**Defending Against Targeted Attacks with Pattern Recognition**

Von der
Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines
**Doktoringenieurs (Dr.-Ing.)**

genehmigte Dissertation

von
Hugo Gascón Polanco
geboren am 25. Februar 1982
in Córdoba, Spanien

| | |
|---|---|
| Eingereicht am: | 17. Dezember 2018 |
| Disputation am: | 2. April 2019 |
| 1. Referent: | Prof. Dr. Konrad Rieck |
| 2. Referent: | Prof. Dr. Felix Freiling |

2019

A Mateo y Pablo.

# Acknowledgements

I am very grateful to my supervisor Prof. Dr. Konrad Rieck and to Prof. Dr. Klaus-Robert Müller, for giving me the chance to come and play. Thanks Konrad for your infinite drive and endless motivation. Leading by example is the best way to lead.

Tammo Krüger, Guido Schwenk and the rest of the IDA team. Thanks for softening the landing. You rock and you know it.

To all the gang, first at the University of Göttingen and later at TU Braunschweig: Daniel Arp, Ansgar Kellner, Alwin Meier, Christian Wressneger, Fabian Yamaguchi and the rest of the SECTUBS team. Time flies when you are having fun. Also when a deadline is approaching.

To Dominik Kühne at TU Berlin, Carmen Scherbaum and Udo Burghardt at the University of Göttingen, and Katja Barkowsky and Frank Rust at TU Braunscheig. Thanks for keeping the engines running smoothly.

To Lukas Rist, Felix Leder, Ryan W. Smith, David Watson, Max Hils, Natalia Stakhanova and rest of the bunch at The Honeynet Project. You are the proof that a highly technical environment can be challenging but also warm and inclusive.

To Andrew Gardner and Walter Bogorad: Life is about detours. Thank you for inviting me to the best one I could dream of. To Reuben Feinman, Aleatha Parker-Wood and the rest of the CAML and SRL teams at Symantec. You can truly make a detour extraordinary. To Lucy and Dave LaPier, for giving me a home very far away from home.

I want to thank the Deutsche Bahn for its double contribution to this work in the forms of a productivity-boosting bahn-bonus area and a mean for regular relief from a purely academic existence.

To Victoria de Miguel, for her primordial roof and Adrian Brox, for his primordial friendship. To my friends in Madrid and everywhere else: you are the place I want to go back to. Special thanks to Juan L. Cantalapiedra for all the *m3ro* evenings and *game-of-life* Macke nights.

To my parents, for their unconditional loving support and eternal patience and to Regina: let's keep the flukes rolling! Mateo, thank you for challenging my time management skills to the extreme and considerably pushing me up towards the level cap. Thanks Franzi, for everything.

Finally, I will always be grateful to Agustin Orfila. This thesis had not been possible without your encouragment. I wish you'd known.

# Zusammenfassung

In unserer heutigen Welt sind alle und alles miteinander vernetzt. Dies bietet mächtigen Angreifern die Möglichkeit, komplexe Verfahren zu entwickeln, die auf spezifische Ziele angepasst sind. Traditionelle Ansätze zur Bekämpfung solcher Angriffe werden damit ineffektiv, was die Entwicklung innovativer Methoden unabdingbar macht.

Die vorliegende Dissertation verfolgt das Ziel, den Sicherheitsanalysten durch eine umfassende Strategie gegen gezielte Angriffe zu unterstützen. Diese Strategie beschäftigt sich mit den hauptsächlichen Herausforderungen in den drei Phasen der Erkennung und Analyse von sowie der Reaktion auf gezielte Angriffe. Der Aufbau dieser Arbeit orientiert sich daher an den genannten drei Phasen. In jedem Kapitel wird ein Problem aufgegriffen und eine entsprechende Lösung vorgeschlagen, die stark auf maschinellem Lernen und Mustererkennung basiert.

Insbesondere schlagen wir einen Ansatz vor, der eine Identifizierung von Spear-Phishing-Emails ermöglicht, ohne ihren Inhalt zu betrachten. Anschliessend stellen wir einen Analyseansatz für Malware Triage vor, der auf der strukturierten Darstellung von Code basiert. Zum Schluss stellen wir MANTIS vor, eine Open-Source-Plattform für Authoring, Verteilung und Sammlung von Threat Intelligence, deren Datenmodell auf einer innovativen konsolidierten Graphen-Darstellung für Threat Intelligence Stardards basiert. Wir evaluieren unsere Ansätze in verschiedenen Experimenten, die ihren potentiellen Nutzen in echten Szenarien beweisen.

Insgesamt bereiten diese Ideen neue Wege für die Forschung zu Abwehrmechanismen und erstreben, das Ungleichgewicht zwischen mächtigen Angreifern und der Gesellschaft zu minimieren.

# Abstract

The speed at which everything and everyone is being connected considerably outstrips the rate at which effective security mechanisms are introduced to protect them. This has created an opportunity for resourceful threat actors which have specialized in conducting low-volume persistent attacks through sophisticated techniques that are tailored to specific valuable targets. Consequently, traditional approaches are rendered ineffective against targeted attacks, creating an acute need for innovative defense mechanisms.

This thesis aims at supporting the security practitioner in bridging this gap by introducing a holistic strategy against targeted attacks that addresses key challenges encountered during the phases of detection, analysis and response. The structure of this thesis is therefore aligned to these three phases, with each one of its central chapters taking on a particular problem and proposing a solution built on a strong foundation on pattern recognition and machine learning.

In particular, we propose a detection approach that, in the absence of additional authentication mechanisms, allows to identify spear-phishing emails without relying on their content. By devising a series of content-agnostic traits, we are able to build characteristic sender profiles and recognize variations from these profiles as spoofing attempts through machine learning classification.

Next, we introduce an analysis approach for malware triage based on the structural characterization of malicious code. We propose two techniques for embedding binary function call graphs that complement each other in terms of explainability and accuracy: an explicit high dimensional mapping inspired by graph kernels and an implicit low dimensional feature space learned through a neural network architecture.

Finally, we introduce MANTIS, an open-source platform for authoring, sharing and collecting threat intelligence, whose data model is based on an innovative unified representation for threat intelligence standards based on attributed graphs. In addition, we devise a similarity algorithm for attributed graphs that enables uncovering relations between threats at different levels of granularity and that, incorporated into our platform, enables MANTIS as an information retrieval system that is capable of retrieving related reports given individual observations from security incidents.

We evaluate our approaches in dedicated experiments that demonstrate their usefulness and potential impact in a real-world setup. For instance, we show how our approach for detection of spear-phishing emails can discriminate thousands of senders, identifying spoofed emails with 90% detection rate and less than 1 false positive in 10,000 emails. Additionally, our approach for structural malware triage enables the analyst to assign an unknown malware sample to its corresponding family with up to 98% accuracy and identify new strains of malware through anomaly detection with more than 75% success at only 1% of known mislabeled samples. Finally, in an evaluation with over 14,000 CyBOX objects, our platform for threat intelligence enables retrieving relevant threat reports with a mean average precision of 80%, given only a single object from an incident, such as a file or an HTTP request. We further illustrate the performance of this analysis in two case studies with the attack campaigns Stuxnet and Regin.

