts the hyperparameters given the prediction vectors from multiple queries to the original model. In particular, the adversary submits k queries. The outputs of the original model are concatenated and provided to the metamodel as input, which predicts the hyperparameters. To increase the success rate, an additional method is presented where the k query inputs are specifically crafted to reveal more information about the hyperparameters. This method is evaluated in the image domain with a one-to-one relation from $\mathcal{Z}$ to $\mathcal{F}$. It is yet open how inputs can be realized in $\mathcal{Z}$ in the general case with no one-to-one relation (as examined in Chapter 3).

An adversary can also reconstruct hyperparameters.

2.2.5 *Poisoning Attack*

In machine learning, the training process is one of the most critical steps due to the impact on all subsequent applications. At this stage, poisoning attacks allow an adversary to change the overall model behavior [e.g., 25, 111] or to obtain targeted responses for specific inputs [e.g., 92, 128, 184] by manipulating the training data or learning model. The attack thus impacts the *integrity* of the training data or model. Such attacks need to be considered in multiple scenarios. First, an adversary might have direct access to the data or model as insider or because the training process is outsourced. Second, a manipulation is possible in transfer-learning where a pre-trained teacher model is made publicly available and users re-train it for a new task. Third, a learning model is trained or continuously updated with external data.

In poisoning attacks, adversaries change the training data or learning model...

Poisoning attacks can be roughly divided into *training-only attacks* and *backdoor attacks* [87]. In the former case, the training data or the model is manipulated, so that the changed behavior applies to all inputs later at deployment time. Thus, there is usually no need to change an input at deployment time, hence the term training-only [87]. Figure 2.10 illustrates the attack with an example from the malware domain. By changing the training dataset, the adversary can control the decision boundary that is learned afterwards. At deployment time, her actual malware sample is misclassified—without modifying it. We will examine another example for a training-only attack in Chapter 4 and Appendix C.1, where the adversary mixes a target image with training images to change the boundary for this particular sample. In general, attacks have been successfully applied in multiple applications, such as network anomaly detection [62, 111], signature generation [149], and crowdsourcing systems [219].

...with the goal to modify the overall model behavior or...

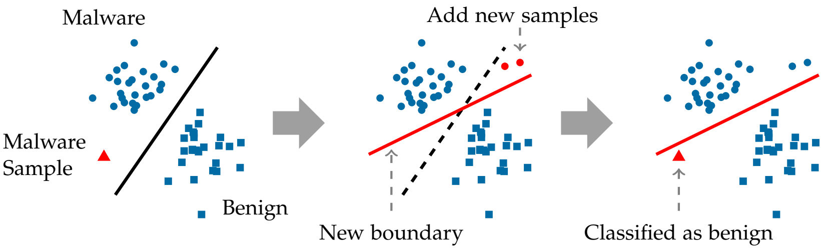


Figure 2.10: Training-only poisoning attack. Left: The adversary wants to evade the detection of her malware sample. Middle: She inserts new training samples, so that another, more favorable classification boundary is created after re-training. Right: The output for her malware sample is changed without modifying the sample.

...to embed a backdoor that can be turned on and off at deployment time.

Another type of poisoning attacks are *backdoor attacks* [e.g., 92, 128, 203, 230]. At training time, the adversary injects a backdoor into a learning model such that this model associates a trigger with a target label. At deployment time, the model behaves normally for benign inputs in the absence of the trigger, but returns the target label if (and only if) the trigger is present in the input. In contrast to training-only poisoning attacks, a backdoor trigger allows an adversary to switch the misbehavior on or off at deployment time by adding or not adding the trigger. Figure 2.11 exemplifies the attack in the image domain with the backdoor method by Gu et al. [92]. We explore this attack in more detail in Chapter 4 and Appendix C.1. Nevertheless, multiple defenses have also been examined against backdoor attacks [e.g., 44, 51, 93, 126, 127, 218]. For instance, the key idea of Wang et al. [218] is that a hidden backdoor provides a short link to its covered target class with relatively few modifications. Only a small, bounded pattern needs to be embedded, as in Figure 2.11. On the contrary, without backdoor, more substantial modifications are required to obtain the target class for any input. A backdoor can thus be detected by checking for such a short link in the learning model.

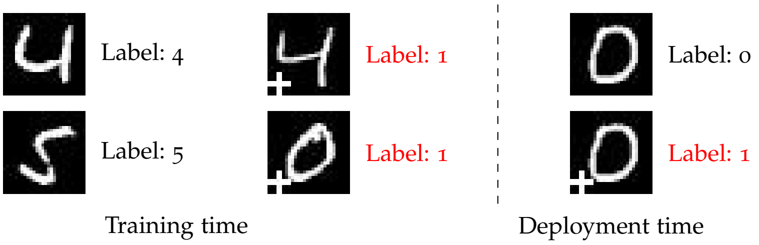


Figure 2.11: Backdoor attack. By changing a few training samples, the learning model will also associate a trigger (a cross here) with the label 1. Without trigger, the learning model behaves normally at deployment time. With trigger, it predicts the target label 1.

2.2.6 *Membership Inference*

An adversary infers if an object was part of the unknown training dataset of a given classifier—solely by relying on the output of the classifier [186]. The underlying idea is that classifiers behave differently on an input that has been used for their training compared to an input obtained for the first time [186]. Membership inference can cause a serious privacy violation. For instance, an adversary could infer that a specific person participated at a medical study by being present in the training dataset. The primary root cause of membership inference attacks is overfitting [186, 231].

With membership inference, an adversary finds out if a sample was used for training.

The adversary operates in a gray-box or black-box threat model [99]. In both cases, she can only send queries to the classifier and typically obtains the prediction scores $g(\cdot)$. She has no knowledge about the model parameters. Neither has she access to the original training data. Yet, she might have a surrogate dataset with the same distribution [99, 180, 186], exploit the query access to collect a synthetic dataset with the same distribution [186], or even use a dataset with a different distribution [180]. In the gray-box scenario, she has more information, such as the kind of learning algorithm or used hyperparameters.

Membership inference attacks were introduced by Shokri et al. [186], and refined in multiple further works [99, 123, 180, 194]. To obtain an intuition how membership inference works, let us shortly consider an attack by Salem et al. [180]. The adversary first derives a threshold for the usual confidence of non-members. To this end, she creates random objects that are unlikely members, queries their confidence values and computes an upper bound as threshold. If the confidence of an object under investigation is above this threshold, it was likely used for training.

Various refinements of the attack...

Defenses can be divided into three groups [99]. First, regularization-based methods try to counter overfitting [see 123, 180, 186]. Second, similar to model-extraction defenses, the confidence scores can be changed [104, 186]. For instance, the output can be truncated [186]. Alternatively, the defender can add noise to the output such that it specifically deceives the adversary's system for membership inference [104]. In other words, the defender herself creates an adversarial example of the output vector [104]. Third, privacy-preserving machine learning methods should be robust by construction [see 186].

...and multiple defenses are available.

2.3 SUMMARY

We are now equipped with a basic knowledge of machine learning and its general attack surface. The following text box shortly summarizes the main takeaways in this chapter.

Main Takeaways.

1. Machine learning allows us to automatically find patterns and relations in data. The data are given as real-world objects z in a problem space $\mathcal{Z}$. For instance, z might be a source code, $\mathcal{Z}$ contains all possible source codes.
2. The learning process takes place along a pipeline with three stages: mapping, training, and output. The overall classification function c_f is given by $c_f(z) = (f \circ \phi \circ \rho)(z)$. With scores, we obtain $c_g(z) = (g \circ \phi \circ \rho)(z)$.
3. The mapping $\phi \circ \rho$ from problem to feature space is the first stage: $\rho : \mathcal{Z} \longrightarrow \mathcal{Z}$ preprocesses $z \in \mathcal{Z}$, then $\phi : \mathcal{Z} \longrightarrow \mathcal{F}$ generates a d-dimensional feature vector $\boldsymbol{x}$ in the feature space $\mathcal{F}$. This stage transforms real-world objects into a suitable format for machine learning.
4. The training stage leads to a learning model θ with a discriminant function $g : \mathcal{F} \longrightarrow \mathbb{R}^{|\mathcal{Y}|}$. It takes $\boldsymbol{x}$ and returns the scores for all classes. The last stage defines the output, for instance, the scores via $g(\boldsymbol{x})$ or class labels via $f(\boldsymbol{x})$, i. e., $f : \mathcal{F} \longrightarrow \mathcal{Y}$.
5. Machine learning itself can be attacked. The research field of adversarial machine learning is concerned with the theory and practice of machine learning under an adversary.
6. An attack is possible against any component of the learning pipeline with adversarial examples, model extraction, hyperparameter extraction, poisoning, adversarial preprocessing, and membership inference.
7. The relation between $\mathcal{Z}$ and $\mathcal{F}$ (via $\phi \circ \rho$) lays the ground for the whole pipeline and is therefore essential to attacks as well. Yet, it has received little attention in research and is thus comprehensively explored in this thesis.

3

ATTACK IN THE PROBLEM SPACE

Equipped with basic knowledge of machine learning and its general attack surface, we can start to systematically explore the relation between the problem and the feature space of machine learning in the security context. Whenever an adversary has to create a real object, such as a source code or PDF file, she jointly operates in both spaces. She has to realize the object in the problem space and needs to attack the classifier in the feature space where the targeted learning model is located. In most application areas in security, however, the mapping from problem to feature space is *not bijective*, i. e., not injective and not surjective [162]. It means that there is no one-to-one mapping between problem and feature space. This introduces a non-trivial hurdle for an attack. For instance, an adversary might identify features that create the wanted classification. Yet, she cannot simply map this vector back to a real object in the problem space as the feature mapping is not invertible. The other way round, although a mapping from problem to feature space exists, the feature outcome may not be predictable or controllable due to side effects. To rename a source code variable, for example, an adversary has to adapt all its locations. This may induce side effects for the feature vector that cannot be computed in advance.

Creating real adversarial objects...

...is challenging due to the relation between $\mathcal{Z}$ and $\mathcal{F}$...

In this chapter, we thoroughly study this relation between problem and feature space. We will see that in order to create a real object, an adversary has to conduct her attack in the problem space, guided by the feature space. This chapter focuses on creating adversarial examples, but the gained insights also concern other attack procedures, such as model extraction and poisoning.

...and requires a problem-space attack that we examine in general and...

To strengthen the insights, we will analyze and conduct the attacks in the context of source code attribution. This application domain is well suited, since the respective constraints and challenges also apply to other application domains, as discussed in this chapter. The empirical results show that adversarial examples can imitate the coding style of developers with high accuracy. Consequently, this chapter does not only illustrate how adversarial learning can be conducted when the problem and the feature space are disconnected, but also pinpoints weaknesses in authorship attribution.

...in the context of source code attribution.

In summary, this chapter discusses the following major aspects:

- *Problem-space attack.* We systematically explore the creation of real adversarial examples in security-sensitive domains. The developed framework includes challenges, constraints, and search strategies. A strategy based on Monte-Carlo Tree Search (MCTS) that guides the creation of real adversarial examples is proposed.

- *Adversarial learning on source code.* We examine the first automatic attack against authorship attribution of source code, which includes targeted as well as untargeted attacks against two state-of-the-art attribution methods.
- *Large-scale evaluation.* An empirical evaluation on a dataset of 204 programmers demonstrates that manipulating the attribution of source code is possible in the majority of the considered cases.

Before considering problem-space attacks, let us briefly review the design of methods for authorship attribution in the following.

3.1 AUTHORSHIP ATTRIBUTION OF SOURCE CODE

To identify the author of source code,...

The learning task in authorship attribution is given by the problem space $\mathcal{Z}$ that contains all possible source codes and a finite set $\mathcal{Y}$ of authors. These authors have developed programs in $\mathcal{Z}$ and need to be identified. Authorship attribution is then the task of identifying the author $y \in \mathcal{Y}$ of a given source code $z \in \mathcal{Z}$ using the classification function c_f such that $c_f(z) = y$. In line with the majority of previous work, it is assumed that the programs in $\mathcal{Z}$ can be attributed to a single author, as the identification of multiple authors is an ongoing research effort [see 64, 143].

Equipped with this basic notation, we proceed to examine the main building blocks of current methods for authorship attribution along the machine learning pipeline given by $c_f(z) = (f \circ \phi \circ \rho)(z)$. We start with the mapping $\phi \circ \rho$ of source code from the problem space $\mathcal{Z}$ to the feature space $\mathcal{F}$, followed by the application of machine learning with the attribution output f.

3.1.1 *Mapping from Code to Features*

As described in Section 2.1.2, the mapping consists of two steps: the preprocessing $\rho : \mathcal{Z} \longrightarrow \mathcal{Z}$ and feature mapping $\phi : \mathcal{Z} \longrightarrow \mathcal{F}$.

...an attribution method preprocesses the code first and...

The preprocessing of source code can be necessary to account for artifacts. In general, artifacts can create shortcuts for separating classes without actually solving the learning task and should be prevented where possible [see 11]. In the case of authorship attribution, the Google Code Jam (GCJ) programming competition [90] is commonly used for evaluation [e.g., 2, 6, 36]. This dataset is reported to have artifacts [11]. Participants reuse their personalized coding templates across the challenges, for instance, with C++ macros. Understandably, this is the result from the nature of the competition, where authors try to solve challenges quickly. These coding templates are often not used, but only added in case they are needed. Models are then learned, that strongly rely on the coding templates as highly discriminative patterns. In consequence, the code attribution is based on artifacts rather than on deeper coding style. From an adversarial perspective, artifacts can also simplify an attack if the adversary can change the classification by exploiting only a few artifacts that are used for identifying a particular developer. In this thesis, we are interested in generating adversarial examples of code where *no* shortcuts, such as artifacts, are exploitable to simplify the attack. As a first step to mitigate artifacts, macros are therefore expanded as preprocessing in the evaluation in Section 3.5. For the same reason, code formatting tools are used as preprocessing to normalize source code. Otherwise, layout features would also provide a simple attack surface for an adversary. This problem becomes clearer in the paragraph after the next when we examine layout features.

The feature mapping ϕ is the second step. The coding habits of a developer can manifest in a variety of stylistic patterns. Consequently, methods for authorship attribution need to extract an expressive set of features from source code that serve as basis for inferring these patterns. In the following, we examine the major types of these features that can be used to define ϕ. The code sample in Figure 3.1 is used as a running example throughout the chapter.

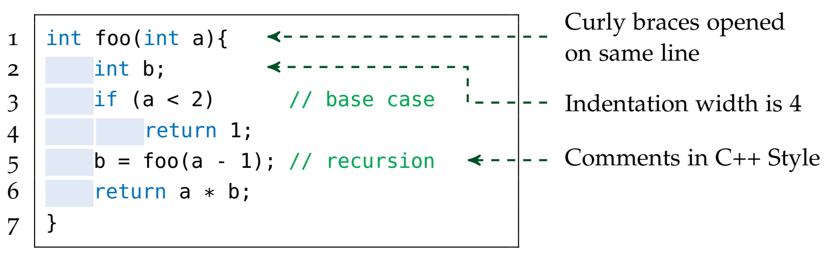
...then extracts various features, such as...

```
int foo(int a){
    int b;
    if (a < 2)      // base case
        return 1;
    b = foo(a - 1); // recursion
    return a * b;
}
```

Figure 3.1: Illustrative code sample as running example in this chapter.

LAYOUT FEATURES Individual preferences of a programmer are often reflected by the layout of the code. Corresponding features are thus a simple way to characterize coding style. Examples for such features are the indentation, the form of comments and the use of brackets. Figure 3.2 highlights the layout features from the running code example: Curly braces are opened on the same line, the indentation width is 4, and comments are provided in C++ style.

...layout patterns,...



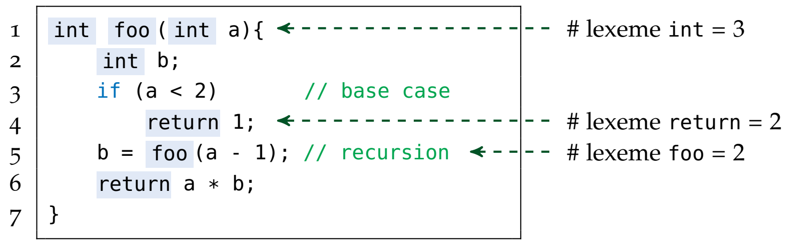
```
int foo(int a){
    int b;
    if (a < 2)      // base case
        return 1;
    b = foo(a - 1); // recursion
    return a * b;
}
```

Figure 3.2: Layout features from code sample in Figure 3.1.

Layout features are trivial to forge by an adversary. For instance, stylistic patterns in the layout can be easily unified by using tools for code formatting, such as `clang-format` [55]. At the same time, the impact of layout features on the attribution accuracy is marginal, as an empirical evaluation on the dataset from Section 3.5 shows. Therefore, a defender can be expected to omit layout features or to apply code formatting tools herself. Hence, we *do not* consider layout features in this thesis. In this way, attacks are examined under a more difficult scenario where no layout features are exploitable.

...string-based patterns covering keywords & symbols, and...

LEXICAL FEATURES A more advanced type of features can be derived from the lexical analysis of source code. In this analysis stage, the source code is partitioned into so-called *lexemes*, tokens that are matched against the terminal symbols of the language grammar [4]. These lexemes give rise to a strong set of string-based features jointly covering keywords and symbols. Figure 3.3 highlights some lexemes from the running code example: the frequency of the lexeme `int` is 3, while it is 2 for the lexeme `foo`. These features can reflect a developer's preference for function names or data types.



```
int foo(int a){
    int b;
    if (a < 2)      // base case
        return 1;
    b = foo(a - 1); // recursion
    return a * b;
}
```

Figure 3.3: Lexical features from code sample in Figure 3.1.

In contrast to code layout, lexical features cannot be easily manipulated, as they implicitly describe the syntax and semantics of the source code. While the lexeme `foo` in the running example could be easily replaced by another string, adapting the lexeme `int` requires a more involved code transformation that introduces a semantically equivalent data type. Such a transformation is introduced in Section 3.4.3.

...syntactic patterns from the AST.

SYNTACTIC FEATURES The use of syntax and control flow also reveals individual stylistic patterns of programmers. These patterns are typically accessed using the abstract syntax tree (AST), a basic data structure of compiler design [4]. As an example, Figure 3.4 shows a simplified AST of the running code example. The AST provides the basis for constructing an extensive set of syntactic features. These features can range from the specific use of syntactic constructs, such as unary and ternary operators, to generic features characterizing the tree structure, such as the frequency of adjacent nodes. In Figure 3.4, there exist 21 pairs of adjacent nodes including, for example, `(func foo)`⟶`(arg int)` and `(return)`⟶`(1)`.

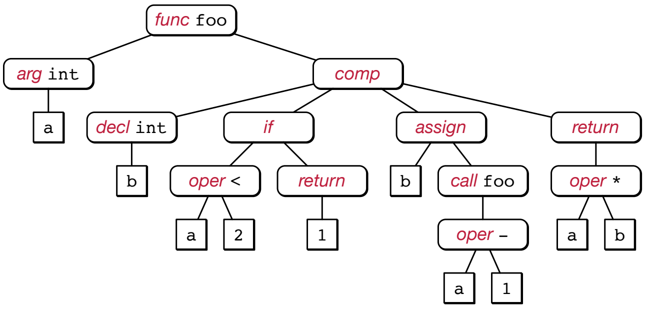


Figure 3.4: Abstract syntax tree (AST) for code sample in Figure 3.1 [169].

Manipulating features derived from an AST is challenging, as even minor tweaks in the tree structure can fundamentally change the program semantics. As a consequence, transformations to the AST need to be carefully designed to preserve the original semantics and to avoid unintentional side effects. For example, removing the node pair `(decl int)`→`(b)` from the AST in Figure 3.4 requires either replacing the type or the name of the variable without interfering with the remaining code. In practice, such transformations are often non-trivial and we examine the details of manipulating the AST in Section 3.4.3.

FROM CODE TO VECTORS The three feature types (layout, lexical, syntactic) provide a broad view on the characteristics of source code and are used by many attribution methods as the basis for applying machine-learning techniques [e.g., 2, 6, 36, 64]. The extracted features can then be used for the mapping of code to a vector space: $\phi : \mathcal{Z} \longrightarrow \mathcal{F} = \mathbb{R}^d$. As example for such a mapping, recall the example in Figure 2.2 in Section 2.1.2. Furthermore, the mapping may include the usage of the TF-IDF weighting which adapts the map to account for the frequency of each feature in the training dataset [2, 36]. Moreover, feature selection algorithms can be used to reduce the number of possible features afterwards [2, 36].

3.1.2 *Multiclass Classification*

After defining the overall mapping $\phi \circ \rho$, we can apply machine learning for identifying the author of a source code. Typically, this is done by training a *multiclass classifier* $g(\boldsymbol{x})$ that returns scores for all authors $\mathcal{Y}$. Different learning algorithms have been used for constructing the multiclass classifier g, as for example, support vector machines [160], random forests [36], and recurrent neural networks [2, 6]. As output, author scores can be returned with $g(\boldsymbol{x})$. Alternatively, the final label is only returned with $f(\boldsymbol{x})$ by computing the author with the highest score. Figure 3.5 visualizes the resulting multiclass setting in $\mathcal{F}$.

Finally, a multiclass classifier is trained and applied.

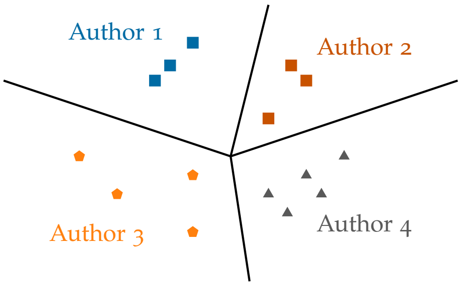


Figure 3.5: Multiclass classification in authorship attribution.

Putting all stages together, the overall attribution system is finally given by $c_g(z)$ that takes z as source code and returns scores for all authors, or by $c_f(z)$ that takes z and returns the finally predicted label y for z.

Attacking each of the possible learning algorithms individually is a tedious task and thus we examine a *black-box attack* for misleading authorship attribution in this chapter. This attack does not require any knowledge of the employed learning algorithm and operates with the output $c_g(z)$ only. Consequently, the approach is agnostic to the learning algorithm as the evaluation in Section 3.5 demonstrates.

3.2 PROBLEM-FEATURE SPACE DILEMMAS

Real adversarial examples have to be created...

...in the problem and the feature space, which is difficult...

With a basic understanding of authorship attribution, we are ready to investigate the creation of real adversarial examples. An adversary has to jointly operate in two spaces. On the one hand, she aims at attacking a classifier in the feature space. On the other hand, she requires an object z to be valid in the problem space. Both spaces are connected by $\phi \circ \rho$. This situation creates several challenges that an adversary has to consider for the design of her attack. In the following, we examine these challenges and use authorship attribution as representative example.

To begin with, the preprocessing ρ can complicate an attack. Yet, this depends on ρ and is application-specific. For source code, for instance, expanding macros can change the code considerably and thus override adversarial modifications. However, an attacker could expand macros herself before computing the necessary modifications and in this way avoid problems around the preprocessing. In other words, she can work with a sample where no preprocessing is necessary.

...due to the feature mapping from $\mathcal{Z}$ to $\mathcal{F}$. Several dilemmas arise:

The major challenge is, however, the feature mapping ϕ. In most application areas, ϕ is not bijective. In other words, there is no one-to-one correspondence between the problem space $\mathcal{Z}$ and the feature space $\mathcal{F}$. Hence, the adversary encounters multiple dilemmas that we examine in the following. The dilemmas are illustrated by Figure 3.6.

3.2.1 *Problem Space $\rightsquigarrow$ Feature Space*

Let us start with the problems in the direction from $\mathcal{Z}$ to $\mathcal{F}$.

Dilemma 1. *Modifications in $\mathcal{Z}$ and thus in $\mathcal{F}$ are limited.*

For instance, the modification of source code is limited. Any change in $\mathcal{Z}$ needs to preserve the syntax and semantics of the program. Arbitrarily removing or rewriting instructions is not possible, as this might produce invalid code. Moreover, we cannot always change a value to any target value without violating the language definition. The problem space defines possible values in $\mathcal{F}$. For instance, take a function-call node in the AST. The depth of this node cannot be changed to 1 if the call needs to be located under a function or method block, as for example for C++. In this case, a feature in $\mathcal{F}$ that measures the depth of a function call cannot be 1.

1) The problem space restricts possible changes and features.

Dilemma 2. *Target-oriented modification of z is difficult. Each change in z can impact a set of features in $\phi(z)$ as side effect.*

Figure 3.6 exemplifies the challenge that the targeted feature vector might not be achievable. Three reasons can be identified for the underlying dilemma.

2) Exactly achieving the wanted features is difficult, if...

- The smallest possible unit of change in $\mathcal{Z}$ can require multiple consolidation operations across z to ensure the validity. Consequently, multiple changes affect multiple features as side effect. If an attacker, for example, adapts the function name `foo` in the running code example in Figure 3.1, she has to consider all usages of the function, such as in line 5. Multiple features are then affected, such as the lexeme `foo` or the subtree under the node `assign` in the AST in Figure 3.4.

...multiple changes affect multiple features,...

- Another reason is the inherent feature correlation if multiple features are (partly) extracted from the same source. At this time, changing only a single statement affects multiple features as side effect. The exact amount of change is generally not controllable. For example, various lexical and syntactic features are extracted from the variable declaration `b` in line 2 of the code example in Figure 3.1. This includes the frequency of the lexeme `b` or the subtree under the node `comp` in the AST in Figure 3.4.

...a single change affects multiple features,...

- Even if an attacker managed to change a single feature only, post-processing steps in ϕ, such as a TF-IDF weighting, can again affect multiple features due to the normalization.

...normalization affects multiple features.

In general, such side effects cannot be computed in advance, which further complicates an attack [162].

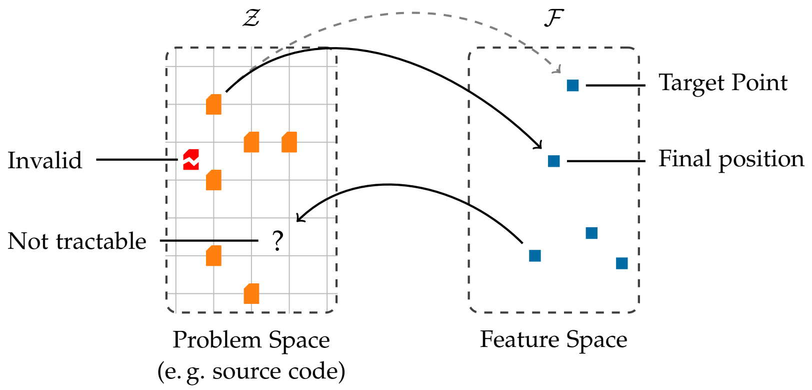


Figure 3.6: Problem-feature space dilemmas: Difficulty of moving in both directions between $\mathcal{Z}$ and $\mathcal{F}$. The grid in $\mathcal{Z}$ exemplifies its discrete nature.

Dilemma 3. *The same transformation in $\mathcal{Z}$ may modify different correlated features depending on the changed location in z.*

3) The impact of a modification varies across the input file.

A source code, for example, can contain multiple `for` loops that differ in their neighborhood, body, and iteration variables. A transformation changing a `for` loop can thus modify different correlated features with each loop. This increases the number of possibly changed features and makes it hard to compute the impact of a transformation in advance.

Taken together, Figure 3.6 illustrates the challenges: the attack operates in a discrete space with limited modifications, and it can be difficult to obtain any targeted feature vector. Appendix B.1 provides more insights on the dilemmas with a detailed example.

3.2.2 *Feature Space $\rightsquigarrow$ Problem Space*

4) The feature mapping is not invertible.

Any change to a feature vector $\phi(z)$ must ensure that there exists a valid object z in the problem space. Unfortunately, ϕ is not invertible in general and we encounter the following dilemma.

Dilemma 4. *Determining z from $\phi(z)$ is intractable for non-bijective feature maps, and it is impossible to directly apply techniques from adversarial learning that operate in the feature space.*

If we calculate the difference of two vectors $\phi(u) = \phi(z) - \phi(z')$, we have no means for determining the resulting source code u. Even worse, it might be impossible to construct u, as the features in $\phi(u)$ can violate the underlying programming language specification, for example, due to feature combinations inducing impossible AST edges.

3.2.3 *Generalization of Dilemmas*

These dilemmas have received little attention so far and a significant fraction of work on attacks has focused on scenarios where the problem and the feature space are mainly identical [see 26, 38, 156]. In these scenarios, changes in the problem space, such as the modification of an image pixel, have a one-to-one effect on the feature space, so that sophisticated gradient-based attacks can be applied (see Section 2.2.2).

By contrast, a one-to-one mapping between problem and feature space cannot be constructed in most security-sensitive areas, such as for text [7, 68, 79, 80, 124, 154], PDF [63, 137, 206, 227, 236], Windows Portable Executable (PE) [8, 112, 175, 176, 197], Android [48, 65, 91, 162, 229], and JavaScript [69]. Consequently, the previously discussed dilemmas are rather the rule than the exception—beyond source code attribution. To underline this insight, let us take a look at the following two application areas. To begin with, consider an attack with PDF files [236]: The file cannot be arbitrarily changed (Dilemma 1). Šrndić and Laskov [236] report the problem of feature correlation (Dilemma 2 and Dilemma 3). An inverse feature mapping does not exist (Dilemma 4). As another example, consider the text domain. An adversary cannot change text arbitrarily (Dilemma 1). She needs to preserve the grammar and meaning. Side effects can occur, for instance, if adding or modifying a single word requires adapting the whole sentence to consider the grammar (Dilemma 2 and Dilemma 3). The feature mapping may not be invertible (Dilemma 4) [e.g., 124].

These problems are rather the rule than the exception in most security domains.

3.3 FORMALIZATION OF PROBLEM-SPACE ATTACKS

The previous dilemmas show the difficulty to conduct an attack when problem and feature space are disconnected. Yet, an attack is possible, but the adversary has to realize her attack in the problem space $\mathcal{Z}$ while being guided from the feature space $\mathcal{F}$. Figure 3.7 illustrates

Thus, an adversary has to use...

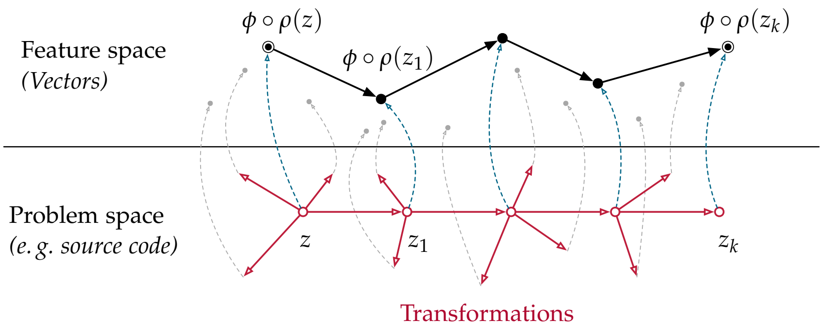


Figure 3.7: Schematic depiction of attack procedure in problem and feature space: The attack is realized by moving in the problem space while being guided from the feature space.

...a problem-space attack that can be...

this two-sided attack procedure. In this thesis, we refer to this attack as *problem-space attack* [162] due to the focus on $\mathcal{Z}$ and to differentiate it from approaches focusing on the feature space only. Note that a problem-space attack is also referred to as *physically realizable attack* in the context of physical objects [185].

Let us take a closer look at how an adversary can realize a problem-space attack. She has to rely on modifications in $\mathcal{Z}$ that we formally define as follows.

Definition 1. *A transformation $T : \mathcal{Z} \longrightarrow \mathcal{Z},\ z \mapsto z'$ takes a problem-space object z and generates a transformed version z'.*

A transformation can have the intention to *add*, *remove* or *change* a single feature or multiple ones. In general, multiple transformations can be necessary for an attack. To chain them together, we define a transformation sequence as follows:

Definition 2. *A transformation sequence $\mathbf{T} = T_k \circ \cdots \circ T_2 \circ T_1$ is the subsequent application of multiple transformations to a problem-space object $z \in \mathcal{Z}$.*

...defined by five constraints and a search strategy.

Given the problem-feature space dilemmas, an adversary has to consider two components when searching for a suitable sequence $\mathbf{T}$: constraints and a search strategy. In the following, we explore both components that have been developed in the context of source code attribution by Quiring et al. [169], and further extended and formalized in a broader context by Pierazzi et al. [162].

3.3.1 *Constraints of Problem-Space Attacks*

The constraints are:

A valid and realistic adversarial example needs to fulfill multiple constraints [162, 169]. They define the allowed movements in the problem space. In particular, five constraints are relevant, which are examined in the following. Table 3.1 provides an overview.

1) An attack does not change the behavior, which is achieved...

PRESERVED SEMANTICS An attack should not change the behavior of an object z. As an example, a modified malware should keep its functionality. In the case of code attribution, the generated source code by an attack should be semantically equivalent to the original code. That is, the two codes produce identical outputs given the same input. In general, semantic equivalence is undecidable [191]. As a result, an adversary has to rely on the following two methods.

...by construction...

First, a transformation T can preserve the semantics by construction. For instance, a `for` loop can be transformed into a semantically-equivalent `while` loop. Moreover, an adversary can exploit a semantic gap and add content in a PDF file such that it does not affect PDF viewers [236]. We will take a closer look on this technique with the fifth constraint later in this section. In general, the impact of transformations that change used content can be difficult to assess due to the

	Constraint	Description
●	Preserved semantics	Modifications do not change the behavior
●	Plausibility	Modifications are inconspicuous
○	Robustness to preprocessing	Adversarial example is robust against changes in preprocessing ρ
○	No exploitation of semantic gap	Adversarial example remains intact if semantic gaps are mitigated or closed
●	Available transformations	Possible modifications of object z in problem space

Table 3.1: Summary of constraints for problem-space attacks. The filled circle ● denotes a mandatory constraint, the circle ○ specifies a recommended constraint for advanced adversarial examples.

complexity of the application domain. If unexpected cases are possible due to modifications, an attacker can also use automated tests to verify the semantics after a transformation.

Second, an attacker can use random manipulations and only keeps mutated objects that pass functionality tests [e.g., 227]. In this case, invalid objects are possible by design and tests have to ensure the semantic equivalence.

...or with automated tests.

If automated tests are used, we need to be aware of their limitation. Although they can reasonably check the semantics, they cannot guarantee strict equivalence in all possible cases. For instance, exchanged API functions that provide the same functionality can differ in unexpected corner cases, such as when the memory is exhausted.

PLAUSIBILITY Depending on the application area, it can be necessary that an adversarial example is inconspicuous for a human user or analyst. In the context of source code, plausibility is important whenever the adversary wants to hide the modification of a source file, for instance, when blaming another developer. Trivial attacks, such as simply copying code snippets from one developer for impersonation or heavily obfuscating source code to avoid any attribution, generate implausible code and are easy to detect. When adversarial examples are generated in the image domain [38], plausibility corresponds to the aspect of imperceptibility. The predicted target class should not be visible in the attack image to avoid raising suspicion. A special note is necessary for malware that naturally contains suspicious modifications. In this case, plausibility for an adversarial example can mean that the modifications do not raise *further* suspicion.

2) An attack should be inconspicuous,...

Plausibility as a subjective constraint can be hard to measure automatically. The visibility of a perturbation in images depends on the respective content (see Section 4.5). Human analysts may judge differently if a code transformation is suspicious. To verify plausibility, an empirical user study can be conducted [e.g., 156, 169, 185].

...which can be measured with user studies.

ROBUSTNESS TO PREPROCESSING A strong adversarial example is robust to changes in the preprocessing ρ, that is, the attack should not fail if ρ is changed [162]. Therefore, an additional constraint can be to *not* exploit fragile features that are affected by ρ. Consider a digital image as example. A strong adversarial example should resist a change in the preprocessing, such as another denoising filter, compression or scaling algorithm [162]. As another example, layout features of source code can be easily normalized with tools for code formatting and a stronger adversarial example does not use them (see Section 3.1.1).

3) An attack is robust to changes in preprocessing.

NO EXPLOITATION OF SEMANTIC GAP An adversary can exploit a *semantic gap*, that is, a different view on z between the learning-based system and the actual application that uses z. In other words, she exploits a discrepancy between the feature mapping ϕ from $\mathcal{Z}$ to $\mathcal{F}$ and the actual application. A stronger adversarial example, however, does not rely on a semantic gap.

4) An attack does not depend on a semantic gap.

To understand this constraint, we first have to examine the different variations of a semantic gap. To begin with, an adversary exploits a semantic gap if she modifies z such that features cannot be correctly extracted by ϕ [48, 103, 206]. For example, Tong et al. [206] exploit that the PDF 1.2 standard also allows a hexadecimal ANSI-code to represent a character. This creates a semantic gap if the feature mapping does not consider this possibility, but PDF viewers handle it. As another type of semantic gap, the adversary can exploit the difference between the used part of z and extracted features ϕ [236]. This means the classifier either extracts too little or too much from z. In the former case, if the feature extraction skips areas in z, an adversary can add malicious functionality into these areas [e.g., 137]. In the latter case, the classifier extracts features from areas that are not relevant for the functionality of z. This allows an adversary to add content there [e.g., 112, 197, 236]. Figure 3.8 exemplifies this semantic gap for PDF files [236]. An adversary can inject specifically-crafted content between the reference table and trailer of a PDF file. This content is not used by the PDF reader, but by the feature extraction for machine learning. Finally, a semantic gap is also possible if the feature extraction does not consider the semantic difference between various parts of z, such as the difference between a code and debug section in a PE file [112]. This can allow simpler modifications.

A gap exists in different variations...

A semantic gap simplifies an attack considerably, since content can be added or changed with more degrees of freedom. Problem-space constraints become easier to realize. For instance, semantics are preserved automatically, if the changed area is not executed by the actual application. A semantic gap also has an impact on the search strategy which is introduced in the next section. Following a gradient from $\mathcal{F}$ can be easier, since less constraints from $\mathcal{Z}$ have to be considered [112]. Informally speaking, $\mathcal{Z}$ and $\mathcal{F}$ are moving closer

...and can simplify problem-space attacks considerably.

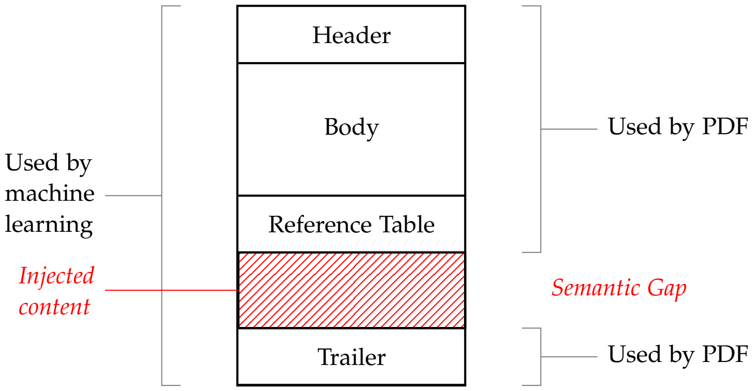


Figure 3.8: Visualization of a simplified PDF file structure and a possible semantic gap. The injected content does not influence the program functionality, but the learning-based system.

to each other. Appendix B.2 provides further insights why a semantic gap simplifies a problem-space attack.

As defenders might adjust their system,...

While it can be necessary to exploit a semantic gap to unveil weak spots in learning-based systems, a stronger attack does not rely on a semantic gap, similar to the preprocessing constraint. Although fixing a semantic gap is by no means trivial [103], a defender can be expected to consider this gap when improving the robustness of a learning-based system. Appended bytes, for example, can be mitigated by identifying the real end of a PE file [172].

...stronger adversarial examples do not exploit a semantic gap.

Therefore, we consider a fifth constraint in this thesis which has not been formalized in the literature so far: An adversarial example remains intact if semantic gaps are mitigated or closed. This constraint can involve considerable work for an adversary. The used parts of an object need to be transformed in such a way that the syntax and semantics are preserved. However, this constraint leads to a strong adversarial example, as characteristics deeper nested and relevant for the functionality of an object z are manipulated.

Remark. The constraints on the preprocessing and semantic gap do not mean that we should not evaluate the whole attack surface. In fact, attacks based on the preprocessing or semantic gap can indicate severe weaknesses of a learning-based system. In Chapter 4, for instance, this thesis demonstrates that preprocessing attacks on the mapping from problem to feature space are simple, but considerably effective attacks. Furthermore, an adversary can be expected to exploit simpler weak spots first. As an example, adversaries started with examining semantic gaps in adversarial learning competitions [172]. Therefore, Table 3.1 marks both constraints as optional. Still, for a thorough security analysis, we need to consider advanced adversarial examples that also fulfill these constraints.

Finally, let us examine the relation between preprocessing and semantic gaps. Preprocessing can help to mitigate a semantic gap, for instance, by removing appended bytes. Yet, it cannot generally fix a different view on z between the learning-based system and actual application. For instance, adjusting the feature mapping can also be necessary if areas—relevant for the functionality—are not extracted as features or if the semantic difference is not considered enough.

5) Modifications are restricted by the problem space...

AVAILABLE TRANSFORMATIONS This constraint defines the possible modifications of $z \in \mathcal{Z}$. These changes are limited, as outlined in Dilemma 1. In the source code domain, for instance, the available transformations are restricted by the programming language specification. A modification can add or replace variables or data types, but not misspell any token. In contrast, misspelling a word or using homoglyphs is possible for normal text [124].

...and the previous attack constraints.

In addition, the available transformations are restricted by the other previously outlined constraints. For instance, adding a large number of comments does not violate the language specification, but leads to implausible code and thus might not be possible to exploit. Using too many spelling mistakes in a text can also raise suspicion or even hinder its actual usage.

Transformations can be divided into three groups, depending on whether they *add*, *remove* or *change* parts in z. A prevalent strategy described in the literature is to use transformations that add content [e.g., 112, 236]. This has the advantage of preserving the semantics. Removing or changing content is more difficult to achieve, as it requires an understanding of the meaning and relevance of the content. For example, adding never-executed program code is guaranteed to not change the semantics, while rewriting code can destroy the program's functionality. However, adding a large amount of unused content can decrease the plausibility and be removable by preprocessing. On the contrary, equivalent content as replacement, such as synonyms in text or equivalent APIs in source code, may not raise further suspicion. These trade-offs further underline the interplay between the different constraints.

3.3.2 *Search Strategy of Problem-Space Attacks*

A search strategy is needed to guide the attack. It can...

The last needed piece for a problem-space attack is the search strategy that guides the transformations in the problem space $\mathcal{Z}$ towards the targeted classification. As the feature mapping ϕ is not invertible, we cannot directly use common gradient-based methods that proceed in the feature space $\mathcal{F}$ only. Therefore, we require a strategy that finds a solution in $\mathcal{Z}$ guided by $\mathcal{F}$. This strategy should account for side effects which are the additional features that were not intended to be changed (see Dilemma 2 and Dilemma 3).

	Problem-driven	**Feature-driven**
Derivative-based	—	Search is done in $\mathcal{F}$ to find a target direction by relying on derivatives (e. g. gradient).
Derivative-free	Search is primarily conducted in $\mathcal{Z}$, using the classifier as black-box oracle.	Search is conducted in $\mathcal{F}$ to find a target point or direction without relying on derivatives.

Table 3.2: Categorization of search strategies.

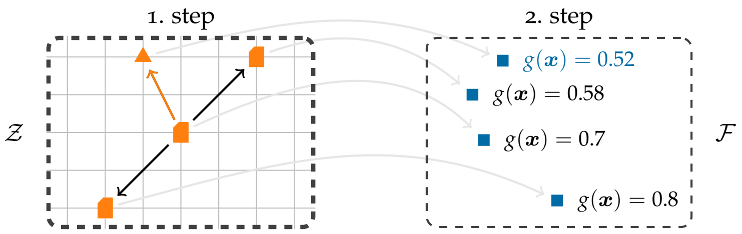
Different strategies are possible that differ in the way how they are guided by $\mathcal{F}$. Pierazzi et al. [162] categorize possible strategies into two types: *problem-driven* and *gradient-driven*. In this thesis, this categorization is extended and a second dimension is added: the usage of derivative information. This allows us to further capture the key characteristics of search strategies. Table 3.2 provides an overview of the categorization that we examine in the following.

PROBLEM-DRIVEN, DERIVATIVE-FREE SEARCH The search primarily operates in the problem space. It performs various transformations in $\mathcal{Z}$ while the classifier is used as a black-box oracle to obtain feedback for each transformation. The attack only relies on the output value of the classifier *without explicitly* using derivative information, such as the gradient. Figure 3.9a illustrates the basic procedure. The attack repeatedly conducts a transformation in $\mathcal{Z}$ and observes the classifier output. It then follows one or multiple directions where the output is promising. In this way, side effects are implicitly considered, as the attack is mainly driven by the problem space.

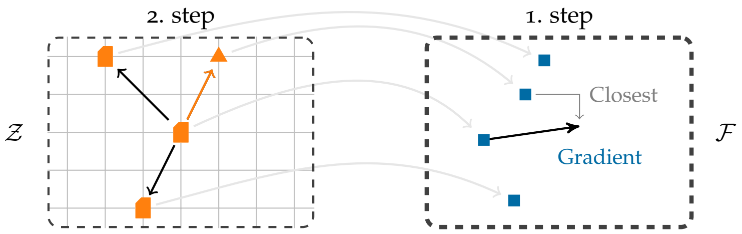
...mainly work in $\mathcal{Z}$ using the classifier as black-box oracle,...

A wide range of optimization techniques are applicable, such as hill climbing, tabu search, simulated annealing, or genetic programming [see 82]. They provide different advantages. For instance, genetic programming as a population-based approach improves multiple candidate objects at the same time. On the contrary, hill climbing improves a single object only. It is thus more affected by local minima than genetic programming, but may find a solution faster.

To obtain some intuition, let us consider how genetic programming can be leveraged to find adversarial examples of PDF [227]. The strategy starts with randomly mutating the initial malicious PDF file, so that a population of mutated PDF files are created. A functionality test is then used to check if the semantics are preserved. Each valid file z' is passed to the classifier $c_g(z')$ to obtain a malware score. A smaller score indicates a promising direction. Thus, the subsequent selection step only keeps the files with smaller scores for the population. In the next round, the kept files are again mutated and the previous steps are repeated. This strategy provides adversarial examples after a few rounds by following multiple promising directions.


1. step
2. step
$\mathcal{Z}$
$\mathcal{F}$
$g(\boldsymbol{x}) = 0.52$
$g(\boldsymbol{x}) = 0.58$
$g(\boldsymbol{x}) = 0.7$
$g(\boldsymbol{x}) = 0.8$

(a) Problem-driven, derivative-free attack



(b) Feature-driven, derivative-based attack

Figure 3.9: Comparison of search strategies. The triangle denotes the selected object in $\mathcal{Z}$. The problem-driven search first performs transformations in $\mathcal{Z}$ with the classifier as oracle, while the feature-driven approach first computes the gradient and then selects transformations in $\mathcal{Z}$. The feature-driven, derivative-free search is omitted.

In Section 3.4.4, we will explore MCTS as a novel search strategy for problem-space attacks. It allows evaluating multiple transformations in advance before deciding on the next move. This makes it particularly suitable for problem-space attacks, where the impact of a transformation may only be visible after multiple other transformations due to side effects or a piecewise constant classification function.

Finally, note that there is no problem-driven strategy that explicitly uses derivative information. In this case, the strategy would automatically be feature-driven, since it operates in $\mathcal{F}$ where derivative information are available only.

...or guide the search in $\mathcal{Z}$ from $\mathcal{F}$...

...with derivative information...

FEATURE-DRIVEN, DERIVATIVE-BASED SEARCH A feature-driven strategy is *explicitly* using information from the feature space[1]. It computes a target direction in $\mathcal{F}$, and then tries to find a way in $\mathcal{Z}$ towards this point or direction. Most feature-driven search strategies are derivative-based (see related work in Section 3.7). Commonly, they make use of the gradient in $\mathcal{F}$ to identify and select promising manipulations in $\mathcal{Z}$. Figure 3.9b illustrates the basic principle. The strategy first computes the gradient in $\mathcal{F}$ which represents the target direction. As ϕ is not invertible, the attack needs to work in the opposite direction: It tries various transformations in $\mathcal{Z}$ and selects the one where its mapped point in $\mathcal{F}$ is close to the gradient in $\mathcal{F}$. The distance can be measured, for instance, with different p-norms (see Appendix A).

---

[1] Pierazzi et al. [162] originally used the term gradient-driven which is here extended to feature-driven, derivative-based.

As an example for a gradient-based search, let us shortly consider an attack against text classification [154]. In a nutshell, for a given word in the text, the adversary computes the gradient in $\mathcal{F}$ to find the direction where the classifier score changes the most. Next, she iterates over a dictionary of words in $\mathcal{Z}$ to find the word whose feature vector is closest to the gradient. This word is then used to replace the original word.

FEATURE-DRIVEN, DERIVATIVE-FREE SEARCH Alternatively, an attack can also perform a derivative-free search in $\mathcal{F}$ to identify a target point or direction. A mimicry attack, for example, first identifies a target feature point from an already-existing object and then performs transformations in $\mathcal{Z}$ to obtain this point approximately [e.g., 236]. The classification score for this new object can be used to check how well the target region was reached.

...or without derivative information.

3.3.3 *Choice of Search Strategy*

Let us proceed with a closer look at when each strategy can be used. To begin with, the choice of the search strategy is restricted by the *threat model* and the *differentiability* of the learning model. In a black-box scenario, no knowledge about the learning-based system is available and consequently no direct derivative information can be used. A problem-driven search strategy is commonly used in this case [63, 124, 169, 227]. Yet, a feature-driven, derivative-based search strategy is also possible by relying on input-output queries. Even if only class predictions $c_f(z)$ are available, a gradient along the decision boundary can be approximated [45, 57]. Such an attack will be discussed in more detail in Chapter 5.

We choose the search strategy regarding...

...the threat model and...

In a white-box scenario, the adversary has full knowledge. In this case, the search strategy further depends on the differentiability of the learning model. If no gradient can be computed, a derivative-based approach is not applicable—even in a white-box scenario. A derivative-free strategy is then required. On the contrary, with available derivative information, all three kinds of search strategy are possible.

...differentiability of the learning model.

As a remedy for the black-box and non-differentiable white-box case, an attacker can also learn an own *surrogate model* that mimics the original model and is differentiable. In this way, she can make use of feature-driven attacks even in a black-box scenario or with a non-differentiable original model. Šrndić and Laskov [236], for instance, learn a differentiable surrogate SVM model to approximate a non-differentiable random forest, and conduct the attack on this surrogate model. Yet, there is a crux. A surrogate-based attack relies on the transferability property. That is, an adversarial example that misleads the surrogate model will probably mislead the original model as well [26, 200]. Hence, the success crucially depends on the quality of the surrogate model [102, 158] (remember the example in Figure 2.9 where the reconstructed decision boundary that optimizes the accuracy is less suited to find adversarial examples).

Last but not least, the choice of the search strategy can depend on whether multiple, different classifiers have to be attacked. A problem-driven strategy has the advantage of being agnostic of the used classifier. For instance, we examine a problem-driven attack against authorship attribution in Section 3.4 that is able to attack different classifiers, as the evaluation in Section 3.5 shows. On the contrary, a derivative-based strategy needs to be adapted to the used learning model.

Taken together, the search strategy needs to be selected with respect to the application scenario. The choice depends on various aspects, such as the threat model, differentiability of the learning model, and the goal to attack multiple classifiers.

3.3.4 *Refining the Search Strategy for Side Effects*

Furthermore, search strategies can be adjusted to account for side effects. First, a strategy can alternately switch between $\mathcal{Z}$ and $\mathcal{F}$. For instance, it first computes a gradient in $\mathcal{F}$, decides on the transformation in $\mathcal{Z}$, and then computes the feature vector in $\mathcal{F}$. Due to side effects, the novel feature vector will be shifted. As the search process is then conducted from this position, side effects are considered by correcting the feature vector in each step. Alternatively, given a target feature vector, the mimicry attack from Šrndić and Laskov [236] conducts all transformations at once. As a result, the impact of the feature dependencies has to be considered only once. Yet, the outcome is less predictable, so that the attack is repeated with multiple feature vectors as target.

Second, a search strategy can measure some side effects of each transformation on a minimal test object in advance [162]. Although the exact side effects will depend on the respective object, this pre-assessment can uncover related features. Hence, an attacker can better prioritize transformations that are tested or selected due to a more realistic mapping to $\mathcal{F}$ [162].

3.3.5 *Final Definition of Problem-Space Attacks*

Taken together, for a real adversarial example,...

We are now ready to put together the different pieces by formally defining a problem-space attack. Given a real object $z \in \mathcal{Z}$, the adversary wants to find a transformation sequence $\mathbf{T}$, so that the classifier c_f predicts the target label y^* for $\mathbf{T}(z)$.

...five constraints...

This attack has to fulfill the previously introduced problem-space constraints (see Table 3.1). Let Γ denote the set of these constraints. To specify that a transformation sequence $\mathbf{T}$ creates a valid problem-space object z with respect to the constraints Γ, the following notation is used: $T(z) \models \Gamma$ [162].

Similar to feature-space attacks (see Section 2.2.2), a problem-space attack can be defined as optimization problem. Without loss of generality, we again focus on the targeted variant in the following, where the classifier predicts the adversary's target class $y^* = y^t$. Comparable to the feature-space setting, two formulations can be differentiated. They primarily define the attack objective function ζ that specifies the direction of optimization for the search strategy.

...and a search strategy, which...

HIGH-CONFIDENCE ATTACK The adversary aims at increasing the classifier's confidence in y^t. The same objective functions ζ can be used as for feature-space attacks. For instance, Equation 2.18 becomes:

...can optimize the classifier confidence...

$$\zeta(\mathbf{T}(z)) = \max_{k \neq y^t}\{g_k(\boldsymbol{x}')\} - g_{y^t}(\boldsymbol{x}') \tag{3.1}$$
$$\text{with } \boldsymbol{x}' = \phi \circ \rho(\mathbf{T}(z))$$

As long as all constrains Γ are fulfilled, there is no need to define a distance metric $\mathcal{D}$ as additional constraint as done for feature-space attacks with Equation 2.20.

MINIMUM-MODIFICATION ATTACK The adversary aims at minimizing the required modifications to obtain y^t. As a result, this version requires the definition of a meaningful distance metric. If we measure the modifications in the problem space, the commonly used L_p distance is no longer applicable, since we do not operate in a vector space anymore. Although a meaningful distance in $\mathcal{Z}$ is rather application-specific, multiple options exist. If z is represented by a discrete structure, it is possible to use kernels [95], such as string kernels [129], tree kernels [56, 173], and graph kernels [81, 96]. Moreover, the edit distance can be used to compare strings in a text [106]. Finally, a general possibility is to count the number of transformations:

...or the required modifications,...

$$\mathcal{D}\left(z, \mathbf{T}(z)\right) = |\mathbf{T}| = |T_k \circ \cdots \circ T_1| = k. \tag{3.2}$$

This distance is a simple approximation for the amount of change and can be used as attack objective function ζ with

$$\zeta\left(\mathbf{T}(z)\right) = \mathcal{D}\left(z, \mathbf{T}(z)\right). \tag{3.3}$$

OPTIMIZATION PROBLEM Both formulations can be summarized as the following optimization problem.

$$\begin{aligned} \arg\min_{\mathbf{T}} \quad & \zeta\left(\mathbf{T}(z)\right) \\ \text{s.t.} \quad & c_f(\mathbf{T}(z)) = y^t, \\ & \mathbf{T}(z) \models \Gamma\,. \end{aligned} \tag{3.4}$$

Taken together, the goal is to find a transformation sequence **T** that optimizes $\zeta(\cdot)$ by manipulating the object z in such a way that the classifier c_f predicts the target label y^t and the problem-space constraints Γ are satisfied. The high-confidence version is obtained with Equation 3.1 as objective function, while the minimum-modification variant can be obtained with Equation 3.3. In order to find a solution to Equation 3.4, the adversary applies one of the three search strategies defined in Section 3.3.2. The choice of the search strategy depends on multiple aspects as defined in Section 3.3.3. The strategy can also account for side effects (Section 3.3.4).

...have to be considered.

3.4 ATTACKING AUTHORSHIP ATTRIBUTION

As next step, we apply this formalization on source code...

To strengthen our insights on problem-space attacks, let us now apply the previous formalization in the context of authorship attribution. We start with the threat model, as the concrete design of a problem-space attack depends on the assumed goals, knowledge and capabilities.

3.4.1 *Threat Model*

...in a black-box scenario...

...with the goal to...

The adversary is assumed to operate in a black-box scenario (see Section 2.2.1). That is, she has black-box access to an attribution method. She can send an arbitrary source code z to the method and retrieve the corresponding prediction $c_g(z)$. The training data, the extracted features, and the employed learning algorithm, however, are unknown to the adversary, and hence the attack can only be guided by iteratively probing the attribution method and analyzing the returned prediction scores. As part of the threat model, let us consider two types of attacks—*untargeted* and *targeted attacks*—that require different capabilities of the adversary and have distinct implications for the involved programmers.

...only mislead the attribution or...

UNTARGETED ATTACKS In this setting, the adversary tries to mislead the attribution of source code by changing the classification into *any* other developer. This attack is also denoted as *dodging* [185] and impacts the correctness of the attribution. As an example, a benign developer might use this attack strategy for concealing her identity before publishing the source code of a program.

...to impersonate other developers...

TARGETED ATTACKS The adversary tries to change the classification into a chosen *target* developer. This attack resembles an *impersonation* and is technically more advanced, as we need to transfer the stylistic patterns from one developer to another. A targeted attack has more severe implications: A malware developer, for instance, could systematically change her source code to blame a benign developer.

In particular, two scenarios are considered for targeted attacks: In the first scenario, the adversary has no access to source code from the target developer and thus certain features, such as variable names and custom types, can only be guessed. In the second scenario, the adversary is assumed to have access to two files of source code from the target developer. Both files are not part of the training or test dataset and act as external source for extracting template information, such as custom variable names. The number of two files is a conservative choice, since two is the minimal number that allows identifying a recurring pattern, for instance, in variable names.

In addition, a scenario is considered where the targeted attack solely rests on a separate training set, without access to the output of the original classifier. This might be the case, for instance, if the attribution method is secretly deployed, but code samples are available from public code repositories. In this scenario, the adversary can learn a surrogate model with the aim that her adversarial example—calculated on the surrogate—also transfers to the original classifier.

3.4.2 *Attack Constraints*

Misleading the attribution of an author can be achieved with different levels of sophistication. For example, an adversary may heavily obfuscate source code for dodging. This trivial attack, however, generates implausible code and is easy to detect. As a consequence, let us define the problem-space constraints from Section 3.3.1 such that it should be hard to identify manipulated source code. An overview over the defined constraints in the following is given by Table 3.3.

...while fulfilling all constraints.

PRESERVED SEMANTICS The source code generated by the attack needs to be semantically equivalent to the original code. The two codes should produce identical outputs given the same input. The developed code transformations in this thesis try to achieve this by construction. As they are based on the compiler frontend *Clang* [55], they have a full view on the source code. Still, unexpected cases are possible that are detected by using automated tests.

PLAUSIBILITY All transformations should change the source code such that the result is readable and plausible. The adversarial example should not be detectable as such under review by a human being. For this reason, no junk code or unusual syntax is included that normal developers would not use. Furthermore, a user study with domain experts is conducted with the finally created adversarial examples to verify that they comply with the plausibility constraint.

Constraint	Description
Preserved semantics	The adversarial source code remains semantically equivalent to the original code, achieved by the design of transformations and by automated tests.
Plausibility	The changed source code is readable and plausible. No junk code or unusual syntax is used. Plausibility is verified afterwards with a user study.
Robustness to preprocessing	Transformations do *not* change layout, as a defender could easily omit adversarially changed layout features.
No exploitation of semantic gap	Transformations directly alter the used source code. Comments are *not* used to add features.
Available transformations	Targeted transformations in problem space, that only change minimal aspects of source code, and enable adding, removing, or changing features.

Table 3.3: Summary of constraints for adversarial examples of source code.

ROBUSTNESS TO PREPROCESSING Layout features of source code, such as the tendency to start lines with spaces or tabs, can be easily normalized with tools for code formatting. If ρ is changed to normalize source code, an attack relying on layout transformations would fail. For this reason, the code transformations are restricted to the forgery of lexical and syntactic features. This leads to a stronger adversarial example, as no fragile layout features are exploitable to mislead the attribution.

NO EXPLOITATION OF SEMANTIC GAP The code transformations do not exploit a semantic gap, as they directly modify the used source code. Comments are not exploited. This leads to a stronger adversarial example, as deeper stylistic patterns are modified.

We continue with the code transformations as last constraint in their own section, as they require a comprehensive introduction.

3.4.3 *Available Code Transformations*

The automatic modification of code is a well-studied problem in compiler engineering and source-to-source compilation [4]. Consequently, we can build code transformations on top of the compiler frontend *Clang* [55], which provides all necessary primitives for parsing, transforming and synthesizing C/C++ source code. Note that code obfuscation methods are *not* used, since their changes are (i) clearly visible, and (ii) cannot mislead a classifier to a targeted programmer.

Following Definition 1, we denote a code transformation by T. It takes a source code x and generates a transformed version x'. While code transformations can serve various purposes in general [4], the fo-

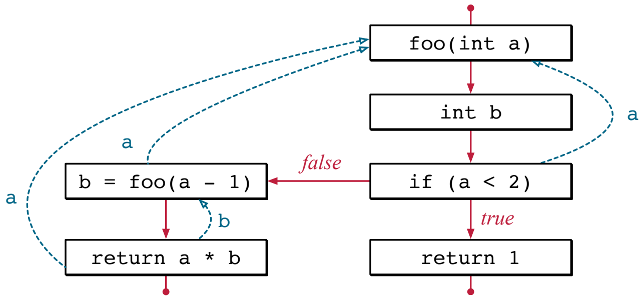


Figure 3.10: Control-flow graph (CFG) with use-define chains (UDCs) for the code sample in Figure 3.1 [169]. The control flow is shown in red (solid), use-define chains in blue (dashed).

cus lies on *targeted* transformations that modify only minimal aspects of source code. If multiple source locations are applicable for a transformation, the adversary is assumed to use a pseudo-random seed to select one location. To chain together targeted transformations, we denote a *code transformation sequence* by **T**—as defined in Definition 2.

To efficiently perform transformations, an attacker can use different program representations, where the AST is the most important one. To ease the realization of involved transformations, however, two additional program representations are usable that augment the view on the source code.

CONTROL-FLOW GRAPH WITH USE-DEFINE CHAINS The control flow of a program is typically represented by a control-flow graph (CFG) where nodes represent statements and edges the flow of control. Using the CFG, it is convenient to analyze the execution order of statements. For the attack, the CFG provided by Clang is further extended with use-define chains (UDCs). These chains unveil dependencies between usages and the definitions of a variable. With the aid of UDCs, it is possible to trace the flow of data through the program and to identify data dependencies between local variables and function arguments. Figure 3.10 shows a CFG with use-define chains.

DECLARATION-REFERENCE MAPPING In addition, a declaration-reference mapping (DRM) extends the AST by linking each declaration to all usages of the declared variable. As an example, Figure 3.11 shows a part of the AST together with the respective DRM for the running code example from Figure 3.1. This code representation enables navigation between declarations and variables, which allows an adversary to efficiently rename variables or to check for the sound transformation of data types. Note the difference between use-define chains and declaration-reference mappings. The former connects variable usages to variable definitions, while the latter links variable usages to variable declarations.

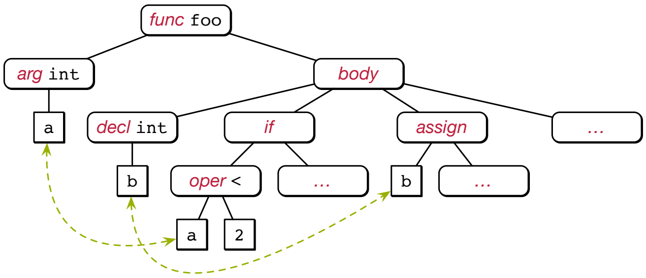


Figure 3.11: Abstract syntax tree (AST) with declaration-reference mapping (DRM) for the code sample in Figure 3.1 [169]. Declaration references are shown in green (dashed).

Transformation family	#	AST	CFG	UDC	DRM
Control transformations	5	•	•	•	
Declaration transformations	13	•			•
API transformations	9	•	•		•
Template transformations	4	•			•
Miscellaneous transformations	4	•			

Table 3.4: Implemented families of code transformations.

The necessary code transformations...

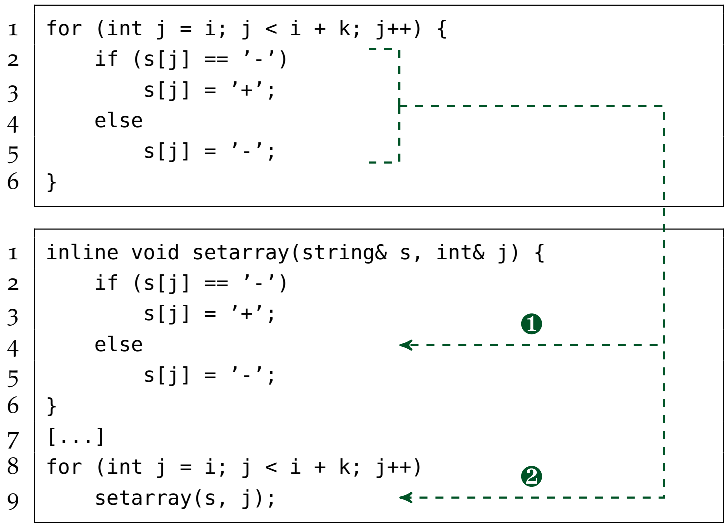
Based on these program representations, we can develop a set of generic code transformations that are suitable for changing different stylistic patterns. In this thesis, 35 transformers are implemented that are organized into five families. Table 3.4 provides an overview of each family together with the program representation used by the contained transformers. In the following, let us briefly examine each of the five families. For a detailed listing of all 35 transformations, the reader is referred to Table B.1 in Appendix B.3.

...adjust the control flow,...

CONTROL TRANSFORMATIONS The first family of source-to-source transformations rewrites control-flow statements or modifies the control flow between functions. In total, the family contains 5 transformations. For example, the control-flow statements `while` and `for` can be mutually interchanged by two transformers. These transformations address a developer's preference to use a particular iteration type. As another example, Figure 3.12 shows the automatic creation of a function. The transformer moves the inner block of the for-statement to a newly created function. This transformation involves passing variables as function arguments, updating their values and changing the control flow of the caller and callee.

...declarations,...

DECLARATION TRANSFORMATIONS The next family consists of 13 transformers that modify, add or remove declarations in source code. For example, in a widening conversion, the type of a variable is changed to a larger type, for example, `float` to `double`. This rewriting



```
for (int j = i; j < i + k; j++) {
    if (s[j] == ’-’)
        s[j] = ’+’;
    else
        s[j] = ’-’;
}
```

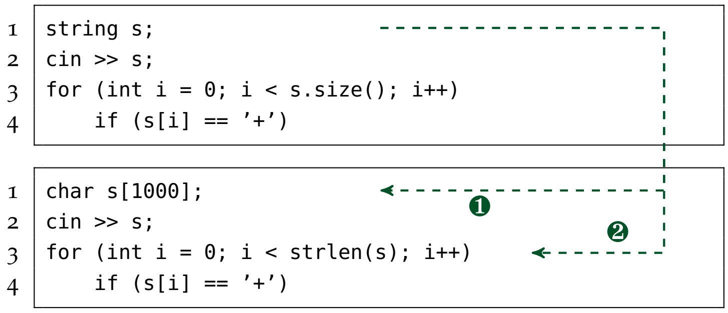
```
inline void setarray(string& s, int& j) {
    if (s[j] == ’-’)
        s[j] = ’+’;
    else
        s[j] = ’-’;
}
[...]
for (int j = i; j < i + k; j++)
    setarray(s, j);
```

Figure 3.12: Example of a control transformation. ❶ moves the compound statement into an own function and passes all variables defined outside the block as parameters. ❷ calls the new function at the previous location.

mimics a programmer’s preference for particular data types. Declaration transformations make it necessary to update all usages of variables which can be elegantly carried out using the DRM representation. Replacing an entire data type is a more challenging transformation, as we need to adapt all usages to the type, including variables, functions and return values. Figure 3.13 shows the replacement of the C++ `string` object by a conventional `char` array, where the declaration and also API functions, such as `size`, are modified. Note that in the implementation of the transformer for this thesis, the `char` array has a fixed size and thus is not strictly equivalent to the C++ `string` object.

API TRANSFORMATIONS The third family contains 9 transformations and exploits the fact that various APIs can be used to solve the same problem. Programmers are known to favor different APIs and

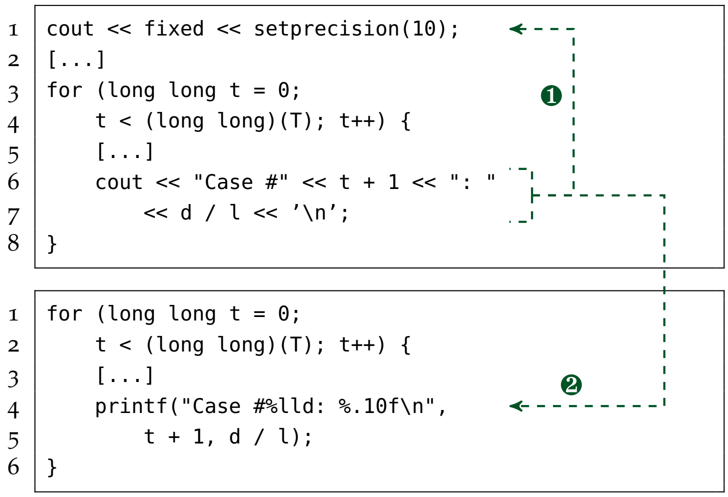
...used APIs,...



```
string s;
cin >> s;
for (int i = 0; i < s.size(); i++)
    if (s[i] == ’+’)
```

```
char s[1000];
cin >> s;
for (int i = 0; i < strlen(s); i++)
    if (s[i] == ’+’)
```

Figure 3.13: Example of a declaration transformation. ❶ replaces the declaration of the C++ `string` object with a `char` array, ❷ adapts all uses of the object.



```
cout << fixed << setprecision(10);
[...]
for (long long t = 0;
    t < (long long)(T); t++) {
    [...]
    cout << "Case #" << t + 1 << ": "
        << d / l << ’\n’;
}
```

```
for (long long t = 0;
    t < (long long)(T); t++) {
    [...]
    printf("Case #%lld: %.10f\n",
        t + 1, d / l);
}
```

Figure 3.14: Example of an API transformation. ❶ determines the current precision for output; ❷ replaces the C++ API with a C-style printf. The format specifier respects the precision and the data type of the variable.

thus tampering with API usage is an effective strategy for changing stylistic patterns. For instance, we can choose between various ways to output information in C++, such as `printf`, `cout`, or `ofstream`.

As an example, Figure 3.14 depicts the replacement of the object `cout` by a call to `printf`. The transformer first checks for the decimal precision of floating-point values that `cout` employs. To this end, the CFG is used to find the last executed `fixed` and `setprecision` statement. Next, the transformer uses the AST to resolve the final data type of each `cout` entry and creates a respective format string for `printf`.

...code patterns inferred from templates,...

TEMPLATE TRANSFORMATIONS The fourth family contains 4 transformations that insert or change code patterns based on given template information (remember the threat model in Section 3.4.1). For example, developers tend to reuse specific variable names, constants, and type definitions. If two template files are given for a target developer, information are extracted and used for transformations. Otherwise, default values that represent general style patterns are employed. For instance, variable names can be iteratively renamed into default names like `i`, `j`, or `k` until a developer's tendency to declare control statement variables is lost (dodging attack) or gets matched (impersonation attack).

...and generic patterns.

MISCELLANEOUS TRANSFORMATIONS The last family contains 4 transformations that conduct generic changes of code statements. For example, the use of curly braces around compound statements is a naive but effective stylistic pattern for identifying programmers. The compound statement transformer thus checks if the body of a control statement can be enclosed by curly braces or the other way round. In this way, a compound statement in the AST can be added or removed.

Another rather simple stylistic pattern is the use of return statements, where some programmers omit these statements in the `main` function and others differ in whether they return a constant, integer or variable. Consequently, a transformer is available that manipulates return statements.

3.4.4 *Search Strategy*

Equipped with different code transformations for changing stylistic patterns, we are ready to determine a sequence of these transformations for untargeted and targeted attacks. We aim at a short sequence, which makes the attack less likely to be detected. Thus, the minimum-modification formulation is applied. The number of transformations is used for the attack objective function. Formally, we adjust the general optimization problem from Equation 3.4 as follows:

We choose to minimize the modifications...

$$\begin{aligned} \arg\min_{\mathbf{T}} \quad & |\mathbf{T}| \\ \text{s.t.} \quad & c_f(\mathbf{T}(z)) = y^*, \\ & \mathbf{T}(z) \models \Gamma\,. \end{aligned} \tag{3.5}$$

For an untargeted attack, y^* is any other developer than the original developer y^s. For a targeted attack, y^* is a particular target author y^t. Due to the black-box threat model, a problem-driven derivative-free search strategy is used. As we explore attacks against multiple learning algorithms in the evaluation, this approach also has the advantage of being agnostic to the learning algorithm.

...with a problem-driven search strategy.

A challenge for a problem-space attack can be that the impact of a single transformation is only visible after multiple other transformations due to side effects or a (nearly) piecewise constant classification function. In the latter case, this means that a single transformation does not necessarily change the score of the classifier. This can happen unintentionally or intentionally, for instance, due to regularization [155] or the choice of learning model, such as a random forest. As a result, simple approximation techniques like Hill Climbing that only evaluate the neighborhood of a sample can fail to provide appropriate solutions.

As the impact of a change might not be directly visible,...

As a remedy, in this thesis, we explore a novel search strategy for problem-space attacks around the concept of *Monte-Carlo Tree Search (MCTS)*—a strong search algorithm that has proven effective in AI gaming with AlphaGo [187]. Similar to a game tree, the variant of MCTS in this thesis creates a search tree for the attack, where each node represents a state of the source code and each edge corresponds to a transformation T. By moving in this tree, we can evaluate the impact of different transformation sequences before deciding on the next move. It is thus particularly suitable for problem-space attacks, as side effects and constant classifier scores are implicitly considered.

...MCTS is proposed as strategy, so that we can simulate...

...multiple changes before deciding on the next one.

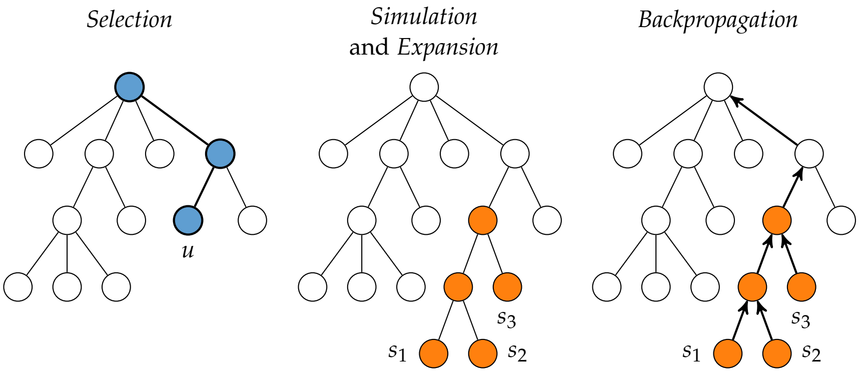


Figure 3.15: Basic steps of Monte-Carlo Tree Search (MCTS).

Figure 3.15 depicts the four basic steps of the algorithm: selection, simulation, expansion, and backpropagation.

SELECTION As the number of possible paths in the search tree grows exponentially, we require a *selection policy* to identify the next node for expansion. This policy balances the tree's exploration and exploitation by alternately selecting nodes that have not been evaluated much (exploration) and nodes that seem promising to obtain a better result (exploitation). Following this policy, we start at the root node and recursively select a child node until we find a node u which was not evaluated before. The left plot in Figure 3.15 illustrates this process. Appendix B.4 gives more information about the used selection policy.

SIMULATION & EXPANSION We continue by generating a set of unique transformation sequences with varying length that start at u. The length of each sequence is bounded by a predefined value. The experiments in Section 3.5 create sequences with up to 5 transformations. For each sequence, we determine the classifier score $c_g(\cdot)$ by providing the modified source code to the attribution method. The middle plot in Figure 3.15 exemplifies the step: three sequences are created where two have the same first transformation. Next, the respective tree nodes are created. As two sequences start with the same transformation, they also share a node in the search tree.

BACKPROPAGATION Finally, we propagate the obtained classifier scores from the leaf node of each sequence back to the root. During this propagation, we update two statistics in each node on the path. First, we increment a counter that keeps track of how often a node has been part of a transformation sequence. In Figure 3.15, we increase the visit count of node u and the nodes above by 3. Second, we store the classifier scores in each node that have been observed in its subtree. For example, node u in Figure 3.15 stores the scores from s_1, s_2 and s_3. Both statistics are used by the selection policy and enable us to balance the exploration and exploitation of the tree in the next iterations.

ITERATION These four basic steps are repeated until a predefined iteration constraint is reached. After obtaining the resulting search tree, we identify the root's child node with the highest average classifier score and make it the novel root node of the tree. We then repeat the entire process again. The attack is stopped if we succeed, if we reach a previously fixed number of iterations, or if we do not obtain any improvement over multiple iterations.

Appendix B.4 provides more implementation details on the developed variant of MCTS. Note that the algorithm is a black-box attack, as the inner working of the classifier c_g is not considered. Moreover, the proposed search strategy is not bounded to source code. It can also be applied to other application areas, as it is only based on generic transformations that correspond to the edges in the search tree.

3.5 EVALUATION

We proceed with an empirical evaluation of the developed problem-space attack and investigate the robustness of source code authorship attribution in a series of experiments. In particular, the impact of untargeted and targeted attacks on two recent attribution methods is investigated (Section 3.5.2 & 3.5.3). Finally, the attack constraints are verified in Section 3.5.4.

The attack is evaluated...

3.5.1 *Experimental Setup*

The empirical evaluation builds on the attribution methods developed by Caliskan et al. [36] and Abuhamad et al. [2], two recent approaches that operate on a diverse set of features and provide superior performance in comparison to other attribution methods. For the evaluation, the same experimental setup of the authors is used, their methods are re-implemented and a similar dataset is applied.

DATASET & SETUP The dataset consists of C++ files from the 2017 *Google Code Jam (GCJ)* programming competition [90]. This contest is divided into various rounds where several participants solve the same programming challenges. This setting enables us to learn a classifier for attribution that separates stylistic patterns rather than artifacts of the different challenges. Moreover, for each challenge, a test input is available that can be used to check the program semantics. Similar to previous work, eight challenges are selected and the respective source codes from all authors who solved these challenges are collected.

...on the GCJ competition with...

In contrast to prior work [2, 36], however, more stringent restrictions on the source code are set, that is, files are filtered out that contain incomplete or broken solutions. Furthermore, a preprocessing step formats each source code with `clang-format` and expands macros. The latter aims at removing artifacts that some authors introduce to write code more quickly during the contest. The final dataset consists of 1,632 files of C++ code from 204 authors solving the same 8 programming challenges of the competition.

The evaluation is based on a *stratified* and *grouped* k-fold cross-validation where the dataset is divided into $k-1$ challenges for training and 1 challenge for testing. This ensures that training is conducted on different challenges than testing. For each of the k folds, feature selection on the extracted features is performed before the respective classifier is trained as described in the original publications. Averaged results over all 8 folds are reported in the following.

...two state-of-the-art code authorship attribution methods.

IMPLEMENTATION The attribution methods and the attack are implemented on top of Clang [55], an open-source C/C++ frontend for the LLVM compiler framework. For the method of Caliskan et al. [36], the AST extraction is re-implemented and the proposed random forest classifier is used for attributing programmers. The approach by Abuhamad et al. [2] uses lexical features that are passed to a LSTM neural network for attribution. Table 3.5 provides an overview of both methods. For further details on the feature extraction and learning process, the reader is referred to the respective publications [2, 36].

As a sanity check, the experiments conducted by Caliskan et al. [36] and Abuhamad et al. [2] are reproduced on the previously described final dataset. Table 3.5 shows the average attribution accuracy and standard deviation over the 8 folds. The re-implementation enables differentiating the 204 developers with an accuracy of 90% and 88% on average, respectively. Both accuracies come close to the reported results with a difference of less than 6%, which can be attributed to omitted layout features and the stricter dataset.

3.5.2 *Untargeted Attack*

Misleading the attribution...

In the first experiment, let us investigate whether an adversary can leverage the developed problem-space attack to manipulate source code such that the original author is not identified. To this end, an untargeted attack is applied to each correctly classified developer from the 204 authors. The attack is repeated for all 8 challenges. Aggregated results are reported.

Method	**Lex**	**Syn**	**Classifier**	**Accuracy**
Caliskan et al. [36]	•	•	Random Forest	90.4% ± 1.7%
Abuhamad et al. [2]	•		LSTM network	88.4% ± 3.7%

Table 3.5: Implemented attribution methods and their reproduced accuracy. (Lex = Lexical features, Syn = Syntactic features)

	Success rate of attack		
Method	**Untargeted**	**Targeted T+**	**Targeted T-**
Caliskan et al. [36]	99.2%	77.3%	71.2%
Abuhamad et al. [2]	99.1%	81.3%	69.1%

Table 3.6: Performance of attack as average success rate. The targeted attack is conducted with template (T+) and without template (T-) information.

ATTACK PERFORMANCE Table 3.6 presents the performance of the attack as the ratio of successful evasion attempts. The attack has a strong impact on both methods and misleads the attribution in 99% of the cases, irrespective of the considered features and learning algorithm. As a result, the source code of almost all authors can be manipulated such that the attribution fails.

...is possible in almost all cases.

ATTACK ANALYSIS To investigate the effect of the attack in more detail, let us examine the ratio of changed features per adversarial sample. Figure 3.16 depicts the distribution over all samples. The method by Caliskan et al. [36] exhibits a bimodal distribution. The left peak shows that a few changes, such as the addition of include statements, are often sufficient to mislead attribution. For the majority of samples, however, the attack alters 50% of the features, which indicates the tight correlation between different features—as outlined by the problem-feature space dilemmas in Section 3.2. A key factor to this correlation is the TF-IDF weighting that distributes minor changes over a large set of features.

The attack changes multiple features, but...

In comparison, less features are necessary to evade the approach by Abuhamad et al. [2], possibly due to the higher sparsity of the feature vectors. Each author has 12.11% non-zero features on average, while 53.12% are set for the method by Caliskan et al. [36]. Thus, less features need to be changed and in consequence each changed feature impacts fewer other features that remain zero.

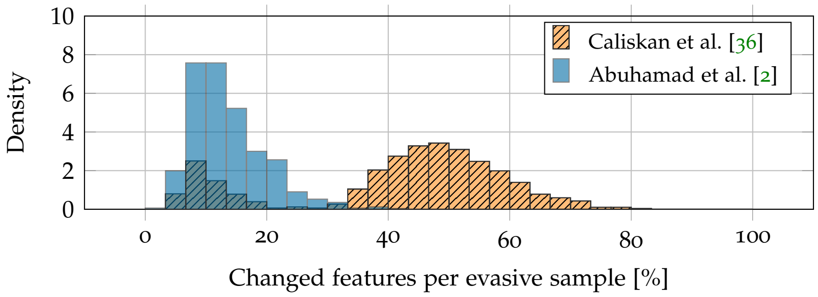


Figure 3.16: Untargeted attack: Histogram over the number of changed features per successful evasive sample for both attribution methods.

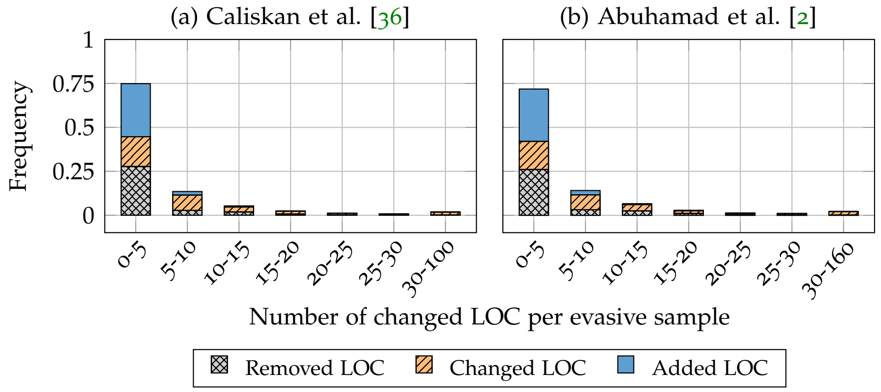


Figure 3.17: Untargeted attack: Stacked histogram over the number of changed lines of code (LOC) per successful evasive sample for both attribution methods. The original source files have 74 lines on average (std: 38.44).

...often only a few lines of code.

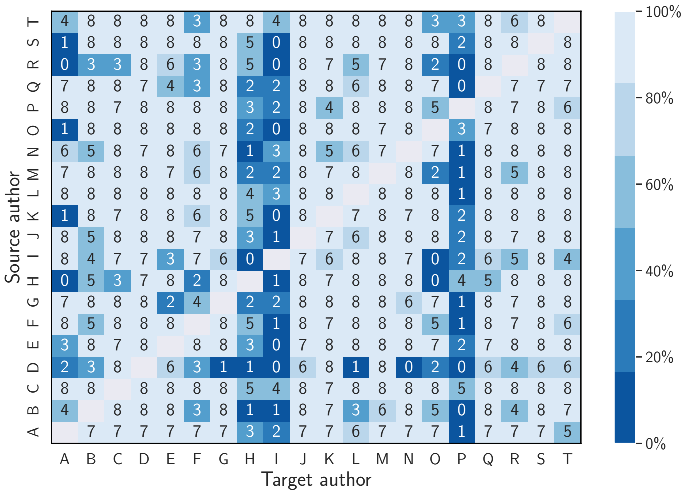
Although we can observe a high number of changed features, the corresponding changes to the source code are minimal. Figure 3.17 shows the number of added, changed and removed lines of code (LOC) determined by a context-diff with difflib [163] for each source file before and after the attack. For the majority of cases in both attribution methods, less than 5 lines of code are added, removed or changed. This low number exemplifies the targeted design of the code transformations that selectively alter characteristics of stylistic patterns.

SUMMARY The results from this experiment demonstrate that the untargeted attack severely impacts the performance of the methods by Caliskan et al. [36] and Abuhamad et al. [2]. We can conclude that other attribution methods employing similar features and learning algorithms also suffer from this problem and hence cannot provide a reliable attribution in presence of an adversary.

3.5.3 *Targeted Attack*

Impersonating a specific target developer...

We proceed to study the targeted variant of the attack. Thus, pairs of programmers are considered in the following, where the code of the source author is transformed until it is attributed to the target author. Due to the quadratic number of pairs, this experiment is performed on a random sample of 20 programmers. This results in 380 source-target pairs each covering the source code of 8 challenges. Table B.2 in Appendix B.5 provides a list of the selected authors. Let us start with the scenario where the adversary has access to template information. That is, she retrieves two samples of source code for each of the 20 programmers from various GCJ challenges—not part of the fixed 8 train-test challenges.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
T	4	8	8	8	8	3	8	8	4	8	8	8	8	8	3	3	8	6	8	
S	1	8	8	8	8	8	8	5	0	8	8	8	8	8	8	2	8	8		8
R	0	3	3	8	6	3	8	5	0	8	7	5	7	8	2	0	8		8	8
Q	7	8	8	7	4	3	8	2	2	8	8	6	8	8	7	0		7	7	7
P	8	8	7	8	8	8	8	3	2	8	4	8	8	8	5		8	7	8	6
O	1	8	8	8	8	8	8	2	0	8	8	8	7	8		3	7	8	8	8
N	6	5	8	7	8	6	7	1	3	8	5	6	7		7	1	8	8	8	8
M	7	8	8	8	7	6	8	2	2	8	7	8		8	2	1	8	5	8	8
L	8	8	8	8	8	8	8	4	3	8	8		8	8	8	1	8	8	8	8
K	1	8	7	8	8	6	8	5	0	8		7	8	7	8	2	8	8	8	8
J	8	5	8	8	8	7	8	3	1		7	6	8	8	8	2	8	7	8	8
I	8	4	7	7	3	7	6	0		7	6	8	8	7	0	2	6	5	8	4
H	0	5	3	7	8	2	8		1	8	7	8	8	8	0	4	5	8	8	8
G	7	8	8	8	2	4		2	2	8	8	8	8	6	7	1	8	7	8	8
F	8	5	8	8	8		8	5	1	8	7	8	8	8	5	1	8	7	8	6
E	3	8	7	8		8	8	3	0	7	8	8	8	8	7	2	7	8	8	8
D	2	3	8		6	3	1	1	0	6	8	1	8	0	2	0	6	4	6	6
C	8	8		8	8	8	8	5	4	8	7	8	8	8	8	5	8	8	8	8
B	4		8	8	8	3	8	1	1	8	7	3	6	8	5	0	8	4	8	7
A		7	7	7	7	7	7	3	2	7	7	6	7	7	7	1	7	7	7	5

Figure 3.18: Impersonation matrix for the attribution method by Caliskan et al. [36]. Each cell indicates the number of successful attack attempts for the 8 challenges. The results for Abuhamad et al. [2] are presented in Figure B.2 in Appendix B.6.