In the present geopolitical landscape of surveillance capitalism, oppressive regimes and democratic institutions with poor accountability, there exist perverse incentives for governments and large corporations to maintain the current state of insecure affairs and keep the door open for targeted threats. Individually, the methods and techniques proposed in this thesis push the boundaries of existing research against targeted attacks by rendering the main entry vector largely ineffective, assisting at better understanding the nature of malicious code and enabling the sharing and correlation of threat data. As a whole, these ideas open new avenues for research on defense mechanisms and represent an attempt to counteract the imbalance between resourceful actors and society at large.

# Table of contents

# List of figures

# List of tables

# Publications

The research presented in this thesis is structured in three blocks that propose solutions for the phases of detection, analysis and response to targeted attacks. Each of these chapters draws on research introduced in the following papers respectively:

> *Reading Between The Lines: Content-Agnostic Detection of Spear-Phishing Emails. Hugo Gascon, Steffen Ulrich, Benjamin Stritter and Konrad Rieck. Proc. of the 21st International Symposium on Research in Attacks, Intrusions and Defenses (RAID)* [54].

> *Structural Detection of Android Malware using Embedded Call Graphs. Hugo Gascon, Fabian Yamaguchi, Daniel Arp, Konrad Rieck. Proc. of the 2013 ACM Workshop on Security and Artificial Intelligence (AISEC)* [56].

> *Mining Attributed Graphs for Threat Intelligence. Hugo Gascon, Bernd Grobauer, Thomas Schreck, Lukas Rist, Daniel Arp and Konrad Rieck. Proc. of the 7th. ACM Conference on Data and Applications Security and Privacy (CODASPY)* [52].

During the completion of this thesis, the expertise developed in tangential problems to the topic of this dissertation has enabled the author to contribute to the fields of mobile security, reverse engineering of network protocols, model-based fuzzing and vulnerability discovery. In particular, specific contributions are discussed in the following publications. Note that while the results of these publications are not part of this dissertation, they are referenced in the text when a relevant link between this work and the ideas proposed in these papers is established.

*Fingerprinting Mobile Devices Using Personalized Configurations. Andreas Kurtz, Hugo Gascon, Tobias Becker, Konrad Rieck and Felix Freiling. Proc. of the 16th Privacy Enhancing Technologies Symposium (PETS)* [88].

*Pulsar: Stateful Black-Box Fuzzing of Proprietary Network Protocols. Hugo Gascon, Christian Wressnegger, Fabian Yamaguchi, Daniel Arp and Konrad Rieck. Proc. of the 11th EAI International Conference on Security and Privacy in Communication Networks (SECURECOMM)* [55].

*Automatic Inference of Search Patterns for Taint-Style Vulnerabilities. Fabian Yamaguchi, Alwin Maier, Hugo Gascon and Konrad Rieck. Proc. of the 36th IEEE Symposium on Security and Privacy (S&P)* [157].

*Continuous Authentication on Mobile Devices by Analysis of Typing Motion Behavior. Hugo Gascon, Sebastian Uellenbeck, Christopher Wolf, and Konrad Rieck. Proc. of the 2014 GI Conference "Sicherheit" (Sicherheit, Schutz und Verläzsslichkeit)* [53].

*Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket. Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon and Konrad Rieck. Proc. of the 2014 Network and Distributed System Security Symposium (NDSS)* [4].

*Chucky: Exposing Missing Checks in Source Code for Vulnerability Discovery. Fabian Yamaguchi, Christian Wressnegger, Hugo Gascon, Konrad Rieck. Proc. of the 20th ACM Conference on Computer and Communications Security (CCS)* [158].

*Learning Stateful Models for Network Honeypots. Tammo Krueger, Hugo Gascon, Nicole Krämer and Konrad Rieck. Proceedings of the 2012 ACM Workshop on Security and Artificial Intelligence (AISEC)* [87].

# 01

# Introduction

The accelerating evolution of technology and our boundless search for connection, entertainment and efficiency have paved the way for technological companies to pervade every aspect of human activity in the first decades of the twenty-first century. The outcome is a hyperconnected world [153] where almost all information is accessible trough networks and computing systems and where every piece of data resulting from interactions between systems and people is digitized, analyzed and stored, regardless of how sensitive. The strong push for digitalization leaves nothing untouched as new services and platforms are made available everyday. Critical infrastructures are plugged online and smartphones stock all traces of a person's behavior in a single device.

At the same time, such a global information-based ecosystem has enabled criminals to carry out their activities with the same mass-reach and efficiency that legit ventures benefit from [see 17, 41]. From the exploratory experiments of the eighties and nineties [117] to the dystopic scenarios of today [159, 160] and led by the opportunity to access a highly profitable market with a low entry barrier, criminals have adopted more specialized roles and steadily improved the sophistication of their tactics. Almost every strain of traditional crime has found its niche in the Internet space. Acquiring technological expertise is thus key for criminals to become effective threat actors and maintaining their competitive advantage over other criminals and, specially, over the security community.

As a consequence, security researchers have helplessly witnessed how attackers have thrived through increasing professionalization and how their motivations to vulnerate networked systems have evolved from pure financial to geopolitical [81]. Despite years of research and improved security mechanisms, the trove of sensitive data held by networked systems makes the potential rewards for persistent attackers even higher and the arms race between attackers and defenders has entered a new phase with the emergence of highly specialized threat actors. Backed by organizations or nation-states, such adversaries invest large resources into attacking much more selective targets with the goal of achieving economic but also political, or strategical gains. Such is the case of attackers involved in industrial espionage [e.g. 41, 103, 30] or the gathering of classified intelligence, sabotage and political repression, activities performed often by some authoritarian regimes [see 113, 102, 33, 66, 101, 40, 109].

Contrary to the logic of mass-oriented criminal markets, such actions are characterized by stealthy operations and performed by skilled groups with plenty of resources at their disposal, making the confident attribution of an attack almost impossible. These groups, often sponsored by nation-states, are organized in specialized subdivisions within a military or intelligence hierarchy and perform structured work in vulnerability research and exploitation, information gathering, maintenance of infrastructure or purely offensive tasks, such as sabotage or support to further extend attacking capabilities [81]. For instance, by stealing signed certificates that allow targeted malware to be installed stealthily.

In this scenario, the paradigmatic asymmetry between attackers and defenders in computer security grows even larger, creating minimal incentives for the dominant actors to disengage from an escalating global conflict that is being kept out of sight from the general public. Furthermore, the high prices paid for zero-day exploits by actors everywhere establish a highly profitable market alternative to responsible disclosure, leaving critical vulnerabilities unpatched and being ultimately most detrimental to civil society.

As a consequence and without diminishing the importance of other forms of organized social, political or economic action, there exists an acute need for technical research into open-source, decentralized and collaborative defensive mechanisms. Solutions that, given the complexity of the problem, must necessarily address the range of issues faced by security analysts from multiple perspectives.

Therefore, in this thesis, we aim at designing a holistic approach against targeted attacks by addressing the challenges encountered in each phase of a comprehensive defense strategy: *detection*, *analysis* and *response*. As we will see, the particularities of targeted attacks defy traditional defenses and ask for innovative approaches that can benefit from the large amount of data generated by systems and their interactions. Accordingly, we put special emphasis in this work on the opportunities created by pattern recognition and machine learning techniques and focus on problems whose solutions achieve the best results through modern data analysis. Thus, we first need to understand how exactly targeted attacks are different and what specific challenges they pose.

## 1.1 Targeted Attacks

Targeted attacks are usually labeled by the media as *advanced persistent threats (APT)*. However, some researchers and vendors require the attack to meet certain criteria of customization, duration, objectives and selection of targets, to be recognized as such [e.g. 29, 138]. In this thesis, we will always refer to targeted attacks that manifest the standard attributes of an APT and, therefore, use both terms interchangeably.

In general, targeted attacks are mostly defined in opposition to the characteristics of non-targeted attacks. While threat actors behind targeted attacks often make use of classic techniques such as malicious emails, compromised sites, exploits and malware, the main differences stem, however, from the amount of resources available and thus in their implementation of the attack.

For example, while financial gain represents the main incentive for criminal organizations in non-targeted attacks, actors involved in APTs are also motivated by industrial espionage, sabotage or intelligence gathering. Accordingly, attackers select their targets carefully among governments, businesses, NGOs, organizations, critical infrastructure, academia and research institutions. In addition, threat actors are able to invest large resources into developing their own techniques and exploitation methods which are tailored to a specific target and improved over time to maximize their efficacy. This implies that, while a non-targeted attack is typically an isolated incident, APTs are conducted in campaigns. Thus, after selecting and compromising a target, attackers iterate over their own methods and follow a strategic approach to obtain and maintain long-term persistence in the target's infrastructure. By focusing on covering their tracks, attackers stay in control of the system of the victim and stealthily

extract any new sensitive information from the network with little risk of being detected. It could be said that, while non-targeted attacks are broad in scope and shallow in sophistication, targeted attacks are narrow and deep.

This characterization is based on the observation, particularly by security vendors [29, 105, 134], that threat actors follow a distinctive set of stages during a persistent targeted attack. While the boundaries of these stages may differ between authors, most steps taken during a targeted operation can be categorized in the following phases:

- **Incursion:** In this initial phase, attackers gather information about the target, whether from public sources or trough traditional covert methods. This information is used to lure the victims into executing or loading malicious code, that often exploits one or more vulnerabilities and then establishes communication with a command-and-control server. In contrast to common attacks, where automation can maximize the gain of the attacker, these initial steps are typically taken manually and are highly focused on each specific victim.

- **Discovery:** Throughout this phase, attackers move laterally throughout the network and map the organization resources in search for unprotected services, as well as more vulnerable nodes. The exploitation of additional systems may require to download extra tools to the victim's network and/or further research.

- **Capture:** During this phase, attackers take steps to obtain persistence in the network by, for instance, disabling software auto-update mechanisms. Moreover, attackers may install rootkits in the target infrastructure allowing them to control the functioning of hardware systems and capturing information as it traverses the network.

- **Exfiltration:** In the final phase, attackers extract the collected data through diverse stealthy mechanisms, for instance, through ordinary services (e.g. email, web) on top of encrypted channels. As part of the ongoing operation, the exfiltrated data is analyzed by the attackers to improve their tactics.

As for academic literature, whereas no formal definition of targeted attacks exists, most authors agree on a similar set of traits. For instance, Blond et al. [11] broadly define these attacks as low-volume, with a focus on socially

engineered communication whose goal is to deceive specific victims into installing malware. In the dataset on which they base their analysis, this communication is exclusively performed through email, and malicious archives or documents are the main mechanism of exploitation. Email represents, thus, one of the most effective vectors for social engineering and the main point of entry in targeted attacks, as more than 90% of successful compromises begin with a specially crafted email [60]. Furthermore, email has become a major target itself, as plenty of strategical and sensitive information is regularly discussed through emails in an informal way [99].

If we take on understanding the origins of APTs, we notice how targeted emails were the first evidence pointing at the appearance of a new strain of advanced intrusions. As Hutchins et al. [72] describe, back in 2005, the U.K. National Infrastructure Security Co-ordination Centre (UK-NISCC) and the U.S. Computer Emergency Response Team (US-CERT) issued technical alert bulletins describing targeted, socially engineered emails including *trojan* documents with the goal of compromising the systems of high-level victims and exfiltrating sensitive information. However, some researchers date the origins of targeted intrusions as early as 1996, with the *Moonlight Maze* operation pioneering a still nascent field by, possibly, the oldest publicly acknowledged state actor [107]. From that time onwards, a substantial amount of ongoing operations has been uncovered as a result of an evolved threat landscape and the appearance of numerous professionalized actors (for a detailed list see [90]). On top of that, attackers have now access to a deluge of public information about targets, thanks to social media and the explosion of online services, which makes constructing effective deceptions easier than ever.

Moreover, nation-states do not have a monopoly on this type of operation. Criminal groups are starting to implement similar techniques and selecting the same type of targets with the goal of selling the stolen information, including to governments. As Pritchard aptly remarks, the distinction between "economic intelligence" and industrial espionage can be a fine one [114]. Ultimately, the main reason behind the increase in targeted attacks lies in the complexity of confidently attributing an operation and the possibility of plausible deniability, which ensures that offensive operations are rarely met with a direct response or lawful retribution.

From a defender's perspective, targeted attacks introduce a series of challenges that can be hardly addressed with traditional approaches. In the first place, a threat actor with large resources can implement social engineered

methods that appear totally inconspicuous to any user. Additionally, the characteristic low volume of a targeted attack does not allow for suspicious traits to be discerned by standard monitoring systems and the use of unknown vulnerabilities and tailored malware prevent detection systems, based on signatures and heuristics, from being effective.

While some technical solutions to these issues have been proposed such as, strong network segmentation, authentication infrastructure, data loss prevention schemes, or standards for documenting and sharing threat intelligence, building effective approaches to thwart persistent targeted attacks remains one of the most challenging problems faced by the security community.

## 1.2 Defense against Targeted Attacks with Machine Learning

In the last two decades, the performance of machine learning and pattern recognition algorithms has experienced an impressive improvement. The increasing availability of data and the declining prices of computational resources, have fueled research in traditional and well established fields, such as image recognition and natural language processing. However, the implementation of out-of-the-box algorithms into open-source frameworks have paved the way for academia and industry to find new potential applications for machine learning techniques.

In the security field, where the collection of relevant threat data has always represented more of a challenge, security researchers have still benefited from the general improvements brought to learning algorithms, and these have emerged as a promising set of tools to address some of the challenges introduced by targeted attacks. As we have discussed, motivated threat actors invest time and resources into customizing their tactics for each one of their targets. Consequently, signatures and rule based approaches, as well as techniques based on heuristics, fail to generalize and capture the precise traits that would allow to block or characterize an infrequent behavior. Machine learning algorithms, on the contrary, build mathematical models based on existing examples but can potentially achieve high accuracy at identifying combinations of attributes that were not present in the training data.

Accordingly, and given the low volume of targeted attacks, the security community has developed standard formats and protocols to share relevant threat intelligence in the form of large amounts of data that can also be used to train machine learning algorithms.