ATTACK PERFORMANCE Table 3.6 depicts the success rate of the attack for both attribution methods. An adversary can transfer the prediction from one to another developer in 77% and 81% of all cases, respectively, indicating that more than three out of four programmers can be successfully impersonated.

...is also possible in the majority of cases.

In addition, for the attribution method by Caliskan et al. [36], Figure 3.18 presents the results as a matrix, where the number of successful impersonations is visually depicted. Note that the value in each cell indicates the absolute number of successful impersonations for the 8 challenges associated with each author pair. The matrix shows that a large set of developers can be imitated by almost every other developer. Their stylistic patterns are well reflected by the developed transformers and thus can be easily forged. By contrast, only the developers I and P have a small impersonation rate, yet 68% and 79% of the developers can still imitate the style of I and P in at least one challenge. The results for the attribution method by Abuhamad et al. [2] are similar. Figure B.2 in Appendix B.6 presents the respective impersonation matrix.

ATTACK ANALYSIS The number of altered lines of code also remains small for the targeted attacks. For both attribution methods, Figure 3.19 shows that in most cases only 0 to 10 lines of code are affected. At the same time, the feature space is substantially changed. Figure 3.20 depicts that both attribution methods exhibit a similar distribution as before in the untargeted attack—except that the left peak vanishes for the method of Caliskan et al. [36]. This means that each source file requires more than a few targeted changes to achieve an impersonation.

Again, often only a few lines of code are affected, but...

...a large number of features are changed.

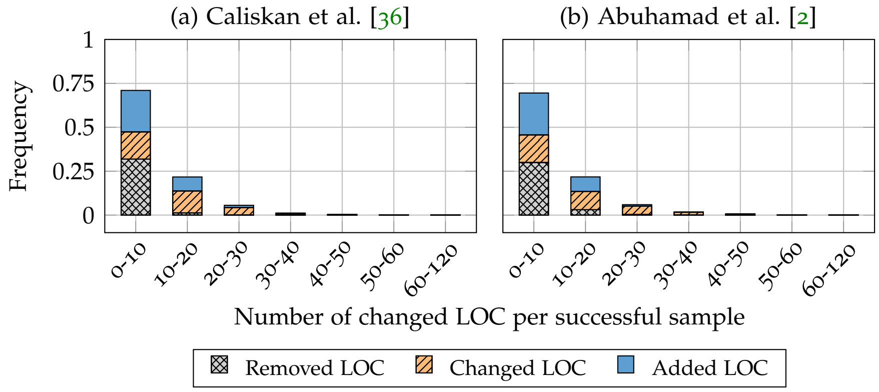


Figure 3.19: Targeted attack: Stacked histogram over the number of changed lines of code (LOC) per successful impersonation for both attribution methods. The original source files have 74 lines on average (std: 38.44).

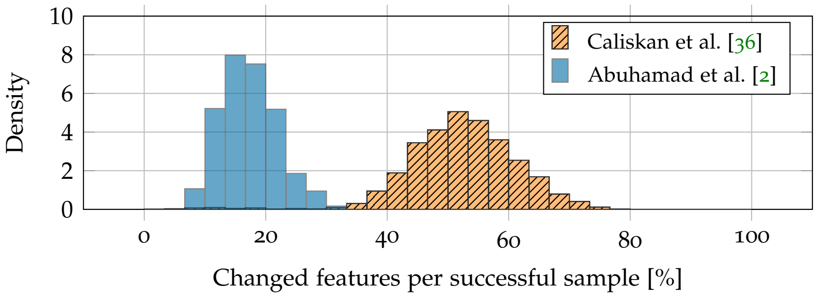


Figure 3.20: Targeted attack: Histogram over the number of changed features per successful impersonation for both attribution methods.

In addition, Table 3.7 shows the contribution of each transformation family to the impersonation success. All transformations are necessary to achieve the reported attack rates. A closer look reveals that the method by Abuhamad et al. [2] strongly rests on the usage of template transformations, while the families are more balanced for the approach by Caliskan et al. [36]. This difference can be attributed to the feature sets. The former method relies on simple lexical features only. Hence, custom declaration names, imported libraries or typedefs are more important than for the method by Caliskan et al. [36] that extracts more involved features from the AST.

To give further intuition for a successful impersonation attack, Appendix B.6 provides a case study from the evaluation.

ATTACK WITHOUT TEMPLATE INFORMATION Let us additionally examine the scenario when the adversary has no access to template information of the target developer. In this case, the template transformers can only try common patterns, such as the iteration variables `i`, `j`, ..., `k` or typedef `ll` for the type `long long`. Table 3.6 shows the results of this experiment as well. Still, an adversary can achieve an impersonation rate of 71% and 69%—solely by relying on the feedback

Transformation Family	Caliskan et al.	Abuhamad et al.
Control Transformers	8.43%	9.72%
Declaration Transformers	14.11%	17.88%
API Transformers	29.90%	19.60%
Template Transformers	38.42%	48.04%
Miscellaneous Transformers	9.15%	4.76%

Table 3.7: Usage of transformation families for impersonation with both attribution methods.

from the classifier. The number of altered lines of code and features correspond to Figure 3.19 and Figure 3.20.

Contrary to expectation, without template information, the approach by Abuhamad et al. [2] is harder to fool than the method by Caliskan et al. [36]. As the lexical features rest more on simple declaration names and included libraries, they are harder to guess without template files. However, if template files are available, this approach is considerably easier to evade.

An attack is also possible without access to the original model...

...by using a surrogate model.

ATTACK WITH SURROGATE MODEL Last but not least, let us evaluate the scenario when the adversary has no access to the prediction of the original classifier, only relying on a surrogate model trained from separate data. To this end, the training set is divided into disjoint sets with three files per author to train the original and surrogate model, respectively. The attack is tested on the method by Caliskan et al. [36], which is the more robust attribution under attack. By the nature of this scenario, the adversary can use two files to support the template transformations. Furthermore, to improve the transferability rate from the surrogate to the original model, we switch to a high-confidence attack by slightly adapting the search strategy. The search does not stop after the first successful adversarial example. Instead, it collects multiple samples and chooses the one with the highest score on the surrogate to be tested on the original classifier (see Appendix B.4 for more information).

Adversarial examples—created with the surrogate model—transfer to the original model in 79% of the cases. That is, attacks successful against the surrogate model are also effective against the original in the majority of the cases. This indicates that the attack successfully changes indicative features for a target developer across models. The success rate of the attack on the original model is 52%. Due to the reduced number of training files in this experiment, the attack is harder, as the coding habits are less precisely covered by the original and surrogate models. Still, an adversary is able to impersonate every second developer with no access to the original classifier.

SUMMARY The findings show that the developed problem-space attack allows an adversary to automatically impersonate a large set of developers. The attack successfully resolves the challenges due to the problem-feature space dilemmas and a restrictive black-box threat scenario. We can conclude that both considered attribution methods can be abused to trigger false allegations—rendering a real-world application dangerous.

3.5.4 *Verification of Attack Constraints*

The created adversarial examples...

The last group of experiments verifies that the generated adversarial code samples comply with the attack constraints specified in Section 3.4.2. It is empirically checked that (i) the semantics of the transformed source code are preserved, and (ii) the generated code is plausible to a human developer.

...preserve their functionality and...

PRESERVED SEMANTICS Automated tests are used to verify the semantics of the transformed source code. In particular, a test file from each challenge of the GCJ competition is used to check that the transformed source code provides the same solution as the original code. In all experiments, the output remains unchanged for each manipulated source code sample before and after the attack.

PLAUSIBILITY To check that the transformations lead to plausible code, a discrimination test with 15 human subjects is conducted. The group consists of 4 undergraduate students, 6 graduate students and 5 professional computer scientists. The structure of the test follows an *AXY-test*: Every participant obtains 9 files of source code—each from a different author but for the same GCJ challenge. These 9 files consists of 3 unmodified source codes as reference (A) and 6 sources codes (XY) that need to be classified as either *original* or *modified*. The participants are informed that 3 of the samples are modified. Then, each participant is asked to identify the unknown samples and to provide a short justification.

...are hard to separate from unmodified source code.

Table 3.8 provides the results of this empirical study. On average, the participants are able to correctly classify 60% of the provided files which is only marginally higher than random guessing. This result highlights that it is hard to decide whether source code has been modified by the attack or not. In several cases, the participants falsely assume that unused `typedef` statements or an inconsistent usage of operators are modifications.

Participant Group	Accuracy	Std
Undergraduate students	66.7%	23.6%
Graduate students	55.6%	15.7%
Professionals	60.0%	24.9%
Total	60.0%	21.8%
Random guessing	50.0%	—

Table 3.8: Study on plausibility of transformed source code.

3.6 LIMITATIONS

The previous experiments demonstrate the impact of the developed problem-space attack on program authorship attribution. Nonetheless, the approach has limitations that are discussed in the following.

VERIFICATION OF SEMANTICS Two programs are assumed to be semantically equivalent if they return the same output for a given input. In particular, the test cases provided by the GCJ competition are applied to verify that the transformed source code is semantically equivalent. As discussed in Section 3.3.1, this approach is reasonable in our setting, but it cannot guarantee strict semantic equivalence in all possible cases.

HARDCODED CODE TRANSFORMATIONS Although Clang as a compiler frontend can be leveraged to obtain a systematic view on source code, the developed code transformations need to be implemented manually. Nevertheless, this approach also has considerable advantages: it allows defining targeted code modifications that change *executed* source code. They preserve semantics by design, are plausible and readable, and do not exploit any preprocessing vulnerability or semantic gap by adding unused code only. The user study demonstrates that it is hard to differentiate between original and modified source code.

MISSING CODE TRANSFORMATIONS Although the attack enables misleading the attribution of programmers in the majority of cases, it is not able to establish a targeted evasion from any author to any other author. This can be attributed to missing code transformations and is not a limitation of the developed approach in general. The developed transformations focus on frequent, generic coding habits. It is possible to extend the set of transformations in such a way that source code from one author can be better transformed to match another author. While we should not rule out that there exist untransferable patterns in lexical and syntactic features, such patterns were not observed during the experiments. Identifying untransferable stylistic patterns is thus a promising direction for future research.

ADVERSARIAL EXAMPLES $\neq$ ANONYMIZATION The attack enables a programmer to hide her identity in source code by misleading an attribution. While such an attack protects the privacy of the programmer, it is not sufficient for achieving anonymity. Note that k-anonymity would require a set of k developers that are equally likely to be attributed to the source code. In our setting, the code of the programmer is transformed to match a different author and an anonymity set of sufficient size is not guaranteed to exist. Still, anonymization is a promising direction for further research, which can build on the concepts of code transformations developed in this chapter.

3.7 RELATED WORK

Conducting problem-space attacks and in particular creating adversarial examples against source code authorship attribution touches different areas of security research. In this section, we review related methods and concepts.

ADVERSARIAL MACHINE LEARNING The security of machine learning techniques is a vivid research field that has unveiled a broad range of possible attacks. In this chapter, we examine the attack surface when the adversary has to create a real adversarial example in $\mathcal{Z}$. A significant fraction of work on adversarial examples has focused on scenarios where the problem and feature space are mainly identical or only focused on the feature space [e.g., 12, 26, 27, 33, 38, 39, 45, 88, 110, 136, 156, 200]. With digital images, for example, pixels in $\mathcal{Z}$ have a one-to-one relation to the feature vector in $\mathcal{F}$. Also related is the approach by Sharif et al. [185] that misleads face recognition systems using painted eyeglasses. The recognition operates in $\mathcal{F}$, but the input image is coming from the real world by taking a photograph of a person. Thus, $\mathcal{Z}$ and $\mathcal{F}$ have no exact but still similar representation. The proposed attack operates in $\mathcal{F}$ but ensures practical feasibility in $\mathcal{Z}$ by refining the optimization problem. In particular, the calculated adversarial perturbations are required to match the form of eyeglasses, to be printable, and to be invariant to slight head movements. The created perturbation in $\mathcal{F}$ thus likely transfers to a person in $\mathcal{Z}$ who wears the eyeglasses.

In most security-sensitive application areas, however, $\mathcal{Z}$ has a very different representation from $\mathcal{F}$, such as for *source code*, *text*, *PDF*, *Windows PE*, *Android*, and *JavaScript*. Table 3.9 lists published problem-space attacks grouped by the application domain. The introduced problem-feature space dilemmas in this thesis are of particular relevance for all these areas.

Application	**References**
Source code	*This thesis*
	Matyukhina et al. [141] and Quiring et al. [169]
Text	Alzantot et al. [7], Ebrahimi et al. [68], Gao et al. [79], Garg and Ramakrishnan [80], Li et al. [124], and Papernot et al. [154]
PDF	Dang et al. [63], Maiorca et al. [137], Tong et al. [206], Xu et al. [227], and Šrndić and Laskov [236]
Windows PE	Anderson et al. [8], Kolosnjaji et al. [112], Rosenberg et al. [175, 176], and Suciu et al. [197]
Android	Chen et al. [48], Demontis et al. [65], Grosse et al. [91], Pierazzi et al. [162], and Yang et al. [229]
JavaScript	Fass et al. [69]

Table 3.9: List of published problem-space attacks per application domain.

The discussed framework of problem-space attacks in Section 3.3 is based on Quiring et al. [169] and Pierazzi et al. [162]. It allows us to categorize and compare the attacks from the different application domains by highlighting their key concepts, strengths and weaknesses. It also allows researchers and practitioners to guide the design of novel problem-space attacks, since necessary and optional constraints together with a search strategy and threat model are pointed out. Table 3.10 shows the categorization of multiple attacks with this framework. Note that the assessment regarding the fulfilled constraints does not imply that an attack relies on advanced methods or not. In fact, a simple attack might fulfill more constraints than a more advanced approach. Appendix B.7 provides detailed information about the categorization of each work in Table 3.10.

Two aspects regarding the research on problem-space attacks are highlighted by Table 3.10. First, a semantic gap is often exploited to create adversarial examples [e.g., 112, 141, 197, 236]. For example, Šrndić and Laskov [236] add content into a PDF file between the cross-reference table and the trailer. A PDF viewer will skip this area, but the attacked classifier extracts features from there (remember Figure 3.8). Suciu et al. [197] add content into slack spaces between sections in a Windows PE file. These slack spaces are not relevant for executing the program, but again extracted as features.

Second, a prevalent concept is to use transformations that only add content, so that problems with the semantics are avoided [e.g., 91, 112, 162, 197]. For instance, a rather simple, yet effective approach appends bytes at the end of a file to mislead malware detection [112, 197]. A more advanced method adds plausible content to Android applications from benign applications through automated code transplantation [162]. By using opaque predicates, a static analysis cannot find out that the added code is not executed at runtime. On the other side, removing or changing used parts is less considered in the literature so far, as it requires an application-specific understanding of the meaning and relevance of modified content. For example, removing

Application	Work	Semantics	Plausibility	Robust Prepr.	No Sem. Gap	Transformers	Search Strategy
Source Code	*This Thesis*, Quiring et al. [169]	●	●	●	●	Rewrite source code with targeted changes (see Section 3.4.3)	PD (MCTS)
	Matyukhina et al. [141]	●	○	○	○	Rewrite code by exploiting layout, comments, control-flow flattening	FD-df
Text	Li et al. [124]	◐	◐	◐	●	Rewrite text with spelling mistakes and synonyms	FD-db, PD
	Papernot et al. [154]	○	○	●	●	Replace word with another word from a dictionary	FD-db
Windows	Kolosnjaji et al. [112]	●	◐	○	○	Append bytes at the end	FD-db
	Suciu et al. [197]	●	◐	○	○	Append bytes at the end, add bytes within file in unused slack spaces	FD-db, PD
PDF	Šrndić and Laskov [236]	●	◐	○	○	Add content in unused area between cross-reference table and trailer	FD-df, FD-db
	Xu et al. [227]	●	○	◐	●	Rewrite PDF tree representation with random mutations	PD
Android	Grosse et al. [91]	●	◐	◐	●	Add unused content in manifest header file	FD-db
	Pierazzi et al. [162]	●	●	●	●	Add unused code via code transplantation, not detectable with static analysis	FD-db

Table 3.10: Overview of representative problem-space attacks for each application domain with considered constraints and search strategy. PD = problem-driven, FD-db = feature-driven, derivative-based, FD-df = feature-driven, derivative-free. The filled circle ● denotes a fulfilled constraint, ◐ a constraint that is partly fulfilled, and ○ a constraint that is not fulfilled.

code can destroy the functionality of a program, while adding a never-executed code statement is not changing the semantics. Using the example of source code, this chapter demonstrates that an attack can automatically change used parts such that the semantics are preserved. A case study further shows that such an attack can well preserve the plausibility—even under review by domain experts who are aware of an attack. To this end, the attack relies on targeted transformations. The downside is that these transformations are implemented manually.

Last but not least, this chapter also introduces MCTS as a novel concept in the portfolio of creating real adversarial examples, previously examined by Wicker et al. [223] in the image context only.

AUTHORSHIP ATTRIBUTION OF SOURCE CODE Identifying the author of a program is a challenging task that has attracted a large body of work. Starting from early approaches experimenting with hand-crafted features [114, 135], the techniques for examining source code have constantly advanced, for example, by incorporating expressive features, such as n-grams [e.g., 2, 35, 73] and ASTs [e.g., 6, 36, 160]. Similarly, techniques for analyzing native code and identifying authors of compiled programs have advanced in the last years [e.g., 5, 37, 143, 177].

Two notable examples for source code are the approach by Caliskan et al. [36] and by Abuhamad et al. [2]. The former inspects features derived from code layout, lexical analysis and syntactic analysis. This work can be considered as the state of the art regarding comprehensiveness at the time of writing. The work by Abuhamad et al. [2] focuses on lexical features as input for recurrent neural networks. Their work covers a large set of authors and makes use of advances in deep learning. Table 3.11 shows the related approaches.

Method	Lay	Lex	Syn	Authors	Results
***Abuhamad et al. [2]**		●		**8903**	**92%**
***Caliskan et al. [36]**	●	●	●	**250**	**95%**
Alsulami et al. [6]			●	70	89%
Frantzeskou et al. [73]	●	●		30	97%
Krsul and Spafford [114]	●	●	●	29	73%
Burrows et al. [35]	●	●		10	77%

Table 3.11: Comparison of approaches for source code authorship attribution. Lay = Layout features, Lex = Lexical features, Syn = Syntactic features. *Attacked in this chapter.

Previous work, however, has mostly ignored the problem of untargeted and targeted attacks. The empirical study by Simko et al. [189] examines how programmers can mislead the attribution by Caliskan et al. [36] by mimicking the style of other developers. While this study provides valuable insights into the risk of forgeries, it does not consider automatic attacks and is thus limited to manipulations by humans. The work by Matyukhina et al. [141] finds adversarial examples automatically, but relies on a rather simple greedy-based search strategy. As transformations, layout features are primarily targeted by manipulating brackets, spaces, and empty lines. Comments are added from the target developer or transformed to another style. Hence, this attack does not fulfill the plausibility, preprocessing, and semantic gap constraint (see Table 3.10). This thesis demonstrates that attacks can be fully automated—on different feature sets and learning algorithms as given by two attribution methods. Plausible, semantics-preserving samples are created which are robust against preprocessing and do not exploit a semantic gap.

3.8 CHAPTER SUMMARY

This chapter thoroughly analyzes attacks against machine learning that have to proceed in the problem space. The domain of source code attribution provides us with practical insights. The text box on the next page shortly summarizes the main takeaways.

Main Takeaways.

1. Adversaries face multiple dilemmas, because the problem and the feature space have no one-to-one correspondence. This is the case in most security-sensitive application domains, such as for source code, text, PDF, Windows PE, or Android files.
2. An attack is possible, but an adversary has to operate in the problem space while being guided by the feature space. Formally, a problem-space attack can be defined as optimization problem that is given by constraints and a search strategy.
3. Five constraints have to be considered: (i) semantics, (ii) plausibility, (iii) preprocessing, (iv) semantic gap, and (v) available transformations.
4. The search strategy finds a solution to the optimization problem. The strategy guides the transformations and is (i) problem-driven, derivative-free, (ii) feature-driven, derivative-based, or (iii) feature-driven, derivative-free. The choice depends on the threat model and differentiability of the learning model.
5. The search strategy based on MCTS provides an effective way to find real adversarial examples. It examines various decision paths before deciding on the next modification, similar to an advanced chess player.
6. The empirical evaluation shows that authorship attribution methods can be undermined. With an untargeted attack, the accuracy drops from over 88% to 1%. With a targeted attack, each developer can be impersonated by 77% to 81% of the others on average. The current state-of-the-art in authorship attribution is insufficient for achieving a robust classification.

4

ATTACK ON THE MAPPING

We proceed to examine the relation between the problem and the feature space of machine learning. In this chapter, we analyze the attack surface that the mapping $\phi \circ \rho$ from problem to feature space can provide. This mapping is of substantial relevance for the whole machine learning pipeline, as it is the first step and thus lays the ground for the subsequent learning process. Consequently, if the adversary can exploit the mapping, she can take over control of the learning pipeline, allowing different attacks during training and deployment. To understand this attack surface, we examine the preprocessing ρ in the context of image scaling in this chapter.

The mapping from $\mathcal{Z}$ to $\mathcal{F}$ also provides an attack surface...

...that we explore in this chapter...

The scaling operation suffers from vulnerabilities that allow an adversary to manipulate an image such that it changes its appearance after downscaling [225]. As an example, Figure 4.1 depicts an attack against the scaling operation of the popular TensorFlow library. The manipulated image (left) changes to the output (center) when scaled to a specific dimension. This image is then used for the subsequent learning process. Attacks on image scaling pose a major threat to machine learning: First, scaling is omnipresent in computer vision, as learning algorithms typically require fixed input dimensions. Second, scaling attacks are agnostic to the learning model, features, and training data. Third, the attacks can be used for poisoning data during training as well as misleading classifiers during deployment. In the previous example from Figure 4.1, a self-driving car might enter a street by falsely assuming a restricted parking area only, causing a risk of collision due to possible opposing traffic. In contrast to adversarial examples that also mislead predictions, image-scaling attacks do not depend on a particular model or feature set, as the downscaling can create a perfect image of the target class. Thus, scaling attacks would remain effective even if neural networks were robust against adversarial examples. Overall, scaling attacks unveil a novel attack category that was coined as *adversarial preprocessing* by Quiring et al. [171].

...in the context of scaling as one form of preprocessing.

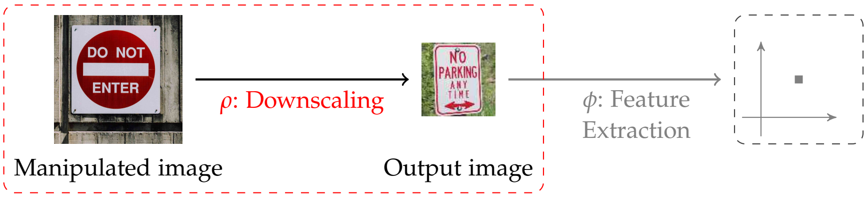


Figure 4.1: Example of an image-scaling attack. Left: a manipulated image showing a *do not enter* traffic sign. Scaling produces the center image with a *no parking* sign, used for feature extraction then.

Fortunately, this chapter also shows that the well-defined structure of scaling algorithms allows identifying the root cause of scaling attacks and developing effective defenses for prevention. This is in stark contrast to the problem-space attacks in the previous chapter which are hard to analyze and defend due to the complexity of learning models. All in all, this chapter demonstrates that an attack on the mapping from $\mathcal{Z}$ to $\mathcal{F}$ can have a considerable impact on machine learning, but can also be understood and mitigated given a thorough root-cause analysis due to the well-defined structure of the mapping.

In summary, this chapter discusses the following major aspects:

- *Attack on the mapping.* We analyze image-scaling attacks as example for an attack on the mapping from $\mathcal{Z}$ to $\mathcal{F}$. The vulnerability underlying scaling attacks is identified in theory and confirmed in practical implementations.
- *Effective Defenses.* We develop a theoretical basis for assessing the robustness of scaling algorithms and designing effective defenses. We also study a novel defense that protects from all possible attack variants.
- *Comprehensive Evaluation.* Scaling algorithms of popular imaging libraries are empirically analyzed under attack. The effectivity of the defenses are demonstrated against adversaries of different strengths.

Before starting the theoretical analysis, let us review the background of image scaling and image-scaling attacks. Note that all figures in this chapter with examples of scaling attacks use real attack images. Thus, a PDF viewer that displays this chapter can be partly affected.

4.1 IMAGE SCALING IN MACHINE LEARNING

In machine learning with digital images,...

Image scaling is a standard procedure in computer vision and a common preprocessing step in machine learning [159]. A scaling algorithm takes a source image S and resizes it to a scaled version D. As many learning algorithms require a fixed-size input, scaling is a mandatory step in most learning-based systems operating on images. For instance, deep neural networks for object recognition, such as VGG19 [190] and Inception V3/V4 [201], expect inputs of 224×224 and 299×299 pixels, respectively. They can only be applied in practice if images are scaled to these dimensions.

...downscaling is often a mandatory preprocessing step.

Generally, we can differentiate *upscaling* and *downscaling*, where the first operation enlarges an image by extrapolation, while the latter reduces it through interpolation. In practice, images are typically larger than the input dimension of learning models and thus adversaries can be expected to focus on downscaling. Table 4.1 lists the most common scaling algorithms. Although these algorithms address the same task, they differ in how the content of the source S is weighted

Framework	Caffe	PyTorch	TensorFlow
Library	OpenCV	Pillow	tf.image
Library Version	4.1	6.0	1.14
Nearest	•	•(‡)	•
Bilinear	•(*)	•(*)	•(*)
Bicubic	•	•	•
Lanczos	•	•	
Area	•	•	•

Table 4.1: Scaling algorithms in deep learning frameworks. (*) denotes the default algorithm, and (‡) the default algorithm if Pillow is used directly without PyTorch.

and smoothed to form the scaled version D. For example, nearest-neighbor scaling simply copies pixels from a grid of the source to the destination, while bicubic scaling interpolates pixels using a cubic function. We examine these algorithms in more detail in Section 4.3 when analyzing the root cause of scaling attacks.

Due to the central role in computer vision, scaling algorithms are an inherent part of several deep learning frameworks. For example, Caffe, PyTorch, and TensorFlow implement all common algorithms, as shown in Table 4.1. Technically, TensorFlow uses its own implementation called *tf.image*, whereas Caffe and PyTorch use the imaging libraries *OpenCV* and *Pillow*, respectively. Other libraries for deep learning either build on these frameworks or use the imaging libraries directly. For instance, Keras uses Pillow, and DeepLearning4j uses OpenCV. As a consequence, the analysis in this chapter focuses on these major imaging libraries.

4.2 IMAGE-SCALING ATTACKS

Xiao et al. [225] show that scaling algorithms are vulnerable to attacks and can be misused to fool machine learning systems. The proposed attack carefully manipulates an image, so that it changes its appearance when scaled to a specific dimension. In particular, the attack generates an image A by slightly perturbing the source image S such that its scaled version matches a target image T. This process is illustrated in Figure 4.2, which also serves as a running example throughout this chapter. In addition, Table 4.2 provides an overview of the notation.

A scaling attack enables controlling the scaling output with imperceptible input changes.

4.2.1 *Threat Model*

The attack is agnostic to the employed learning model and does not require knowledge of the training data or extracted features. Yet, the adversary needs to know two parameters: (i) the used scaling algo-

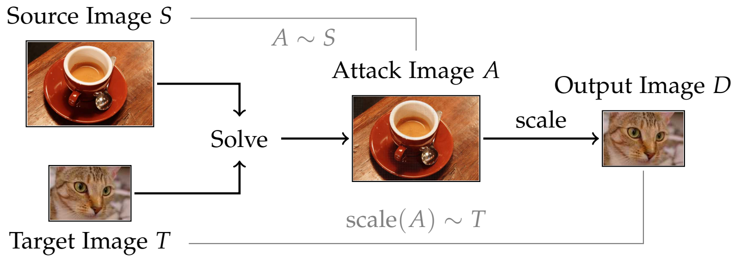


Figure 4.2: Principle of image-scaling attacks: An adversary computes A such that it looks like S but downscales to T.

Symbol	Size	Description
S	$m \times n$	The source image that is used to create the attack image.
T	$m' \times n'$	The target image that the adversary wants to obtain after scaling.
A	$m \times n$	The attack image, a slightly perturbed version of S
D	$m' \times n'$	The output image of the scaling function scale.

Table 4.2: Table of symbols for scaling attacks.

rithm and (ii) the target size $m' \times n'$ of the scaling operation. Xiao et al. describe how an adversary can easily deduce both parameters with black-box access to the machine learning system by sending specifically crafted images [see 225]. Moreover, Table 4.1 shows that common open-source libraries have a limited number of scaling options and use default algorithms if users are not actively selecting an algorithm. Taken together, only a few attempts may be necessary to discover the correct setup. In some settings, a fixed algorithm can even be enforced by specific image sizes. Section 4.6.1 provides more information about this attack variant.

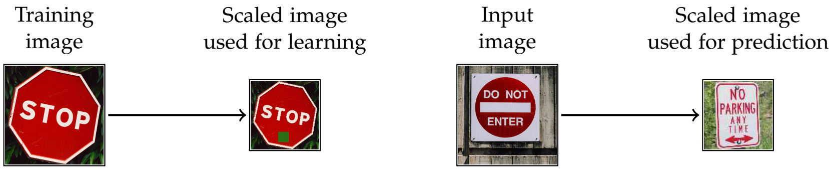
The attack does not depend on features, training data, and learning models,...

...and only requires knowledge of the scaling parameters.

Image-scaling attacks allow achieving different goals. As the attack is targeting the preprocessing ρ, the image is manipulated before any feature extraction. Hence, scaling attacks can effectively mislead all subsequent steps in a machine-learning pipeline, allowing different attacks during training and deployment time. That is, an attacker can conceal poisoning attacks or trigger false predictions during the application of a learning model. Let us review both scenarios in the following.

This attack allows the adversary...

DATA POISONING In a general poisoning attack, the adversary tries to control the learning process by manipulating the training data or learning model (see Section 2.2.5). Although various training-only and backdoor poisoning methods are particularly effective with a few changed training instances, they often have the major shortcoming that the manipulation is still visible [e.g., 92, 184, 230]. Hence, the attack can be easily detected if the dataset, for example, is audited by human beings. Furthermore, backdoor attacks also require adding



(a) Data poisoning

(b) Misleading prediction

Figure 4.3: Applications of scaling attacks. (a) Enhanced poisoning: The green box as backdoor is only present in the downscaled image used for training. (b) Controlled predictions: The adversary creates a perfect image of the target class that is passed to the classifier.

a trigger to an input at deployment time, which can also unveil an attack. As image-scaling attacks provide a new means for creating novel content after downscaling, they allow an adversary to create less visible, data-modifying poisoning attacks. Figure 4.3a exemplifies the combination by showing an adapted backdoor attack. The adversary modifies a training image such that a backdoor trigger is only present in the downscaled image. The trigger is not visible in the full-size training image that a human analyst may use. Consequently, the learning algorithm unnoticeably associates the backdoor trigger with the stop sign during training. Appendix C.1 provides further information about the combination of scaling attacks and poisoning, including the adapted backdoor attack from Figure 4.3a. Finally, note that the combined attack necessarily inherits the threat model from the poisoning attack, such as the required access to the training data or a model.

...to disguise poisoning attacks at training time, and...

MISLEADING PREDICTIONS Furthermore, the adversary can control the predictions during the application of a learning model—without modifying the training data or model. To this end, she uses the scaling attack, so that the downscaling leads to an image of another, targeted class. Compared to adversarial examples, both attacks accomplish the same goal. However, image-scaling attacks considerably differ in the threat model: The attacks are model-independent and do not depend on knowledge of the learning model, features, or training data. Furthermore, image-scaling attacks would be effective even if neural networks were robust against adversarial examples, as the downscaling can create a perfect image of the target class. Figure 4.3b exemplifies the usage of scaling attacks to mislead predictions.

...to mislead the prediction at deployment time...

...with less effort compared to adversarial examples.

Finally, we should note that scaling attacks are of particular concern in all security-related applications where images are scaled and processed automatically. All in all, the discussed attacks underline the possible *security implications if an adversary exploits the mapping from problem to feature space*.

4.2.2 *Attack Strategy*

Image-scaling attacks can be solved...

Image-scaling attacks can be implemented with a strong and a weak strategy [225]. In the strong strategy, the adversary can choose the source and target image. In the weak version, the adversary can only choose the target, and the calculated attack image is meaningless and easily detectable. We thus focus on the stronger attack strategy in the following, which is relevant for real-world applications.

OBJECTIVES Formally, image-scaling attacks need to pursue the following two objectives:

Objective O1. *The downscaling operation on A needs to produce the target image:* $\text{scale}(A) \sim T$.

Objective O2. *The attack image A needs to be indistinguishable from the source image: $A \sim S$*

The first objective ensures that the target image T is obtained during scaling, while the second objective aims at making the attack hard to detect. We can verify the first objective by checking if the prediction of a neural network corresponds to the target image's class. Note that without the second objective, the attack would be trivial, as the adversary could simply overwrite S with an upscaled or larger version of T. In this case, however, the attack would be easily detectable and thus not effective in practice.

...as optimization problem.

STRONG ATTACK STRATEGY The adversary seeks a minimal perturbation Δ of S such that the downscaling of $\Delta + S = A$ produces an output similar to T. Both goals can be summarized as the following optimization problem:

$$\begin{aligned} & \min(\|\Delta\|_2^2) \\ \text{s. t.} \quad & \|\text{scale}(S + \Delta) - T\|_\infty \leqslant \epsilon\ . \end{aligned} \tag{4.1}$$

Additionally, each pixel value of A needs to remain within the fixed range (e. g., $[0, 255]$ for 8-bit images). This optimization problem can be solved with Quadratic Programming [31]. When successful, the adversary obtains an image A that looks like the source but matches the target after scaling.

HORIZONTAL AND VERTICAL OPTIMIZATION Common imaging libraries, such as OpenCV or Pillow, implement downscaling by first resizing images horizontally and then vertically. This implementation technique enables approximating the scaling operation from Equation 4.1 by a closed-form expression which is based on a simple matrix multiplication:

$$D = \text{scale}(S + \Delta) = L \cdot (S + \Delta) \cdot R \tag{4.2}$$

with $L \in \mathbb{R}^{m' \times m}$, $R \in \mathbb{R}^{n \times n'}$ and $D \in \mathbb{R}^{m' \times n'}$. The matrices L and R contain fixed coefficients that depend on the selected scaling algorithm. Both matrices can be computed in advance and are reusable. The reader is referred to Xiao et al. [225] for a description how to calculate L and R.

Based on this matrix multiplication, the attack can also be decomposed into a horizontal and vertical manipulation, which are conducted in reverse order to the scaling, as shown in Figure 4.4. The attack proceeds by first computing a resized version of S, that is, $S' = \text{scale}(S) \in \mathbb{R}^{m \times n'}$. Here, we solve Equation 4.1 with S' as source image and T as target. We have $A' = S' + \Delta'$. Due to the decomposition, we only need the coefficient matrix L and thus arrive at the following optimization problem

$$\min(\|\Delta'\|_2^2) \;\; \text{s.t.} \; \|L \cdot (S' + \Delta') - T\|_\infty \leqslant \epsilon \, . \tag{4.3}$$

Next, the horizontal direction is considered. To this end, the adversary calculates the final attack image A with S as source image, but A' as target, analogue to Equation 4.3.

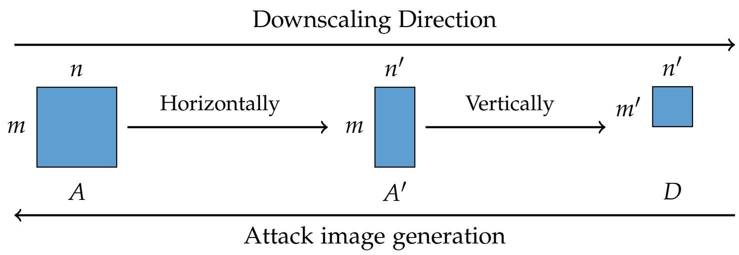


Figure 4.4: Libraries resize an image horizontally first, and then vertically. The attack creates A in reverse order: first the intermediate image A', and then A.

COLUMN-BASED OPTIMIZATION In order to decrease the computational effort, the optimization can be further decomposed into individual dimensions. We start again with the vertical scaling direction where we resize $S' \in \mathbb{R}^{m \times n'}$ to $D \in \mathbb{R}^{m' \times n'}$. Instead of considering the whole matrix, we solve the problem from Equation 4.3 for each column of S' separately:

$$\min(\|\Delta'_{*,j}\|_2^2) \;\; \text{s.t.} \; \|L \cdot \left(S'_{*,j} + \Delta'_{*,j}\right) - T_{*,j}\|_\infty \leqslant \epsilon \, , \tag{4.4}$$

where the subscript in $X_{*,j}$ specifies the j-th matrix column of a matrix X. This optimization is repeated for the horizontal direction and finally computed for all color channels.

4.3 ATTACK ANALYSIS

Having this background on image-scaling attacks, we are ready to investigate their inner workings in more detail. The goal is to find

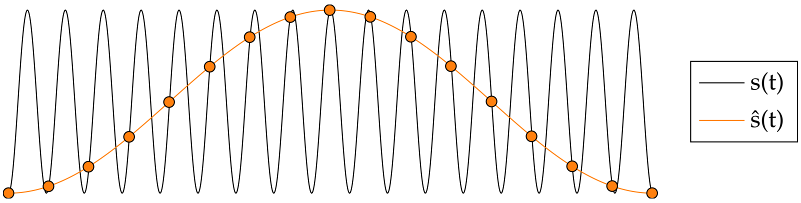


Figure 4.5: Example of an undersampled signal $s(t)$. Based on the sampling points, it is not possible to distinguish between $s(t)$ and $\hat{s}(t)$.

The attack is possible, since...

out which vulnerability image-scaling attacks exactly exploit to be successful. We start off by observing that the presented attacks must exploit a vulnerability that is shared by many scaling algorithms. As the implementations of the algorithms differ, this vulnerability needs to be linked to the general concept of scaling. To better grasp this concept, we require a broader perspective on image scaling and thus examine it from the viewpoint of signal processing.

4.3.1 *Scaling as Signal Processing*

Images can be viewed as a generic signal, similar to audio and video. While audio is described by a one-dimensional time series, an image represents a discrete and two-dimensional signal. Typically, images are encoded in the *spatial domain* with pixels. However, any signal can be described by a sum of sinusoids of different frequencies, and hence images can also be represented in the *frequency domain* [e.g., 151, 192].

Scaling reduces the dimension of an image. As a result, the frequency mixture of the image changes and higher frequencies are lost. This process is closely related to *downsampling* in signal processing, where a high-frequency signal is transformed to a lower frequency. A major problem of downsampling is that the reduced resolution might not be able to describe all relevant frequencies in the image. According to the Nyquist–Shannon theorem [151], it is only feasible to reconstruct a signal $s(t)$ from a discrete number of sampled points if the sampling rate f_T is at least twice as high as the highest frequency f_{max} in the signal: $f_T \geq 2 \cdot f_{max}$.

If the frequency f_T is below that threshold, the signal cannot be unambiguously reconstructed. In this case, the sampled points do not provide enough information to distinguish between the original signal and other possible signals. Figure 4.5 shows an example of this phenomenon, where it is impossible to decide which one of the two signals $s(t)$ and $\hat{s}(t)$ is described by the sampled points. Ultimately, the reconstructed signal can differ significantly from the original signal, which is known as the *aliasing effect* [151]. As we see in the next sections, image-scaling attacks build on this very effect by cleverly manipulating a signal such that its downsampled version becomes a new signal.

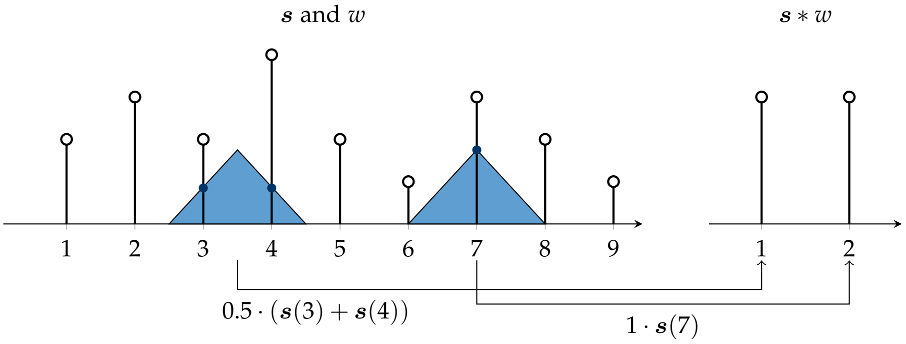


Figure 4.6: Scaling with convolution. The triangle illustrates the kernel with its relative weighting. It has a width of 2 and is shifted by a step size of $\beta = 3.5$.

4.3.2 *Scaling and Convolution*

It is clear that scaling algorithms do not merely reduce the frequencies in an image. These algorithms carefully interpolate the pixels of the source image before downscaling it in order to mitigate the aliasing effect. This computation can be described as a *convolution* between the source signal and a kernel function [151]. For each position in the scaled image, the kernel combines a set of pixels (samples) from the source using a specific weighting. All scaling algorithms given in Table 4.1 can be expressed using this concept. Note that kernels are weighting functions in the image-scaling context in contrast to kernel functions in machine learning.

...scaling algorithms...

Without loss of generality, let us focus on the horizontal scaling of a single row in the following, that is, a row $s \in \mathbb{R}^n$ from the source image is scaled to $d \in \mathbb{R}^{n'}$. We denote by β the respective scaling ratio: $\beta = n/n'$. The goal of downscaling is to determine the value for each pixel in d from a set of samples from s. This process can be described using a kernel function w as follows

$$(s * w)(t) = \sum_{u \in \mathbb{Z}} w\,(t-u)\,s(u). \tag{4.5}$$

Intuitively, w represents a weighting function that is moved over s as a sliding window. We denote the size of this window as the *kernel width* σ. Each pixel within this window is multiplied by the respective weight at this position. Figure 4.6 exemplifies this process for a bilinear kernel with $\sigma = 2$. The first pixel in d is the aggregated result from the third and fourth pixel in s, while the second pixel in d is only estimated from the seventh pixel in s.

...move over the input image with a sliding window...

As the downscaling of an image produces a smaller number of pixels, the window of the kernel function needs to be shifted on s by a specific step size, similar to the process of sampling in signal pro-

...and aggregate pixels in each window to get...

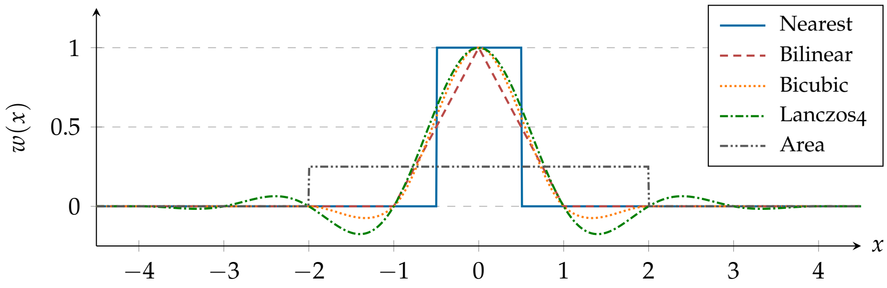


Figure 4.7: Visualization of kernel functions for common scaling algorithms.

cessing. The scaling ratio defines this step size so that each sampling position is given by

$$\nu(p) = p \cdot \beta, \tag{4.6}$$

...a respective output pixel.

where p is the target pixel in $\boldsymbol{d}$ and $\nu(p)$ a position in $\boldsymbol{s}$ around which we place the kernel window. Note that the position $\nu(p)$ is not necessarily discrete and can also fall between two pixels, as shown in Figure 4.6. The downscaled output image is then computed as follows:

$$\boldsymbol{d}_p = (\boldsymbol{s} * w)(\nu(p)) \quad p = 0, 1, \ldots, n' - 1. \tag{4.7}$$

Each scaling algorithm is defined by a particular kernel function. Figure 4.7 depicts the standard kernels for common scaling algorithms. For instance, nearest-neighbor scaling builds on the following kernel function:

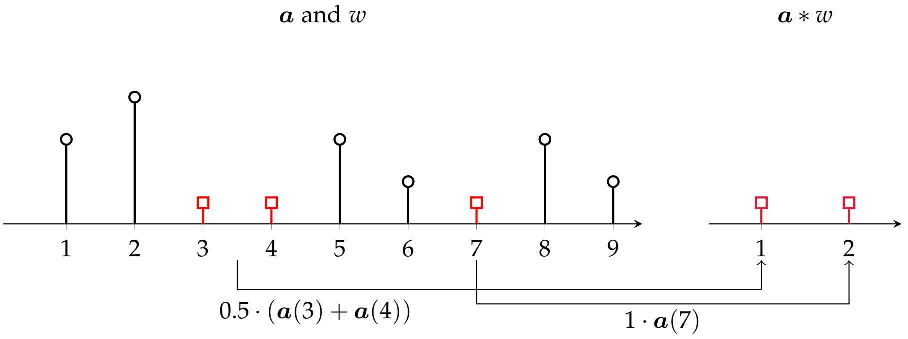
$$w(x) = \begin{cases} 1 & \text{for } -0.5 \leqslant x < 0.5, \\ 0 & \text{otherwise .} \end{cases} \tag{4.8}$$

Only the value that is the closest to $\nu(p)$ is used by this scaling algorithm. In other words, nearest-neighbor scaling simply copies pixels from $\boldsymbol{s}$ on a discrete grid to $\boldsymbol{d}$. Overall, each kernel differs in the number of pixels that it uses and the respective weighting of the considered pixels.