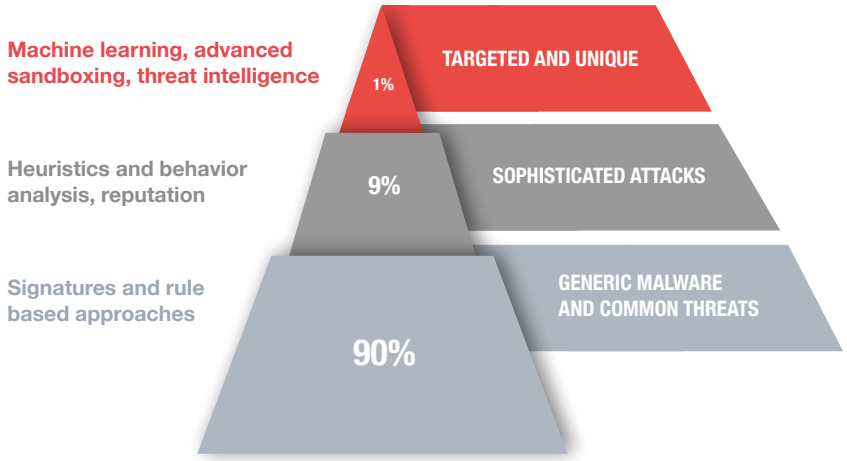



Fig. 1.1 Distribution of threats by sophistication and their corresponding defense mechanisms [89].

Therefore, machine learning, in combination with advanced sandboxing and threat intelligence, emerges not as a substitute for traditional approaches which are effective at mitigating low sophisticated threats like generic malware and phishing, but as a complement to assist the analyst detect and understand the most sophisticated adversaries in a predictive manner (see Figure 1.1).

Academic security research have proposed numerous solutions to address diverse security problems from a defense perspective by taking advantage of machine learning and pattern recognition. However, it is not until recently that researchers have begun proposing specific solutions to address the challenges imposed by targeted attacks. First, by performing exploratory analyses to understand the nature of this type of threats [e.g. 66] and assessing the response of users to social engineering [19, 151]. Next, by focusing on specific technical solutions and designing approaches for which learning algorithms can provide a vantage point for the defenders.

For instance, spear-phishing emails, being the most common entry vector, have received a lot of attention. Given that these emails attempt to impersonate known senders [11], some researchers have focused on blocking messages through behavior modeling and data analysis [e.g. 141, 2, 70]. Moreover, other researchers have suggested that learning based approaches can provide an effective solution to help attributing malicious code to a specific nation-state during the investigation of an APT attack [104, 130].

While most of these approaches provide interesting solutions to important problems, there exists plenty of room for research to overcome some of their

limitations and, most importantly, an urgent need to provide solutions that help security analysts address the threats of targeted attacks in a comprehensive manner across all their phases and through continuous defense.

## 1.3 Thesis Contribution

In this thesis, we aim at providing a holistic strategy for defenders against targeted attacks spanning the phases of detection, analysis and response. Moreover, we aim at designing solutions that can take advantage of existing patterns in data by relying extensively on learning algorithms. Consequently and seen from a longitudinal perspective, the methods proposed in this thesis share a common technical pattern. As shown in Figure 1.2, each one of the approaches proposed in each phase is based on the formalization of a particular abstraction which is created to let pattern recognition and machine learning techniques operate on the structured data of the input problem space.

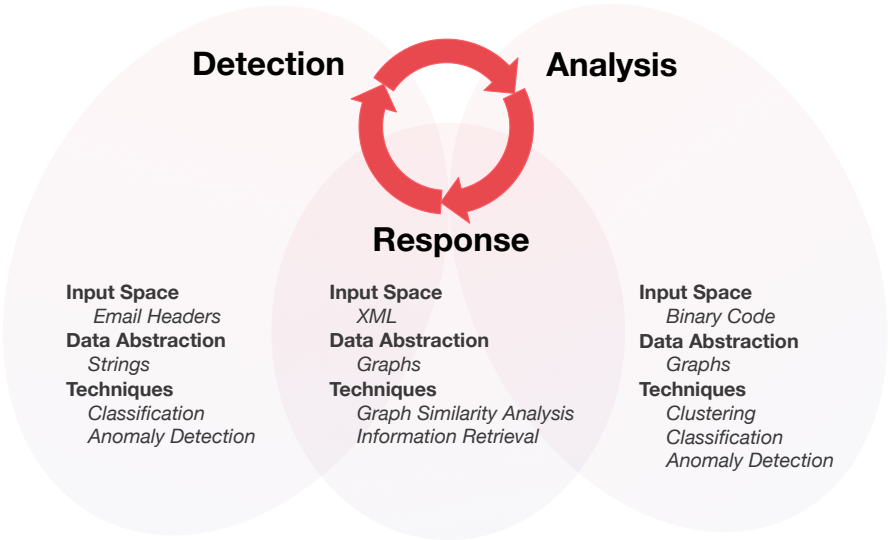



Fig. 1.2 Phases of our holistic strategy against targeted attacks and technical schema of each one of the corresponding solutions proposed in this thesis.

In particular, we address the problems of identifying spear-phishing emails in different adversarial settings (detection), performing malware triage at scale (analysis) and facilitating the authoring, sharing and correlation of threat intelligence (response).

In the course of this thesis, we propose solutions for each one of these problems by introducing a series of techniques and methodologies that are made possible through the following specific contributions:

- **Content-agnostic identification of email senders.** We propose a series of traits to characterize email senders without relying on the textual content of their messages. Then, by combining concepts of machine learning classification and anomaly detection, and based solely on header data observed at the mailbox of a recipient, we devise an approach to identify spoofed emails as a proxy for detecting spear-phishing attacks with high accuracy and against attackers with increasing resources (Chapter 2).

- **Structural embeddings and triage of binary code.** We introduce a generic representation for binary code based on function call graphs and two complementary approaches for graph embedding: an explicit high dimensional mapping that allows for explainability and an implicit low dimensional feature map learned through a deep neural network architecture. These vector representations enable us to take advantage of machine learning algorithms for clustering, classification and anomaly detection in the graph space as means to triage x86 malware based on the structural properties of its binary code (Chapter 3).

- **Unified representation and correlation of threat intelligence.** We introduce a unified representation for threat intelligence standards based on attributed graphs and design a similarity algorithm that operates on its structural representation. Then, we incorporate these concepts into an open-source platform for threat intelligence to devise an information retrieval system that is capable of retrieving related reports given individual observations from security incidents (Chapter 4).

## 1.4 Thesis Organization

This thesis consists of five chapters, from which four of them remain. In each of the first three chapters, we focus on specific challenges faced by the analyst during the detection, analysis and response phases of a comprehensive strategy against targeted attacks. Accordingly, we propose a series of complementary solutions to the problems of spear-phishing emails, malware triage and correlation of threat intelligence that are designed to be implemented on top of each other

and, therefore, recommend the reader to proceed by reading the chapters in their standard order. Nevertheless, the experimental setup within each chapter has been designed independently and its discussion can be read individually without loss of context. The last chapter summarizes and concludes this thesis.

**Chapter 2** is concerned with the detection phase. In this chapter we address the problem of detecting spear-phishing attacks. In particular, we assume that resourceful actors can always craft seemingly authentic messages and propose and evaluate an innovative method to identify spoofed emails without relying on their textual content.