4.3.3 *Root-Cause Analysis*

If the windows do not use all pixels in the input image,...

Based on the insights from signal processing, we can start to investigate the root cause of image-scaling attacks. We observe that not all pixels in the source image equally contribute to its scaled version. Only those pixels close to the center of the kernel receive a high weighting, whereas all remaining pixels play a limited role during scaling. If the step size exceeds the kernel width, some pixels are even ignored and irrelevant for the scaling operation. Figure 4.6 illustrates this situation: Only three out of nine pixels are considered for computing the scaled output.



$\boldsymbol{a}$ and w
$\boldsymbol{a} * w$
1
2
3
4
5
6
7
8
9
1
2
$0.5 \cdot (\boldsymbol{a}(3) + \boldsymbol{a}(4))$
$1 \cdot \boldsymbol{a}(7)$

Figure 4.8: Illustration of scaling attack on previous example from Figure 4.6.

This imbalanced influence of the source pixels provides a perfect ground for image-scaling attacks. The adversary only needs to modify those pixels with high weights to control the scaling and can leave the rest of the image untouched. This strategy is sufficient for achieving both objectives of the attack: (O1) a modification of pixels with high weights yields scale$(A) \sim T$, and (O2) depending on the sparsity of those pixels the attack image A visually matches the source image S. Figure 4.8 exemplifies these insights on the scaling example from Figure 4.6. The adversary creates her attack signal $\boldsymbol{a}$ by only modifying the three considered pixels in such a way that the downscaled values are her target values. The other pixels keep their value.

...the attack can only modify the pixels that are used...

...while leaving the rest unmodified.

From the perspective of signal processing, image-scaling attacks can thus be interpreted as malicious aliasing, where the adversary selectively manipulates those regions of the signal that are sampled during downscaling. These regions create a high-frequency signal in the source image that is not visible in the spatial domain but precisely captures the sampling rate of the downscaling process.

We can deduce that the success of image-scaling attacks depends on the sparsity of pixels with high weight. If these pixels are dense, the adversary may still achieve O1 but will fail to satisfy O2, as the attack becomes visible. Reviewing the general concept of scaling, two factors determine the sparsity of these pixels: the scaling ratio β and the kernel width σ. For images, we can formally bound the ratio r of pixels that are considered during scaling by

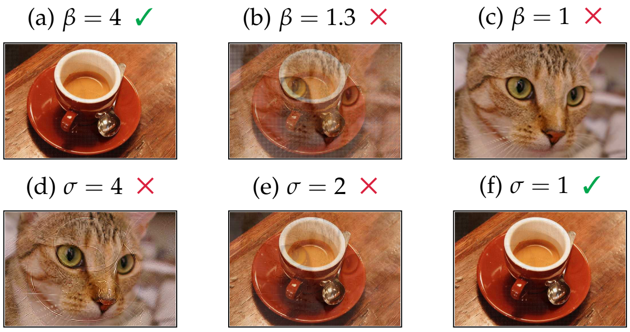
Thus, the attack is not visible if...

$$r \leq \frac{\sigma_h \, \sigma_v}{\beta_h \, \beta_v} \, . \tag{4.9}$$

The terms β_h, β_v as well as σ_h and σ_v denote the respective scaling ratio and kernel width horizontally and vertically. If the direction is irrelevant, quadratic images are considered for the analysis, so that we can simply use β and σ for both axes. Moreover, note that the right term may exceed one if the windows of the kernels overlap and pixels in the source are considered multiple times.

...the number of unmodified pixels is large enough...

SCALING RATIO The larger the ratio β, the fewer pixels are considered during scaling if the kernel width is bounded. In particular, the



(a) $\beta = 4$ ✓ (b) $\beta = 1.3$ ✗ (c) $\beta = 1$ ✗

(d) $\sigma = 4$ ✗ (e) $\sigma = 2$ ✗ (f) $\sigma = 1$ ✓

Figure 4.9: Influence of the scaling ratio and kernel size (see Figure 4.2 for the setting of this example); β and σ are the same horizontally and vertically. Plot (a)–(c) show manipulated images under varying ratios. Plot (d)–(f) show manipulated images under varying kernel sizes. The symbols ✓and ✗ indicate if the attack is successful.

number of pixels that are discarded growths quadratically with β. An adversary can thus easily control the ratio r by increasing the size of the source image.

...due to the scaling ratio...

Figure 4.9 (a)-(c) show the influence of the scaling ratio on the attack for a kernel with $\sigma = 1$. All images fulfill Objective O1, that is, the images are scaled down to the “cat” image. Depending on the scaling ratio, however, their success to Objective O2 changes. For a large ratio of $\beta = 4$, the attack image looks like the source, and the cat is not visible. For a smaller scaling ratio, the manipulated image becomes a mix of the source and target. For $\beta = 1$, the attack obviously fails.

KERNEL WIDTH The smaller the kernel width σ, the fewer pixels are considered during each convolution. While σ is typically not controlled by the adversary, several implementations of scaling algorithms make use of very small constants for this parameter. For example, the nearest-neighbor, bilinear, and bicubic kernels of the TensorFlow framework have a width of 1, 2, and 4, respectively.

...or the width of the scaling window.

Figure 4.9 (d)-(f) depict the influence of the kernel width on the attack for a fixed scaling ratio of $\beta = 4$. Again, all images fulfill Objective O1 and are scaled down to the “cat” image. For $\sigma = 1$, the attack also satisfies Objective O2 and is invisible. If two pixels are considered by the kernel, however, the cat becomes visible. For $\sigma = 4$, all pixels need to be manipulated and the attack fails.

The presented analysis is not limited to the scaling algorithms that are considered in this chapter. Any algorithm is vulnerable to image-scaling attacks if the ratio r of pixels with high weight is small enough. Developers are thus able to check quickly if their algorithms are vulnerable to these attacks. Overall, this general understanding of scaling attacks enables us to compare different scaling algorithms and to develop effective defense strategies in the following.

4.4 DEFENSES

We continue with the development of defenses that build on the previous analysis and address the root cause of image-scaling attacks—rather than fixing their symptoms. The developed defenses aim to *prevent* attacks without interfering with the typical pipeline of machine learning frameworks. They can thus serve as a plug-in for existing scaling algorithms.

This understanding of the attack's root cause allows...

Note that the mere *detection* of attacks is not sufficient here, as the scaling operation would need to be able to reject inputs as attack. This would change the API of imaging libraries. It would also require adapting existing learning pipelines. Prevention aims at a secure scaling operation by design—without rejecting inputs.

Consequently, we first derive requirements for secure scaling and use these to validate the robustness of existing algorithms (Defense 1). As only a few algorithms realize a secure scaling, we proceed to examine a generic defense that reconstructs the source image and thereby is applicable to any scaling algorithm as preprocessing (Defense 2).

...developing two defenses for prevention.

4.4.1 *Attacker Model*

For the construction and evaluation of the defenses, two types of adversaries are considered: a *non-adaptive* adversary who uses existing image-scaling attacks, and an *adaptive* adversary who is aware of the defense and adapts the attack strategy accordingly. Both adversaries have full knowledge of the scaling algorithm and the target size. In the adaptive scenario, the adversary additionally has full knowledge of the applied defense. Finally, the adversary is expected to freely choose the source and target image so that she can find the best match for conducting attacks in a given setup.

These assumptions are realistic due to the open-source nature of deep learning frameworks and the use of several well-known learning models in practice, such as VGG19 and Inception V3/V4. With black-box access to the scaling and the learning models, an adversary can even deduce the scaling algorithm and target size by sending a series of specially crafted images to the learning system [see 225].

4.4.2 *Defense 1: Robust Scaling Algorithms*

Let us start with the conception of an ideal robust scaling algorithm which serves as a prototype for analyzing the properties of existing algorithms.

AN IDEAL SCALING ALGORITHM In the ideal case, an algorithm investigates each pixel of the source image at least once for downscaling. The robustness of the scaling increases further if the employed convolution kernels overlap, and thus one pixel of the source contributes to multiple pixels of the scaled version. Technically, this requirement can be realized by *dynamically* adapting the kernel width σ to the scaling ratio β such that $\sigma \geq \beta$ holds. That is, the larger the ratio between the source and the scaled image, the wider the convolution kernel needs to become to cover all pixels of the image.

First, we can derive an ideal, robust scaling algorithm...

In addition to processing all pixels, an ideal algorithm also needs to weight all pixels equally; otherwise, a kernel with small support would leave pixels untouched if their weights become zero. For example, pixels close to the edge of the convolution window typically receive a very low weighting, as shown in Figure 4.7. Hence, the convolution of an ideal algorithm should be *uniform* and combine all pixels in the current kernel window with equal weight.

Although both properties—considering all pixels and a uniform convolution—can be technically implemented, they introduce challenges that can limit their practical utility: First, processing all pixels of an image slows down the scaling process. This is not necessarily a problem in applications where large neural networks are trained, and the overhead of scaling is minimal anyway. However, in real-time settings, it might be prohibitive to go over all pixels during scaling. Second, the flattened weighting of the convolution can blur the image content and remove structure necessary for recognizing objects. As a consequence, we identify a trade-off between security and performance in image scaling.

EXISTING SCALING ALGORITHMS Based on the concept of an ideal algorithm, it is possible to analyze existing scaling algorithms with respect to the processed pixels and the employed convolution kernels. Table 4.3 shows the results for the three considered imaging libraries after examining their source code. In particular, the source code of OpenCV version 4.1, Pillow 6.0, and tf.image 1.14 from TensorFlow were analyzed.

...and check existing algorithms if they correspond to it.

Library	**OpenCV**	**tf.image**	**Pillow**
Nearest	1	1	1
Bilinear	2	2	$2 \cdot \beta$
Bicubic	4	4	$4 \cdot \beta$
Lanczos	8	—	$6 \cdot \beta$
Area	β	β	β

Table 4.3: Kernel width σ for the scaling algorithms implemented by the imaging libraries OpenCV, tf.image (TensorFlow), and Pillow.

The results show that several scaling algorithms are implemented with fixed-size convolution kernels. For example, OpenCV and TensorFlow implement nearest-neighbor, bilinear, and bicubic scaling with a kernel width of 1, 2, and 4, respectively. Consequently, these algorithms become vulnerable once the scaling ratio exceeds the kernel width, and pixels of the source image are omitted during scaling.

Fortunately, one algorithm is implemented with a dynamic kernel width of β in all libraries: *area scaling*. This algorithm scales an image by simply computing the average of all pixels under the kernel window, which corresponds to a uniform convolution, as shown in Figure 4.7 for $\beta = 4$. Moreover, area scaling corresponds to a low-pass filter which mitigates the aliasing effect. As a result, area scaling provides strong protection from image-scaling attacks, and the algorithm is a reasonable defense if the uniform weighting of the convolution does not impact later analysis steps. The evaluation in Section 4.5 also empirically demonstrates the robustness of area scaling.

This shows that area scaling and...

The analysis provides another interesting finding: Pillow stands out from the other imaging library, as it implements a dynamic kernel width for all algorithms except for nearest-neighbor scaling. The dynamic kernel width σ is chosen such that the convolution windows substantially overlap, for example, for bicubic and Lanczos scaling by a factor of 4 and 6, respectively. Although the used convolutions are not uniform for these algorithms, this overlap creates a notable obstruction for the attacker, as dependencies between the overlapping windows need to be compensated. Figure 4.10 schematically shows the dynamic kernel width of Pillow in comparison to the implementations of OpenCV and TensorFlow.

...most of Pillow's scaling algorithms should be secure.

DISADVANTAGES While area scaling and the Pillow library provide a means for robust scaling, they also induce drawbacks. As exemplified in Figure 4.11, the algorithms cannot entirely remove all traces from the attacks. Small artifacts can remain, as the manipulated pixels are not cleansed and still contribute to the scaling, though with limited impact. The evaluation in Section 4.5 shows that these remnants are not enough to fool the neural network anymore. The predicted class for the scaled images, however, is not always correct due to the noise of the attack remainings. As a remedy, we examine an alternative defense in the next section that reconstructs the source image and is thus applicable to any scaling algorithm. This reconstruction removes attack traces, so that the classifier predicts the original class again.

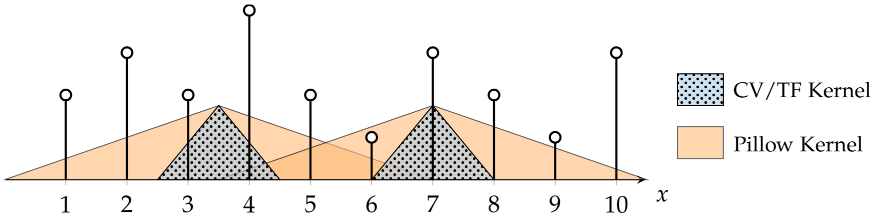


Figure 4.10: Comparison of bilinear scaling for Pillow, OpenCV and TensorFlow. The latter two fix σ to 2, while Pillow uses a dynamic kernel width.

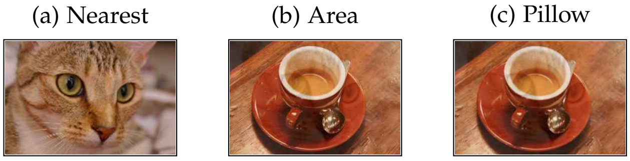

(a) Nearest (b) Area (c) Pillow

Figure 4.11: Comparison of scaling algorithms: (a) insecure nearest-neighbor scaling, (b) robust area scaling, and (c) robust scaling from Pillow. Note the visible attack traces in (b) and (c).

4.4.3 *Defense 2: Image Reconstruction*

Second, we can reconstruct manipulated pixels...

The following defense is constructed around the main working principle of image-scaling attacks: The attacks operate by manipulating a small set of pixels that controls the scaling process. With knowledge of the scaling algorithm, we can precisely identify this set of pixels in the attack image. A naive defense strategy that removes this set would effectively block any attack, yet it would corrupt the scaling, as all relevant pixels are removed. Instead, a better strategy is to identify all pixels processed by a scaling algorithm and then to reconstruct their content using the remaining pixels of the image.

Reconstructing pixels in images is a well-known problem in image processing, and there exist several methods that provide excellent performance in practice, such as techniques based on wavelets and shearlets [e.g., 181, 198]. These involved approaches, however, are difficult to analyze from a security perspective, and their robustness is hard to assess. Hence, let us examine two simple reconstruction methods for the considered pixels that possess transparent security properties: a *selective median filter* and a *selective random filter*.

...with the median of their neighborhood...

SELECTIVE MEDIAN FILTER Given a scaling algorithm and a target size, this filter identifies the set of pixels $\mathcal{P}$ in the input image that is processed during scaling. For each of the pixels $p \in \mathcal{P}$, it determines a window W_p around p, similar to a convolution kernel, and computes the median pixel value for this window. To make the computation robust, the size of this window is defined as $2\,\beta_h \times 2\,\beta_v$, which ensures that half of the pixels overlap between the different windows and thus hinders existing scaling attacks. In addition, other manipulated points $p' \in \mathcal{P}$ in W_p are considered by excluding them from the computation of the median. Figure 4.12 depicts the basic principle of the selective median filter.

In comparison to other approaches for reconstructing the content of images, this defense builds on the statistical robustness of the median operation. Small groups of pixels with high or low values are compensated by the median. On average, the adversary is required to change about 50% of the pixels in a window to reach a particular target value for the median. The evaluation demonstrates that non-adaptive as well as adaptive adversaries are not capable of effectively

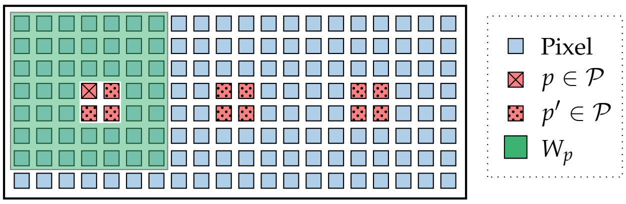


Figure 4.12: Image reconstruction based on a selective median filter.

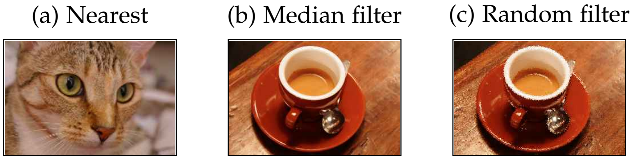


Figure 4.13: Examples of second defense: (a) insecure nearest-neighbor scaling, (b) robust scaling via selective median filter, and (c) selective random filter. Note that attack traces are not visible anymore.

manipulating these median values without introducing strong visible artifacts (see Section 4.5).

The robustness of the median filter comes at a price: Computing the median for all pixels in each window W_p for all $p \in \mathcal{P}$ yields a run-time complexity of $\mathcal{O}(|\mathcal{P}| \cdot \beta_h \cdot \beta_v)$. That is, the run-time growths quadratically with the scaling ratio. While this overhead might be neglectable when working with large neural networks, there also exist applications in which more efficient scaling is necessary. Providing secure and efficient scaling, however, is a challenging task, as the robustness of a scaling algorithm increases with the number of considered pixels.

SELECTIVE RANDOM FILTER A selective random filter represents an alternative that tackles the problem of efficiency by taking a random point from each window instead of the median. This method, however, comes with two problems. First, the reconstruction becomes non-deterministic. Second, the scaled image might suffer from poor quality. This filter is therefore suitable for applications that demand a very efficient run-time performance and can tolerate a loss in visual quality. Appendix C.2 outlines the filter in more detail.

...or with a random pixel from the neighborhood.

In summary, two reconstruction methods are presented that target the core of image-scaling attacks. As exemplified by Figure 4.13, both restore the pixels that an adversary changes. These methods prevent the attacks and can be easily used in front of existing scaling algorithms, so that almost no changes are necessary to the typical workflow of machine learning systems.

4.5 EVALUATION

The defenses are evaluated regarding...

We continue with an empirical evaluation of the defenses against image-scaling attacks. In Section 4.5.2 and 4.5.3, we study the security of robust scaling algorithms (Defense 1). In Section 4.5.4 and 4.5.5, we examine the novel defense based on image reconstruction (Defense 2). For each defense, we study the evaluation with a non-adaptive adversary that performs regular image-scaling attacks and then proceed to investigate an adaptive adversary who tries to circumvent the defenses.

4.5.1 *Experimental Setup*

...the scaling output and visibility of input changes.

To evaluate the efficacy of the defenses, we have to consider Objective O1 and Objective O2 of image-scaling attacks. If a defense is capable of impeding one of these objectives, the attack fails. For example, if the control of the adversary over the source is restricted such that the classification of the scaled version is not changed, the defense has foiled O1. Similarly, if the embedded target image becomes clearly visible, the defense has thwarted O2. Consequently, the experiments are designed along these two objectives.

The evaluation includes different scaling ratios and...

DATASET & SETUP The ImageNet dataset [178] with a pre-trained VGG19 model [190] is used for evaluation. This deep neural network is a standard benchmark in computer vision and expects input images of size $224 \times 224 \times 3$. From the dataset, 600 images as an unmodified reference set and 600 source images for conducting attacks are randomly sampled. For each source image, a target image from the dataset is randomly selected, ensuring that both images have different classes and predictions. To investigate different scaling ratios, the images are sampled such that 120 images are obtained for each of the following five intervals of ratios: $[2,3), [3,4), [4,5), [5,7.5), [7.5,10)$. As we have one ratio for the vertical and one for the horizontal direction for each image, the minimum of both is considered for this assignment.

The image-scaling attacks are implemented in the strong variant (cf. Section 4.2.2), but with a slight improvement to the original attacks: Instead of using a fixed value for ϵ, its value is gradually increased from 1 up to 50 if the quadratic programming solver cannot find a solution. During the evaluation, cases were observed where single columns or rows required a larger ϵ to find a feasible solution. In this way, the attack's success rate can be increased, if only a single part of an image may require a higher ϵ value.

...algorithms from OpenCV, Pillow, and TensorFlow.

As scaling algorithms, the implementations of nearest-neighbor, bilinear, bicubic, and area scaling from the libraries OpenCV (version 4.1), Pillow (version 6.0), and tf.image (version 1.13) from TensorFlow are considered. The Lanczos algorithm is omitted, as it provides comparable results to bicubic scaling in the experiments due to the similar convolution kernel and kernel width (see Figure 4.7).

EVALUATION OF O1: PREDICTIONS USING VGG19 It is checked if the deep neural network VGG19 predicts the same class for the scaled image scale(A) and the target image T. As there are typically minor fluctuations in the predicted classes when scaling with different ratios, the commonly used top-5 accuracy is applied. That is, O1 is fulfilled if a match exists between the top-5 predictions for the target image T and the scaled image scale(A).

EVALUATION OF O2: USER STUDY The second objective is investigated with a user study with 36 human subjects who have different professional background, ranging from students to teachers and researchers. The participants obtain 3 attack images for each interval of scaling ratio and are asked to visually identify one or more of three classes, where one class corresponds to the source image, one to the embedded target image and the third to an unrelated class. An attack is considered as successful, if a participant selects the class of the source image only and does not notice the target image.

EVALUATION OF O2: PSNR As quantitative measurement, the peak signal to noise ratio (PSNR), a common metric in image processing [74], is additionally used to measure the difference between the unmodified source image and its attack image. Formally, the PSNR for the attack image A and the source image S is defined as

$$\mathrm{PSNR}(A, S) = 10\ \log_{10}\left(\frac{I_{\mathrm{max}}^2}{\frac{1}{N} \parallel A - S \parallel_2^2}\right). \tag{4.10}$$

The denominator represents the mean squared error between both images with N as the total number of pixels. I_{max} is the maximum of the pixel range. A high PSNR value (larger than 25 dB) indicates a strong match between two images. As a conservative choice, the attack is considered unsuccessful if the PSNR value is below 15 dB. Experiments were also conducted with more advanced methods for comparing the quality of images, such as feature matching based on scale-invariant feature transform (SIFT) analysis [133]. This technique, however, shows the same trends as the simple PSNR measurement and is thus omitted in the following.

4.5.2 *Defense 1: Non-Adaptive Attacks*

In the first experiment, we study the robustness of existing scaling algorithms from OpenCV, TensorFlow, and Pillow against image-scaling attacks. Note that area scaling is examined in the following Section 4.5.3, as it is not vulnerable to standard image-scaling attacks.

Existing scaling algorithms enable...

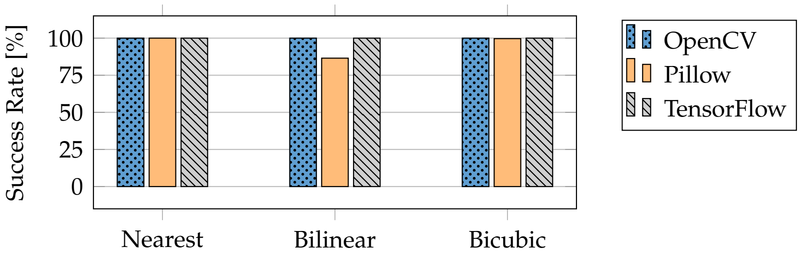


Figure 4.14: Success rate of image-scaling attacks regarding O1: the number of classifications with target class after scaling.

...an adversary to control the scaling output while...

EVALUATION O1 Figure 4.14 shows the performance of the attack as the ratio of classifications with the wanted target class after scaling. The attack is successful with respect to O1 for all scaling algorithms from OpenCV, TensorFlow, and Pillow. An exception is Pillow's bilinear scaling where the success rate is 87%, as a feasible solution is not found for all source and target pairs here. Overall, the results confirm that an attacker can successfully manipulate an image such that its scaled version becomes a target image, irrespective of the scaling algorithm or library. This manipulation, however, is not sufficient for a successful attack in practice, as visual traces may clearly indicate the manipulation and undermine the attack. Thus, O2 is also evaluated in this experiment.

...input changes are not visible with a large scaling ratio,...

...except for Pillow.

EVALUATION O2 Figure 4.15 shows the results from the user study that investigates the visual perception of the generated attack images. In line with the previous theoretical analysis, the attack is successful against OpenCV and TensorFlow, once a certain scaling ratio is reached (red bars in Figure 4.15). We can observe that for ratios exceeding 5, most attack images are not detected by the participants. However, for the implementations of bilinear and bicubic scaling in the Pillow library, the participants always spot the attack and identify the embedded target class in the source image. This result confirms the analysis of the implementations in Section 4.4.2 and the vital role of the dynamic kernel width used by Pillow.

In addition, Figure C.2 in Appendix C.3 reports the PSNR values between the attack and source image over the entire dataset. We observe the same trend as in the user study. For OpenCV and TensorFlow, the images become similar to each other with a larger β, reaching PSNR values above 25 dB. On the contrary, the PSNR values for Pillow's bilinear and bicubic scaling algorithm remain below 15 dB irrespective of β, underlining that O2 is not achieved even with a larger scaling ratio.

SUMMARY The results confirm that image-scaling attacks are effective against several scaling algorithms in popular imaging libraries. The attacks succeed in crafting images that are classified as the target

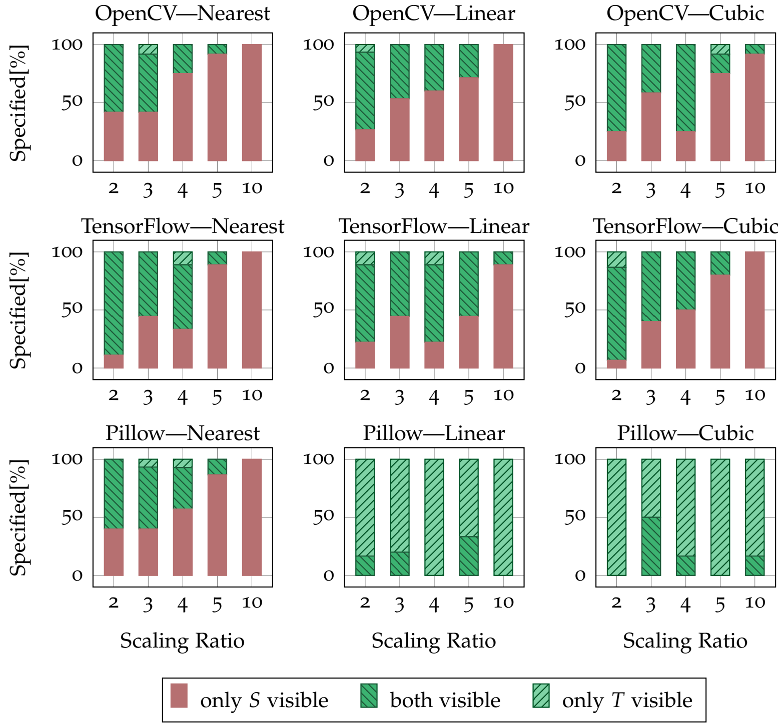


Figure 4.15: User study on image-scaling attacks regarding O2. The attack is successful if only the source image S is visible (red).

class. However, the visibility of the attacks depends on the scaling ratio and the kernel width. In the case of Pillow, the attack fails to hide the manipulations from a human viewer for bilinear and bicubic scaling. Thus, these implementations of scaling algorithms can be considered robust against a non-adaptive adversary in practice.

4.5.3 *Defense 1: Adaptive Attacks*

In the second experiment, we consider an adaptive adversary that specifically seeks means for undermining robust scaling. To this end, first an attack against the implementation of the Pillow library is constructed (Section 4.5.3.1) and then attacks against area scaling are examined (Section 4.5.3.2 and 4.5.3.3).

As even an adaptive adversary...

4.5.3.1 *Attacking the Pillow Library*

The previous analysis shows that image-scaling attacks fail to satisfy Objective O2 when applied to the Pillow library. The dynamic kernel width forces the attack to aggressively change pixels in the source, so that the target image becomes visible. As a remedy, an adversary can limit the number of changed pixels. To build on the successful attacks against OpenCV and TensorFlow, the following adaptive strategy is tested: 2 pixels in each kernel window are allowed to be freely changed while using images with $\beta \in [4, 5)$. The goal is to find a modification for these pixels such that the convolution over the whole kernel yields the target value. To increase the chances to obtain a feasible solution, the remaining pixels are allowed to be changed by 10 at most. The experiment from the previous section is again conducted with this new constraint and results for 120 image pairs with $\beta \in [4, 5)$ for bilinear and bicubic scaling are reported, respectively.

...cannot bypass Pillow's bilinear and bicubic scaling...

RESULTS The added constraint severely impacts the success rate of the attack. The rate drops to 0% for bilinear scaling and to 0.83% for bicubic scaling. That is, Objective O1 is not reached anymore. In the majority of cases, no feasible solution exists and several columns of the source image are not modified. Only in a single case, the attack is successful for bicubic scaling. However, the attack image shows obvious traces from the target image, clearly revealing the attack.

4.5.3.2 *Attacking Area Scaling*

Area scaling stands out from the other algorithms as it employs a uniform weighting of pixels and operates on rectangular blocks instead of columns and rows. As a result, the original attack by Xiao et al. [225] is not applicable to this scaling algorithm. To attack area scaling, two novel attack strategies are thus examined in the following.

The first strategy aims at slightly changing all pixels of a block to control its average. That is, the adversary seeks a minimal perturbation under the L_1 norm such that the average of the block becomes the targeted value. For a target value t, she solves the following optimization problem:

$$\min(\|\tilde{\Delta}\|_1) \text{ s.t. } \left\|\operatorname{avg}(\tilde{S} + \tilde{\Delta}) - t\right\|_\infty \leqslant \epsilon \,, \tag{4.11}$$

where $\tilde{S}$ is the current block, $\tilde{\Delta}$ its perturbation and ϵ a small threshold. The results for the L_2 norm are equivalent and thus omitted.

The second strategy aims at adapting only a few pixels of a block while leaving the rest untouched. To this end, the adversary can optimize the L_0 norm, so that only the number of changed pixels counts. The attack works as follows for a current image block: if the target value is larger than the current average, the adversary iteratively sets pixels in the source to $I_{\max}$ until the target is reached. If the target is smaller, she iteratively sets pixels to 0. Note that the last value generally needs to be adapted such that the average becomes the target value.

RESULTS With respect to Objective O1, both the L_1 and L_0 attack are successful in 100% of the images. However, both variants fail reaching Objective O2 in all of the cases. A manual inspection of the images reveals that the source is largely overwritten by both attacks and parts of the target become visible in all attack images. Figure 4.16a provides results on this experiment by showing the distribution of PSNR values

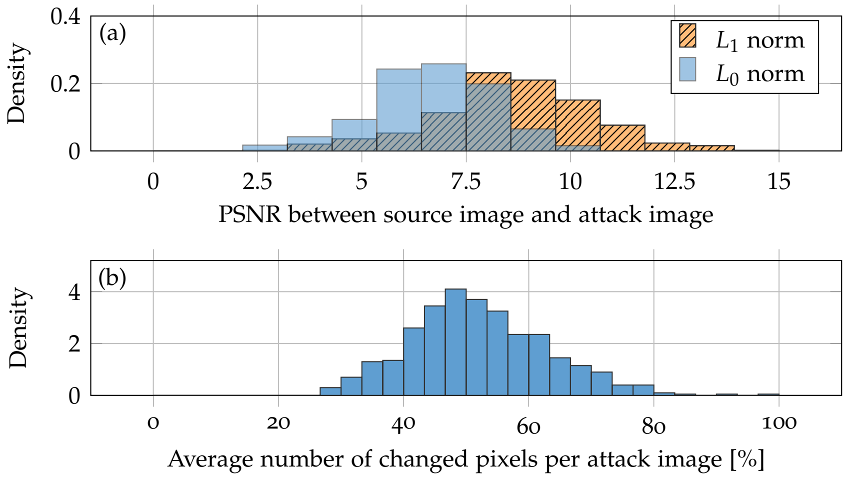


Figure 4.16: Adaptive attack against area scaling: (a) Distribution of PSNR values and (b) the average number of changed pixels by the L_0-based attack.

over all source-attack image pairs. The average PSNR is 8.6 dB for L_1 and 6.7 dB for L_0, which corresponds to a very low similarity between the source and the attack image. In addition, Figure 4.16b depicts the distribution of changed pixels for the L_0 attack. While for the majority around 50% of the pixels are changed, a few images only require to change 28%. Still, this is too much to achieve O2. Figure C.3 in Appendix C.3 shows the five best images from the evaluation with the smallest number of changed pixels. In all cases, the source image cannot be recognized anymore.

...as well as area scaling,...

4.5.3.3 *Selective Source Image*

In addition to the two adaptive attacks, we also examine area scaling under a more challenging scenario. In this scenario, the adversary selects the most suitable source image for a fixed target. As a result, the class of the source image is arbitrary and potentially suspicious, yet the attack becomes stronger due to the selected combination of source and target. This strategy is implemented as follows: For each target image T, the adversary chooses the source image S, for which the scaled version has the smallest average distance to the target image. Fewer changes are thus required to obtain a similar output after scaling. Results are reported for the 100 best novel source-target pairs in the following.

RESULTS As before, both the L_1 and L_0 attack are successful in 100% of all cases regarding Objective O1. However, the attack again largely overwrites the source image such that the target is visible in all cases. The examples from Figure C.4 in Appendix C.3 underline that the attack fails to keep the changes minimal, although the source and target are similar to each other. The average PSNR value is 16 dB for L_1 and 12 dB for L_0. Both are slightly higher than in the non-selective scenario but still far too low compared to successful examples from Section 4.5.2.

...these algorithms can be recommended, confirming the theoretical analysis.

SUMMARY We can conclude that Pillow's bilinear and bicubic algorithm also withstand an adaptive adversary. Likewise, area scaling is robust against different adaptive attacks, including the selection of source images. The attacks in this section are a best effort for assessing the security of scaling and confirm the theoretical analysis from Section 4.4.2. In summary, scaling algorithms in Pillow (except for nearest scaling) as well as area scaling can be recommended.

4.5.4 *Defense 2: Non-Adaptive Attacks*

The reconstruction defense also prevents...

We proceed with evaluating the novel defense for reconstructing images from Section 4.4.3. In particular, the selective median or random filter is combined with a vulnerable scaling algorithm and the robustness of the combination is tested. As attacks, all manipulated images from Section 4.5.2 that satisfy the objectives O1 and O2 for one scaling algorithm are considered. This includes attacks against nearest-neighbor scaling from all imaging libraries as well as attacks against bilinear and bicubic scaling from OpenCV and TensorFlow.

...an attacker from controlling the scaling output...

EVALUATION O1 The two filters prevent all attacks. When they are employed, no attack image succeeds in reaching Objective O1 for the respective scaling algorithm. The image reconstruction effectively removes the manipulated pixels and thereby prevents a misclassification of the images.

...while restoring the visual quality of the attack image...

EVALUATION O2 As the original image content is reconstructed, the visual difference between the source and the reconstructed images are minimal. Figure 4.17 depicts the distribution of PSNR values between each source and attack image—before and after reconstruction. The quality considerably increases after restoration and reaches high PSNR values above 25 dB. Figure C.5 in Appendix C.3 provides some examples before and after reconstruction.

...and recovering the original image's prediction.

RECONSTRUCTION ACCURACY Table 4.4 depicts the success rate of reconstructing the attack image's original prediction, that is, obtaining the prediction of the actual source image. The median filter recovers the predictions in almost all cases successfully. For the random filter, the success rate is slightly reduced due to the loss in visual quality.

In addition, the impact of both filters on benign, unmodified images is measured. The median filter runs with almost no loss of accuracy.

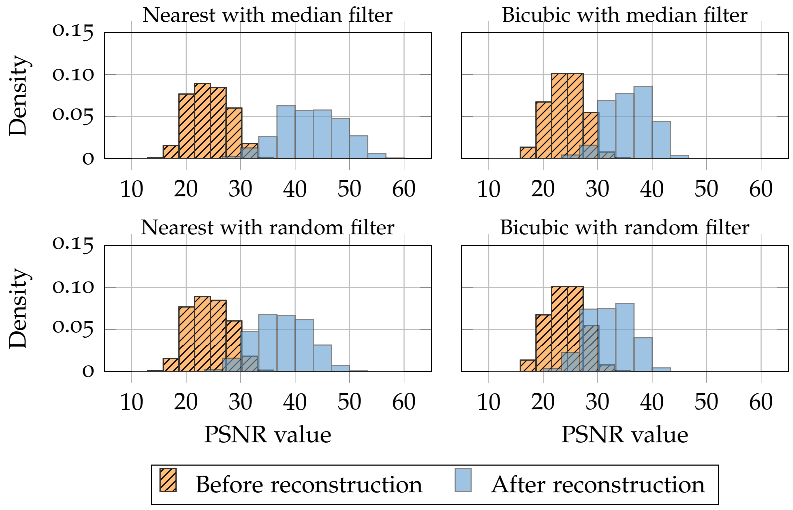


Figure 4.17: PSNR distribution before and after attack image reconstruction for median and random filter on OpenCV's scaling algorithms. Results for the other scaling algorithms are similar and thus omitted.

The random filter induces a small loss which can be acceptable if a low run-time overhead of a defense is an important criterion in practice.

All in all, the results show that it is possible to prevent the attack, and to recover the original prediction in addition.

RUN-TIME EVALUATION Finally, the run-time performance of both filters is evaluated. To this end, the median and the random filter are each applied in combination with nearest-neighbor scaling. For comparison, nearest-neighbor scaling alone, area scaling, and a forward pass of VGG19 are evaluated as well. Each setting is applied to the same 2,000 images and the average run-time per image is measured. The test system is an Intel Xeon E5-2699 v3 with 2.4 GHz. Figure 4.18

Library	**Algorithm**	**Median**		**Random**	
		Attacks	**Benign**	**Attacks**	**Benign**
OpenCV	Nearest	99.6%	99.0%	89.3%	89.1%
	Bilinear	100.0%	99.4%	97.7%	98.0%
	Bicubic	100.0%	99.2%	91.4%	93.4%
TensorFlow	Nearest	99.6%	99.0%	88.9%	89.1%
	Bilinear	100.0%	98.9%	97.7%	97.7%
	Bicubic	100.0%	99.4%	91.7%	92.0%
Pillow	Nearest	100.0%	99.6%	88.1%	90.4%

Table 4.4: Performance of defense in terms of recovering correct outputs from the attack images, as well as impact on benign, unmodified images.

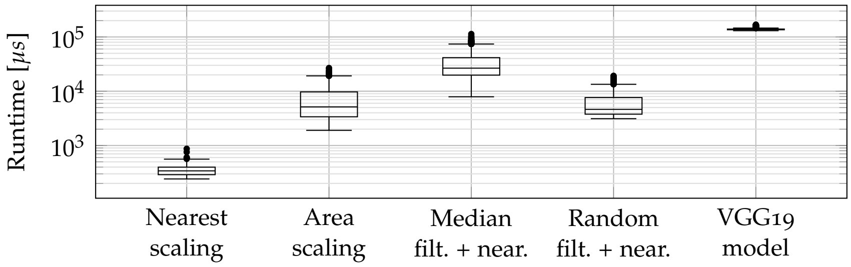


Figure 4.18: Run-time performance of nearest-neighbor and area scaling as well as the reconstruction defenses in combination with nearest-neighbor scaling. A forward pass of VGG19 is also shown.

shows the measurements on a logarithmic scale in microseconds. Area scaling as well as the median and the random filter introduce a notable overhead and cannot compete with the insecure nearest-neighbor scaling in performance. However, in comparison to a pass through the VGG19 model, they are almost an order of magnitude faster and induce a neglectable overhead for deep learning systems.

SUMMARY This experiment shows that both the median and random filter provide effective defenses against non-adaptive attacks. In contrast to robust scaling, the filters prevent the attack and reconstruct the original prediction.

4.5.5 *Defense 2: Adaptive Attacks*

Even an adaptive adversary...

Finally, it remains to investigate the robustness of the proposed reconstruction methods against an adaptive adversary who is aware of the methods and adapts her attack accordingly. We thus analyze two strategies that aim at misleading the image reconstruction of attack images. Both strategies attempt to manipulate the reconstruction of the pixels $p \in \mathcal{P}$ such that they keep their value after applying the median or random filter.

MEDIAN FILTER The attack strategy for the median filter is as follows: Given a window W_p around $p \in \mathcal{P}$, we denote by m the current median of W_p. Note that p is not part of W_p (see Figure 4.12). The adversary seeks a manipulation of the pixels in W_p, so that $m = p$. Hence, applying the median filter will not change p and the adversarial modification remains. Without loss of generality, let us assume that $m < p$. In order to increase m, the adversary needs to set more pixels to the value of p. She starts with the highest pixel value that is smaller than p and sets it to p. She continues with this procedure until the median equals p. Appendix C.4 shows that this attack strategy is *optimal* regarding the L_0, L_1, and L_2 norm if the windows W_p do not overlap. A smaller number of changes to the image cannot ensure

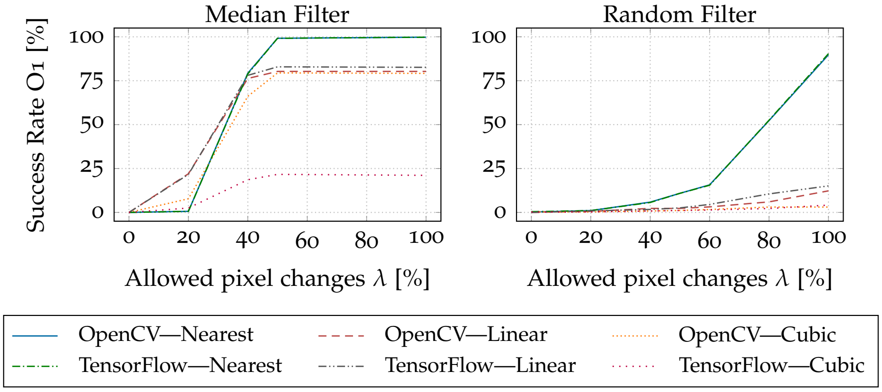


Figure 4.19: Success rate of the adaptive attacks against the reconstruction defense regarding O1. Note O2 is not satisfied (see Figure 4.20).

that $m = p$. These results give a first insight on the robustness of the median filter. A considerable rewriting is necessary to change the median, even in the overlapping case where an adversary can exploit dependencies across windows.

In the experiments, the maximum fraction λ of changeable pixels per window is varied. This bound allows measuring the filter's robustness depending on the L_0 norm.

RANDOM FILTER For the random filter, the attack strategy increases the probability that the target value in a window W_p is selected. To this end, the adversary is assumed to set a fraction λ of all pixels in W_p to p. To minimize the number of changes to the image, she replaces only those pixels in the window with the smallest absolute distance to p. This strategy is optimal in the sense that manipulation with fewer changes would result in a lower probability for hitting the target value p.

RESULTS Figure 4.19 shows the success rate of the adaptive attacks regarding Objective O1 for OpenCV and TensorFlow. The results for Pillow's nearest-neighbor scaling are similar and thus omitted. The adaptive attacks need to considerably modify pixels so that the manipulated images are classified as the target class. The median filter is robust until 40% of the pixels in each window can be changed. Against the random filter, a higher number of changed pixels is necessary to increase the probability of being selected.

...cannot bypass the defense without...

With respect to Objective O2, both filters withstand the adaptive attacks and thus remain secure. Rewriting 20% of the pixels already inserts clear traces of manipulation, as exemplified by Figure C.6 in Appendix C.3. In all cases, the attack image is a mix between source and target class. The results for the random filter are similar.

...clearly visible input changes.

In addition, Figure 4.20 shows the results from a user study for the median filter. The participants identify the attacks in the vast

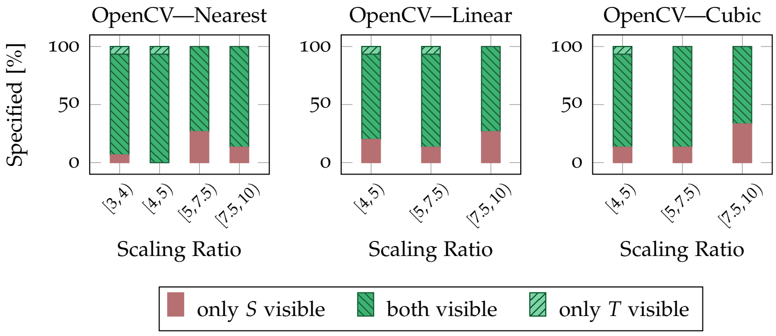


Figure 4.20: User study to determine the success rate of the adaptive attack against the median filter with respect to O2. If the source class was only recognized, the distortion was too strong to determine the target class (see Figure C.7). O2 is thus violated in any case.

majority of the cases. In a few cases, the participants only recognized the source class. A closer analysis reveals that the distortion in these cases is so strong that the detection of particular classes is difficult. As a result, the participants did not specify the target class. Figure C.7 in Appendix C.3 shows examples for these cases.

SUMMARY The results from the adaptive attack provide strong empirical evidence for the robustness of the defense based on image reconstruction. If a vulnerable scaling algorithm needs to be used in a machine learning system or the reconstruction of the original class is essential, using one of the filters as a preprocessing step is recommended.

4.6 DISCUSSION

We continue with a discussion of a side effect that allows enforcing the usage of nearest-neighbor scaling. Then, limitations of the analysis in this chapter are discussed.

4.6.1 *Downgrade Attack to Nearest Scaling*

Some libraries always apply nearest scaling for specific input sizes, which...

The root-cause analysis also reveals a side effect in the implementation of $\nu(p)$ (see Equation 4.6) in OpenCV and TensorFlow. An adversary can enforce the usage of nearest scaling by choosing a respective scaling factor although the library is supposed to use bilinear, bicubic or Lanczos scaling. In particular, if the scaling ratio is an uneven integer, $\beta = 2x + 1,\ x \in \mathbb{N}$, OpenCV is effectively using nearest scaling. In TensorFlow, each integer with $\beta \in \mathbb{N}$ leads to the same effect. Thus, if the adversary can control the source image size, she can resize her image before to obtain the respective scaling factor. This in turn allows her to perform a more powerful scaling attack by creating attack images with less distortion, as the ratio of considered pixels decreases (see Section 4.3.3). Note that this issue is not exploited in the evaluation. A variety of scaling factors are evaluated to draw general conclusions on scaling attacks.

...allows reducing the input changes.

Library	$\nu(\cdot)$
OpenCV	$\nu(p) = (p + 0.5) \cdot \beta - 0.5$
TensorFlow	$\nu(p) = p \cdot \beta$ (*)
Pillow	$\nu(p) = (p + 0.5) \cdot \beta$

(*) The scaling function in TensorFlow can be changed to the definition from OpenCV. However, this option is not exposed in `tf.image.resize_images`, the high level resizing API.

Table 4.5: Implementation of $\nu(p)$ in OpenCV, TensorFlow and Pillow.

To understand the reason for this attack possibility, we need to consider the mapping $\nu(p)$ and the kernel w. Table 4.5 shows the slightly different implementations of $\nu(p)$ in OpenCV, TensorFlow and Pillow. For OpenCV, for instance, if β is an uneven integer, $\nu(p)$ will always be an integer. Thus, only one pixel will be used for the convolution. A closer look on the definition of the kernels in Figure 4.7 reveals the underlying reason. Common kernels $w(x)$ are zero at all integer positions $x \in \mathbb{Z} \backslash \{0\}$. Hence, if $\nu(p)$ is an integer and the kernel is positioned around it, each neighboring pixel obtains a weight of zero. Thus, only the pixel at position $\nu(p)$ is used. This behavior corresponds to nearest scaling and can be well observed in Figure 4.6 with the second window. In practice, the effect is observed for bilinear, bicubic and Lanczos scaling in OpenCV and TensorFlow. On the contrary, Pillow makes use of a dynamic kernel width, so that this behavior is not observed in this case.

4.6.2 *Limitations*

Next, the limitations of the analysis in this chapter are discussed.

FOCUS ON SCALING We focus on the scaling operation in this chapter. It gives a general understanding about the security impact of the mapping from $\mathcal{Z}$ to $\mathcal{F}$. Yet, other operations, such as normalization, data augmentation or file parsing, can also induce an attack surface and should be explored in future work systematically. The next section provides a further example from the text domain to underscore that not only scaling is vulnerable (Section 4.7.1).

SIMILAR SOURCE-TARGET PAIR The findings show that scaling attacks can be blocked reliably. To circumvent the defenses, an adversary would need to choose a source-target pair where the pixel difference is sufficiently small. In this case, however, both images would effectively show the same content, so that the impact of this attack is questionable. The main threat of a scaling attack is to create an unrelated output image after downscaling, so that all further steps in the learning pipeline can work correctly and do not need to be exploited. For instance, assume that an adversary creates an adversarial example and hides the adversarial perturbations with a scaling attack. This combination misleads the prediction. It also circumvents the proposed scaling-attack defenses, since the source-target pair effectively shows the same content. However, the threat model is now weaker, as it substantially depends on the knowledge and capabilities to create an adversarial example. In this case, the attack becomes dependent on the feature extraction and learning model.

CONSIDERED SCALING ALGORITHMS Moreover, the analysis in this chapter focuses on common scaling algorithms that are provided by the imaging libraries OpenCV, Pillow and TensorFlow (see Table 4.1). Yet, the insights are not limited to these scaling algorithms. Any algorithm is vulnerable if only a subset of pixels is used for scaling. The root-cause analysis enables developers to quickly access the risk for other scaling algorithms.

DETECTION DEFENSES This chapter analyzes various defense concepts for preventing scaling attacks. Detection strategies are another concept that decide if an image under investigation was manipulated to cause another result after downscaling [225]. This can be used to find out that an attack is going on. For instance, it allows us to scan a dataset in advance to spot a poisoning attack. However, for machine learning systems in a production mode, the rejection of images would subtly change the API, which might not be acceptable. Defenses that prevent the attack as proposed in Section 4.4 can thus be more easily employed. Moreover, prevention blocks an attack by design in any case while currently developed detection defenses cannot spot attacks without false negatives or false positives [166, 225].

4.7 FURTHER ATTACK SURFACES IN THE MAPPING

Finally, we examine the security impact of the mapping in further cases. In particular, we explore another attack on the mapping from the text domain, and then examine fragile camera fingerprints as novel perspective on the mapping.

4.7.1 *Attack on the Mapping in the Text Domain*

Although we focused on scaling, the mapping...

The scaling operation is not the only mapping that can induce an attack surface. Let us consider the following example from the text domain to obtain a better intuition how the mapping can be relevant in other applications.

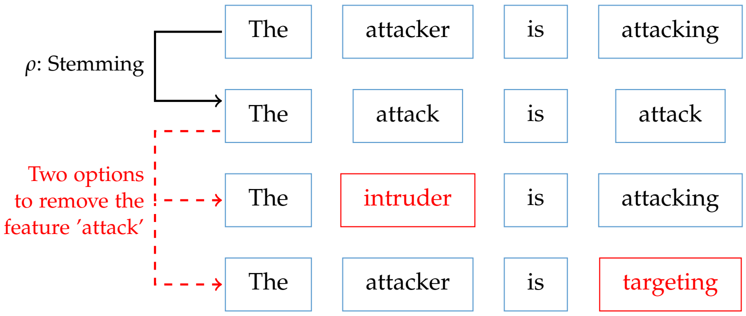


Figure 4.21: Text stemming as example for an attack on the mapping ρ in the text domain. The adversary wants to remove the feature *attack* from the text for her adversarial example. Stemming provides her with two options to achieve this.

Stemming is a preprocessing step to normalize text [130]. It reduces words to their word stem. For example, the word *attacker* is reduced to *attack*. If learning algorithms, for instance, classify the topic of a text, they can benefit from stemming, since redundant words and thus features are unified [e.g., 43].

Although reasonable for text classification, it gives an adversary more possibilities to manipulate text. Different words are mapped to the same word stem. Figure 4.21 exemplifies that both words *attacking* and *attacker* are reduced to the same stem. Let us assume that the adversary has to reduce the occurrence of *attack* by 1 to create an adversarial example. Stemming allows her to either replace *attacker* or *attacking*, giving her more degrees of freedom for manipulation. This can simplify finding inconspicuous locations to replace words, so that the plausibility constraint can be fulfilled (remember Section 3.3). Although text stemming does not create a comparable attack surface to scaling, it simplifies an attack and highlights that other mappings can also induce an attack surface.

...in other areas such as text processing also provides an attack surface.

4.7.2 *Fragile Camera Fingerprints*

Not only the adversary, but also a defender is able to exploit particularities in the mapping. As example, let us examine the concept of fragile camera fingerprint in the following. To begin with, a camera fingerprint is a signal that is unnoticeably present in any image taken by the same camera, but differs between images from different cameras. It has found widespread applications in forensics to attribute digital images to their source camera [75]. Recent works have also proposed to use the fingerprint as a means to link mobile device authentication to inherent hardware characteristics of the mobile device [14, 212]. In practice, however, these use cases face the problem of *fingerprint copy-attacks* [85, 134]. If a camera owner shares images from her camera with the public, an adversary can estimate the fingerprint, plant it into her own images, and pretend that an arbitrary image was captured by the owner's camera. As a proactive defense, Quiring and Kirchner [164] have introduced the concept of *fragile camera fingerprint* that exploits an *asymmetry* in the quality of accessible data. The camera owner only shares JPEG-compressed images with the public while retaining her uncompressed images private. As a result, by using the public JPEG images, the adversary's estimate of the camera owner's fingerprint will only contain the part that is robust to lossy JPEG compression. The estimate will lack the component that is fragile to compression. Quiring et al. [170] analytically and empirically establish that the fragile component cannot be estimated from JPEG-compressed images with common compression levels.

Therefore, it is possible to apply a mapping that only extracts the fragile information before deducing the fingerprint. As a result, the camera owner will always be able to use a fingerprint that the adversary cannot compute. Taken together, this example shows that the mapping can also be considered to increase security, for example, by proactively adjusting the mapping in an asymmetric data scenario. Such scenario can be relevant in machine learning if data should be used for private learning, and made publicly available to some extent. For example, this can be the case with personal information on a mobile device as discussed by Kurtz et al. [116].

4.8 RELATED WORK

Image-scaling attacks represent a novel threat to the security of machine learning systems. As a consequence, there exists only a small body of related approaches that are discussed in the following.

Chen et al. [50] extend the original scaling attack [225] by mainly studying different norms for Equation 4.1. The attack analysis in Section 4.3 shows that this does not affect the attack's working principle and thus the proposed defenses. Quiring and Rieck [166] elaborate on the application of scaling attacks for the poisoning scenario. In particular, the work shows that scaling attacks allow an adversary to significantly conceal image manipulations of a backdoor attack [92] and overlay-poisoning attack [184]. Note that both enhanced attacks are shortly introduced in Appendix C.1. Moreover, the work examines the detection defenses by Xiao et al. [225] for scaling attacks. First, Quiring and Rieck [166] show that the detection methods cannot reliably detect scaling attacks in the backdoor scenario. Second, although attacks in the overlay-poisoning scenario can be detected, an adaptive variant for the scaling attack is derived that bypasses the detection again.

This chapter extends this line of research. We comprehensively study scaling attacks as an attack on the mapping from $\mathcal{Z}$ to $\mathcal{F}$, together with the root cause of scaling attacks and defenses for prevention.

In comparison to prior work in adversarial learning, scaling attacks differ in two important aspects: (i) They affect all further steps of a machine learning system. Scaling attacks are thus agnostic to feature extraction and learning models, giving rise to general adversarial examples and poisoning. (ii) This chapter shows that the exploited vulnerability can be effectively mitigated by defenses. This rare success of defenses in adversarial machine learning is rooted in the well-defined structure of image scaling that fundamentally differs from the high complexity of learning models. Finally, note that image-scaling attacks further bridge the gap between adversarial learning and multimedia security where the latter also considers adversarial signal manipulations [18, 168]. We further examine this relation in the next chapter when linking learning and digital watermarking.

4.9 CHAPTER SUMMARY

The following text box summarizes the main takeaways in this chapter.

Main Takeaways.

1. An attack on the mapping $\phi \circ \rho$ from $\mathcal{Z}$ to $\mathcal{F}$ can have a considerable security impact, as the mapping provides the basis for the whole learning process.
2. Scaling attacks exploit vulnerabilities in the preprocessing ρ. An adversary can create arbitrary image outputs after downscaling. While humans work with the original image, the learning method uses a different, downscaled image.
3. As preprocessing attack, scaling attacks are agnostic to the learning model, features, and training data. They allow concealing poisoning attacks and misleading predictions. Compared to adversarial examples, the threat model is based on less assumptions.
4. A root-cause analysis shows that the attack is possible if only a subset of pixels is used for scaling. Only these pixels have to be modified to control the downscaling output. If the ratio of modified and unmodified pixels in the input image is large enough, the modifications become unnoticeable.
5. Two defense mechanisms for prevention are presented. First, the requirements for secure scaling are analyzed and used to validate the robustness of existing scaling algorithms. Second, two filters are developed to make vulnerable algorithms robust.
6. Empirical results show that these defenses are robust even under an adaptive adversary with full knowledge about them.
7. Further perspectives on the mapping are explored with an attack from the text domain, and a defense based on asymmetric data access.

5

LINKING FEATURE AND MEDIA SPACE

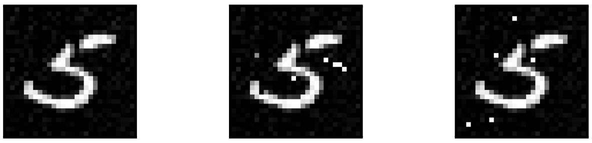
While the previous two chapters underline the major importance of the problem and the feature space for machine learning, this chapter demonstrates that the relation to other spaces should also be considered in adversarial learning. In particular, we will examine that it is possible to link the feature space $\mathcal{F}$ of machine learning with the *media space* $\mathcal{M}$ of digital watermarking.

Not only $\mathcal{Z}$ and $\mathcal{F}$ are relevant for secure learning,...

This linkage allows us to systematically study the similarities of attacks against learning and watermarking methods. In fact, both operate in an adversarial environment. In watermarking, a pattern is embedded in a signal, such as an image, in the presence of an adversary [60]. This adversary seeks to extract or remove the information from the signal, thereby reversing the watermarking process and obtaining an unmarked copy of the signal, for example, for illegally distributing copyrighted content. As a consequence, similar to machine learning, methods for digital watermarking naturally operate in an adversarial environment and several types of attacks and defenses have been proposed for watermarking methods, such as sensitivity and oracle attacks [e.g., 17, 57, 59, 76].

...but also the media space $\mathcal{M}$ of digital watermarking.

To illustrate the similarity, let us consider the simplified attacks shown in Figure 5.1: The middle plot corresponds to an attack that creates an *adversarial example* against a learning method, similar to the method proposed by Papernot et al. [156]. A few pixels of the target image have been carefully manipulated such that the digit 5 is misclassified as 8. By contrast, the right plot shows an *oracle attack* against a watermarking method, similar to the attacks developed by Westfeld [221] and Cox & Linnartz [59]. Again, a few pixels have been changed; this time, however, to mislead the watermark detection in the target image.



Target image with watermark.

Adv. example (Misclassified as 8).

Oracle attack (Broken watermark).

Figure 5.1: Examples of attacks against learning and watermarking [168]. Middle: the target is modified such that it is misclassified as 8. Right: the target is modified such that the watermark is destroyed.

While both attacks address different goals, the underlying attack strategy is surprisingly similar. Both attacks aim at minimally modifying the target such that a decision boundary is crossed. In the case of machine learning, this boundary separates different classes, such as the digits. In the case of digital watermarking, the boundary discriminates watermarked from unmarked signals. Although the previous example illustrates only a single attack type, it becomes apparent that there is a conceptual similarity between attacks in machine learning and attacks in watermarking. In this chapter, we systematically study the similarities of black-box attacks in $\mathcal{F}$ and $\mathcal{M}$. To this end, a unified notation for these attacks is derived. It enables us to transfer attacks, defenses, and lessons learned between both fields.

In summary, this chapter discusses the following major aspects:

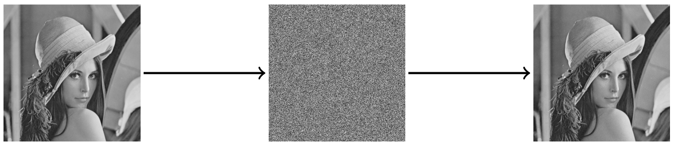
- *Relation between feature space and media space.* A novel formal view on learning and watermarking shows that the respective feature space $\mathcal{F}$ and media space $\mathcal{M}$ can be linked. This view exposes previously unknown similarities between both research fields.
- *Transfer of attacks and defenses.* Attack strategies and defenses can be transferred between $\mathcal{F}$ and $\mathcal{M}$. This correspondence gives rise to novel attacks and defenses.
- *Transfer of knowledge.* The unified notation enables transferring knowledge. Each field has established lessons learned that are relevant for the other field as well.
- *Two case studies.* Based on the unified view, two novel defenses are derived to hinder model-extraction attacks in machine learning and oracle attacks in watermarking.

Before examining the relation between $\mathcal{F}$ and $\mathcal{M}$, a brief primer on digital watermarking is provided in the following.

5.1 DIGITAL WATERMARKING

In watermarking, a pattern is embedded into a signal...

Digital watermarking allows for verifying the authenticity of digital media, like images, music or videos. Digital watermarks are frequently used for copyright protection and identifying illegally distributed content [60]. Technically, a watermark is attached to a medium by embedding a pattern into the signal of the medium such that the pattern is *imperceptible* and *inseparable*. A particular challenge for this embedding is the robustness of the watermark, which should persist under common media processing, such as compression and denoising. There exist several approaches for creating robust watermarks and the reader is referred to Cox et al. [60] for a comprehensive overview. To obtain further intuition, let us examine a simple watermarking scheme as example in the following.


Original Image

Watermark

Target image with watermark

Figure 5.2: Example of a digital watermark. A random noise pattern is added to the image in the spatial domain. The pattern is not observable but detectable.

5.1.1 *Example of a Watermarking Scheme*

Figure 5.2 shows a simple watermarking scheme where a random pattern is added to the pixels of an image. The induced changes remain (almost) unnoticeable, yet the presence of the watermark can be detected by correlating the watermarked image with the original watermark.

Let us consider this scheme in more detail. The watermarking process can be divided into two phases: embedding and detection. For the former phase, the *additive spread spectrum technique* is used. In this scheme, the watermarking parameter ω consists of a pseudorandom pattern $\boldsymbol{v} \in \mathbb{R}^d$ and a threshold η. The watermarked version $\tilde{\boldsymbol{x}}$ of a signal $\boldsymbol{x}$ is then created by adding the watermarking vector $\boldsymbol{v}$ onto $\boldsymbol{x}$ element-wise, that is,

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{v} \; . \tag{5.1}$$

To decide if a signal contains the particular watermark, a *linear correlation detector* can be employed that uses the following decision function:

$$f_{\mathcal{M}}(\tilde{\boldsymbol{x}}) = \tilde{\boldsymbol{x}}^{\mathsf{T}} \boldsymbol{v} \; \gtrless \eta \; = \; \{1, -1\} \; . \tag{5.2}$$

The function computes a weighted sum between $\tilde{\boldsymbol{x}}$ and the watermark $\boldsymbol{v}$. If watermark and signal match, the correlation exceeds a pre-defined threshold η and a positive label is returned. Geometrically, each signal corresponds to a point in a vector space where the watermark induces a decision boundary. The result are two subspaces—one for the watermark's presence, one for its absence. The detection thus works by determining which subspace an input signal is currently in.

5.1.2 *Attacks Against Watermarking*

Similar to machine learning, watermarking methods need to account for the presence of an adversary and withstand different forms of attacks [59, 76]. While there exist several attacks based on information leaks and embedding artifacts that are unique to digital watermarking [e.g., 19, 60], two attack classes can be identified, that correspond to adversarial examples and model-extraction attacks in the black-box scenario.

...in the presence of an adversary who...

ORACLE ATTACKS In this attack scenario, the adversary has access to a watermark detector that can be used to check whether a given media sample contains a watermark [59]. Such a detector can be an online platform verifying the authenticity of images as well as a media player that implements digital rights management. Given this detector, the attacker can launch an *oracle attack* in which she iteratively modifies a watermarked signal until the watermark is undetectable. The attack thus impacts the *integrity* of the pattern embedded in the signal.

...wants to evade the watermark detection...

While it is trivial to destroy the pattern and the coupled signal, for example using massive changes to the signal, carefully removing the watermark while preserving the original signal is a notable challenge. As a consequence, a large variety of different attack strategies has been proposed [e.g., 57, 59, 61, 109]. A prominent example is the *Blind Newton Sensitivity Attack*, where no prior knowledge about the detector's decision function is required and which has been successfully applied against several watermarking schemes (see Appendix D.1).

WATERMARK ESTIMATION In the second attack setting, the adversary also has access to a watermark detector, yet her goal is to estimate the watermark instead of only removing it from the target signal [53, 139]. The attack has an impact on the *confidentiality* of the watermark. Consequently, an adversary cannot only remove the watermark from the signal, but also forge it onto arbitrary other data. This *watermark estimation* therefore represents a considerable threat to watermarking methods, as it can undermine security mechanisms for copyright protection and access control.

...or estimate the watermark itself.

5.2 UNIFYING ADVERSARIAL LEARNING AND WATERMARKING

It is evident from the previous section that watermarking methods share similarities to attacks against machine learning—an observation that has surprisingly been overlooked by the two research communities [18]. This section systematically identifies the similarities and shows that it is possible to transfer knowledge about attacks and defenses from one field to the other. An overview of this systematization is presented in Figure 5.3. The systematization of machine learning and digital watermarking is guided along the following five concepts:

1. *Data Representation.* Machine learning and watermarking make use of similar data representations, which enables putting corresponding learning and detection methods into the same context (Section 5.2.1).
2. *Problem setting.* Watermarking can be seen as a special case of a binary classification. Consequently, binary classifiers and watermarking techniques tackle a similar problem (Section 5.2.2).

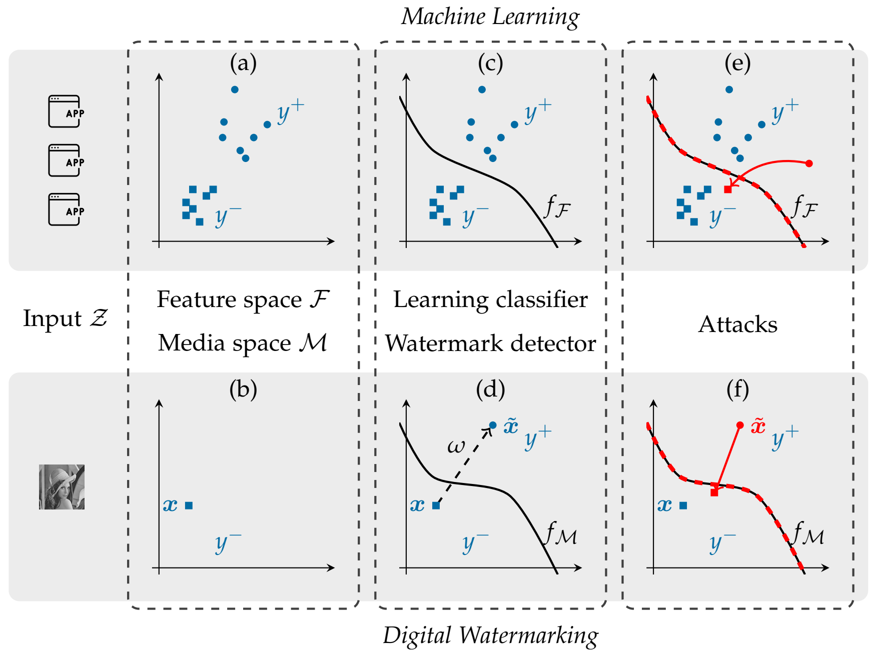


Figure 5.3: A unified view on machine learning and digital watermarking. Top: A machine learning setup including a feature space, a learning classifier, and corresponding attacks. Bottom: A watermarking setup including the media space, the watermark detector, and corresponding attacks. The red dashed line illustrates model extraction/watermark estimation, while the red arrow shows an adversarial example generation/oracle attack.

3. *Attacks.* Due to the similar representation and problem setting, attacks overlap between both fields, as we examine for adversarial examples (Section 5.2.3) and model extraction (Section 5.2.4).

4. *Defenses.* Defenses developed in one research field often fit the corresponding attack in the other field and thus can be transferred due to the similar data representation and problem setting (Section 5.2.5).

5. *Differences.* Both fields naturally have differences that—together with the similarities—yield a clear picture of both research fields (Section 5.2.6).

In the following, we examine each of these concepts in more detail, where the concept is first formalized for machine learning and then for digital watermarking. Note that learning concepts and attack principles from Chapter 2 are briefly repeated to obtain a compact systematization in one place. Furthermore, to not exaggerate the usage of symbols, the notation from machine learning is used when possible. $\mathcal{M}$ and $\mathcal{F}$ are then added as subscript to highlight the respective field.

5.2.1 *Feature Space vs. Media Space*

To begin with, both fields operate on objects from a problem space $\mathcal{Z}$. In machine learning, $\mathcal{Z}$ consists of, for example, images, source codes or programs. In watermarking, $\mathcal{Z}$ consists of digital media, such as images or videos. Each object must be converted to a suitable data representation.

As $\mathcal{F}$ is typically a vector space...

MACHINE LEARNING Problem-space objects are mapped to a *feature space* $\mathcal{F}$ that captures the characteristics of the objects to be analyzed and learned. The features in $\mathcal{F}$ usually correspond to vectors $\boldsymbol{x} \in \mathbb{R}^d$ and in the case of *classification* are assigned to a class label $y \in \mathcal{Y}$ that needs to be learned and predicted, such as y^+ and y^- in Figure 5.3(a).

...and $\mathcal{M}$ is also a vector space,...

...we have $\mathcal{F} \cong \mathcal{M}$.

DIGITAL WATERMARKING Similar to machine learning, digital media are mapped to a *media space* $\mathcal{M}$ where the watermarking methods operate on a signal, such as the pixels of an image or the audio samples of a recording. Without loss of generality, a signal in $\mathcal{M}$ can be described as a vector $\boldsymbol{x} \in \mathbb{R}^d$ and thus the media space corresponds to the feature space used in machine learning: $\mathcal{F} \cong \mathcal{M}$. Note that advanced watermarking schemes often map the signal to other spaces, such as frequency or random subspace domains [60, 76]. Still, the mapped signals can be described as points in a vector space.

Note that the mapping from $\mathcal{Z}$ to $\mathcal{M}$ is an identity function in the considered cases of this thesis. For each valid signal, a respective digital medium can be created. This is similar for digital media in machine learning where the feature mapping can also be an identity function, for instance, if pixels are directly used as features (see Section 2.1.2). Finally, note that $\mathcal{F} \cong \mathcal{M}$ does not depend on the application context. For instance, $\mathcal{M}$ can work in an audio context while $\mathcal{F}$ describes features for malware detection. We focus on the situation in a vector space.

5.2.2 *Classifier vs. Watermark Detector*

A classifier divides $\mathcal{F}$ with a decision boundary,...

MACHINE LEARNING After embedding the training data into a feature space, the actual learning process is performed using a learning algorithm, such as a support vector machine or a neural network. In the case of classification, this learning algorithm tries to infer functional dependencies from the training data to separate data points of different classes. These dependencies are captured by a learning model with model parameters θ. These parameters lead to a decision function.

Given a vector $\boldsymbol{x}$, this function predicts a class label based on an underlying decision boundary in the vector space:

$$f_{\mathcal{F}} : \mathcal{F} \longmapsto \{-1, 1\} \,. \tag{5.3}$$

To highlight that the decision function f belongs to machine learning, $f_{\mathcal{F}}$ with subscript $\mathcal{F}$ is used in the following. Moreover, for a simpler comparison with watermarking, $f_{\mathcal{F}}$ is a binary classifier, but the discussed concepts are also applicable to multiclass classifiers.

DIGITAL WATERMARKING The media space in watermarking is divided into two separate subspaces as depicted in Figure 5.3(d) where the marked and unmarked versions of the signal represent the two classes. Note that a robust watermark should ideally survive signal processing steps, such as compression and denoising of images or audio. Therefore, the watermark class implicitly contains variations as well, just as machine learning captures the variations of objects from a class through its generalization.

...and a watermark detector also divides $\mathcal{M}$ with a decision boundary.

If we denote an unmarked signal as $\boldsymbol{x}$ and a watermarked signal as $\tilde{\boldsymbol{x}}$, the relation between $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$ is given by a parameter ω that defines the pattern of the watermark. As a consequence, a watermark detector also employs a function $f_{\mathcal{M}}(\boldsymbol{x})$ to determine which subspace a signal is in and thus whether it contains the watermark:

If the classifier and detector only return the predicted class,...

$$f_{\mathcal{M}} : \mathcal{M} \longmapsto \{-1, 1\} \,. \tag{5.4}$$

Similar to machine learning, the function $f_{\mathcal{M}}$ may induce a linear or non-linear decision boundary, such as a polynomial [78] or fractalized surface [138].

Although the functions $f_{\mathcal{F}}$ and $f_{\mathcal{M}}$ share similarities, the creation process of the underlying decision boundary fundamentally differs. In machine learning, the boundary needs to separate the training data as good as possible which restricts the boundary's shape. In contrast, the boundary in watermarking schemes can be created under more degrees of freedom as long as the underlying watermark is reliably detectable. After that the boundary is created, an attacker, however, faces the same situation in both fields. As Figure 5.3(c)–(d) highlight, a decision boundary—not necessarily the same in $\mathcal{F}$ and $\mathcal{M}$—divides the respective vector space into two subspaces:

$$\mathcal{F} = \left\{ \boldsymbol{x} \in \mathbb{R}^d | f_{\mathcal{F}}(\boldsymbol{x}) \ = y^- \right\} \cup \left\{ \boldsymbol{x} \in \mathbb{R}^d | f_{\mathcal{F}}(\boldsymbol{x}) \ = y^+ \right\} \tag{5.5}$$

$$\mathcal{M} = \left\{ \boldsymbol{x} \in \mathbb{R}^d | f_{\mathcal{M}}(\boldsymbol{x}) = y^- \right\} \cup \left\{ \boldsymbol{x} \in \mathbb{R}^d | f_{\mathcal{M}}(\boldsymbol{x}) = y^+ \right\} \tag{5.6}$$

Consequently, black-box attacks that work through input-output observations are transferable between machine learning and digital watermarking. The following sections discuss this similarity and provide a mapping between machine learning and watermarking attacks, which lays the ground for transferring defenses from one field to the other.

...we get a similar black-box attack surface in $\mathcal{F}$ & $\mathcal{M}$.

5.2.3 *Adversarial Example vs. Oracle Attack*

Attackers who mislead the output...

As the first attack mapping, we consider the pair of *adversarial examples* and *oracle* attacks in a black-box setting. In this attack scenario, an adversary targets the integrity of the classifier's/detector's response by inducing a misclassification from an iteratively collected set of input-output pairs.

...with adversarial examples in $\mathcal{F}$...

MACHINE LEARNING As we have a correspondence between $\mathcal{F}$ and $\mathcal{M}$, we consider only *feature-space attacks* as introduced in Section 2.2.2. To obtain an adversarial example, the adversary tries to manipulate a feature vector $\boldsymbol{x}$ with minimal changes, so that it is misclassified as y^t by the decision function $f_{\mathcal{F}}$. Formally, the attack can thus be described as an optimization problem,

$$\arg\min_{\boldsymbol{\delta}} \quad \mathcal{D}(\boldsymbol{x}, \boldsymbol{x}+\boldsymbol{\delta}) \quad \text{s.t.} \quad f_{\mathcal{F}}(\boldsymbol{x}+\boldsymbol{\delta}) = y^t \;, \tag{5.7}$$

where $\mathcal{D}$ is a distance function, and $\boldsymbol{\delta}$ the modifications.

...or with oracle attacks in $\mathcal{M}$...

DIGITAL WATERMARKING In an oracle attack, an adversary tries to disturb or even remove the watermark embedded in a signal. The attack setting is closely related to adversarial examples. Formally, the underlying optimization problem is given by

$$\arg\min_{\boldsymbol{\delta}} \quad \mathcal{D}(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{x}}+\boldsymbol{\delta}) \quad \text{s.t.} \quad f_{\mathcal{M}}(\tilde{\boldsymbol{x}}+\boldsymbol{\delta}) = y^- \;, \tag{5.8}$$

where $\mathcal{D}$ measures the changes on the watermarked signal $\tilde{\boldsymbol{x}}$ and y^- corresponds to no detection.

Note that the constraint $(\boldsymbol{x}+\boldsymbol{\delta}) \in [\boldsymbol{x}_{\text{lb}},\ \boldsymbol{x}_{\text{ub}}]$ is neglected in Equation 5.7 and Equation 5.8 for the sake of a compact comparison. In both attacks, the final values in the feature and signal vector also need to remain within a certain range $[\boldsymbol{x}_{\text{lb}},\ \boldsymbol{x}_{\text{ub}}]$.

...use similar black-box methods that are transferable between $\mathcal{F}$ and $\mathcal{M}$.

MACHINE LEARNING ↔ DIGITAL WATERMARKING The optimization problems in Equation 5.7 and Equation 5.8 are equivalent. In geometrical terms, this allows similar attack strategies in both fields whenever the adversary aims at crossing the decision boundary in the vector space towards the wanted class based on binary outputs only (see Figure 5.3(e)–(f)).

As introduced in Section 2.2.2, black-box attacks against learning methods can be categorized into *direct attacks* and *learning-based attacks*. In the first category, the attacker directly uses the classifier output to construct an adversarial example. The watermarking literature has extensively developed strategies to find signals in this way [e.g., 57, 59, 61, 108, 109]. For example, the Blind Newton Sensitivity Attack (BNSA) [57] iteratively follows the decision boundary. To this end, it computes the decision boundary's first and second derivative by observing how the detector's output varies for minimal changes at a decision boundary location. This attack is straight applicable against learning classifiers when we replace the binary output $f_{\mathcal{M}}$ with $f_{\mathcal{F}}$. Appendix D.1 recaps the attack in more detail. Another example is the attack by Kalker [108]. It estimates the normal vector to a linear decision boundary by creating a circle of vectors around a position on this boundary. Depending on the detector's output, the vectors are averaged and the normal vector can be estimated to find a way towards y^-. This attack is also straight applicable against learning classifiers. Appendix D.2 recaps this attack.

In the context of adversarial learning, direct black-box attacks have recently been examined as well [33, 45]. Brendel et al. [33] implement a random walk around the decision boundary. Chen et al. [45] iteratively follow the boundary, similar to the BNSA. Yet, they compute the first derivative by using a circle of vectors around a respective boundary position, similar to Kalker [108]. These attacks are also applicable against watermark detectors by replacing $f_{\mathcal{F}}$ with $f_{\mathcal{M}}$. In summary, direct black-box attacks in adversarial learning and watermarking use similar concepts due to the equivalent problem setting. We arrive at the following insights:

Insight 1. *The unification provides new direct black-box attacks in the attack portfolio of adversarial learning and watermarking. They should also be considered when attacking a classifier or detector.*

Insight 2. *Beyond the algorithms, novel concepts can also be transferred, such as the usage of the second derivative by the BNSA.*

The second attack category, the learning-based attack, is based on a surrogate model and the transferability property: an adversarial example that misleads the adversary's surrogate model will probably also mislead the original model [26, 158, 200]. Due to the same attack objective and the same geometrical structure, such a strategy is also possible against watermarking schemes: An adversary learns a surrogate model to approximate the watermark's decision function and then creates an adversarial example on that model instead of the watermark detector. In this way, the adversary can exploit the full access to the model to apply white-box attacks. Quiring & Rieck [165] and Quiring et al. [168] demonstrate the feasibility of this novel attack against the advanced watermarking scheme Broken Arrows [76]. In this regard, lessons learned from the adversarial learning community are also relevant for watermarking. The community concluded that learning-based attacks can bypass defenses on the original model, such as gradient masking, due to the transferability property of the surrogate model [159]. Let us note the following insight:

Insight 3. *Attacks with surrogate models can also be used against watermark detectors. Such attacks can bypass defenses on the original system.*

5.2.4 *Model Extraction vs. Watermark Estimation*

Likewise, attacks that estimate...

As the second attack mapping, we consider the pair of *model extraction* and *watermark estimation*. In the black-box scenario, the adversary aims at compromising the confidentiality of a learning model or digital watermark by sending specifically crafted queries to a given classifier/detector and observing the respective binary output over multiple iterations.

...the decision boundary in $\mathcal{F}$...

MACHINE LEARNING Model-extraction attacks center on an effective strategy for querying a classifier such that the underlying model can be reconstructed with few queries. The adversary can pursue three different goals (remember Section 2.2.3). For the transfer to watermarking, fidelity and functionally equivalent extraction are relevant. Geometrically, both goals can be described as finding a function $\hat{f}_{\mathcal{F}}$ such that its decision boundary is as close as possible to the original one of $f_{\mathcal{F}}$. Formally, the fidelity goal is to maximize

$$\Pr_{\boldsymbol{x}\sim\mathcal{X}}\left[\mathbb{1}(\hat{f}_{\mathcal{F}}(\boldsymbol{x}) = f_{\mathcal{F}}(\boldsymbol{x}))\right], \tag{5.9}$$

where $\mathcal{X}$ is the expected data distribution of the inputs. The functionally equivalent extraction is given by

$$\hat{f}_{\mathcal{F}}(\boldsymbol{x}) = f_{\mathcal{F}}(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \mathcal{F}. \tag{5.10}$$

...or the decision boundary in $\mathcal{M}$...

DIGITAL WATERMARKING Watermark estimation represents the counterpart to model extraction. In this attack scenario, the adversary seeks to reconstruct the watermark from a marked signal $\tilde{\boldsymbol{x}}$. If successful, the adversary is not only capable of perfectly removing the watermark from the signal $\tilde{\boldsymbol{x}}$, but also of embedding it in other signals, thereby effectively creating forgeries. Geometrically, this attack can again be described by reconstructing a decision boundary, but this time in the media space:

$$\hat{f}_{\mathcal{M}}(\boldsymbol{x}) = f_{\mathcal{M}}(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \mathcal{M}. \tag{5.11}$$

...correspond to each other and...

MACHINE LEARNING ↔ DIGITAL WATERMARKING While learning models and watermarks are conceptually very different, Equation 5.9, 5.10, and 5.11 emphasize that the extraction of an underlying decision boundary in a vector space represents a common adversarial goal in both research fields. Figure 5.3(e)–(f) illustrates the correspondence.

Both fields examine direct-extraction attacks that aim at a reconstruction of the model θ or watermark ω. Consequently, both aim at a functionally equivalent extraction (Equation 5.10 ↔ Equation 5.11).

A commonly used strategy in both fields consists in localizing the decision boundary through a line search and then combining various gathered points to reconstruct θ or ω precisely [53, 132, 139]. The extraction of non-linear classifiers, such as decision trees, also exploits localized boundary points for reconstruction [207]. The latter attack is discussed in more detail in Section 5.3 when presenting a novel defense against it. This defense is inspired by concepts from digital watermarking. Overall, we arrive at the following insight:

...can be transferable between $\mathcal{F}$ and $\mathcal{M}$.

Insight 4. *Attacks in both fields directly reconstruct a decision boundary in a vector space. This leads to new functionally equivalent attacks in the attack portfolio of adversarial learning and digital watermarking.*

The second group of attacks is learning-based and has been primarily investigated in adversarial machine learning for fidelity extraction (Equation 5.9). An attacker collects a number of input-output pairs with queries either scattered over the feature space or created adaptively [e.g., 41, 102, 107, 152, 158, 207]. These observations allow the adversary to learn a surrogate model. As previously described, this approach can be part of an attack to create adversarial examples. The unified view in this chapter underlines that an adversary can also use this group of attacks in digital watermarking: She substitutes a watermarking scheme by a learning model that approximates the decision boundary of $f_{\mathcal{M}}$. Although this only yields an approximation of the original watermark, the attacker is able to remove or add digital watermarks by using white-box attack strategies from adversarial learning, as demonstrated by Quiring & Rieck [165] and Quiring et al. [168]. To sum up, learning-based attacks from adversarial learning that aim at fidelity extraction can also be relevant in watermarking (see Insight 3).

5.2.5 *Defenses*

The communities of both research fields have extensively worked on developing defenses to fend off the attacks presented in the previous sections. However, it is usually much easier to create an attack that compromises a security goal, than devising a defense that effectively stops a class of attacks. As a result, several of the developed defenses only protect from very specific attacks and it is still an open question how learning methods and watermark detectors can generally be protected from the influence of an adversary. As the previous sections highlight, an attacker geometrically works at the decision boundary in both fields. Moreover, attackers often collect feedback by repeatedly querying the system in both fields. Consequently, various defense strategies are not restricted to one particular research field. In this section, these similarities are formally described, and the implications as means of novel research directions are outlined (see Table 5.1). We also consider defenses from adversarial learning that were ini-

The common attack surface also leads to transferable defense strategies, such as...

Defense Technique	**Adv. Learning**			**Watermarking**
		Ensembles		
Multiple classifiers/detectors	[22, 23, 161, 220]			[52, 214]
Union of watermarks		◁		[76]
		Randomized responses		
Randomized boundary	[233]	◁	▷	[76, 125]
		Boundary modification		
Non-parametric boundary		◁		[138]
Non-parametric random. boundary		◁		[167]
Snake traps		◁		[76]
		Blind-spot detection		
Data enclosing	[28, 179]		▷	(CS1)
		Stateful analysis		
Closeness-to-the-boundary	(CS2)	◁		[17, 204]
Line search detection		◁		[17, 205]
Locality-sensitive hashing		◁		[215]
Learning-based encoding with kNN	[46]		▷	

◁ = Possible transfer from watermarking to machine learning;
▷ = Possible transfer from machine learning to watermarking;
CS1, CS2 = Defense transfer demonstrated as case study in Section 5.3;

Table 5.1: Transfer of defense techniques introduced by adversarial learning and digital watermarking.

tially presented against white-box adversaries, but also work when an adversary acts in a black-box setting.

...using multiple classifiers/detectors,...

ENSEMBLES The first joint defense strategy applies *multiple learning classifiers or watermark detectors*. In machine learning, for instance, each classifier can be built from a random subset of the feature set [22, 23, 161, 220]. The binary prediction is then retrieved from the numerical output of all classifiers $f_{\mathcal{F}}^{(i)}$ through an aggregation function $E_{\mathcal{F}}$:

$$f_{\mathcal{F}}(\boldsymbol{x}) = E_{\mathcal{F}}\left(f_{\mathcal{F}}^{(1)}(\boldsymbol{x}), f_{\mathcal{F}}^{(2)}(\boldsymbol{x}), \dots, f_{\mathcal{F}}^{(k)}(\boldsymbol{x})\right). \tag{5.12}$$

A corresponding strategy has been examined against oracle attacks. The binary prediction is obtained from the numerical output of several detectors $f_{\mathcal{M}}^{(i)}$ where each is built from a random subset of signal samples. The final detector output is then obtained from an aggregation function $E_{\mathcal{M}}$, for instance the median, which yields [52, 214]:

$$f_{\mathcal{M}}(\boldsymbol{x}) = E_{\mathcal{M}}\left(f_{\mathcal{M}}^{(1)}(\boldsymbol{x}), f_{\mathcal{M}}^{(2)}(\boldsymbol{x}), \dots, f_{\mathcal{M}}^{(k)}(\boldsymbol{x})\right). \tag{5.13}$$

A comparison of Equation 5.12 and Equation 5.13 reveals that both fields employ a similar defense strategy with the same intention: An adversary has to attack different classifiers/detectors at the same time and cannot be sure whether a specific feature/sample influences the returned output.

However, the watermarking literature has already discussed weaknesses of this defense by creating a so-called *p-boundary* that acts as a surrogate boundary [54]. As a result, such an attack can be relevant to machine learning as well if multiple classifiers are used.

The watermarking literature also provides further ensemble-based defenses. The *Broken Arrows* watermarking scheme creates several watermarks that form a *union of watermarks*. During detection, only the watermark with the smallest distance to the current signal is applied [76]. This mitigates the risk that an adversary could compare multiple signals with the same watermark. This defense can also be applied to learning methods. It would correspond to an ensemble of classifiers where the aggregation function $E_{\mathcal{F}}$ just chooses one classifier depending on the input.

Insight 5. *Ensemble defenses are studied in both fields and can be transferred.*

RANDOMIZED RESPONSES A simple yet effective strategy to impede attacks against classifiers and watermark detectors builds on randomized outputs. While this defense cannot rule out successful attacks, the induced indeterminism obstructs simple attack strategies and requires more thorough concepts. Against model extraction, Zheng et al. [233] propose to return randomized responses if a query lies within a margin around the decision boundary. This randomized region obstructs the boundary's reconstruction, since the adversary cannot exactly localize the boundary. The watermarking field has also examined this defense idea [76, 125], and has already discussed its weaknesses [54, 108, 167]. In particular, an adversary can exploit the *p-boundary* [54]. Moreover, the attack by Kalker [108] enables an adversary to estimate the normal vector to the decision boundary despite a randomized region [108, 167]. As most part of the vector circle is outside this region, the computations are not affected (see Appendix D.2). Overall, we should keep in mind the following lessons learned:

...building on randomness,...