**Chapter 3** addresses the analysis phase. In this chapter we acknowledge malicious code as a source for threat intelligence and focus on the problem of establishing a link between new malware samples and known families. In particular, we introduce a method based on the structural analysis of binary code and derive graph representations that enable the analyst to perform malware triage effectively at scale by means of machine learning algorithms.

**Chapter 4** is concerned with the response phase. In this chapter we introduce a platform developed in collaboration with Siemens CERT for authoring and collecting standardize threat data and, most importantly, a method that enables the correlation of heterogeneous information based on a generic attributed graph representation for threat intelligence.

**Chapter 5** concludes this thesis. In this final chapter, we summarize the main results presented in this work, draw overall conclusions of our proposed approaches and experiments and discuss possible directions for future research.

# 02

# Detection

Emails are a prevalent attack vector for infiltrating companies and organisations. As documents and links are regularly exchanged via email within and across these environments, they are a perfect vehicle for transmitting malicious payloads to a victim [74, 24]. To increase their success, attackers specifically target individual members of an organization using carefully crafted emails—a technique referred to as *spear-phishing*. For example, an attacker may pick an appropriate topic, use correct wording and spoof a well-known sender to convince the recipient of the veracity of an email [61]. These targeted emails are more advanced than regular phishing or spam campaigns, as they are individually adapted to the environment and behavior of the victim. Consequently, there exist only few similarities between different targeted emails which makes it hard to construct effective defenses. As a result, more than 90% of targeted attacks begin through spear-phishing [60].

Although users are increasingly aware of the risk they are exposed to, they have to rely on hints provided by the email client to detect spoofed content. In the default setup, several clients, like Microsoft Outlook and Mozilla Thunderbird, display only little information for identifying the sender, such as the `From` and `Reply-To` fields. Emails from unknown senders can be marked accordingly and specifically dealt with but these and other fields can be forged, making it hard even for a skilled user to distinguish legitimate content from well-crafted attacks [151, 19]. While inconsistent combinations of these

fields can be easily detected and used to notify the user of a potential threat, the situation becomes challenging if all fields are correctly adapted by the adversary, such that the email appears totally legitimate in its content as well as its headers.

Common anti-spoofing techniques such as the Sender Policy Framework [SPF, 125], DomainKeys Identified Mail [DKIM, 124] and the more recent Domain Message Authentication Reporting & Conformance [DMARC, 126] can help to validate the sender of an email in this situation. Similarly, techniques for digital signing of emails, such as PGP [121] and S/MIME [123], enable to verify the sender. Unfortunately, these techniques are still not widely adopted in practice. While we notice several email domains in our evaluation data with SPF entries, less than 5% of the collected 700.000 emails contain corresponding DKIM headers or even digital signatures. Moreover, all of these techniques need to be implemented at the sending side, which renders it difficult to protect from spoofing if not all communication parties adopt the technology [108, 51]. Therefore, given an attacker that is able to exactly match the address of a known sender, the user is unable to detect the attack and might be tricked into opening a malicious file or link.

As a result, there is a demand for alternative approaches that are able to protect users from highly targeted spear-phishing emails in the threat landscape. These approaches need to address three major challenges: First, they need to operate under the assumption that a skilled adversary can almost arbitrarily forge the data within emails. Second, these approaches must not depend on changes at the sending side and operate at the recipient only. Third, they need to account for the large variability of textual content used in spear-phishing attacks that is hard if not impossible to identify by a detection system. A method recently proposed by Ho et al. [70] focuses, for instance, on the identification of credential phishing and is designed to identify attacks from unseen senders. However, their approach ignores the problem of address spoofing and requires the victim to interact with the targeted email by clicking on a link.

In this chapter, we tackle these challenges and propose an approach that is able to verify, without relying on its content, if an email matching the address of a known sender truly originates from its legit source. Our method builds on the observation that a sender leaves characteristic traits in the structure of an email, which are independent from textual content and often persist over time. These traits significantly differ between senders and reflect peculiarities of the user behavior, email client and delivery path, such as particular header combinations,

encoding formats and attachment types. Based on this observation, we develop a detection method that receives the mailbox of a user as input and applies machine learning techniques to generate profiles for all senders in the mailbox, even if only a few emails are available. These profiles provide a content-agnostic view on the sender and enable us to spot spoofed emails as deviations from the learned profiles.

We empirically evaluate our approach on a collection of 92 mailboxes from twelve different domains, covering over 700,000 emails from 16,000 senders. We demonstrate that our method can discriminate thousands of senders in one mailbox and enables identifying spoofed emails with 90% detection rate and less than 1 false positive in 10,000 emails. Moreover, we can show that the individual traits of a sender observed at the recipient's end are hard to guess and spoofing attempts only succeed if entire emails of the sender as delivered to the recipient are known to the adversary. Although our approach cannot generally rule out spoofing due to leaked emails, it considerably raises the bar for targeted attacks and—in absence of widely deployed server-side solutions—provides an effective protection for companies and organisations targeted by spear-phishing attacks.

In summary, we make the following contributions:

- **Characteristic sender profiles:** We identify traits which enable us to characterize the senders of email without relying on textual content. The resulting profiles are expressive enough to distinguish thousands of senders while accounting for the diversity of individual emails.

- **Detection of spear-phishing emails:** We demonstrate how the learned profiles of senders can be used for identifying spoofed emails and help to mitigate the risk of spear-phishing attacks in absence of stronger server-side solutions in practice.

- **Evaluation and evasion experiments:** We evaluate the performance of our method through a series of increasingly adverse scenarios where the attacker becomes stronger by obtaining more information about the target and building a better model of the spoofed sender.

The rest of this chapter is organized as follows: In Section 2.1 we present traits observable in the structure of emails and describe in Section 2.2 how these can be used to construct profiles for senders. We evaluate the resulting

detection method in Section 2.3 and discuss its impact and limitations in Section 2.4. Related work is reviewed in Section 2.5 and Section 2.6 concludes the chapter.

## 2.1 Traits in Email Structure

The identification of spoofed emails is a challenging problem of network security. An attacker can almost arbitrarily manipulate the structure and content of emails, ranging from a trivially spoofed `From` field to carefully crafted sequences of fake `Received` headers [see 122]. In absence of exact detection techniques in practice, such as DKIM and DMARC, it is thus hard to discriminate legitimate from forged emails.

The freedom available for constructing a spoofed email, however, may also turn against the attacker and pose an obstacle. We argue that it is non-trivial to mimic an email from a particular sender without detailed knowledge and that minor irregularities in the email structure may provide valuable clues for identifying spear-phishing attacks. If the attacker has access to emails from a sender known to the victim, she can simply copy the email structure, yet if this information is not fully available, she needs to make a good guess and hope that the forged structure mimics the original communication well.

For uncovering such forgeries, we identify three groups of traits that can characterize the sender of an email: First, when writing an email the sender introduces *behavior features* that reflect individual preferences and peculiarities. Second, the email client generates *composition features*, identifying the particular client and its configuration. Third, the delivery of an email leaves *transport features* that capture details of the sending and receiving infrastructure. In the following, we describe these groups of traits in more detail and use the simplified email in Figure 2.1 as a running example through out this section.