Insight 6. *Randomization-based defenses have been studied in both fields.*

Insight 7. *Research in watermarking shows that randomization only mitigates an attack. The gained insights should also be considered in adversarial learning, such as the possibility to bypass a randomized region.*

BOUNDARY MODIFICATION Another defense strategy from watermarking increases the decision boundary's complexity. A linear boundary, for instance, can be replaced by a non-parametric version along the previous boundary so that the new boundary cannot be estimated with a finite number of known points [138]. The non-parametric boundary can be implemented, for instance, by using a

...making the decision boundary more complex,...

fractal [138]. In addition, Furon and Bas [76] have introduced small indents called *snake traps* at the decision boundary in order to stop attacks based on random walks along the detection region [61, 76]. These defenses are applicable in machine learning as well, as they replace an existing boundary by a more complex version that lies along the previous boundary. In this way, the learned separation of the classifier is not changed, but black-box attacks are obstructed. A non-parametric boundary, for instance, would increase the difficulty to conduct a model-extraction attack that has the goal to obtain a functionally equivalent $\hat{f}_{\mathcal{F}}$. Attacks from adversarial learning based on random walks [33] might also be affected by snake traps. Last but not least, Quiring and Schöttle [167] examined the combination of non-parametric and randomized boundary, so that the deterministic outer boundaries of a randomized region cannot be exploited by an adversary.

However, research in watermarking has also unveiled weaknesses of these boundary-based defenses. In particular, an adversary can overcome a non-parametric boundary, such as a fractalized boundary, by using its *envelope* [57, 167, 205]. The combination of non-parametric and randomized boundary also remains vulnerable to this kind of attack [167].

Insight 8. *Research in watermarking also studies defenses to modify the decision boundary, e. g., by using non-parametric boundaries or adding snake traps against random walks. They are transferable to adversarial learning.*

Insight 9. *Yet, research shows that oracle attacks can bypass boundary-based defenses. These lessons learned are also relevant for adversarial learning.*

...detecting inputs outside the expected data distribution,...

BLIND-SPOT DETECTION A defense strategy in machine learning against adversarial examples consists in detecting and rejecting *blind-spot* inputs that are outside the expected data distribution in $\mathcal{F}$ (see Section 2.2.2). In the case of malware detection, this defense implies that an evasion attack needs to contain plausible features of the benign class without losing the malicious functionality. Russu et al. [179] implement this defense strategy using non-linear kernel functions, while Biggio et al. [28] realize a tighter and more complex boundary through the combination of two-class and one-class models. Although invented against white-box and gray-box attacks, these countermeasures also tackle black-box attacks that need to probe the feature space with blind-spot inputs outside the data distribution. Section 5.3.1 shows that this strategy also addresses an oracle attack, where an adversary may also probe the watermark detector with artificial inputs [222].

Insight 10. *Preventing the adversary from exploiting blind-spots, as introduced in adversarial learning, can also be used in watermarking.*

STATEFUL ANALYSIS If the learning method or watermark detector is outside of the attacker's control, an active defense strategy becomes possible, in which the defender seeks to identify sequences of malicious queries. For instance, a cloud service providing machine learning as a service may monitor incoming queries for patterns indicative of adversarial-example generations or model-extraction attacks. Stateful analysis of queries has been already thoroughly applied in digital watermarking for detecting oracle and watermark-estimation attacks [17, 204, 205, 215], while this concept has only recently been examined in adversarial machine learning by Chen et al. [46].

...or analyzing the whole sequence of queries.

The defenses in both fields exploit the fact that an adversary will typically follow a specific strategy to locate the decision boundary due to the inherent binary output restriction. For example, an adversary may use a line search to localize the boundary or perform several queries close to the boundary in order to exactly locate its position. Formally, we have a meta-detector ϑ that works alongside the usual decision function. It analyzes the sequence of the current and prior inputs $\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \dots, \boldsymbol{x}_{t-l}$ in parallel to infer whether the system is subject to an attack. For machine learning, we obtain a new classifier that integrates $f_{\mathcal{F}}(\boldsymbol{x}_t)$ as follows:

$$\tilde{f}_{\mathcal{F}}(\boldsymbol{x}_t) = \Psi\left(f_{\mathcal{F}}(\boldsymbol{x}_t), \vartheta(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \dots, \boldsymbol{x}_{t-l})\right). \tag{5.14}$$

In watermarking, we obtain a new detector that integrates $f_{\mathcal{M}}(\boldsymbol{x}_t)$ as follows:

$$\tilde{f}_{\mathcal{M}}(\boldsymbol{x}_t) = \Psi\left(f_{\mathcal{M}}(\boldsymbol{x}_t), \vartheta(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \dots, \boldsymbol{x}_{t-l})\right). \tag{5.15}$$

The classifier $f_{\mathcal{F}}(\boldsymbol{x}_t)$ and detector $f_{\mathcal{M}}(\boldsymbol{x}_t)$ are not influenced by the meta-detector. The function Ψ either forwards the true decision value (from $f_{\mathcal{F}}(\boldsymbol{x}_t)$ or $f_{\mathcal{M}}(\boldsymbol{x}_t)$) or initiates another defense if ϑ detects an attack. For instance, it may return misleading outputs or block further access.

A comparison of both equations underlines that the proposed defense strategies from digital watermarking are directly applicable to machine learning and vice versa. The meta-detector ϑ can be reused and just the decision function $f_{\mathcal{M}}$ needs to be replaced by $f_{\mathcal{F}}$ or vice versa. Both fields have examined different strategies to implement the meta-detector which provides an opportunity for transferring concepts (see Table 5.1). Section 5.3 demonstrates this opportunity with a case study where model-extraction attacks are mitigated with the closeness-to-the-boundary concept from watermarking. Overall, we arrive at the following insight:

Insight 11. *The unification provides new concepts to implement a stateful defense in adversarial learning and watermarking.*

An important conclusion in watermarking was that stateless defenses do not prevent an oracle attack and researchers continued with stateful defenses [17, 204, 205]. These lessons learned might be relevant for adversarial learning as well. Thus, we arrive at the following insight:

Insight 12. *As stateless defenses mitigate, but do not prevent an attack, researchers in watermarking continued with stateful defenses. This might motivate a stronger focus on this defense in adversarial learning as well.*

5.2.6 *Differences*

For successfully transferring concepts between machine learning and digital watermarking, however, researchers also need to account for the difference of both areas. First, as described in Section 5.2.2, the decision boundary in machine learning needs to be adjusted to existing training data in contrast to digital watermarking. Thus, defenses from watermarking that introduce a completely new decision boundary [e.g., 77, 78] are not necessarily applicable to machine learning. Second, the white-box setting from machine learning where an attacker knows internals such as the model or fractions of the training data is not directly transferable to digital watermarking. If the original signal or the watermark are known, an adversary has already succeeded. Third, specific attacks are unique to the respective field. Reconstructing the watermark as a noise signal from a set of images, for example by averaging images, is unique to digital watermarking [e.g., 60]. The poisoning scenario known from adversarial machine learning where the attacker manipulates a fraction of the training data is in turn not transferable to digital watermarking.

In summary, machine learning and digital watermarking have different goals and the learning process with real-world data differs to the artificial watermark embedding process. Nevertheless, both operate in a corresponding vector space and, although the decision boundary can be different, the black-box scenario leads to a common attack surface: An adversary tries to change the vector subspace or to estimate the boundary just from binary outputs with multiple queries. Therefore, similar attack strategies and defenses are usable.

5.3 CASE STUDIES

Two case studies underline the efficacy of this unified view.

Equipped with a unified notation for black-box attacks and defenses, we are ready to study the transfer of concepts from one research field to the other in practical scenarios. In the first case study, a concept for securing machine learning is applied to a watermark detector, so that the resulting defense mitigates an oracle attack. In the second case study, the concept of closeness-to-the-boundary is applied to machine learning, so that the resulting defense blocks model-extraction attacks.

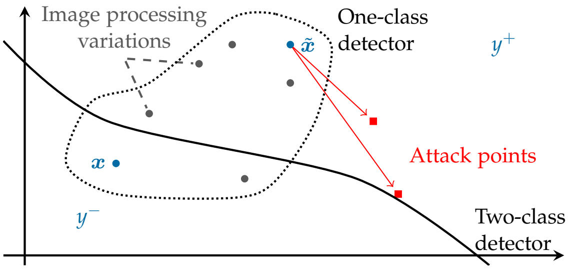


Figure 5.4: Transfer from machine learning to watermarking: A 1½-detector combining a one-class and two-class detection method.

5.3.1 *Transfer of Defense: ML → DW*

In the first case study, we transfer a defense against evasion attacks from the area of machine learning. This defense detects blind-spot inputs by combining a two-class and one-class classifier—a concept denoted as *1½-classifier* [28]. Instead of just discriminating objects into two classes, the defense additionally learns a one-class model for the underlying data distribution. The combined classifier discriminates two classes but also requires all inputs to lie within the learned region of normality. As a result, evasion attacks become more difficult, as the adversary cannot exploit blind-spots any longer. She needs to stay within normal regions when locating and moving towards the decision boundary.

First, a concept from machine learning is transferred...

This simple yet effective idea has not been applied in the context of digital watermarking so far. While existing watermarking schemes provide an accurate detection of marked content, they ignore how signals are distributed in the media space $\mathcal{M}$ and hence an adversary can explore the full space for exploring properties of the watermark. Broadly speaking, an "image does not have to look nice" [222] in an attack and thus attack points resemble distorted or implausible media. For example, many oracle attacks move along random directions or set samples of a signal to constant values when locating the decision boundary [57] (see Figure 5.5).

...to watermarking, which checks for...

To exploit this characteristic, a *1½-detector* is introduced in the following. This detector identifies watermarks but additionally spots implausible signals, that is, inputs too far away from reasonable variations of the original signal. The detector rests on the concept of Biggio et al. [28] and only provides a correct decision if the input lies within the learned region of normality. If signals outside the region are provided, the detector returns a *random decision*, thereby foiling attack strategies that move along random directions or use constant values. Figure 5.4 depicts this defense and the resulting combination of boundaries.

...blind-spot inputs being distorted or implausible.

To generate a suitable model of normality, the one-class model in the detector is trained with examples of common variations of the target

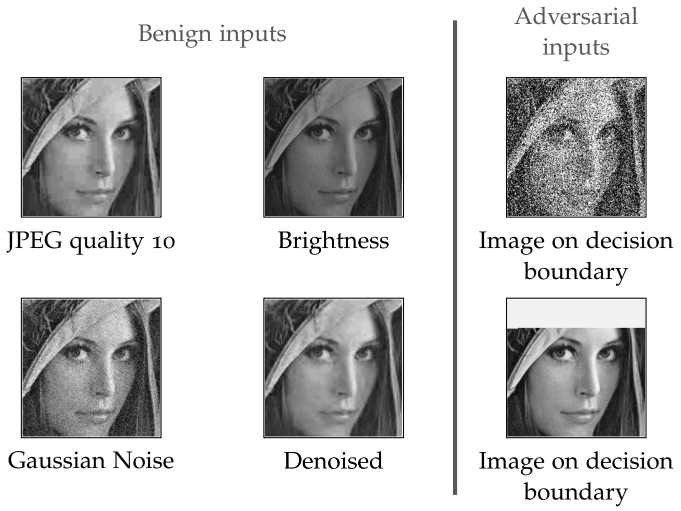


Figure 5.5: Distortions of the target image. The left four plots show plausible image distortions, whereas the right plots depict attack images.

signal. If the media space corresponds to images, different changes of brightness, scaling, contrast, compression and denoising can be applied to the target image $\tilde{\boldsymbol{x}}$. Similarly, other plausible variations of the signal can be added into the one-class model. Figure 5.5 shows different variations of a target image that are correctly identified by the 1½-detector.

In an empirical evaluation...

EXPERIMENTAL SETUP For the evaluation, we focus on the image domain. A 1½-detector is implemented, that uses a linear watermarking scheme (see Section 5.1.1) and a one-class model based on the neighborhood of a signal. Given an image $\tilde{\boldsymbol{x}}$, this model computes the distance d to the k-nearest variation of $\tilde{\boldsymbol{x}}$, that is,

$$\mathsf{d}(\tilde{\boldsymbol{x}}) = \frac{1}{\Lambda k} \sum_{\boldsymbol{u} \in \mathcal{N}_{\tilde{\boldsymbol{x}}}} \|\boldsymbol{u} - \tilde{\boldsymbol{x}}\| \tag{5.16}$$

where $\mathcal{N}_{\tilde{\boldsymbol{x}}}$ are the k-nearest neighbors of $\tilde{\boldsymbol{x}}$. For normalization purposes, each distance is divided by the maximum distance in the media space, Λ. The image is marked as implausible if the distance to its k-nearest variations reaches a given threshold ξ. For the study, $k = 3$ is simply used.

As dataset for the evaluation, 50 images from the Dresden Image Database are selected [84]. All images are converted to grayscale and cropped to a common size of 128×128 pixels and tagged with a digital watermark. To obtain training data for the one-class model, different variations of the watermarked images are created by applying common image processing techniques, such as noise addition, denoising, JPEG compression, and contrast/brightness variation.

To attack the marked images, the Blind Newton Sensitivity Attack (BNSA) is implemented, a state-of-the-art oracle attack that suc-

Threshold	Success of attacks	False positive rate
No defense	100%	—
$\xi = 0.46$	6%	0%
$\xi = 0.31$	6%	0%
$\xi = 0.23$	0%	0%
$\xi = 0.18$	0%	0%
$\xi = 0.12$	0%	0.14%
$\xi = 0.03$	0%	2.27%
$\xi = 0.02$	0%	4.47%

Table 5.2: Detection performance of the 1½-detector.

cessfully defeats several existing defenses (see Appendix D.1). For different configurations of the 1½-detector, the attack is launched against the selected 50 images and results are averaged over the 50 runs, respectively.

...with a strong oracle attack,...

DEFENSE EVALUATION The results of this experiment are presented in Table 5.2. If no defense is deployed, the implemented oracle attack is capable of removing the watermark from all images, thereby demonstrating the efficacy of the BNSA. However, if the one-class model in the 1½-detector is enabled and a threshold below 0.31 is picked, the attack fails to remove the watermark in all cases. As the defense returns random decisions outside the normal regions, the attack is not able to compute the correct gradient and thus does not converge to the correct watermarking pattern. The correlation between the watermark extracted from the final attack outcome and the original watermark is thereby zero in all cases. A threshold $\xi \geq 0.31$ however enlarges the extent of the normal regions, so that the chances increase that the attack works on the decision boundary within the normal region without disturbance again.

...the defense prevents the watermark removal in all cases...

FALSE POSITIVES Table 5.2 also shows the false-positive rate induced by the new detector. To this end, variations of the selected images that have not been used for training are tested. If a low threshold is picked, the learned model is too restrictive and some of the generated variations lie outside the normal region. Starting with a threshold of 0.18, however, the defense does not identify any benign variation as attack and thus allows us to separate legitimate variations of an image from malicious inputs generated by the BNSA.

...while benign inputs are still detected correctly.

SUMMARY In summary, we can identify a range of suitable thresholds where the detector does not misclassify benign variations and is successfully able to obstruct the watermark removal in all cases. The proposed defense is generally applicable by other watermarking schemes because the objective is to spot adversarially crafted images without changing the underlying watermark detection process. Moreover, as the defense already impedes the initial boundary localization process which is not unique to the BNSA, other oracle attacks [e.g., 53, 108] are likely to be affected as well.

5.3.2 *Transfer of Defense: DW → ML*

Second, a stateful defense concept from watermarking...

In the second case study, the concept of closeness-to-the-boundary from the area of digital watermarking is transferred to machine learning. In particular, this case study demonstrates that the resulting defense effectively mitigates the risk of model extraction by identifying sequences of malicious queries to a learning method.

Before presenting this novel defense, the tree-extraction attack proposed by Tramèr et al. [207] is shortly summarized. The attack reconstructs decision trees by performing targeted queries on the APIs provided by a platform providing machine learning as a service. The attack is possible, since the platform does not only return the class label for a submitted query but also a confidence score for a particular leaf node. This enables an adversary to distinguish between the leaves. For each leaf and for each of its features, a recursive binary search locates the leaf's decision boundary in that direction. As the binary search covers the whole feature range, other leaf regions are discovered as well and extracted subsequently. In this way, an adversary can extract all possible paths of the decision tree. Note that the attack needs to fix all features except for the one of interest, as otherwise the attack may miss a leaf during the binary search.

As a countermeasure to this attack, a stateful defense is devised that observes the closeness of queries to the decision boundary, as already used in digital watermarking [17, 204]. In this scenario, the system does not only check for the presence of a watermark, but simultaneously counts the number of queries falling inside a margin surrounding the boundary. An attacker conducting an oracle attack—thereby working around the boundary necessarily—creates an unusually large number of queries inside this margin. As a result, the analysis of the input sequences allows the identification of unusual activity. The exact parameters of the security margin are derived from statistical properties of the decision function [17].

...is applied against model-extraction attacks...

Although this defense strategy has been initially designed to protect watermark detectors, it can be extended to secure decision trees as well. Figure 5.6 illustrates the transferred concept where margins are added to all boundaries of a decision tree. The width of these margins is determined for each region separately depending on the statistical distribution of the data. Overall, this security margin is defined alongside the original decision tree and does not require changes to

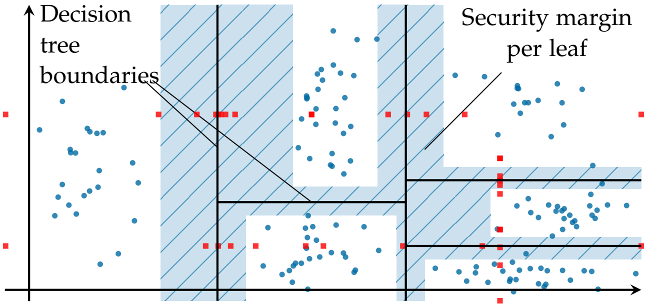


Figure 5.6: Transfer from watermarking to machine learning: A stateful defense using the closeness-to-the-boundary concept.

its implementation. Appendix D.3 provides more information on the margin's creation process.

Formally, a stateful learning-based system $\tilde{f}_{\mathcal{F}}(\boldsymbol{x}_t)$ is built—following the definition in Equation 5.14: When the decision tree returns the predicted class $f_{\mathcal{F}}(\boldsymbol{x}_t)$ for a query $\boldsymbol{x}_t$, a meta-detector ϑ checks if the query falls inside the security margin. To determine whether the tree is subject to an attack, ϑ keeps track of a history of queries and computes the ratio between points inside and outside the margin for each leaf. The averaged ratio over all leaves, χ, is an indicator for the plausibility of the current input sequence. As an example, Figure 5.6 shows a typical query sequence from the tree-extraction algorithm (red squares). The adversary has to work within the margin to localize the decision boundary, in contrast to the distribution of benign queries (blue circles).

...by counting the number of queries near the boundary,...

...which is higher in the case of an attack.

EXPERIMENTAL SETUP To evaluate this defense in practice, the publicly available tree-extraction implementation by Tramèr et al. [207] is used. Table 5.3 summarizes the applied datasets. Each dataset is divided into a training set (50%) and test set (50%), where the first is used for learning a decision tree and calibrating the security margins. The meta-detector ϑ identifies an attack if the query ratio χ exceeds the threshold 0.3. The test set is used to simulate the queries of an honest user. In this way, the risk of false positives can be determined, that is, declaring an honest input sequence as malicious. Next, the tree-

In an empirical evaluation, this defense...

Dataset	Samples	Features	∅ Leaves
Iris	150	4	4.6
Carseats	400	8	13.2
College	777	17	18.8
Orange Juice	1,070	11	59.0
Wine Quality	1,599	11	89.4

Table 5.3: Datasets for evaluation. The number of leaves from the learned decision tree are averaged over the repetitions.

Dataset	**Original**		**Blocking**		**Random**		**Adaptive**	
	Q	p	Q	p	Q	p	Q	p
Iris	108	100%	38	09%	*	09%	4,412	100%
Carseats	871	100%	148	20%	*	20%	15,156	46%
College	2,216	100%	244	10%	*	10%	8,974	08%
Orange J.	4,804	100%	846	20%	*	20%	86,354	48%
Wine Qual.	9,615	100%	978	11%	*	11%	37,406	11%

Table 5.4: Effectiveness of the transferred closeness-to-the-boundary defense based on queries Q and extracted leaves p for different attack variations and possible reactions after detecting the attack.

extraction attack is applied against the learned tree without and with the meta-detector ϑ. In the latter case, two reactions after detecting an attack sequence are considered: (a) the system blocks further access and (b) the system returns random decisions. An attack is stopped after 1 Million queries (denoted by *). Each experiment is repeated 5 times and aggregated results are presented in the following.

...prevents a full decision-tree extraction.

DEFENSE EVALUATION To determine the knowledge gain by the adversary, Table 5.4 reports the percentage of successfully extracted leaves p together with the required number of queries Q. Without any defense, the original attack extracts the whole tree ($p = 100\%$). In contrast, the blocking strategy based on ϑ allows the system to block the tree extraction at the very beginning. With random decisions, the attack's binary search recursively locates an exponential number of boundaries erroneously, without any improvement regarding the extraction. At the same time, the final query ratio χ after submitting an honest sequence was not higher than 0.2 in all datasets, so that ϑ does not mark a benign query sequence as an attack by mistake. As a result, ϑ can effectively separate legitimate from malicious input sequences.

It also mitigates adaptive attacks that cover the traces...

...with additional, random queries or...

ADAPTED ATTACK In practice, an adversary will adapt the attack strategy to the particular defense, so that we study possible attack variations in the following. The adversary is assumed to create *cover queries* outside the margin by selecting random values in the range of each feature. The intention is to keep the query ratio below the threshold. Results with 40 cover queries for each tree extraction query are presented, which highlights the effect of a substantial increase in queries. Table 5.4 shows the performance of this adapted attack. Despite the increase in queries, the whole tree can still not be extracted. Only half of the leaves are recovered before ϑ spots the attack and the system blocks further access.

There are two practical problems that explain the attack's failure. Without knowledge of the training data distribution, the adversary

Dataset	**Cover Queries**	**Extracted leaves p [%]**				
		Known training data [%]				
		10	20	30	40	50
Iris	1x	17	21	21	21	22
	5x	64	85	89	92	94
	40x	76	91	94	97	100
Carseats	1x	28	29	28	29	30
	5x	39	60	69	82	89
	40x	50	87	97	100	100
College	1x	12	12	12	12	12
	5x	17	26	28	29	32
	40x	29	64	85	94	100
Orange Juice	1x	28	29	29	29	29
	5x	39	63	88	98	99
	40x	46	92	100	100	100
Wine Quality	1x	20	22	22	23	24
	5x	33	55	88	98	100
	40x	43	91	100	100	100

Table 5.5: Percentage of extracted leaves p with an informed attacker knowing a certain percentage of the training data.

cannot know where a decision boundary could be located and thus where the margin could be. Another problem is that the adversary needs to control the ratio in almost each leaf. It is not sufficient to send just one fixed well-chosen cover query all the time since this query would only affect one leaf. These problems make the smart selection of cover queries challenging since the adversary has to perform initial queries to localize a first set of leaves. Thus, the presented defense can spot the attack before the adversary collects more information to formulate smarter cover queries.

WELL-INFORMED ATTACK We finally consider the situation where an adversary may even have access to parts of the training data. This makes a defense clearly challenging since the adversary can already make assumptions about a possible learning model. The adversary is assumed to create cover queries from the leaked training data. Table 5.5 summarizes the percentage of extracted leaves p for varying amounts of known training data and cover queries. The defense can still block an adversary even if training data are leaked partly. If just 10% of the data are known, even 40 cover queries between each attack query do not suffice to extract the whole tree. However, if the adversary knows

...queries based on training data that are leaked to some extent.

more data points, the cover queries spread more equally over all leaves and the attack chances increase.

SUMMARY The evaluation demonstrates that the transferred stateful defense can effectively obstruct model-extraction attacks. It is not limited to a decision tree and can be applied to models, such as an SVM, where an attacker tries to locate the decision boundary through queries. The defense can be easily deployed in practice by implementing it alongside an existing classifier.

5.4 RELATED WORK

Taken together, this chapter links machine learning & watermarking,...

...but more research fields also deal with an adversary,...

Machine learning and digital watermarking are not the only research areas that have to deal with an adversary. The identified similarities between both research fields can be seen as part of a bigger problem: *Adversarial Signal Processing* as stated by Barni and Pérez-González [18]. More fields such as information forensics, multimedia forensics, biometrics, or steganography and steganalysis also deal with an adversary's presence [18]. By working in parallel, research communities unnoticeably re-invent similar attack and defense strategies and do not make use of lessons learned, leading to an avoidable lower pace in research.

...motivating a unification across all these fields in future.

Unfortunately, a unification has not been carried out so far. This thesis marks a first effort to bring two communities together by establishing a comprehensive, conceptional link between the feature space and the media space. This enables us to transfer attacks, defenses, and lessons learned from watermarking to machine learning and vice versa. Following the effort to bring communities together, Schöttle et al. [183] establish a link between steganalysis and adversarial examples. In a proof-of-concept, they adjust a steganalysis method to detect adversarial examples in the image domain. Still, more work is needed to systematically develop a comprehensive unified view between the various research fields coping with an adversary, as done in this thesis. Hopefully, this eventually leads to a common theory of *Adversarial Data Processing*[1].

5.5 SUMMARY

The text box on the next page shortly summarizes the main takeaways in this chapter.

---

[1] The term signal processing does not perfectly fit to machine learning, so that the proposed term here might better convey the unification across the research fields.

Main Takeaways.

1. The feature space $\mathcal{F}$ of machine learning can be linked with the media space $\mathcal{M}$ of digital watermarking. $\mathcal{F}$ and $\mathcal{M}$ are a vector space, respectively. For classification and watermark detection, decision boundaries divide both spaces into subspaces. If the classifier and detector only provide the predicted class, we get a similar black-box attack surface.
2. Black-box attacks based on input-output queries are transferable between both fields. A correspondence between adversarial examples and oracle attacks, as well as between model extraction and watermark estimation is shown.
3. In both fields, attackers work at the decision boundary and repeatedly query the system. Thus, multiple defenses can be transferred as well.
4. Knowledge can also be exchanged, as each field has lessons learned that are relevant for the other field as well.
5. Two case studies empirically demonstrate the benefit of the unified view. First, the transferred concept of blind-spot detection from adversarial learning successfully prevents an oracle attack from removing a watermark. Second, a stateful defense from digital watermarking also blocks model-extraction attacks in machine learning.

6

CONCLUSION

Machine learning has become a key element in computer science and engineering, with applications ranging from autonomous driving and speech recognition to more security-oriented applications such as intrusion detection and malware analysis. Despite great potential, machine learning itself can pose a security risk if an adversary actively targets the learning pipeline. The application of machine learning thus requires reasoning about potential attacks and possible defenses.

This thesis marks a step towards secure learning by having analyzed the relation between the problem space $\mathcal{Z}$, feature space $\mathcal{F}$, and media space $\mathcal{M}$. First, we have studied the relation between $\mathcal{Z}$ and $\mathcal{F}$. In particular, we have thoroughly examined problem-space attacks that create real-world objects in $\mathcal{Z}$ while misleading learning methods in $\mathcal{F}$. In addition, we have analyzed the mapping itself from $\mathcal{Z}$ to $\mathcal{F}$. Using the example of image scaling, we have examined the potential attack surface, its root cause, and possible defenses. Second, we have studied the conceptional link between $\mathcal{F}$ and $\mathcal{M}$. This has allowed us to transfer attacks, defenses, and lessons learned between machine learning and digital watermarking, thereby bundling methods and insights between various research fields coping with an adversary.

Taken all together, this thesis provides a comprehensive view on the security of learning-based systems, from the input in $\mathcal{Z}$ to the classification output in $\mathcal{F}$ with its linkable space $\mathcal{M}$. The benefit of this view and analysis is also practically underlined by the Microsoft Machine Learning Security Evasion Competition [196]. The knowledge about problem-space attacks and the mapping allowed Quiring et al. [172] to develop a learning-based system for malware detection that also resisted evasion attacks from real-world attackers and won the defender challenge of the competition.

6.1 SUMMARY OF RESULTS

In the following, the main contributions of this thesis towards secure machine learning are summarized, spanning the Chapters 2–5.

CHAPTER 2 This chapter laid the groundwork for the thesis by providing two major contributions. First, *the basic concepts of machine learning were introduced along a learning pipeline*. This view demonstrated the importance of the relation between $\mathcal{Z}$ and $\mathcal{F}$. Second, *attacks at each component of this pipeline* were examined. This allowed us to put the relation between $\mathcal{Z}$ and $\mathcal{F}$ into the context of different attacks.

Chapter 3 Although machine learning is often successfully attacked in $\mathcal{F}$, considering $\mathcal{Z}$ to realize the attack is equally important. In most applications, there is no bijective mapping between $\mathcal{Z}$ and $\mathcal{F}$. This complicates an attack, but is often neglected by prior research. As third contribution, we analyzed *the relation between $\mathcal{Z}$ and $\mathcal{F}$. In particular, we explored problem-space attacks that create real objects in $\mathcal{Z}$ and that mislead learning-based systems in $\mathcal{F}$.* We studied four dilemmas and five constraints of these attacks, as well as three types of search strategies that guide the attacks. Using the example of source code attribution, we practically learned how attacks can be conducted. A new strategy based on MCTS was developed to find adversarial examples of source code. An empirical evaluation showed that adversarial examples can be successfully created. In the untargeted scenario, the attribution accuracy drops from over 88% to 1%. In the targeted scenario, more than three out of four developers can be impersonated.

Chapter 4 As next contribution, we proceeded with *a security analysis of the mapping itself from $\mathcal{Z}$ to $\mathcal{F}$, also overlooked in most prior work*. In the context of image scaling, we learned that an adversary can precisely control the scaling output and thus the input to the subsequent learning pipeline. Yet, based on a root-cause analysis of this attack, defenses for prevention were developed. First, we studied requirements for secure scaling and found that some implemented scaling algorithms in libraries are robust against attacks. Second, two filters were developed to make vulnerable algorithms robust. These defenses also withstand an adaptive adversary in the evaluation. Finally, we studied further attack surfaces in the mapping.

Chapter 5 As fifth contribution, we explored the relation between $\mathcal{F}$ and $\mathcal{M}$. We established that $\mathcal{F} \cong \mathcal{M}$. In the black-box scenario, adversaries in both fields also work at the decision boundary in a vector space and repeatedly query the system. *This allowed transferring attacks, defenses, and lessons learned.* These insights were summarized in twelve takeaways, including novel potential methods for adversarial examples and model extraction. Two cases studies empirically underlined the transfer. First, a defense that spots implausible inputs in $\mathcal{F}$ was transferred to $\mathcal{M}$. Second, a stateful defense from watermarking was applied against model extraction in machine learning.

6.2 FUTURE WORK

Inherent to research is that further questions remain or arise once we have explored a problem. Hence, let us finally examine possible research directions—keeping a concluding remark of Alan Turing in mind: "*We can only see a short distance ahead, but we can see plenty there that needs to be done*" [210].

PROBLEM-SPACE DEFENSES This thesis focused on problem-space attacks, so that we can understand the attack surface of learning-based systems in depth. The next step should also include defenses that consider the relation between $\mathcal{Z}$ and $\mathcal{F}$. Similar to research on attacks, most prior work on defenses has focused on $\mathcal{F}$. While a feature-space defense can increase the robustness of a learning-based system against real-world attacks, it does not have to. As an example, consider a regularized model that uses many features with evenly-distributed weights [65]. This model can be harder to evade than a model that relies on a few features only. Yet, an attack might still be able to create a real object that changes many features, for instance, due to a semantic gap. Fortunately, considering the problem space can also provide novel insights on defenses. Attackers have to fulfill challenging constraints in $\mathcal{Z}$, such as preserving the plausibility and semantics (see Chapter 3). This, in turn, can lead to new defenses by considering these constraints in the feature design. In summary, although researchers have recently started to explore problem-space defenses [e.g., 49], further studies in the context of $\mathcal{Z}$ and $\mathcal{F}$ should follow.

FURTHER ATTACKS ON THE MAPPING Chapter 4 introduced a novel attack surface in machine learning: the mapping from $\mathcal{Z}$ to $\mathcal{F}$. This surface is rather application-specific, so that this thesis focused on image scaling, followed by two further examples from the text and forensic domain to underline the overall threat. Based on these results, researchers and practitioners ought to analyze the mapping in every application that makes use of different operations to preprocess objects and to extract features for machine learning.

TRANSFER OF KNOWLEDGE Chapter 5 showed that concepts and knowledge between adversarial learning and digital watermarking can be transferred. Future work should build on this unification and make use of the linked attacks and defenses. Developed concepts can also serve as guidance for new methods. Eventually, the two communities can learn from each other and combine the *best of both worlds*.

ADVERSARIAL DATA PROCESSING Not only adversarial learning and digital watermarking deal with an adversary. More research fields such as multimedia forensics or steganography operate under an adversary's presence. Hence, it might be possible to link the problem or feature space from machine learning with spaces from other fields. This can allow the transfer of concepts and knowledge, as successfully demonstrated in this thesis. Ultimately, a unified view and cooperation across all these adversarial research fields can bundle resources and knowledge, thereby allowing us to make more powerful steps together—increasing the distance we can see ahead.

A

ADDITIONAL BACKGROUND INFORMATION

In the following, we shortly examine three common distance metrics in adversarial learning, that quantify similarity between two vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$ [38, 88, 156]. Note that the individual features of a feature vector $\boldsymbol{x} \in \mathbb{R}^d$ are given by x_i, that is, $\boldsymbol{x} = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$.

A distance function $\mathcal{D}\left(\boldsymbol{x}, \boldsymbol{x}'\right)$ can use the L_p distance metric that is given as $\|\boldsymbol{x} - \boldsymbol{x}'\|_p$, where $\|\cdot\|_p$ is the p-norm:

$$\|\boldsymbol{u}\|_p = \left(\sum_{i=1}^{d} |u_i|^p\right)^{\frac{1}{p}}. \tag{A.1}$$

Three common distance metrics are based on a different p-norm:

- The L_0 distance is given by

  $$\|\boldsymbol{x} - \boldsymbol{x}'\|_0 = |\{i \mid x_i \neq x_i'\}|. \tag{A.2}$$

  This distance only counts the elements that differ between $\boldsymbol{x}$ and $\boldsymbol{x}'$. It does not consider the amount of each difference. In the case of two images, for example, it only counts the number of pixels that are different. If used for adversarial examples, this distance usually leads to a few, but strong changes [38].

- The L_2 distance is given by

  $$\|\boldsymbol{x} - \boldsymbol{x}'\|_2 = \sqrt{\sum_{i=1}^{d} (x_i - x_i')^2} \tag{A.3}$$

  This distance corresponds to the Euclidean distance between two vectors. It penalizes large differences. If used for adversarial examples, it usually leads to evenly distributed, small changes [38].

- The L_∞ distance is given by

  $$\|\boldsymbol{x} - \boldsymbol{x}'\|_\infty = \max(|x_1 - x_1'|, \ \ldots \ , |x_d - x_d'|). \tag{A.4}$$

  This distance only uses the largest difference over all dimensions. For adversarial examples in the image domain, for instance, the adversary can use this distance to define a tolerable, maximum modification. All pixels can be arbitrarily changed up to this maximum.

Each distance can be used for an attack. Yet, note that none of them perfectly matches the human perception [38], so that multiple metrics may have to be evaluated.

B

ATTACK IN THE PROBLEM SPACE

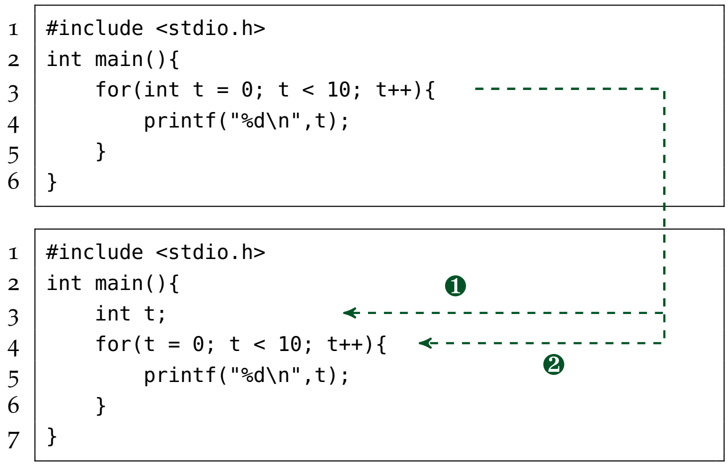
This chapter provides additional information about adversarial examples in the problem space, examined in Chapter 3.

B.1 EXAMPLE FOR PROBLEM-FEATURE SPACE DILEMMAS

The following section provides more insights on the problem-feature space dilemmas when creating adversarial examples of source code. To this end, let us use the source code example in Figure B.1. In the following, 289 features are extracted with the attribution method by Caliskan et al. [36]. To measure the inherent feature correlation, the raw frequency of occurrence of each feature is used. The common TF-IDF-normalization in authorship attribution is not applied [36]. Otherwise, the change of a single feature would automatically affect other features.

Let us further assume that an adversary needs to decrease only the value of the AST bigram feature `ForStmt-DeclStmt`. This feature is part of the syntactic feature set (see Section 3.1.1). It reflects a developer's tendency to declare a variable inside or outside of a `for` statement. In order to remove the bigram, the declaration of the iteration variable needs to be moved out of the `for` block. Although this is a very targeted modification, it impacts 17 of 289 features. Figure B.1 shows the change of a few selected features.

This example underlines the problem that plausible and valid modifications of source code are limited and features are inherently correlated. Hence, it is difficult to obtain a targeted feature vector. In this example, the adversary wants to decrease `ForStmt-DeclStmt` by 1 only, but multiple other features change as well.



```
#include <stdio.h>
int main(){
    for(int t = 0; t < 10; t++){
        printf("%d\n",t);
    }
}
```

```
#include <stdio.h>
int main(){
    int t;
    for(t = 0; t < 10; t++){
        printf("%d\n",t);
    }
}
```

Feature	Family	Before	After
ForStmt—DeclStmt	AST Node Bigram	1.00	0.00
DeclRefExpr	AST Node Type	4.00	5.00
DeclRefExpr	Average depth of AST Node Type	6.25	6.00
t	Code in AST leaves	4.00	5.00
int	Average depth of code in AST leaves	4.50	4.00

Figure B.1: Problem-feature space dilemmas. In order to rewrite the `ForStmt-DeclStmt` AST bigram, we need to move the declaration in front of the for statement (steps ❶ and ❷). This necessarily changes other, correlated features. The table shows one randomly selected feature per family together with its feature value before and after transformation.

B.2 SEMANTIC GAP IN CODE ATTRIBUTION

Let us shortly analyze why a semantic gap can simplify an attack using source code manipulations as example. Assume that the adversary adds lexemes as a comment. They are not used for the program, but can influence the syntactic feature set. This semantic gap simplifies various constraints. Transformations are easier to implement. The adversary is relieved from implementing transformations that are semantics-preserving. Side effects are mitigated, since adding an existing variable name in a comment, for instance, does not require any consolidation operations. Taken together, this semantic gap reduces the gap between problem and feature space.

However, exploiting this semantic gap also has disadvantages: a defender could easily remove all comments when extracting syntactic features. Furthermore, the plausibility constraint might suffer, since a large number of unusual lexemes in the comments or a large amount of added bytes can directly uncover an attack.

B.3 LIST OF CODE TRANSFORMATIONS

A list of all 35 developed code transformations is presented in Table B.1. The transformers are grouped accordingly to the family of their implemented transformations, i. e., transformations altering the control flow, transformations of declarations, transformations replacing the used API, template transformations, and miscellaneous transformations.

Control Transformations

Transformer	Description of Transformations
For statement	Replaces a `for`-statement by an equivalent `while`-statement.
While statement	Replaces a `while`-statement by an equivalent `for`-statement.
Function creator	Moves a whole block of code to a standalone function and creates a call to the new function at the respective position. The transformer identifies and passes all parameters required by the new function. It also adapts statements that change the control flow (e.g. the block contains a `return` statement that also needs to be back propagated over the caller).
Deepest block	Moves the deepest block in the AST to a standalone function.
If statement	Split the condition of a single `if`-statement at logical operands (e.g., `&&` or `||`) to create a cascade or a sequence of two `if`-statements depending on the logical operand.

Declaration Transformations

Transformer	Description of Transformations
Array	Converts a static or dynamically allocated array to a C++ vector object.
String	*Array option:* Converts a char array (C-style string) into a C++ string object. The transformer adapts all usages in the respective scope, for instance, it replaces all calls to `strlen` by calling the instance method `size`. *String option:* Converts a C++ string object into a char array (C-style string). The transformer adapts all usages in the respective scope, for instance, it deletes all calls to `c_str()`.
Floating-point type	Converts `float` to `double` as next higher type.
Boolean	*Bool option:* Converts true or false by an integer representation to exploit the implicit casting. *Int option:* Converts an integer type into a boolean type if the integer is used as boolean value only.
Typedef	*Convert option:* Convert a type from source file to a new type via `typedef`, and adapt all locations where the new type can be used. *Delete option:* Deletes a type definition (`typedef`) and replace all usages by the original data type.
Include-Remove	Removes includes from source file that are not needed.
Unused code	*Function option:* Removes functions that are never called. *Variable option:* Removes global variables that are never used.
Init-Decl	*Move into option:* Moves a declaration for a control statement if defined outside into the control statement. For instance, `int i; ...; for(i = 0; i < N; i++)` becomes `for(int i = 0; i < N; i++)`. *Move out option:* Moves the declaration of a control statement's initialization variable out of the control statement.

Table B.1: List of code transformations.

API Transformations

Transformer	Description of Transformations
Input interface	*Stdin option:* Instead of reading the input from a file (e.g. by using the API `ifstream` or `freopen`), the input to the program is read from stdin directly (e.g. with `cin` or `scanf`). *File option:* Instead of reading the input from stdin, the input is retrieved from a file.
Output interface	*Stdout option:* Instead of printing the output to a file (e.g. with `ofstream` or `freopen`), the output is written directly to stdout (e.g. with `cout` or `printf`). *File option:* Instead of writing the output directly to stdout, the output is written to a file.
Input API	*C++-Style option:* Substitutes C APIs used for reading input (e.g., `scanf`) by C++ APIs (e.g., usage of `cin`). *C-Style option:* Substitutes C++ APIs used for reading input (e.g., usage of `cin`) by C APIs (e.g., `scanf`).
Output API	*C++-Style option:* Substitutes C APIs used for writing output (e.g., `printf`) by C++ APIs (e.g., usage of `cout`). *C-Style option:* Substitutes C++ APIs used for writing output (e.g., usage `cout`) by C APIs (e.g., `printf`).
Sync-with-stdio	Enable or remove the synchronization of C++ streams and C streams if possible.

Template Transformations

Transformer	Description of Transformations
Identifier	Renames an identifier, i.e., the name of a variable or function. If no template is given, default values are extracted from the 2016 Code Jam Competition set that was used by Caliskan et al. [36] and that is not part of the training and test set. Default values, such as `T`, `t`, ..., `i`, are then tested.
Include	Adds includes at the beginning of the source file. If no template is given, the most common includes from the 2016 Code Jam Competition are used as defaults.
Global declaration	Adds global declarations to the source file. Defaults are extracted from the 2016 Code Jam Competition.
Include-typedef	Inserts a type using `typedef`, and updates all locations where the new type can be used. Defaults are extracted from the 2016 Code Jam Competition.

Miscellaneous Transformations

Transformer	Description of Transformations
Compound statement	*Insert option:* Adds a compound statement (`{...}`). The transformer adds a new compound statement to a control statement (`if`, `while`, etc.) given their body is not already wrapped in a compound statement. *Delete option:* Deletes a compound statement (`{...}`). The transformer deletes compound statements that have no effect, i.e., compound statements containing only a single statement.
Return statement	Adds a return statement. The transformer adds a return statement to the main function to explicitly return 0 (meaning success). Note that main is a non-void function and is required to return an exit code. If the execution reaches the end of main without encountering a return statement, zero is returned implicitly.
Literal	Substitutes a return statement returning an integer literal by a statement that returns a variable. The new variable is declared by the transformer and initialized accordingly.

Table B.1: List of code transformations (continued).

B.4 MONTE-CARLO TREE SEARCH

This section provides further details about the developed variant of Monte-Carlo Tree Search. Algorithm 1 gives an overview of the attack. The procedure `Attack` starts with the root node r_0 that represents the original source code z. The algorithm then works in two nested loops:

- The outer loop in lines 3–5 repetitively builds a search tree for the current state of source code r, and takes a single move (i.e. a single transformation). To do so, in each iteration, the child node with the highest average classifier score is chosen. This process is repeated until the attack succeeds or a stop criterion is fulfilled (a fixed number of outer iterations is reached or no improvement over multiple iterations is observed) (line 3).
- The procedure `MCTS` represents the inner loop. It iteratively builds and extends the search tree under the current root node r. As this procedure is the main building block of the attack, let us examine the individual steps in more detail in the following.

Algorithm 1 Monte-Carlo Tree Search

```
 1: procedure ATTACK($r_0$)
 2:     $r \leftarrow r_0$
 3:     while not SUCCESS($r$) and not STOPCRITERION($r$) do
 4:         MCTS($r$)                          ▷ Extend the search tree under $r$
 5:         $r \leftarrow$ CHILDWITHBESTSCORE($r$)          ▷ Perform next move
 6: procedure MCTS($r$)
 7:     for $i \leftarrow 1, N$ do
 8:         $u \leftarrow$ SELECTION($r, i$)
 9:         $\mathcal{T} \leftarrow$ SIMULATIONS($u$)
10:         EXPANSION($u, \mathcal{T}$)
11:         BACKPROPAGATION($\mathcal{T}$)
```

SELECTION Algorithm 2 shows the pseudocode to find the next node which is evaluated. The procedure recursively selects a child node according to a selection policy. It stops if the current node has no child nodes or is not marked as visited. The procedure finally returns the node that will be evaluated next.

As the number of possible paths grows exponentially (we have up to 35 transformations as choice at each node), we cannot evaluate all possible paths. The tree creation thus crucially depends on a selection policy. A simple heuristic is used to approximate the *Upper Confidence Bound for Trees* algorithm that is often used as selection policy [see 34]. Depending on the current iteration index i of `Selection`, the procedure `SelectionPolicy` alternately returns the decision rule to choose the child with the highest average score, with the lowest visit count, or with the highest score standard deviation. This step balances the *exploration* of less-visited nodes and the *exploitation* of promising nodes with a high average score.

Algorithm 2 Selection Procedure of MCTS

```
procedure Selection($r$, $i$)
    $D \leftarrow$ SelectionPolicy($i$)
    $u \leftarrow r$
    while $u$ has child nodes do
        $v \leftarrow$ SelectChild($u, D$)                ▷ Child of $u$ w.r.t. to $D$
        if $v$ not marked as visited then
            Mark $v$ as visited
            return $v$
        else
            $u \leftarrow v$
    return $u$
```

SIMULATIONS Equipped with the node u that needs to be evaluated, the next step generates a set of transformation sequences $\mathcal{T}$ that start at u:

$$\mathcal{T} = \{\mathbf{T}_j \mid j = 1, \ldots, k \text{ and } |\mathbf{T}_j| \leq M\} \,, \tag{B.1}$$

where $|\mathbf{T}_j|$ is the number of transformations in $\mathbf{T}_j$. The sequences are created randomly and have a varying length which is, however, limited by M. The evaluation in Section 3.5 uses $M = 5$ to reduce the number of possible branches.

In contrast to the classic game use-case, the returned scores are available as early feedback, so that we do not need to play out a full game. In other words, it is not necessary to evaluate the complete path to obtain feedback. The classifier score is determined for each sequence by passing the modified source code at the end of the respective sequence to the attribution method. Note a further difference to the general MCTS algorithm. Instead of evaluating only one path, the attack creates a batch of sequences that can be efficiently executed in parallel. This reduces the computation time while obtaining the scores for various paths.

EXPANSION Algorithm 3 shows the pseudocode to insert the respective transformations from the sequences as novel tree nodes under u. For each sequence $\mathbf{T}$, the procedure starts with the respective first transformation. It checks if a child node with the same transformation already exists under u. If not, a new node v is created and added as child under u. Otherwise, the already existing node v is used. This step is repeated with v and the next transformation. Figure 3.15 from Section 3.4.4 exemplifies this expansion step.

BACKPROPAGATION Algorithm 4 shows the last step that backpropagates the classifier scores to the root. For each sequence, the

Algorithm 3 Expansion Procedure of MCTS

```
procedure EXPANSION($u$, $\mathcal{T}$)
    for $\mathbf{T}$ in $\mathcal{T}$ do                    ▷ For each sequence
        $z \leftarrow u$
        for $T$ in $\mathbf{T}$ do                          ▷ For each transformer
            if $z$ has no child with $T$ then
                $v \leftarrow$ CREATENEWNODE($T$)
                $z$.add_child($v$)
            else
                $v \leftarrow z$.GETCHILDWITH($T$)
            $z \leftarrow v$
```

Algorithm 4 Backpropagation Procedure of MCTS

```
procedure BACKPROPAGATION($\mathcal{T}$)
    for $\mathbf{T}$ in $\mathcal{T}$ do
        $s \leftarrow$ GETSCORE($\mathbf{T}$)
        get $n$ as tree leaf of current sequence
        while $n$ is not None do                            ▷ Backpropagate to root
            $n$.visitCount $\leftarrow$ $n$.visitCount + 1  ▷ Increase visit count
            $n$.scores = $n$.scores $\cup$ $s$              ▷ Append score
            $n \leftarrow n$.parent                         ▷ Will be None for root node
```

procedure first determines the last node n of the current sequence and the observed classifier score s at node n. Next, all nodes on the path from n to the root node of the search tree are updated. First, the visit count of each path node is incremented. Second, the final classifier score s is added to the score list of each path node. Both statistics are used by `SelectChild` to choose the next promising node for evaluation. Furthermore, `ChildWithBestScore` uses the score list to obtain the child node with the highest average score.

VARIATION FOR SURROGATE MODEL A slight variation is used for the scenario with a surrogate model (see threat model in Section 3.4.1 and evaluation in Section 3.5.3). To improve the transferability rate from the surrogate to the original model, the attack does not terminate at the first successful adversarial example. Instead, it collects all successful samples and stops the outer loop after a predefined number of iterations. The sample with the highest score on the surrogate is tested on the original classifier.

This variation leads to a high-confidence attack instead of the previous minimum-modification variant in Equation 3.5. Formally, for a targeted attack, we now optimize the classifier output for class y^t, so that the adjusted attack is given as:

$$\begin{aligned} \underset{\mathbf{T}}{\arg\max} \quad & [c_g(\mathbf{T}(z))]_{y^t} \\ \text{s.t.} \quad & c_f(\mathbf{T}(z)) = y^t, \\ & \mathbf{T}(z) \models \Gamma\,. \end{aligned} \tag{B.2}$$

B.5 LIST OF DEVELOPERS FOR IMPERSONATION

Table B.2 maps the letters to the 20 randomly selected programmers from the 2017 GCJ contest.

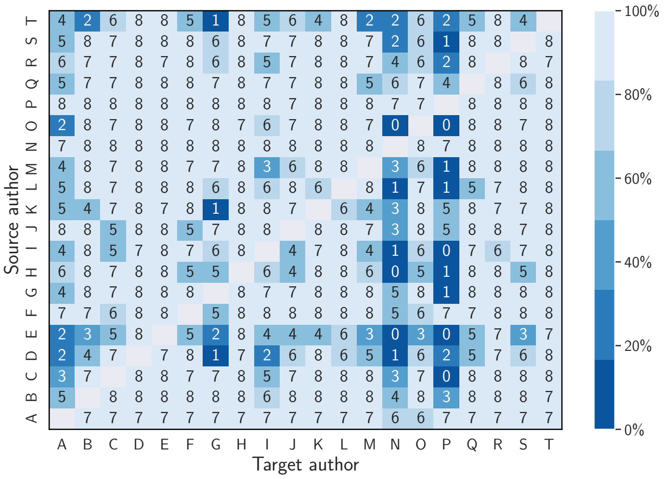
Letter	Author	Letter	Author
A	4yn	K	chocimir
B	ACMonster	L	csegura
C	ALOHA.Brcps	M	eugenus
D	Alireza.bh	N	fragusbot
E	DAle	O	iPeter
F	ShayanH	P	jiian
G	SummerDAway	Q	liymouse
H	TungNP	R	sdya
I	aman.chandna	S	thatprogrammer
J	ccsnoopy	T	vudduu

Table B.2: List of developers for impersonation.

B.6 ADDITIONAL EVALUATION RESULTS

This section provides further results from the evaluation in Section 3.5.

TARGETED ATTACK Figure B.2 presents the impersonation matrix for the targeted attack against the attribution method by Abuhamad et al. [2]. The results are similar to the impersonation results against the attribution method by Caliskan et al. [36] (see Figure 3.18).



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
T	4	2	6	8	8	5	1	8	5	6	4	8	2	2	6	2	5	8	4	
S	5	8	7	8	8	8	6	8	7	7	8	8	7	2	6	1	8	8		8
R	6	7	7	8	7	8	6	8	5	7	8	8	7	4	6	2	8		8	7
Q	5	7	7	8	8	8	7	8	7	7	8	8	5	6	7	4		8	6	8
P	8	8	8	8	8	8	8	8	8	7	8	8	8	7	7		8	8	8	8
O	2	8	7	8	8	7	8	7	6	7	8	8	7	0		0	8	8	7	8
N	7	8	8	8	8	8	8	8	8	8	8	8	8		8	7	8	8	8	8
M	4	8	7	8	8	7	7	8	3	6	8	8		3	6	1	8	8	8	8
L	5	8	7	8	8	8	6	8	6	8	6		8	1	7	1	5	7	8	8
K	5	4	7	8	7	8	1	8	8	7		6	4	3	8	5	8	7	7	8
J	8	8	5	8	8	5	7	8	8		8	8	7	3	8	5	8	8	7	8
I	4	8	5	7	8	7	6	8		4	7	8	4	1	6	0	7	6	7	8
H	6	8	7	8	8	5	5		6	4	8	8	6	0	5	1	8	8	5	8
G	4	8	7	8	8	8		8	7	7	8	8	8	5	8	1	8	8	8	8
F	7	7	6	8	8		5	8	8	8	8	8	8	5	6	7	7	8	8	8
E	2	3	5	8		5	2	8	4	4	4	6	3	0	3	0	5	7	3	7
D	2	4	7		7	8	1	7	2	6	8	6	5	1	6	2	5	7	6	8
C	3	7		8	8	7	7	8	5	7	8	8	8	3	7	0	8	8	8	8
B	5		8	8	8	8	8	8	6	8	8	8	8	4	8	3	8	8	8	7
A		7	7	7	7	7	7	7	7	7	7	7	7	6	6	7	7	7	7	7

Figure B.2: Impersonation matrix for the attribution method by Abuhamad et al. [2]. Each cell indicates the number of successful attack attempts for the 8 challenges.

CASE STUDY To provide an intuition for a successful impersonation, Figure B.3 shows a case study from the evaluation. The upper listing shows the code from the source author in original form, and the middle listing its modified version that is classified as the target author. For comparison, the lower listing shows the original source text from the target author for the same challenge, which is not available to the attacker. The table lists four conducted transformations. For instance, the target author has the stylistic pattern to use `while` statements, C functions for the output, and particular typedefs. By changing these patterns, the attack succeeds in misleading the attribution method.

Source Author I

```
 1  cout << std::fixed;
 2  for (long long ccr = 1; ccr <= t; ++ccr) {
 3      double d, n, ans = INT_MIN;
 4      cin >> d >> n;
 5      for (double i = 0; i < n; ++i) {
 6          double k, s;
 7          cin >> k >> s;
 8          [...]
 9      }
10      ans = d / ans;
11      cout << "Case #" << ccr << ": " << setprecision(7) << ans << "\n";
12  }
```

Source → Target

```
 1  typedef double td_d;                                        ❶
 2  [...]
 3  long long ccr = 1;
 4  while (ccr <= t) {                                          ❷
 5      td_d d, n, ans = INT_MIN;
 6      cin >> d >> n;
 7      td_d i;                                                 ❸
 8      for (i = 0; i < n; ++i) {
 9          td_d k, s;
10          cin >> k >> s;
11          [...]
12      }
13      ans = d / ans;
14      printf("Case #%lld: %.7f\n", ccr, ans);                 ❹
15      ++ccr;
16  }
```

Target Author P

```
 1  int T, cas = 0;
 2  cin >> T;
 3  while (T--) {
 4      int d, n;
 5      cin >> d >> n;
 6      double t = 0;
 7      while (n--) {
 8          int k, s;
 9          cin >> k >> s;
10          t = max((1.0 * d - k) / s, t);
11      }
12      double ans = d / t;
13      printf("Case #%d: %.10f\n", ++cas, ans);
14  }
```

Transformer	**Description**
❶ Typedef	adds typedef and replaces all locations with previous type by novel typedef.
❷ For statement	converts `for`-statement into an equivalent `while`-statement, as target tends to solve problems via `while`-loops.
❸ Init-Decl	moves a declaration out of the control statement which mimics the declaration behavior of while-statements.
❹ Output API	substitutes C++ API for writing output by C API `printf`. To this end, it determines the precision of output statements by finding the commands: `fixed` (line 1) and `setprecision` (line 11).

Figure B.3: Impersonation example from the evaluation for the GCJ problem *Steed 2: Cruise Control*.

B.7 ADDITIONAL INFORMATION FOR RELATED WORK

This section explains the categorization of constraints for the different papers in Table 3.10.

We begin with source code. Regarding Matyukhina et al. [141], the created adversarial examples preserve the semantics. However, most transformations aim at layout features, for instance, by manipulating brackets, spaces, and empty lines. Thus, a robustness against preprocessing is not given. Moreover, transformations also add comments from the target developer. This exploits a semantic gap. Together with control-flow flattening, this also affects the plausibility. Therefore, the plausibility, preprocessing, and semantic gap constraints are marked as not fulfilled.

Let us continue with text. Li et al. [124] create adversarial examples of text by modifying words. The transformations add spelling mistakes. For instance, a single character is removed within a word or two adjacent characters are swapped. Synonyms are also used to replace a whole word. The former transformation based on mistakes can impact the semantics and plausibility. As a result, semantics and plausibility are marked as partly fulfilled. Li et al. [124] show in the evaluation that spelling mistakes are largely affected by spell checkers while repairing synonyms is more difficult. Preprocessing is thus marked as partly fulfilled. As the visible text is directly rewritten, no semantic gap is exploited. Papernot et al. [154] also create adversarial examples of text. However, this attack replaces a word by another, arbitrary one from a dictionary such that the attack moves towards the gradient direction. Thus, the semantics and plausibility constraints are marked as not fulfilled. A replaced word is difficult to repair via preprocessing, so that this constraint is marked as fulfilled. As the visible text is directly rewritten, no semantic gap is exploited.

We continue with the Windows domain. Kolosnjaji et al. [112] evade malware detection of Windows PE files. Their approach is to append bytes at the end of the file. This preserves the semantics. Although this is not directly visible in the file, a security expert can easily detect the bytes by checking the end of the file. Plausibility is thus marked as partly fulfilled. The end of a PE file can also be detected and removed [172], so that the preprocessing constraint is not fulfilled. The attack exploits a semantic gap, since the added bytes are not used for executing the program, but are considered by the learning-based system. Similarly, Suciu et al. [197] evade detection by appending bytes. They also add bytes in the slack spaces between the sections in a PE file. Yet, this can also be detected and removed [172]. The constraint assignment thus corresponds to Kolosnjaji et al. [112].

We proceed with PDF files. Šrndić and Laskov [236] evade PDF malware detection by adding adversarial content between the cross-reference table and the trailer of a PDF file. A PDF viewer skips this area, so that the semantics are preserved by design, but a semantic gap is exploited. Although the user does not see this modification, a security expert could locate this area by analyzing the file. The plausibility is thus marked as partly fulfilled. The unused area should be removable, so that the approach is not robust to preprocessing. Xu et al. [227] evade PDF malware detection by relying on genetic programming with random mutations. Dynamic tests are used to check if the modifications preserve the functionality of the malware. Thus, the semantics constraint is marked as fulfilled. As the attack rewrites any object in the PDF tree structure, this might lead to visible modifications in the file. Plausibility was not explicitly tested in the paper and the constraint is thus marked as not fulfilled. Moreover, after conducting the attack, Xu et al. [227] find that the attack also relies on artifacts. For instance, font objects are missing in the malware dataset, so that a classifier learns that the presence of font objects is an indicator for a benign input. The attack also makes use of it by increasing the font count. In principle, such spurious features can be repaired and removed [162]. Nevertheless, the attack also uses other transformations. It is thus not clear how well preprocessing affects the adversarial changes in general. Thus, the preprocessing constraint is marked as partly fulfilled in Table 3.10. Finally, the approach rewrites any object in a PDF file and thus does not exploit a semantic gap.

Finally, we examine the Android domain. Grosse et al. [91] evade Android malware detection by adding content to the Android manifest file. This preserves the semantics. The added content is visible and might be implausible, so that this constraint is marked as partly fulfilled. Yet, a further analysis could check if the added, unused content is needed and remove it. This might require a dynamic analysis, so that the preprocessing constraint is marked as partly fulfilled. By rewriting the manifest file, no semantic gap is exploited. Pierazzi et al. [162] also evade Android malware detection by adding plausible content to Android applications from benign samples through automated code transplantation. By using opaque predicates, a static analysis cannot find out that the added code is not executed at runtime. Hence, the attack preserves the semantics by design and is robust to preprocessing. No semantic gap is exploited by changing the code of the Android file. By using code from other benign samples, the added code is realistic by construction. In contrast to the code transformations in this thesis, an analyst could spot the principle behind the opaque predicates if looking at the code. Yet, Android samples are typically large, so that this might still be challenging. Plausibility is therefore marked as fulfilled.

C

ATTACK ON THE MAPPING

This chapter provides additional information about the discussed attacks on the mapping, examined in Chapter 4.

C.1 COMBINING POISONING WITH IMAGE-SCALING ATTACKS

In this section, we examine the combination of poisoning and image-scaling attacks in more detail. Scaling attacks are suitable for poisoning attacks in two threat models [166].

- *Stealthiness during training time.* The adversary manipulates the training data. If poisoning attacks leave visible traces in the training data, image-scaling attacks can be useful to conceal these manipulations during training time.
- *Stealthiness during deployment time.* The adversary trains the learning model, for instance, as insider, due to outsourcing or in a transfer-learning scenario. The adversary can thus arbitrarily change the training data (or the model). Image-scaling attacks in this case are not needed (or applicable). However, common backdoor attacks require adding a visible backdoor trigger in samples at deployment time [e.g., 92, 128, 230]. If such samples are examined, a visible backdoor trigger would directly reveal that a model has been tampered. These attacks can thus benefit from image-scaling attacks at deployment time.

Both threat models are relevant for backdoor attacks, but only the first is relevant for training-only poisoning attacks.

In the following, we study both groups of poisoning attacks (backdoor and training-only attack) and their combination with image-scaling attacks in the different threat models. We start with a backdoor attack, and then examine a training-only poisoning attack.

BACKDOOR ATTACK As first attack, we study the *BadNets* backdoor method from Gu et al. [92]. The adversary chooses a target label and a small, bounded backdoor trigger, such as a small green square. This trigger is added to a limited number of training images and the respective label is changed to the target label. In this way, the learning algorithm associates this trigger with the target class later at training time. Another possibility is to add the trigger only to images of the target class, so that no label changes are necessary.

To hide the backdoor trigger, the adversary can apply an image-scaling attack. The original image without trigger represents the source

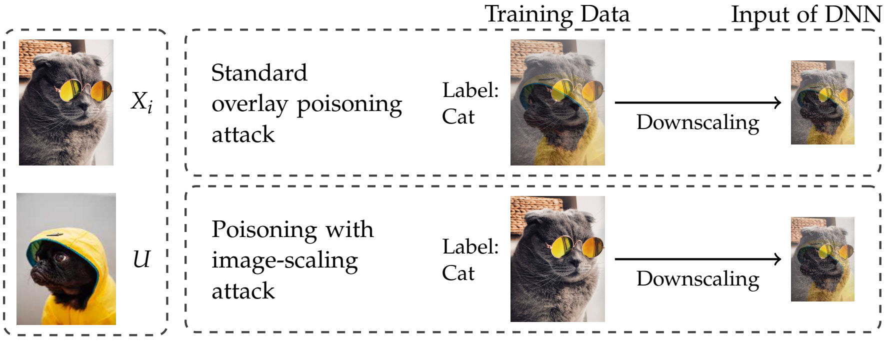


Figure C.1: Example of an overlay-poisoning attack [184]: a neural network learns to classify the particular dog as cat by blending the dog with multiple cat images—without changing training labels.

image S, its version with the trigger in the network's input dimensions is the target image T. By solving Equation 4.1, the adversary obtains the attack image A that is saved as training image. The trigger is only present after downscaling. Figure 4.3a in Section 4.2.1 exemplifies the combination.

Scaling attacks allow more insidious backdoor attacks in both threat models introduced in this section. At training time, the backdoor trigger in the training data can be disguised with scaling attacks. Thus, it is harder for a human analyst to detect the manipulations based on the full-size images. The learning algorithm, however, uses the downscaled images and associates the backdoor trigger with a respective target label. At deployment time, the adversary normally activates the backdoor with the trigger—with or without scaling attack. Moreover, scaling attacks are also applicable in the second threat model alone if a backdoor trigger only needs to be concealed in the input at deployment time.

A particular advantage of backdoor attacks is that a backdoor trigger can be easily activated at deployment time in the physical world [92]. In the example of Figure 4.3a, the adversary could put a green post-it on a road sign. The post-it resembles the green square backdoor trigger from the training time. This attack can also benefit from scaling attacks. The trigger is disguised in the training data by using image-scaling attacks, and activated in the physical world—necessarily without a scaling attack.

TRAINING-ONLY POISONING ATTACK As second attack, we study the overlay-poisoning attack by Shafahi et al. [184]. In particular, they present a clean-label attack, as the label of the modified training samples is not changed. As a result, this poisoning strategy becomes more powerful in combination with image-scaling attacks: The manipulated images keep their correct class label and show no obvious traces of manipulation.

In particular, the adversary's objective is that the model classifies a specific and unmodified sample U as a chosen target class y^* at deployment time. To this end, the adversary chooses a set of images X_i with the class y^*. Similar to watermarking, she embeds a low-opacity version of U into each image X_i:

$$X_i' = \alpha \cdot U + (1 - \alpha) \cdot X_i. \tag{C.1}$$

If the parameter α, for instance, is set to 0.3, features of U are blended into X_i while the manipulation is less visible. For an image-scaling attack, the adversary chooses X_i as source image S, and creates X_i' as respective target image T in the network's input dimensions. The computed attack image A serves as training image then. The changed images are finally added to the training set together with their correct label y^*. As a result, the learning algorithm associates U with y^*. At deployment time, U can be passed to the learning system without any changes and is classified as y^*.

As an example, Figure C.1 shows an overlay-poisoning attack on the popular TensorFlow library. The target class y^* is *cat*, the dog image is U, and one selected training image X_i showing a cat is visible. The network will learn to classify the particular dog U as cat if this dog is repeatedly inserted into varying cat-images during training. In the attack's standard version, the slight manipulation of the training image is still noticeable. Yet, image-scaling attacks conceal the manipulation of the training data effectively. The dog appears only in the downscaled image which is finally used by the neural network.

Overall, the overlay attack shows a possible combination between poisoning and scaling attacks if the entire image is slightly changed instead of adding a small and bounded trigger.

C.2 SELECTIVE RANDOM FILTER

The random filter is identical to the selective median filter, except for that it takes a random point from each window instead of the median. That is, given a point $p \in \mathcal{P}$, we consider a window W_p around p of size $2\beta_h \times 2\beta_v$ and randomly select a point as a reconstruction of p. Again, we exclude points $p' \in \mathcal{P}$ from this window to limit the attacker's influence.

Randomly selecting a point for reconstruction comes with two problems. First, the reconstruction becomes non-deterministic. Second, the scaled image might suffer from poor quality. The evaluation in Section 4.5 shows that applying the random filter on benign images leads to a loss of accuracy between 2% and 10.9%. This might be acceptable for the benefit of a very efficient run-time performance. The filter reconstructs an image with a complexity of $\mathcal{O}(|\mathcal{P}|)$, which is independent of the scaling ratio. Furthermore, the filter also provides strong protection from attacks. If an image contains $|\mathcal{P}|$ relevant points,

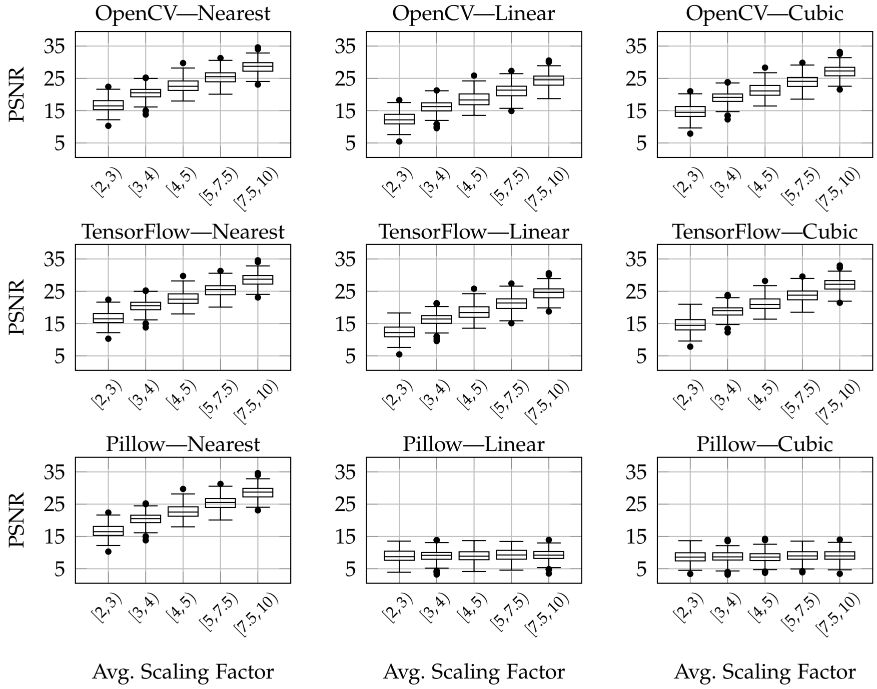


Figure C.2: Success rate of attack regarding O2: the similarity between source image and attack image, measured by the PSNR value.

there exist $|\mathcal{P}| \cdot 4\,\beta_h \beta_v$ possible combinations for its reconstruction. If we consider a scaling ratio of 5 and a target size of 200×200, this already amounts to 4 million different combinations an attacker needs to guess from.

C.3 ADDITIONAL EVALUATION RESULTS

Figures C.2 to C.7 give further information and examples from the evaluation. In particular, they provide visual examples of successful and failed attacks, thereby highlighting the working principle of image-scaling attacks.

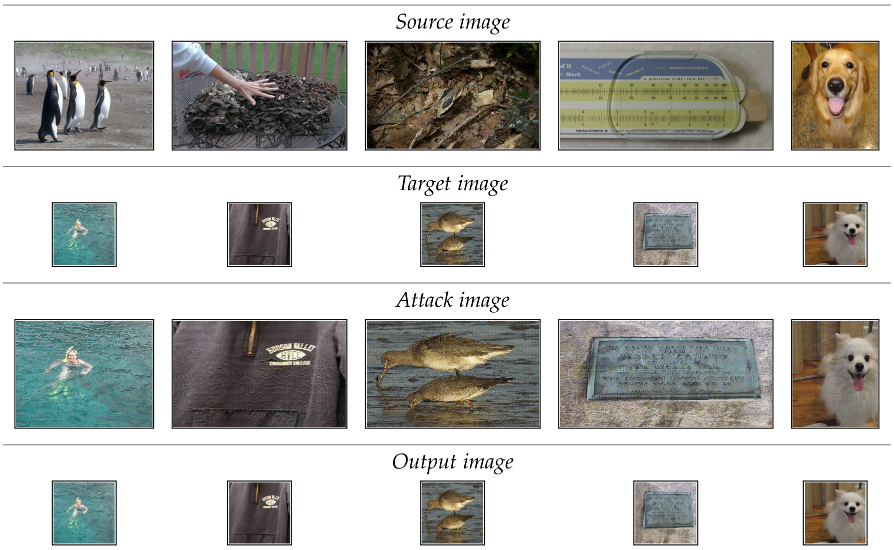


Figure C.3: Best images of the L_0 version of the adaptive attack against area scaling (plotted here column-wise per example). The attack fails in all cases with respect to Objective O2, as each attack image is not similar to the source image anymore.

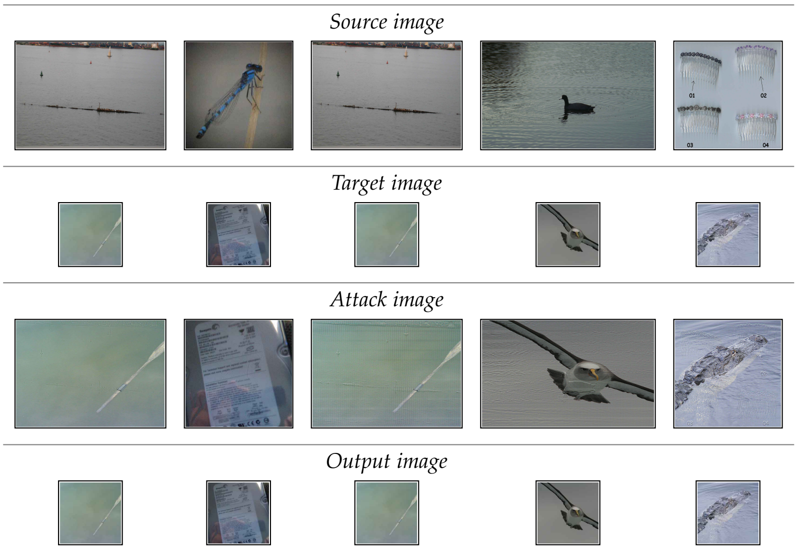
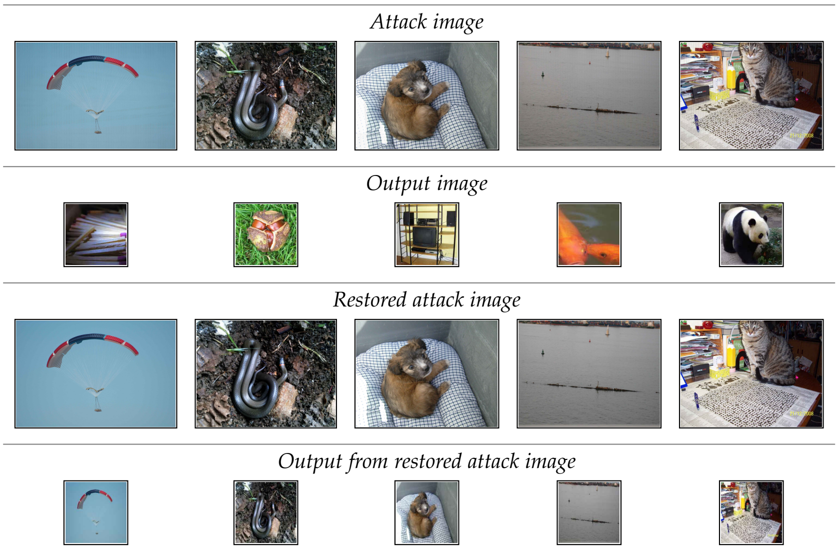


Figure C.4: Selective source scenario against area scaling with the L_1 attack (first two columns) and L_0 attack (last three columns). The attack fails in all cases with respect to Objective O2. While traces from the source image are visible, the attack image overwrites the source image considerably.



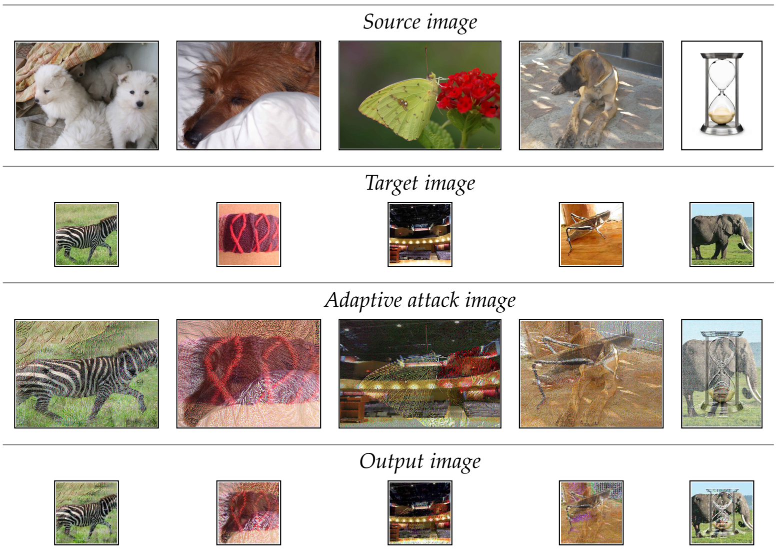
Attack image

Output image

Restored attack image

Output from restored attack image

Figure C.5: Randomly selected examples before and after restoration with median filter (first three columns) and random filter (last two columns). Without restoration, the attack is successful, as the downscaling of the attack image produces an unrelated target image (1st and 2nd row). With restoration, the attack fails in all cases with respect to Objective O1, as the downscaled output from the restored attack image produces the respective content and not an unrelated image (3rd and 4th row). Moreover, the filtering improves quality, as it removes traces from the attack.



Source image

Target image

Adaptive attack image

Output image

Figure C.6: Successful examples regarding Objective O1 from the adaptive attack against the median filter if 20% of the pixels in each block can be changed. The target class is detected, but the attack image is a mix between source and target class. The results thus violate Objective O2.

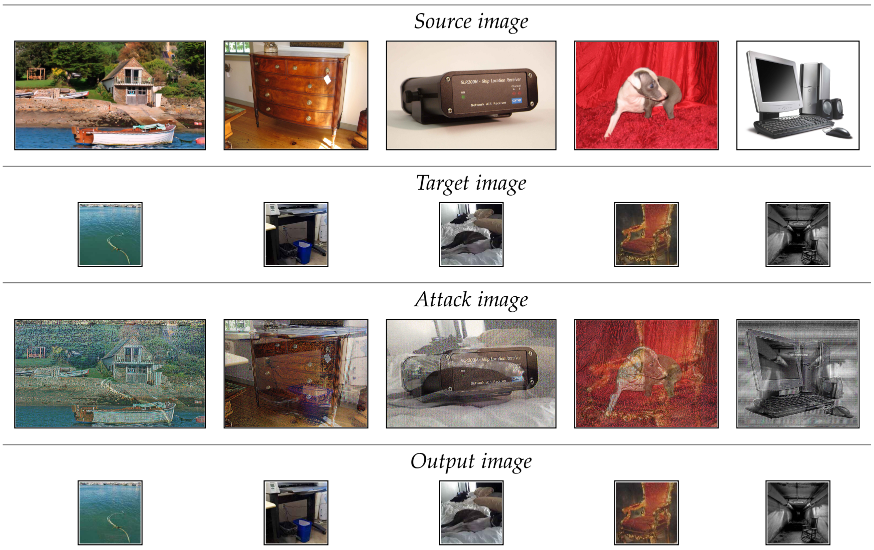


Figure C.7: Examples from the adaptive attack against the median filter where the participants from the user study only recognized the source class. Although the target class was not recognized, it is visible that the attack image is still a mix between source and target class which makes it difficult to determine particular classes. The results thus violate Objective O2.

C.4 ADAPTIVE ATTACK AGAINST MEDIAN FILTER

In the following, we analyze the adaptive attack against the median-based defense. It is demonstrated that the attack is optimal regarding the L_0, L_1, and L_2 norm if each window W_p does not overlap with other windows. An adversary cannot make less changes to control the output of the median filter.

For a given attack image and window W_p, the adversary seeks to manipulate the pixels in W_p such that the median m over W_p still corresponds to p. In this way, the modifications from the image-scaling attack remain even after applying the median filter. Without loss of generality, let us assume that $m < p$ and further unroll W_p to a one-dimensional signal. We consider a signal with uneven length k and denote the numerical order by brackets, so that the signal is given by:

$$x_{(1)}, \cdots, x_{\left(\frac{k-1}{2}\right)}, m_{\left(\frac{k+1}{2}\right)}, x_{\left(\frac{k+3}{2}\right)}, \cdots, x_{(l)}, \cdots, x_{(k)} \tag{C.2}$$

Let us denote by $x_{(l)}$ the largest pixel in the sorted signal that is smaller than p. The objective is to change the signal with the fewest possible changes so that $m = p$.

We start by observing that we need to change $l - \frac{k+1}{2} + 1$ pixels to move the median to p. Less changes do not impact the numerical order sufficiently. We can thus conclude that the minimal L_0 norm for an attack is given by

$$L_0 = l - \tfrac{k+1}{2} + 1 \,. \tag{C.3}$$

Next, it is shown that setting all pixels between m and $x_{(l)}$ to p successfully moves the median as well as minimizes the L_1 and L_2 norm in addition. First, we observe that if we replace pixels with indices in $[1, (k-1)/2]$ by a value smaller than m, the median is not changed. Likewise, replacing pixels larger than $x_{(l)}$ by a value larger than m does not change the median. Two methods can move the median to p: (1) We can replace pixels with indices in $[1, (k+1)/2]$ by p. (2) We can set all pixels with index $[(k+1)/2,\ l]$ to p. Note that we could also use a value larger than p in method (1) or (2), but this would change more than necessary. Comparing both methods, (2) induces less changes regarding the L_1/L_2 norm, as these values are closer to p. Thus, the adaptive attack uses the optimal strategy for the L_1/L_2 norm by setting all pixels between m and $x_{(l)}$ to p. Furthermore, we can derive a simple bound for the L_2 norm:

$$(L_2)^2 = \sum_{(\frac{k+1}{2}) \leqslant i \leqslant l} \left(x_{(i)} - p\right)^2 \leqslant L_0\, (m-p)^2 \ . \tag{C.4}$$

Overall, we can exactly compute the number and amount of required changes for a successful attack. The analysis, however, also shows that the attack always depends on the concrete pair of a source and a target image, and there is no notion of a class boundary. Consequently, a general bound cannot be derived, as achieved with certifiable defenses against adversarial examples. Yet, the empirical results in Section 4.5.5 demonstrate that the necessary changes are very large if target and source images show realistic content, so that the median m and the target value p are not close to each other.

D

LINKING FEATURE AND MEDIA SPACE

This chapter provides additional information about the relation between feature space and media space, examined in Chapter 5.

D.1 BLIND NEWTON SENSITIVITY ATTACK

This section briefly recaps the BNSA [57] in a simplified version. This oracle attack solves Equation 5.8. For simplicity, let us slightly rewrite this optimization problem as follows:

$$\arg\min_{\boldsymbol{\delta}} \ \mathsf{d}(\boldsymbol{\delta}) \quad \text{s.t.} \quad f_{\mathcal{M}}(\tilde{\boldsymbol{x}} + \boldsymbol{\delta}) = y^{-} \ , \tag{D.1}$$

We directly instantiate the distance function $\mathcal{D}$ from Equation 5.8 by using $\mathsf{d}(\cdot)$ that quantifies the modifications with the squared Euclidean norm:

$$\mathsf{d}(\boldsymbol{\delta}) = \|\boldsymbol{\delta}\|_2^2 \ . \tag{D.2}$$

Note that other distance functions could be used for the BNSA as well [see 57].

The adversary has no access to the real-valued output that $f_{\mathcal{M}}(\tilde{\boldsymbol{x}})$ internally computes before returning the binary decision. Therefore, the idea is to rewrite the optimization problem from Equation D.1 into the following unconstrained version:

$$\arg\min_{\boldsymbol{\delta}} \ (\mathsf{d} \circ \mathsf{b}_{\tilde{\boldsymbol{x}}})(\boldsymbol{\delta}). \tag{D.3}$$

The surjection $\mathsf{b}_{\tilde{\boldsymbol{x}}}(\boldsymbol{\delta})$ reflects the prior constraint to find a solution in the other subspace. As a position on the boundary is sufficient, $\mathsf{b}_{\tilde{\boldsymbol{x}}}(\boldsymbol{\delta})$ maps each input to the boundary. In particular, $\mathsf{b}_{\tilde{\boldsymbol{x}}}(\boldsymbol{\delta})$ (i) takes a vector $\boldsymbol{\delta}$ as input, (ii) finds a scalar α such that $\tilde{\boldsymbol{x}} + \alpha\boldsymbol{\delta}$ lies on the decision boundary, (iii) and returns $\alpha\boldsymbol{\delta}$ as output. To highlight the dependence on $\tilde{\boldsymbol{x}}$, it is added as subscript in $\mathsf{b}_{\tilde{\boldsymbol{x}}}$. To implement $\mathsf{b}_{\tilde{\boldsymbol{x}}}(\boldsymbol{\delta})$, a bisection algorithm can be used.

However, $\mathsf{b}_{\tilde{\boldsymbol{x}}}(\boldsymbol{\delta})$ has to map each input vector to the boundary explicitly by running the bisection algorithm each time. Thus, a closed form to solve the problem in Equation D.3 is not applicable. Therefore, numerical iterative methods, such as Newton's method or gradient descent, have to be used. The attack consists of the following steps.

1. The attack first needs to find a signal $\boldsymbol{x}_b$ on the decision boundary. Note that $\boldsymbol{x}_b$ does not need to resemble $\tilde{\boldsymbol{x}}$ [59]. For example, using an image, we can reduce the contrast until the image

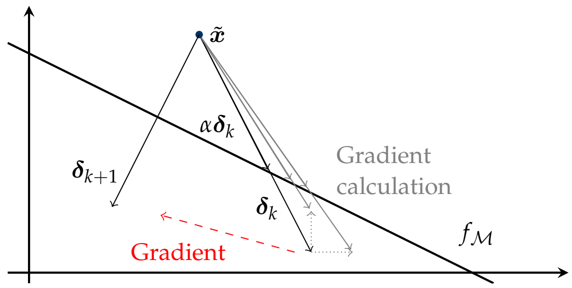


Figure D.1: Blind Newton Sensitivity Attack (BNSA). Queries around a boundary position reveal the function's gradient at this position to minimize the distance between the manipulated signal and the original one.

crosses the boundary and the detector changes the detector output [59]. Alternatively, we can employ a bisection algorithm to find a scalar γ such that $\gamma\tilde{\boldsymbol{x}}$ lies on the decision boundary [57]:

$$\boldsymbol{x}_b = \gamma\tilde{\boldsymbol{x}}. \tag{D.4}$$

Then, the attack computes the start value $\boldsymbol{\delta}_0 = (\gamma - 1) \cdot \tilde{\boldsymbol{x}}$.

2. Based on the current position, the attack finds the direction in which the objective in Equation D.3 decreases the fastest. For example, using the gradient-based version of the BNSA, we can summarize an update step with the following equation:

$$\boldsymbol{\delta}_{k+1} = \boldsymbol{\delta}_k - \underbrace{\xi_k \cdot}_{\text{step length}} \underbrace{\nabla(\mathsf{d} \circ \mathsf{b}_{\tilde{\boldsymbol{x}}})(\boldsymbol{\delta}_k)}_{\text{descent direction}}. \tag{D.5}$$

As an analytic calculation of the gradient is not possible, the gradient needs to be approximated numerically. To this end, the attack slightly changes $\boldsymbol{\delta}_k$ at one dimension, maps the vector to the boundary again via $\mathsf{b}_{\tilde{\boldsymbol{x}}}$ and records the distance through this change. By repeating this procedure for each dimension in the vector space, the attack is able to calculate the gradient. More formally, the i-th dimension of the gradient is calculated as:

$$[\nabla(\mathsf{d} \circ \mathsf{b}_{\tilde{\boldsymbol{x}}})(\boldsymbol{\delta}_k)]_i = \frac{(\mathsf{d} \circ \mathsf{b}_{\tilde{\boldsymbol{x}}})(\boldsymbol{\delta}_k + \lambda \boldsymbol{e}_i) - (\mathsf{d} \circ \mathsf{b}_{\tilde{\boldsymbol{x}}})(\boldsymbol{\delta}_k)}{\lambda}. \tag{D.6}$$

$\boldsymbol{e}_i$ is here the i-th vector of the canonical basis, and λ a small positive number. Figure D.1 exemplifies this calculation in the 2-dimensional case.

3. Then, either a new iteration starts in the current position, or the attack terminates based on a certain criterion. In Figure D.1, $\boldsymbol{\delta}_{k+1}$ already shows the shortest way out of the subspace of y^+. It only needs to be scaled to the decision boundary to get the minimal modifications, so that the watermark in $\tilde{\boldsymbol{x}}$ is not detected anymore.

In summary, the attack does not require a priori knowledge about the detector's decision function and works only with a binary output. The optimal solution is guaranteed for convex boundaries, but suitable results are also reported for non-linear watermarking schemes—with e.g. polynomial or fractalized decision boundaries—by following the boundary's envelope [57, 58]. Moreover, multiple adjustments are discussed to decrease the required queries and computation time with a gradient and Newton's method [see 57]. Finally, note that Comesaña and Pérez-González [58] present a pseudocode for the attack.