### 2.1.1 Behavior Features

When a user writes an email, several of her individual preferences can manifest in the structure of the email—aside from her writing style and habits [44, 141]. For example, some senders are frequently including recipients using the `CC` header, whereas others avoid this and prefer to address all recipients directly using the `To` field. Similarly, senders differ in the type and amount of files they are attaching to emails in conversations. While some of these features are

```
Return-Path: <john@doe.com>
Received: from [93.184.216.34] (HELO example.com)
    by example.com with ESMTP id 69815728;
    Tue, 16 May 2017 14:06:48 +0200
To: Jane Dee <jane@example.com>
Date: Tue, 16 May 2017 14:00:02 +0200
Message-Id: <20170516133920.23212@doe.com>
Subject: Security Conference
From: John Doe <john@doe.com>
In-Reply-To: <1405590537$56fe@example.com>
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="boundary"

-boundary
Content-Type: text/plain

FYI, interesting conference: https://tinyurl.com/ktmqtgh

-boundary
Content-Type: application/octet-stream; name="foo.exe"
Content-Transfer-Encoding: base64

TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAKCkdyZWV0aW5ncyw
gUmV2aWV3ZXIhCsKvXF8o44OEKV8vwq8KCg==
-boundary-
```

Fig. 2.1 Simplified email as running example.

volatile and change between different contexts, other features may persist over time and provide a first basis for constructing a profile of the sender.

For our analysis, we identify 13 feature types that characterize the behavior of a sender in the structure of an email, including

1. the type, number and order of attachments, for example when multiple documents are exchanged,
2. the relation to other emails and recipients, for example in form of `References` and `In-Reply-To` headers,
3. digital signatures and certificates attached to the email as well as corresponding PGP and S/MIME fields, and
4. the amount of text in the main part and the amount of quoted text in email responses.

A complete list of all 13 features is provided in Table A.1 of the appendix. Note that the cardinality of these features differs, where some may appear

multiple times in an email, such as the type of attachments and others only once, such as the depth of the MIME structure. As an example, the email in Figure 2.1 shows the attachment of an executable file (line 20) and the reference to a previous conversation (line 10)—two features that are rarely used in combination.

### 2.1.2 Composition Features

The second source for traits in the structure of an email is the mail user agent (email client) that converts the provided addresses, text and attachments into a suitable format for delivery. As emails have been originally restricted to ASCII characters, there exists a wealth of encoding schemes for converting binary data to a compatible ASCII representation [e.g., 118, 119, 120]. These schemes are selected by the email client and often slightly vary in implementation, thus providing features that characterize the composition of an email. For example, the Base64 encoding [120] does not enforce a fixed text length and thus clients differ in the formatting of the corresponding text blocks. Similarly, there exists several minor variations in the construction of multi-part MIME messages that provide clues about the client and its configuration.

For our analysis, we identify 22 feature types that capture peculiarities of the email client and its configurations, including

1. the type, order and syntax of common headers, such as the `From`, `To`, `Subject` and `Message-Id` headers,

2. the type, order and syntax of headers in MIME parts, including fields like `Content-Type` and `Content-Disposition`,

3. the syntax of address fields, such as the formatting and quoting of names and email addresses,

4. the encoding of international characters in the subject field, in address fields and filenames,

5. the type and location of textual content, such as HTML and plain parts in the email,

6. client-specific behavior, such as the length of line breaks, missing and superfluous encodings of characters,

7. individual details of the MIME structure, such as the depth and the order of different MIME parts, and

8. the structure of the `Message-Id` header and the structure of MIME boundaries.

A complete list of the 22 composition features is provided in Table A.2 of the appendix. While these features alone are clearly not sufficient to identify attacks, in combination with behavior and transport features they sharpen the view on a sender and thereby obstruct the spoofing of email addresses. As an example, the email in Figure 2.1 shows a unique order of the `From`, `To` and `Subject` field (line 5–9) which indicates a rare email client. Furthermore, the Base64-encoded attachment is formatted using a 60 character line length (line 23).

### 2.1.3 Transport Features

A third group of traits can be attributed to the delivery path of an email. As the email moves from the sending to the receiving mail transport agent, often passing multiple hops, different headers are added to the structure. These headers describe the individual mail hops in form of `Received` headers and provide information about available delivery features, such as delivery protocols, TLS or the time zone of the mail server. These headers and features, again, generate a series of traits that can help to distinguish different senders and spot irregularities in the delivery process.

Although an attacker can insert fake headers prior to the delivery of an email, it is not possible to change or remove headers added by hops on the delivery path. As a consequence, an attacker can only forge these headers by either connecting directly to the receiving server or, alternatively, attempting to inject emails early into the delivery process—a tractable but non-trivial task in practice, as it would require having access to the same delivery infrastructure as the sender that the attacker is trying to spoof.

We identify 11 transport features that enable us to reconstruct the delivery path of an email and spot deviations from past emails of the same sender.

These features include

1. the number and order of `Received` headers, where each hop is represented by the hash of its hostname,
2. the path of time zones from the first to the last hop during the delivery process,
3. the delivery protocols and TLS features available in some `Received` headers,
4. the validity of DKIM records added by the servers and their relation to the claimed sender of the email, and
5. non-standard headers added by spam filters or anti-virus services during the delivery of the email.

Table A.3 in the appendix provides a list of all 11 transport features. As an example of traits introduced by the delivery process, the email in Figure 2.1 contains a detailed `Received` header (line 2–4). This header defines the mail hop, delivery protocol and delivery time. This information is available with any mail passing the hop and thus can leak to the attacker. However, we show in Section 2.3 that knowledge of transport features alone is insufficient to evade our detection method and that the attacker needs access to original emails delivered to the recipient for successfully spoofing a sender.

## 2.2 Content-Agnostic Spear-Phishing Detection

Equipped with three groups of traits for characterizing the sender of an email, we are ready to develop a corresponding detection method using machine learning techniques. The application of learning methods spares us from manually constructing detection rules for each of the senders and thereby allows for scaling our approach to thousands of senders, as we demonstrate in Section 2.3.

### 2.2.1 Feature Extraction and Embedding

The proposed groups of traits provide detailed information about the structure of emails from each sender in the recipient's mailbox. In order to learn a profile from the traits, however, we require a numerical representation that can be used in combination with common learning methods. As a remedy, we apply

the concept of a *bag-of-words model*—a technique originating from information retrieval [133] and natural language processing [78, 77]—and adapt it to the traits extracted from the structure of emails.

To this end, we represent each of the extracted traits as a feature string and build a joint set $F$ that comprises all observable strings from the three groups of traits:

$$F := F_{\text{behavior}} \cup F_{\text{composition}} \cup F_{\text{transport}}. \tag{2.1}$$

Table A.1, A.2 and A.3 in the appendix show some of these feature strings as examples in the rightmost column.