D.2 KALKER'S ATTACK

Kalker [108] introduces an oracle attack against a normalized correlation detector with a bipolar watermark. With slight modifications, this attack is applicable to a larger class of watermarking schemes [60], such as the presented scheme in Section 5.1.1.

The idea is to calculate the normal vector $\boldsymbol{m}$ to the decision boundary in order to find the shortest path out of the subspace of y^+. Figure D.2 exemplifies this attack which can be divided into three phases [60, 108]:

1. The starting position $\boldsymbol{x}_b$ is any random signal on or near the decision boundary. It can be calculated as for the BNSA.

2. The attacker creates a zero-mean random vector $\boldsymbol{u}_i$ with components $\{-k, +k\}$. By adding $\boldsymbol{u}_i$ to $\boldsymbol{x}_b$, the resulting vector ends either in the y^+ or y^- subspace. If the detector returns the watermark's presence, we assume a positive correlation between $\boldsymbol{u}_i$ and the normal vector $\boldsymbol{m}$ and set a scalar $v_i = 1$. If the detector returns the watermark's absence, $\boldsymbol{u}_i$ points to the opposite direction of $\boldsymbol{m}$ (negative correlation) and we set $v_i = -1$. The attacker repeats this process to generate a certain number of such random vectors. As each vector has only two possible component values,

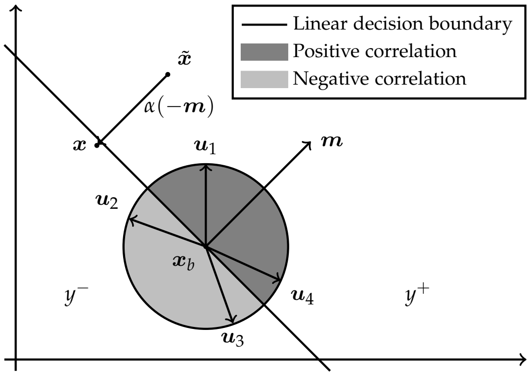


Figure D.2: Geometric view of Kalker's attack [108].

all random vectors have equal length. Effectively, we create a circle of vectors around the starting position $\boldsymbol{x}_b$.

3. By averaging over all random vectors, the attacker can estimate the normal vector $\boldsymbol{m}$:

$$\boldsymbol{m} = \sum_i \nu_i \cdot \boldsymbol{u}_i \, . \tag{D.7}$$

Due to the scalar ν_i, each random vector lies on the side with the positive correlation. By averaging over all vectors, the normal vector as middle can be computed.

In the case of a linear decision boundary, $(-\boldsymbol{m})$ represents the direction of the shortest path towards y^-. Thus, we subtract $\boldsymbol{m}$ with a scaling factor α from $\tilde{\boldsymbol{x}}$ such that we obtain a position on the boundary. In doing so, we obtain an unwatermarked signal with minimal distortion. In other words, $\alpha(-\boldsymbol{m})$ represents the optimal modification $\boldsymbol{\delta}$ in Equation 5.8.

Originally, Kalker [108] interprets $\boldsymbol{m}$ as watermark vector. To estimate a bipolar watermark, $\boldsymbol{m}$ is scaled such that its quantized components are in the target set $\{-1, 0, 1\}$. However, using the normal vector interpretation, this attack also deals with other watermarking schemes, for which the normal vector provides information about the shortest path towards y^- [60].

Furthermore, this attack can circumvent defenses that build on a randomized region around the boundary (see Section 5.2.5), since it can create a vector circle whose largest part is outside of the randomized region.

D.3 SECURITY MARGIN CONSTRUCTION

The construction of the security margin works as follows: First, we choose a tree region and select the training data that fall inside this particular region. Next, we estimate the distribution of the selected training data at each dimension through a kernel-density estimation. In this way, no a priori assumptions about their distribution are required. Finally, the distribution in each dimension is used to define the margin at the boundary in this dimension. To this end, we set the margin to the feature value where the probability of occurrence is smaller than a certain threshold. In Figure 5.6, for example, the lower-right tree region has a smaller security margin, since more training data are near the boundary. On the contrary, the region on the very left exhibits fewer training samples near the boundary, so that a larger margin can be defined. By defining the security margin in this statistical way, we can control the false positive rate that an honest query falls inside the margin. We repeat the process for each tree region.

BIBLIOGRAPHY

[1] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin. *Learning from Data: A Short Course*. AMLBook, 2012.

[2] M. Abuhamad, T. AbuHmed, A. Mohaisen, and D. Nyang. "Large-Scale and Language-Oblivious Code Authorship Identification." In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2018.

[3] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet. "Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring." In: *Proc. of USENIX Security Symposium*. 2018.

[4] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley, 2006.

[5] S. Alrabaee, P. Shirani, L. Wang, M. Debbabi, and A. Hanna. "On Leveraging Coding Habits for Effective Binary Authorship Attribution." In: *Proc. of European Symposium on Research in Computer Security (ESORICS)*. 2018.

[6] B. Alsulami, E. Dauber, R. E. Harang, S. Mancoridis, and R. Greenstadt. "Source Code Authorship Attribution Using Long Short-Term Memory Based Networks." In: *Proc. of European Symposium on Research in Computer Security (ESORICS)*. 2017.

[7] M. Alzantot, Y. Sharma, A. Elgohary, B.-J. Ho, M. Srivastava, and K.-W. Chang. "Generating Natural Language Adversarial Examples." In: *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2018.

[8] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth. *Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning*. arXiv:1801.08917. 2018.

[9] D. Arp. "Efficient and Explainable Detection of Mobile Malware with Machine Learning." PhD thesis. TU Braunschweig, 2019.

[10] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck. "Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket." In: *Proc. of Network and Distributed System Security Symposium (NDSS)*. 2014.

[11] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck. "Dos and Don'ts of Machine Learning in Computer Security." In: *Proc. of USENIX Security Symposium*. 2022 (to appear).

[12] A. Athalye, N. Carlini, and D. Wagner. "Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples." In: *Proc. of Int. Conference on Machine Learning (ICML)*. 2018.

[13] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok. "Synthesizing Robust Adversarial Examples." In: *Proc. of Int. Conference on Machine Learning (ICML)*. 2018.

[14] Z. Ba, S. Piao, X. Fu, D. Koutsonikolas, A. Mohaisen, and K. Ren. "ABC: Enabling Smartphone Authentication with Built-in Camera." In: *Proc. of Network and Distributed System Security Symposium (NDSS)*. 2018.

[15] D. Bahdanau, K. Cho, and Y. Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." In: *International Conference on Learning Representations (ICLR)*. 2015.

[16] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. "Automated Classification and Analysis of Internet Malware." In: *Proc. of International Symposium on Recent Advances in Intrusion Detection (RAID)*. 2007.

[17] M. Barni, P. Comesaña-Alfaro, F. Pérez-González, and B. Tondi. "Are you threatening me?: Towards smart detectors in watermarking." In: *Proceedings of SPIE* 9028 (2014).

[18] M. Barni and F. Pérez-González. "Coping with the enemy: Advances in adversary-aware signal processing." In: *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. 2013.

[19] P. Bas and A. Westfeld. "Two Key Estimation Techniques for the Broken Arrows Watermarking Scheme." In: *Proc. of the Workshop on Multimedia and Security (MM&Sec)*. 2009.

[20] A. Bendale and T. Boult. "Towards Open Set Deep Networks." In: *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[21] B. Biggio and F. Roli. "Wild patterns: Ten years after the rise of adversarial machine learning." In: *Pattern Recognition* 84 (2018).

[22] B. Biggio, G. Fumera, and F. Roli. "Adversarial pattern classification using multiple classifiers and randomisation." In: *Structural, Syntactic, and Statistical Pattern Recognition*. Springer, 2008.

[23] B. Biggio, G. Fumera, and F. Roli. "Multiple classifier systems for robust classifier design in adversarial environments." In: *International Journal of Machine Learning and Cybernetics* 1.1 (2010).

[24] B. Biggio, B. Nelson, and P. Laskov. "Support Vector Machines Under Adversarial Label Noise." In: *Proc. of Asian Conference on Machine Learning (ACML)*. 2011.

[25] B. Biggio, B. Nelson, and P. Laskov. "Poisoning Attacks against Support Vector Machines." In: *Proc. of Int. Conference on Machine Learning (ICML)*. 2012.

[26] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. "Evasion Attacks against Machine Learning at Test Time." In: *Machine Learning and Knowledge Discovery in Databases*. Springer, 2013.

[27] B. Biggio, G. Fumera, and F. Roli. "Security Evaluation of Pattern Classifiers under Attack." In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 26.4 (2014).

[28] B. Biggio, I. Corona, Z. He, P. P. K. Chan, G. Giacinto, D. S. Yeung, and F. Roli. "One-and-a-Half-Class Multiple Classifier Systems for Secure Learning Against Evasion Attacks at Test Time." In: *Proc. of International Workshop on Multiple Classifier Systems (MCS)*. 2015.

[29] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

[30] B. E. Boser, I. M. Guyon, and V. N. Vapnik. "A Training Algorithm for Optimal Margin Classifiers." In: *Proc. of the Annual ACM Workshop on Computational Learning Theory (COLT)*. 1992.

[31] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2009.

[32] L. Breiman. "Random Forests." In: *Machine Learning* 45.1 (2001).

[33] W. Brendel, J. Rauber, and M. Bethge. "Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models." In: *International Conference on Learning Representations (ICLR)*. 2018.

[34] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. "A Survey of Monte Carlo Tree Search Methods." In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012).

[35] S. Burrows, A. L. Uitdenbogerd, and A. Turpin. "Application of Information Retrieval Techniques for Source Code Authorship Attribution." In: *Proc. of Conference on Database Systems for Advanced Applications (DASFAA)*. 2009.

[36] A. Caliskan, R. Harang, A. Liu, A. Narayanan, C. R. Voss, F. Yamaguchi, and R. Greenstadt. "De-anonymizing Programmers via Code Stylometry." In: *Proc. of USENIX Security Symposium*. 2015.

[37] A. Caliskan, F. Yamaguchi, E. Tauber, R. Harang, K. Rieck, R. Greenstadt, and A. Narayanan. “When Coding Style Survives Compilation: De-anonymizing Programmers from Executable Binaries.” In: *Proc. of Network and Distributed System Security Symposium (NDSS)*. 2018.

[38] N. Carlini and D. A. Wagner. “Towards Evaluating the Robustness of Neural Networks.” In: *Proc. of IEEE Symposium on Security and Privacy (S&P)*. 2017.

[39] N. Carlini and D. Wagner. “Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods.” In: *Proc. of ACM Workshop on Artificial Intelligence and Security (AISEC)*. 2017.

[40] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. Goodfellow, A. Madry, and A. Kurakin. *On Evaluating Adversarial Robustness*. arXiv:1902.06705. 2019.

[41] V. Chandrasekaran, K. Chaudhuri, I. Giacomelli, S. Jha, and S. Yan. “Exploring Connections Between Active Learning and Model Extraction.” In: *Proc. of USENIX Security Symposium*. 2020.

[42] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-Supervised Learning*. MIT Press, 2006.

[43] L. Charlin and R. S. Zemel. “The Toronto paper matching system: an automated paper-reviewer assignment system.” In: *ICML Workshop on Peer Reviewing and Publishing Models*. 2013.

[44] H. Chen, C. Fu, J. Zhao, and F. Koushanfar. “DeepInspect: A Black-box Trojan Detection and Mitigation Framework for Deep Neural Networks.” In: *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*. 2019.

[45] J. Chen, M. I. Jordan, and M. J. Wainwright. “HopSkipJumpAttack: A Query-Efficient Decision-Based Attack.” In: *Proc. of IEEE Symposium on Security and Privacy (S&P)*. 2020.

[46] S. Chen, N. Carlini, and D. Wagner. “Stateful Detection of Black-Box Adversarial Attacks.” In: *Proc. of the ACM Workshop on Security and Privacy on Artificial Intelligence (SPAI)*. 2020.

[47] T. Chen and C. Guestrin. “XGBoost: A Scalable Tree Boosting System.” In: *Proc. of the ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)*. 2016.

[48] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren. “Android HIV: A Study of Repackaging Malware for Evading Machine-Learning Detection.” In: *IEEE Transactions on Information Forensics and Security (TIFS)* 15 (2020).

[49] Y. Chen, S. Wang, D. She, and S. Jana. "On Training Robust PDF Malware Classifiers." In: *Proc. of USENIX Security Symposium*. 2020.

[50] Y. Chen, C. Shen, C. Wang, Q. Xiao, K. Li, and Y. Chen. "Scaling Camouflage: Content Disguising Attack Against Computer Vision Applications." In: *IEEE Transactions on Dependable and Secure Computing (TDSC)* (2020).

[51] E. Chou, F. Tramèr, and G. Pellegrino. "SentiNet: Detecting Localized Universal Attacks Against Deep Learning Systems." In: *Deep Learning and Security Workshop (DLS)*. 2020.

[52] M. E. Choubassi and P. Moulin. "On the fundamental tradeoff between watermark detection performance and robustness against sensitivity analysis attacks." In: *Proceedings of SPIE* 6072 (2006).

[53] M. E. Choubassi and P. Moulin. "Noniterative Algorithms for Sensitivity Analysis Attacks." In: *IEEE Transactions on Information Forensics and Security (TIFS)* 2.2 (2007).

[54] M. E. Choubassi and P. Moulin. "On Reliability and Security of Randomized Detectors Against Sensitivity Analysis Attacks." In: *IEEE Transactions on Information Forensics and Security (TIFS)* 4.3 (2009).

[55] *Clang: C Language Family Frontend for LLVM*. LLVM Project, `https://clang.llvm.org`. Last visited June 2021. 2021.

[56] M. Collins and N. Duffy. "Convolution Kernels for Natural Language." In: *Advances in Neural Information Proccessing Systems (NIPS)*. Vol. 14. 2002.

[57] P. Comesaña, L. Pérez-Freire, and F. Pérez-González. "Blind Newton sensitivity attack." In: *IEE Proceedings – Information Security* 153.3 (2006).

[58] P. Comesaña and F. Pérez-González. "Breaking the BOWS Watermarking System: Key Guessing and Sensitivity Attacks." In: *EURASIP Journal on Information Security* 2007.1 (2007).

[59] I. J. Cox and J.-P.M. G. Linnartz. "Public watermarks and resistance to tampering." In: *IEEE International Conference on Image Processing (ICIP)*. 1997.

[60] I. J. Cox, M. Miller, J. Bloom, J. Fridrich, and T. Kalker. *Digital watermarking and steganography*. Morgan Kaufmann Publishers, 2002.

[61] S. Craver and J. Yu. "Reverse-engineering a detector with false alarms." In: *Proceedings of SPIE* 6505 (2007).

[62] G. F. Cretu, A. Stavrou, M. E. Locasto, S. J. Stolfo, and A. D. Keromytis. “Casting out Demons: Sanitizing Training Data for Anomaly Sensors.” In: *Proc. of IEEE Symposium on Security and Privacy (S&P)*. 2008.

[63] H. Dang, Y. Huang, and E.-C. Chang. “Evading Classifiers by Morphing in the Dark.” In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2017.

[64] E. Dauber, A. Caliskan, R. Harang, G. Shearer, M. Weisman, F. Nelson, and R. Greenstadt. “Git Blame Who?: Stylistic Authorship Attribution of Small, Incomplete Source Code Fragments.” In: *Proceedings on Privacy Enhancing Technologies (PETS)* 2019.3 (2019).

[65] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli. “Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection.” In: *IEEE Transactions on Dependable and Secure Computing (TDSC)* PP.99 (2017).

[66] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. “ImageNet: A large-scale hierarchical image database.” In: *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009.

[67] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. Second. John Wiley & Sons, 2000.

[68] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou. “HotFlip: White-Box Adversarial Examples for Text Classification.” In: *Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL), (Short Papers)*. 2018.

[69] A. Fass, M. Backes, and B. Stock. “HideNoSeek: Camouflaging Malicious JavaScript in Benign ASTs.” In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2019.

[70] P. Fogla and W. Lee. “Evading Network Anomaly Detection Systems: Formal Reasoning and Practical Techniques.” In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2006.

[71] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. “Polymorphic Blending Attacks.” In: *Proc. of USENIX Security Symposium*. 2006.

[72] G. Forman. “A Pitfall and Solution in Multi-class Feature Selection for Text Classification.” In: *Proc. of Int. Conference on Machine Learning (ICML)*. 2004.

[73] G. Frantzeskou, E. Stamatatos, S. Gritzalis, and S. Katsikas. “Effective identification of source code authors using byte-level information.” In: *Proc. of International Conference on Software Engineering (ICSE)*. 2006.

[74] J. Fridrich. *Steganography in Digital Media: Principles, Algorithms, and Applications*. Cambridge University Press, 2010.

[75] J. Fridrich. “Sensor Defects in Digital Image Forensic.” In: *Digital Image Forensics: There is More to a Picture Than Meets the Eye*. Ed. by H. T. Sencar and N. Memon. Springer, 2013, pp. 179–218.

[76] T. Furon and P. Bas. “Broken arrows.” In: *EURASIP Journal on Information Security* 2008 (2008).

[77] T. Furon, I. Venturini, and P. Duhamel. “Unified approach of asymmetric watermarking schemes.” In: *Proceedings of SPIE* 4314 (2001).

[78] T. Furon, B. Macq, N. Hurley, and G. Silvestre. “JANIS: Just another N-order side-informed watermarking scheme.” In: *IEEE International Conference on Image Processing (ICIP)*. 2002.

[79] J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi. “Black-Box Generation of Adversarial Text Sequences to Evade Deep Learning Classifiers.” In: *IEEE Security and Privacy Workshops (SPW)*. 2018.

[80] S. Garg and G. Ramakrishnan. “BAE: BERT-based Adversarial Examples for Text Classification.” In: *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2020.

[81] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck. “Structural Detection of Android Malware Using Embedded Call Graphs.” In: *Proc. of ACM Workshop on Artificial Intelligence and Security (AISEC)*. 2013.

[82] M. Gendreau and J.-Y. Potvin. *Handbook of Metaheuristics, 3rd ed.* Springer International Publishing, 2019.

[83] A. Globerson and S. Roweis. “Nightmare at Test Time: Robust Learning by Feature Deletion.” In: *Proc. of Int. Conference on Machine Learning (ICML)*. 2006.

[84] T. Gloe and R. Böhme. “The Dresden Image Database for Benchmarking Digital Image Forensics.” In: *Journal of Digital Forensic Practice* 3.2–4 (2010).

[85] T. Gloe, M. Kirchner, A. Winkler, and R. Böhme. “Can we Trust Digital Image Forensics?” In: *International Conference on Multimedia*. 2007.

[86] Y. Goldberg. *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers, 2017.

[87] M. Goldblum, D. Tsipras, C. Xie, X. Chen, A. Schwarzschild, D. Song, A. Madry, B. Li, and T. Goldstein. *Dataset Security for Machine Learning: Data Poisoning, Backdoor Attacks, and Defenses*. arXiv:2012.10544v2. 2020.

[88] I. J. Goodfellow, J. Shlens, and C. Szegedy. "Explaining and harnessing adversarial examples." In: *International Conference on Learning Representations (ICLR)*. 2015.

[89] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[90] *Google Code Jam*. https://codingcompetitions.withgoogle.com/codejam. Last visited June 2021. 2021.

[91] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel. "Adversarial Examples for Malware Detection." In: *Proc. of European Symposium on Research in Computer Security (ESORICS)*. 2017.

[92] T. Gu, B. Dolan-Gavitt, and S. Garg. *BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain*. arXiv: 1708.06733. 2017.

[93] W. Guo, L. Wang, Y. Xu, X. Xing, M. Du, and D. Song. "Towards Inspecting and Eliminating Trojan Backdoors in Deep Neural Networks." In: *Proc. of the International Conference on Data Mining (ICDM)*. 2020.

[94] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer series in statistics. New York, N.Y.: Springer, 2001.

[95] D. Haussler. *Convolution Kernels on Discrete Structures*. Tech. rep. UCSC-CRL-99-10. UC Santa Cruz, 1999.

[96] S. Hido and H. Kashima. "A Linear-Time Graph Kernel." In: *Proc. of the International Conference on Data Mining (ICDM)*. 2009.

[97] G. Hinton et al. "Deep Neural Networks for Acoustic Modeling in Speech Recognition." In: *IEEE Signal Processing Magazine* 29.6 (2012).

[98] L. Huang, A. D. Joseph, B. Nelson, B. I. P. Rubinstein, and J. D. Tygar. "Adversarial Machine Learning." In: *Proc. of ACM Workshop on Artificial Intelligence and Security (AISEC)*. 2011.

[99] B. Hui, Y. Yang, H. Yuan, P. Burlina, N. Z. Gong, and Y. Cao. "Practical Blind Membership Inference Attack via Differential Comparisons." In: *Proc. of Network and Distributed System Security Symposium (NDSS)*. 2021.

[100] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin. "Black-box Adversarial Attacks with Limited Queries and Information." In: *Proc. of Int. Conference on Machine Learning (ICML)*. 2018.

[101] U. Iqbal, P. Snyder, S. Zhu, B. Livshits, Z. Qian, and Z. Shafiq. "AdGraph: A Graph-Based Approach to Ad and Tracker Blocking." In: *Proc. of IEEE Symposium on Security and Privacy (S&P)*. 2020.

[102] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot. "High Accuracy and High Fidelity Extraction of Neural Networks." In: *Proc. of USENIX Security Symposium*. 2020.

[103] S. Jana and V. Shmatikov. "Abusing File Processing in Malware Detectors for Fun and Profit." In: *Proc. of IEEE Symposium on Security and Privacy (S&P)*. 2012.

[104] J. Jia, A. Salem, M. Backes, Y. Zhang, and N. Z. Gong. "Mem-Guard: Defending against Black-Box Membership Inference Attacks via Adversarial Examples." In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2019.

[105] A. D. Joseph, B. Nelson, B. I. P. Rubinstein, and J. D. Tygar. *Adversarial Machine Learning*. Cambridge, UK: Cambridge University Press, 2019.

[106] D. Jurafsky and J. H. Martin. *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., 2009.

[107] M. Juuti, S. Szyller, S. Marchal, and N. Asokan. "PRADA: Protecting Against DNN Model Stealing Attacks." In: *Proc. of IEEE European Symposium on Security and Privacy (EuroS&P)*. 2019.

[108] T. Kalker. "Watermark estimation through detector observations." In: *Proc. of IEEE Benelux Signal Processing Symposium*. 1998.

[109] T. Kalker, J.-P.M. G. Linnartz, and M. van Dijk. "Watermark estimation through detector analysis." In: *IEEE International Conference on Image Processing (ICIP)*. 1998.

[110] A. Kantchelian, J. D. Tygar, and A. Joseph. "Evasion and Hardening of Tree Ensemble Classifiers." In: *Proc. of Int. Conference on Machine Learning (ICML)*. 2016.

[111] M. Kloft and P. Laskov. "Online Anomaly Detection under Adversarial Impact." In: *JMLR Workshop and Conference Proceedings, Volume 9: AISTATS*. 2010.

[112] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli. "Adversarial Malware Binaries: Evading Deep Learning for Malware Detection in Executables." In: *European Signal Processing Conference (EUSIPCO)*. 2018.

[113] A. Krizhevsky and G. Hinton. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[114] I. Krsul and E. H. Spafford. "Authorship analysis: identifying the author of a program." In: *Computers & Security* 16.3 (1997).

[115] A. Kurakin, I. J. Goodfellow, and S. Bengio. "Adversarial examples in the physical world." In: *International Conference on Learning Representations (ICLR) (Workshop Track)*. 2017.

[116] A. Kurtz, H. Gascon, T. Becker, K. Rieck, and F. Freiling. "Fingerprinting Mobile Devices Using Personalized Configurations." In: *Proceedings on Privacy Enhancing Technologies (PETS)* 2016.1 (2016).

[117] M. Lapin, M. Hein, and B. Schiele. "Top-k Multiclass SVM." In: *Advances in Neural Information Proccessing Systems (NIPS)*. 2015.

[118] E. Le Merrer, P. Pérez, and G. Trédan. "Adversarial frontier stitching for remote neural network watermarking." In: *Neural Computing and Applications* 32.13 (2020).

[119] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. "Handwritten Digit Recognition with a Back-Propagation Network." In: *Advances in Neural Information Proccessing Systems (NIPS)*. 1990.

[120] Y. LeCun, Y. Bengio, and G. Hinton. "Deep Learning." In: *Nature* 521.7553 (2015).

[121] T. Lee, B. Edwards, I. Molloy, and D. Su. "Defending Against Neural Network Model Stealing Attacks Using Deceptive Perturbations." In: *Deep Learning and Security Workshop (DLS)*. 2019.

[122] J. Levinson et al. "Towards fully autonomous driving: Systems and algorithms." In: *Proc. of IEEE Intelligent Vehicles Symposium (IV)*. 2011.

[123] J. Li, N. Li, and B. Ribeiro. "Membership Inference Attacks and Defenses in Classification Models." In: *Proc. of ACM Conference on Data and Applications Security and Privacy (CODASPY)*. 2021.

[124] J. Li, S. Ji, T. Du, B. Li, and T. Wang. "TextBugger: Generating Adversarial Text Against Real-world Applications." In: *Proc. of Network and Distributed System Security Symposium (NDSS)*. 2019.

[125] J.-P.M. G. Linnartz and M. van Dijk. "Analysis of the sensitivity attack against electronic watermarks in images." In: *Proc. of Information Hiding Conference*. Vol. 1525. 1998.

[126] K. Liu, B. Dolan-Gavitt, and S. Garg. "Fine-Pruning: Defending Against Backdooring Attacks on Deep Neural Networks." In: *Proc. of Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*. 2018.

[127] Y. Liu, Y. Xie, and A. Srivastava. "Neural Trojans." In: *IEEE International Conference on Computer Design (ICCD)*. 2017.

[128] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang. "Trojaning Attack on Neural Networks." In: *Proc. of Network and Distributed System Security Symposium (NDSS)*. 2018.

[129] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. "Text classification using string kernels." In: *Journal of Machine Learning Research (JMLR)* 2 (2002).

[130] J. B. Lovins. "Development of a stemming algorithm." In: *Mechanical Translation and Computational Linguistics* 11.1-2 (1968).

[131] D. Lowd and C. Meek. "Good Word Attacks on Statistical Spam Filters." In: *Conference on Email and Anti-Spam (CEAS)*. 2005.

[132] D. Lowd and C. Meek. "Adversarial Learning." In: *Proc. of the ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)*. 2005.

[133] D. G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints." In: *International Journal of Computer Vision* 60.2 (2004).

[134] J. Lukáš, J. Fridrich, and M. Goljan. "Digital Camera Identification from Sensor Pattern Noise." In: *IEEE Transactions on Information Forensics and Security (TIFS)* 1.2 (2006).

[135] S. MacDonell, A. Gray, G. MacLennan, and P. Sallis. "Software forensics for discriminating between program authors using case-based reasoning, feed-forward neural networks and multiple discriminant analysis." In: *Proc. of International Conference on Neural Information Processing (ICONIP)*. 1999.

[136] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. "Towards Deep Learning Models Resistant to Adversarial Attacks." In: *International Conference on Learning Representations (ICLR)*. 2017.

[137] D. Maiorca, I. Corona, and G. Giacinto. "Looking at the Bag is Not Enough to Find the Bomb: An Evasion of Structural Methods for Malicious PDF Files Detection." In: *Proc. of ACM Asia Conference on Computer and Communications Security (ASIA CCS)*. 2013.

[138] M. F. Mansour and A. H. Tewfik. "Improving the security of watermark public detectors." In: *Proc. of International Conference on Digital Signal Processing (DSP)*. 2002.

[139] M. F. Mansour and A. H. Tewfik. "LMS-based attack on watermark public detectors." In: *IEEE International Conference on Image Processing (ICIP)*. 2002.

[140] E. Mariconti, L. Onwuzurike, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini. "MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models." In: *Proc. of Network and Distributed System Security Symposium (NDSS)*. 2017.

[141] A. Matyukhina, N. Stakhanova, M. Dalla Preda, and C. Perley. "Adversarial Authorship Attribution in Open-Source Projects." In: *Proc. of ACM Conference on Data and Applications Security and Privacy (CODASPY)*. 2019.

[142] D. Meng and H. Chen. “MagNet: A Two-Pronged Defense against Adversarial Examples.” In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2017.

[143] X. Meng, B. P. Miller, and K.-S. Jun. “Identifying Multiple Authors in a Binary Program.” In: *Proc. of European Symposium on Research in Computer Security (ESORICS)*. 2017.

[144] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff. “On Detecting Adversarial Perturbations.” In: *International Conference on Learning Representations (ICLR)*. 2017.

[145] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. “Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection.” In: *Proc. of Network and Distributed System Security Symposium (NDSS)*. 2018.

[146] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. “DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks.” In: *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[147] A. Moser, C. Kruegel, and E. Kirda. “Limits of static analysis for malware detection.” In: *Proc. of Annual Computer Security Applications Conference (ACSAC)*. 2007.

[148] B. Nelson, B. I. P. Rubinstein, L. Huang, A. D. Joseph, S. J. Lee, S. Rao, and J. D. Tygar. “Query Strategies for Evading Convex-Inducing Classifiers.” In: *Journal of Machine Learning Research (JMLR)* 13 (2012).

[149] J. Newsome, B. Karp, and D. Song. “Paragraph: Thwarting Signature Learning by Training Maliciously.” In: *Proc. of International Symposium on Recent Advances in Intrusion Detection (RAID)*. 2006.

[150] S. J. Oh, M. Augustin, M. Fritz, and B. Schiele. “Towards Reverse-Engineering Black-Box Neural Networks.” In: *International Conference on Learning Representations (ICLR)*. 2018.

[151] A. V. Oppenheim, J. R. Buck, and R. W. Schafer. *Discrete-Time Signal Processing; 2nd ed.* Prentice-Hall, 1999.

[152] T. Orekondy, B. Schiele, and M. Fritz. “Knockoff Nets: Stealing Functionality of Black-Box Models.” In: *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[153] T. Orekondy, B. Schiele, and M. Fritz. “Prediction Poisoning: Towards Defenses Against DNN Model Stealing Attacks.” In: *International Conference on Learning Representations (ICLR)*. 2020.

[154] N. Papernot, P. McDaniel, A. Swami, and R. Harang. “Crafting adversarial input sequences for recurrent neural networks.” In: *IEEE Military Communications Conference (MILCOM)*. 2016.

[155] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami. "Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks." In: *Proc. of IEEE Symposium on Security and Privacy (S&P)*. 2016.

[156] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. "The Limitations of Deep Learning in Adversarial Settings." In: *Proc. of IEEE European Symposium on Security and Privacy (EuroS&P)*. 2016.

[157] N. Papernot, P. McDaniel, and I. Goodfellow. *Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples*. arXiv:1605.07277. 2016.

[158] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. Berkay Celik, and A. Swami. "Practical Black-Box Attacks against Machine Learning." In: *Proc. of ACM Asia Conference on Computer and Communications Security (ASIA CCS)*. 2017.

[159] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman. "SoK: Security and Privacy in Machine Learning." In: *Proc. of IEEE European Symposium on Security and Privacy (EuroS&P)*. 2018.

[160] B. N. Pellin. *Using Classification Techniques to Determine Source Code Authorship*. Tech. rep. Department of Computer Science, University of Wisconsin, 2000.

[161] R. Perdisci, G. Gu, and W. Lee. "Using an Ensemble of One-Class SVM Classifiers to Harden Payload-based Anomaly Detection Systems." In: *Proc. of the International Conference on Data Mining (ICDM)*. 2006.

[162] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro. "Intriguing Properties of Adversarial ML Attacks in the Problem Space." In: *Proc. of IEEE Symposium on Security and Privacy (S&P)*. 2020.

[163] Python Software Foundation. *difflib – Helpers for computing deltas*. `https://docs.python.org/3/library/difflib.html`. Last visited June 2021.

[164] E. Quiring and M. Kirchner. "Fragile Sensor Fingerprint Camera Identification." In: *IEEE International Workshop on Information Forensics and Security (WIFS)*. 2015.

[165] E. Quiring and K. Rieck. "Adversarial Machine Learning Against Digital Watermarking." In: *European Signal Processing Conference (EUSIPCO)*. 2018.

[166] E. Quiring and K. Rieck. "Backdooring and Poisoning Neural Networks with Image-Scaling Attacks." In: *Deep Learning and Security Workshop (DLS)*. 2020.

[167] E. Quiring and P. Schöttle. "On the Combination of Randomized Thresholds and Non-Parametric Boundaries to Protect Digital Watermarks against Sensitivity Attacks." In: *Proc. of the ACM Workshop on Information Hiding and Multimedia Security (IH&MMSec)*. 2014.

[168] E. Quiring, D. Arp, and K. Rieck. "Forgotten Siblings: Unifying Attacks on Machine Learning and Digital Watermarking." In: *Proc. of IEEE European Symposium on Security and Privacy (EuroS&P)*. 2018.

[169] E. Quiring, A. Maier, and K. Rieck. "Misleading Authorship Attribution of Source Code using Adversarial Learning." In: *Proc. of USENIX Security Symposium*. 2019.

[170] E. Quiring, M. Kirchner, and K. Rieck. "On the Security and Applicability of Fragile Camera Fingerprints." In: *Proc. of European Symposium on Research in Computer Security (ESORICS)*. 2019.

[171] E. Quiring, D. Klein, D. Arp, M. Johns, and K. Rieck. "Adversarial Preprocessing: Understanding and Preventing Image-Scaling Attacks in Machine Learning." In: *Proc. of USENIX Security Symposium*. 2020.

[172] E. Quiring, L. Pirch, M. Reimsbach, D. Arp, and K. Rieck. *Against All Odds: Winning the Defense Challenge in an Evasion Competition with Diversification*. arXiv:2010.09569. 2020.

[173] K. Rieck. "Machine Learning for Application-Layer Intrusion Detection." PhD thesis. Berlin Institute of Technology (TU Berlin), 2009.

[174] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov. "Learning and Classification of Malware Behavior." In: *Proc. of Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*. 2008.

[175] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici. "Generic Black-Box End-to-End Attack Against State of the Art API Call Based Malware Classifiers." In: *Proc. of Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*. 2018.

[176] I. Rosenberg, A. Shabtai, Y. Elovici, and L. Rokach. "Query-Efficient Black-Box Attack Against Sequence-Based Malware Classifiers." In: *Proc. of Annual Computer Security Applications Conference (ACSAC)*. 2020.

[177] N. E. Rosenblum, X. Zhu, and B. P. Miller. "Who Wrote This Code? Identifying the Authors of Program Binaries." In: *Proc. of European Symposium on Research in Computer Security (ESORICS)*. 2011.

[178] O. Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge." In: *International Journal of Computer Vision (IJCV)* 115.3 (2015).

[179] P. Russu, A. Demontis, B. Biggio, G. Fumera, and F. Roli. "Secure Kernel Machines Against Evasion Attacks." In: *Proc. of ACM Workshop on Artificial Intelligence and Security (AISEC)*. 2016.

[180] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes. "ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models." In: *Proc. of Network and Distributed System Security Symposium (NDSS)*. 2019.

[181] S. Sardy, P. Tseng, and A. G. Bruce. "Robust Wavelet Denoising." In: *IEEE Transactions on Signal Processing* 49 (2001).

[182] B. Schölkopf and A. J. Smola. *Learning with Kernels*. Cambridge, MA: MIT Press, 2002.

[183] P. Schöttle, A. Schlögl, C. Pasquini, and R. Böhme. "Detecting Adversarial Examples - a Lesson from Multimedia Security." In: *European Signal Processing Conference (EUSIPCO)*. 2018.

[184] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. "Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks." In: *Advances in Neural Information Proccessing Systems (NIPS)*. 2018.

[185] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter. "Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition." In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2016.

[186] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. "Membership Inference Attacks against Machine Learning Models." In: *Proc. of IEEE Symposium on Security and Privacy (S&P)*. 2017.

[187] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search." In: *Nature* 529 (2016).

[188] P. Simard, D. Steinkraus, and J. Platt. "Best practices for convolutional neural networks applied to visual document analysis." In: *Proc. of International Conference on Document Analysis and Recognition*. 2003.

[189] L. Simko, L. Zettlemoyer, and T. Kohno. "Recognizing and Imitating Programmer Style: Adversaries in Program Authorship Attribution." In: *Proceedings on Privacy Enhancing Technologies (PETS)* 2018.1 (2018).

[190] K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv:1409.1556. 2014.

[191] M. Sipser. *Introduction to the Theory of Computation, 2nd ed.* Thomson Course Technology, 2006.

[192] S. W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997.

[193] R. Sommer and V. Paxson. "Outside the Closed World: On Using Machine Learning For Network Intrusion Detection." In: *Proc. of IEEE Symposium on Security and Privacy (S&P)*. 2010.

[194] C. Song and V. Shmatikov. "Auditing Data Provenance in Text-Generation Models." In: *Proc. of the ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)*. 2019.

[195] Y. Song, M. E. Locasto, A. Stavrou, and S. J. Stolfo. "On the infeasibility of modeling polymorphic shellcode." In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2007.

[196] J. Stanley. *Machine Learning Security Evasion Competition 2020 Invites Researchers to Defend and Attack*. `https://msrc-blog.microsoft.com/2020/06/01/machine-learning-security-evasion-competition-2020-invites-researchers-to-defend-and-attack/`. Last visited June 2021. 2020.

[197] O. Suciu, S. E. Coull, and J. Johns. "Exploring Adversarial Examples in Malware Detection." In: *IEEE Security and Privacy Workshops (SPW)*. 2019.

[198] C. Sun, C. Tang, X. Zhu, X. Li, and L. Wang. "An efficient method for salt-and-pepper noise removal based on shearlet transform and noise detection." In: *AEUE - International Journal of Electronics and Communications* 69.12 (2015).

[199] I. Sutskever, O. Vinyals, and Q. V. Le. "Sequence to Sequence Learning with Neural Networks." In: *Advances in Neural Information Proccessing Systems (NIPS)*. 2014.

[200] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. *Intriguing properties of neural networks*. arXiv:1312.6199. 2013.

[201] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the Inception Architecture for Computer Vision." In: *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[202] S. Szyller, B. G. Atli, S. Marchal, and N. Asokan. *DAWN: Dynamic Adversarial Watermarking of Neural Networks*. arXiv: 1906.00830. 2019.

[203] R. Tang, M. Du, N. Liu, F. Yang, and X. Hu. "An Embarrassingly Simple Approach for Trojan Attack in Deep Neural Networks." In: *Proc. of the ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)*. 2020.

[204] B. Tondi, P. Comesaña-Alfaro, F. Pérez-González, and M. Barni. "On the effectiveness of meta-detection for countering oracle attacks in watermarking." In: *IEEE International Workshop on Information Forensics and Security (WIFS)*. 2015.

[205] B. Tondi, P. Comesaña-Alfaro, F. Pérez-González, and M. Barni. "Smart Detection of Line-Search Oracle Attacks." In: *IEEE Transactions on Information Forensics and Security (TIFS)* 12.3 (2017).

[206] L. Tong, B. Li, C. Hajaj, C. Xiao, N. Zhang, and Y. Vorobeychik. "Improving Robustness of ML Classifiers against Realizable Evasion Attacks Using Conserved Features." In: *Proc. of USENIX Security Symposium*. 2019.

[207] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. "Stealing Machine Learning Models via Prediction APIs." In: *Proc. of USENIX Security Symposium*. 2016.

[208] F. Tramèr, N. Carlini, W. Brendel, and A. Madry. "On Adaptive Attacks to Adversarial Example Defenses." In: *Advances in Neural Information Proccessing Systems (NeurIPS)*. 2020.

[209] D. Tsipras, S. Santurkar, L. Engstrom, A. Ilyas, and A. Madry. *From ImageNet to Image Classification: Contextualizing Progress on Benchmarks*. arXiv:2005.11295. 2020.

[210] A. M. Turing. "I.—Computing Machinery and Intelligence." In: *Mind* LIX.236 (1950).

[211] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh. "Embedding Watermarks into Deep Neural Networks." In: *Proc. of the ACM on International Conference on Multimedia Retrieval (ICMR)*. 2017.

[212] D. Valsesia, G. Coluccia, T. Bianchi, and E. Magli. "User Authentication via PRNU-Based Physical Unclonable Functions." In: *IEEE Transactions on Information Forensics and Security (TIFS)* 12.8 (2017).

[213] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data [in Russian]*. (English translation: Springer Verlag, New York, 1982). Moscow: Nauka, 1979.

[214] R. Venkatesan and M. H. Jakubowski. "Randomized detection for spread-spectrum watermarking: Defending against sensitivity and other attacks." In: *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. Vol. 2. 2005.

[215] I. Venturini. "Oracle attacks and covert channels." In: *Proc. of International Workshop on Digital Watermarking*. Vol. 3710. 2005.

[216] D. Wagner and P. Soto. "Mimicry attacks on host based intrusion detection systems." In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2002.

[217] B. Wang and N. Z. Gong. "Stealing Hyperparameters in Machine Learning." In: *Proc. of IEEE Symposium on Security and Privacy (S&P)*. 2018.

[218] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. "Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks." In: *Proc. of IEEE Symposium on Security and Privacy (S&P)*. 2019.

[219] G. Wang, T. Wang, H. Zheng, and B. Y. Zhao. "Man vs. Machine: Practical Adversarial Detection of Malicious Crowdsourcing Workers." In: *Proc. of USENIX Security Symposium*. 2014.

[220] K. Wang, J. J. Parekh, and S. J. Stolfo. "Anagram: A Content Anomaly Detector Resistant To Mimicry Attack." In: *Proc. of International Symposium on Recent Advances in Intrusion Detection (RAID)*. 2006.

[221] A. Westfeld. "A Workbench for the BOWS Contest." In: *EURASIP Journal on Information Security* 2007.1 (2008).

[222] A. Westfeld. "Fast Determination of Sensitivity in the Presence of Countermeasures in BOWS-2." In: *International Workshop on Information Hiding*. 2009.

[223] M. Wicker, X. Huang, and M. Kwiatkowska. "Feature-Guided Black-Box Safety Testing of Deep Neural Networks." In: *Tools and Algorithms for the Construction and Analysis of Systems*. 2018.

[224] D. H. Wolpert. "The Lack of a Priori Distinctions between Learning Algorithms." In: *Neural Computation* (1996).

[225] Q. Xiao, Y. Chen, C. Shen, Y. Chen, and K. Li. "Seeing is Not Believing: Camouflage Attacks on Image Scaling Algorithms." In: *Proc. of USENIX Security Symposium*. 2019.

[226] H. Xu, C. Caramanis, and S. Mannor. "Robustness and Regularization of Support Vector Machines." In: *Journal of Machine Learning Research (JMLR)* 10 (2009).

[227] W. Xu, Y. Qi, and D. Evans. "Automatically Evading Classifiers: A Case Study on PDF Malware Classifiers." In: *Proc. of Network and Distributed System Security Symposium (NDSS)*. 2016.

[228] W. Xu, D. Evans, and Y. Qi. "Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks." In: *Proc. of Network and Distributed System Security Symposium (NDSS)*. 2018.

[229] W. Yang, D. Kong, T. Xie, and C. A. Gunter. "Malware Detection in Adversarial Settings: Exploiting Feature Evolutions and Confusions in Android Apps." In: *Proc. of Annual Computer Security Applications Conference (ACSAC)*. 2017.

[230] Y. Yao, H. Li, H. Zheng, and B. Y. Zhao. "Latent Backdoor Attacks on Deep Neural Networks." In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2019.

[231] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha. "Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting." In: *IEEE Computer Security Foundations Symposium (CSF)*. 2018.

[232] J. Zhang, Z. Gu, J. Jang, H. Wu, M. P. Stoecklin, H. Huang, and I. Molloy. "Protecting Intellectual Property of Deep Neural Networks with Watermarking." In: *Proc. of ACM Asia Conference on Computer and Communications Security (ASIA CCS)*. 2018.

[233] H. Zheng, Q. Ye, H. Hu, C. Fang, and J. Shi. "BDPL: A Boundary Differentially Private Layer Against Machine Learning Model Extraction Attacks." In: *Proc. of European Symposium on Research in Computer Security (ESORICS)*. 2019.

[234] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li, and S. Hu. "Traffic-Sign Detection and Classification in the Wild." In: *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[235] N. Šrndić and P. Laskov. "Detection of Malicious PDF Files Based on Hierarchical Document Structure." In: *Proc. of Network and Distributed System Security Symposium (NDSS)*. 2013.

[236] N. Šrndić and P. Laskov. "Practical Evasion of a Learning-Based Classifier: A Case Study." In: *Proc. of IEEE Symposium on Security and Privacy (S&P)*. 2014.