Making use of this set $F$, we define an $|F|$-dimensional vector space that takes values 0 or 1 in each dimension. Each email $e$ is then mapped to this space by building a vector $\varphi(e)$, such that for each feature $f$ extracted from $e$ the corresponding dimension is set to 1, while all other dimensions are set to 0. Formally, this map can be defined for all emails $M$ as

$$\varphi : M \longrightarrow \mathbb{R}^{|F|}, \quad \varphi(e) \longmapsto (I_f(e))_{f \in F} \tag{2.2}$$

where the auxiliary function $I$ simply indicates whether the feature $f$ is present in $e$, that is,

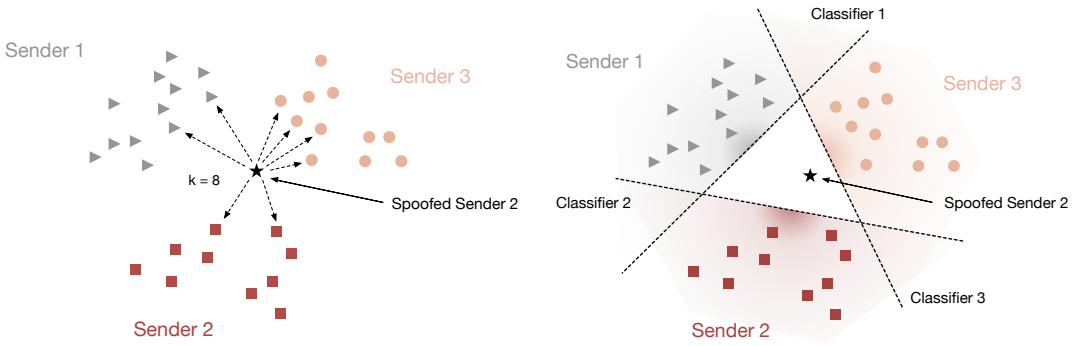
$$I_f(e) = \begin{cases} 1 & \text{if email } e \text{ contains feature } f \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

The resulting binary vector space $\mathbb{R}^{|F|}$ allows us to represent each email as a vector of the contained traits of its sender. In the following, we describe how we use this representation to train a machine learning classifier that, based on these features, is able to assign each email to its corresponding sender and indicate possibly spoofed emails.

### 2.2.2 Model Learning and Classification

Several learning methods can be applied for classifying data in a vector space. To operate in our setting, however, a learning method needs to address additional requirements: First, the method has to be able to operate in a high-dimensional vector space, as the set $F$ may cover thousands of different traits. Second, the methods needs to be capable of learning a classification model, even if only very few training data is available, such as a couple of emails only.

In view of these requirements, we select the following two learning methods for our detection approach: (a) a k-nearest-neighors classifier (kNN) that can generate good classification results with very few training data and (b) a multi-class support vector machine (SVM) which is known for effectively operating in high-dimensional vector spaces [see 42].




(a) kNN classifier (b) SVM one-vs-all classifier

Fig. 2.2 Schematic overview of the detection: A classifier is used to identify emails as spoofed when a mismatch between the output of the classifier and the origin sender address occurs.

**K-Nearest Neighbors Classifier** The kNN algorithm is a simple yet effective learning method for classification. It computes the distance between a test sample and all existing samples in a training set and makes a decision through voting on the labels of its k-nearest samples after applying a weight function (see Figure 2.2a). Such instance-based learning algorithms do not construct an explicit learning model and thus can be applied even if only a single email is available for a sender. For our approach, we label each feature vector with the address of the originating sender address. When a new email is received, we compute the distance between this sample and the feature vectors of all existing emails as follows

$$d(e_x, e_y) = \left|\left| \varphi(e_x) - \varphi(e_y) \right|\right|_1 = \sum_{f \in F} | I_f(e_x) - I_f(e_y) | , \tag{2.4}$$

where $d$ corresponds to the Manhattan or $L_1$ distance. A mismatch between the incoming sender address and the prediction of the classifier is then flagged by our method as a spoofing attempt.

The advantage of making predictions with very few training data, however, comes at a price. The distance between each new email and all existing emails

needs to be computed before making a decision, which is computationally expensive on large mailboxes. Fortunately, this problem can be addressed in two ways: First, one can implement the classifier using special data structures for reducing the number of distance computations, such as ball trees and cover trees [10]. Second, if the number of training instances reaches a certain limit, one can simply switch to another learning method, such as a support vector machine or, when possible, sample the training data according to a distribution that maintains the classifier performance.

**Multi-Class Support Vector Machines** As second learning method, we employ a linear multi-class SVM algorithm [46]. The algorithm computes a series of maximum-margin hyperplanes that separate the emails from one sender from the emails of all other senders (see Figure 2.2b). That is, given $N$ different senders, $N$ hyperplanes are determined, each one of them represented by a vector $w \in \mathbb{R}^{|F|}$ and a scalar $b$ in the vector space.

If a new email arrives, we simply determine the position to the learned hyperplanes and pick the sender with the best match, that is, the largest value of

$$h(e) = \langle \varphi(e), w \rangle + b = \sum_{f \in F} I_f(e) \cdot w_f + b. \tag{2.5}$$

Note that this function can be computed efficiently, if the feature vector $\varphi(e)$ is sparse, as only non-zero dimensions $I_f(e)$ contribute to the output. As a result, we can compute $h(e)$ in linear time in the number of traits $|e|$ extracted from $e$ and the overall run-time for analyzing an email is $O(N|e|)$. In contrast to the kNN algorithm, the run-time for the prediction of a linear SVM is independent of the size of the training set and thus this learning method is suitable if more emails are available from particular senders [see 46].

To demonstrate the efficacy of our method, we assess in the following how our proposed set of features is able to capture the differences between different senders and then, evaluate the performance of our approach for detection of spoofed emails on real data from a large set of recipient mailboxes.

## 2.3 Evaluation

We proceed to evaluate our detection method on a large dataset of real-world emails. In particular, we are interested in studying the ability of our method to characterize the sender of an email based on its structure and to identify spoofed emails under different levels of knowledge of the adversary. Before presenting these experiments, we first introduce our dataset (Section 2.3.1) and define the corresponding attacker model (Section 2.3.2).

### 2.3.1 Evaluation Data

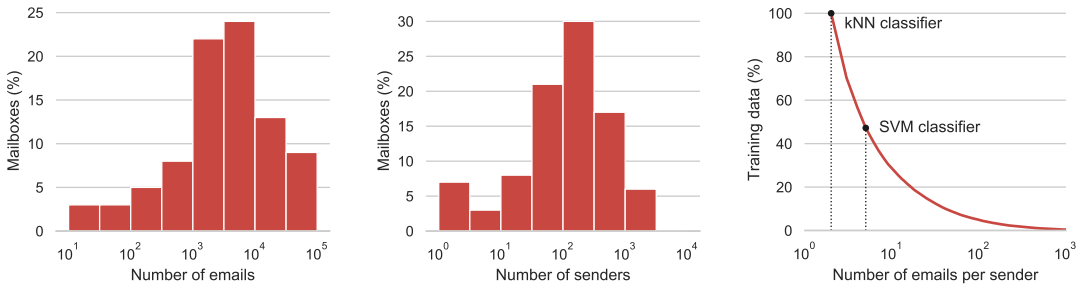
For our evaluation, we have gathered anonymized features extracted from 92 mailboxes from twelve different domains, including enterprise and commercial email services. To evaluate the efficacy of our detection method, we require at least one email for learning and one for testing from each sender. Consequently, we discard all emails from senders that have sent only a single email. Our final dataset comprises a total of 760,603 emails from 17,381 senders, where each sender has authored at least two emails. These emails are described by a total of 617,960 features extracted using the traits defined in Section 2.1. Table 2.1 provides an overview of the statistics of our evaluation data.

Table 2.1 Statistics of email data used for evaluation.

| **Basic statistics** | | | **Total** |
|---|---|---|---|
| Mailboxes | | | 92 |
| Emails | | | 760,603 |
| Senders | | | 17,381 |
| Features | | | 617,960 |
| **Detailed statistics** | **Min.** | **Mean** | **Max.** |
| Emails per mailbox | 2 | 8,267 | 50,924 |
| Emails per sender | 2 | 43 | 44,204 |
| Senders per mailbox | 1 | 279 | 2,144 |
| Features per email | 5 | 69 | 183 |
| Emails per sender and mailbox | 2 | 29 | 10,304 |

Figure 2.3 depicts in more detail how emails and senders are distributed within our dataset. From Figure 2.3a and 2.3b we can see that over 50% of the mailboxes in our dataset contain between $10^3$ to $10^4$ emails and between $10^2$ to $10^3$ different senders. This large corpus of emails provides a good basis for evaluating the performance of our method. Depending on the applied learning model, however, we require a minimum number of emails per sender and thus

not all senders might be available for training. Figure 2.3c shows the amount of training data available to a learning method depending on the minimum number of emails per sender. While for the kNN classifier all senders can be used for evaluation, in the case of the SVM classifier, we need to restrict our experiments to 46% of the data, as we require at least 5 emails for training.




(a) (b) (c)

Fig. 2.3 Overview of the evaluation data: (a) distribution of emails and (b) distribution of senders in the 92 mailboxes; (c) training data available for learning with varying emails per sender.

To prepare our experiments, we extract feature vectors from all emails in our evaluation data. This may seem as an intractable task at first glance, as the resulting vector space has over 600,000 dimensions. However, the majority of these dimensions is zero and each email contains only between 5 to 183 features (see Table 2.1). As a result, we can make use of efficient data structures for operating with these sparse feature vectors [see 127].

As a sanity check whether our representation is suitable for learning a classification, we first study how senders in a mailbox differ from each other and then analyze how emails from a specific sender change over time. To this end, we first calculate a simple statistic: For each sender, we compute the average of its feature vectors and measure the distances between the resulting 17,381 mean vectors within each mailbox. We make use of the Manhattan distance ($L_1$ distance) for comparing the mean vectors. The distance can be interpreted as the average number of features differing between the senders and thus provides an estimate for the quality of extracted traits.

Figure 2.4 shows the distribution of the Manhattan distances between all senders in each mailbox. It can be observed that most senders are separated from each other by a distance larger than 40 on average. This demonstrates that several of the extracted traits are highly specific and capture nuances of the email structure suitable for discriminating the senders.

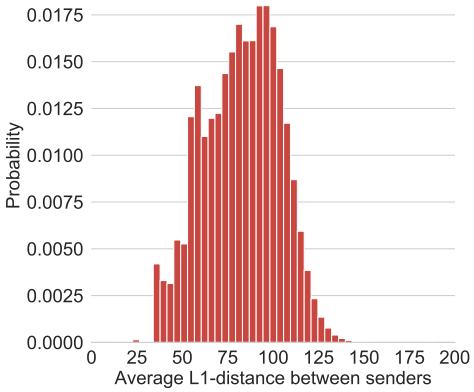



Fig. 2.4 Average distance between senders

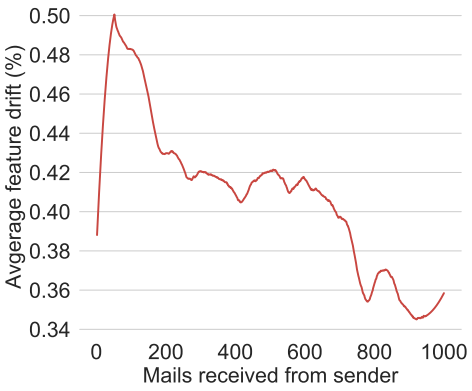



Fig. 2.5 Feature drift over time

Multiple sources may introduce variability and noise into the email traits of a sender, such as software updates, network configurations and changing devices. We thus study how emails from an individual sender change over time. In particular, we want to answer the question how many features change in a new email when it is compared with existing emails from the same sender. For this, we measure the Manhattan distance between each email received at a certain point in time in a mailbox and all emails previously received from the same sender. The average number of differing features is then presented as a percentage of the feature space dimensionality.

Figure 2.5 shows that a slight feature drift exits. It can be observed how the variability grows rapidly at first with the initial emails received from a sender. However, when an increasing number of emails is received each class becomes more compact and the average percentage of different features in a new email decreases. Note that although individual profiles become more stable during time, they also tend to differ considerably between senders as shown in Figure 2.4.

As the final preparation step, we determine the presence of anti-spoofing techniques in the 760,603 emails using corresponding email client and transport features. Table 2.2 shows the percentage of emails in our dataset that contain anti-spoofing techniques, where we additionally report statistics from the top million web domains listed at the monitoring service *BuiltWith* [15]. Although the adoption of SPF [125] has reached almost 40% by now, the overall implementation of anti-spoofing techniques is still low in both data sources. In particular, recent techniques, such as DKIM [124] and DMARC [126] are used in less than 5% of the emails, thereby emphasizing the need for alternative protection measures.

Table 2.2 Anti-spoofing techniques in our evaluation data and as reported by the monitoring service *BuiltWith*.

| **Anti-spoofing technique** | **Our data** | **Top 1M** [15] |
|---|---|---|
| SPF | — | 39.9% |
| DKIM | 4.3% | 0.1% |
| DMARC | — | 1.3% |
| PGP, S/MIME | 0.88% | — |

### 2.3.2 Attacker Model

In the absence of anti-spoofing techniques, a skilled adversary is able to forge most of the data contained in an email. However, we argue that, by inferring a sender profile based on traits of the email structure, an attacker is forced to mimic such profile to effectively masquerade as the sender. As a consequence, the success of such spoofing depends on how much information of the email structure is available to the adversary and if the attacker has access to the senders delivery infrastructure.

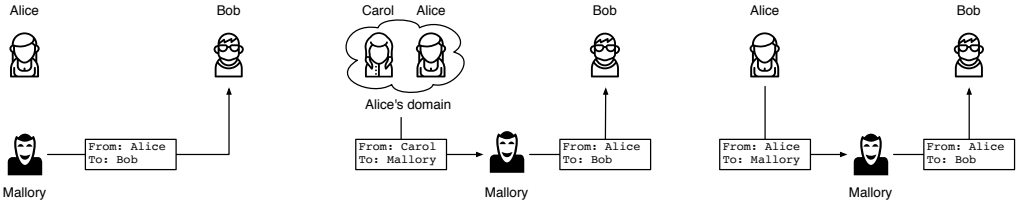
Therefore, we begin the evaluation of our approach by measuring in a controlled experiment how an attacker may affect the detection performance by spoofing an increasing number of features from a sender's profile (i.e. all features extracted from all emails received from a specific sender in a mailbox). To this end, we first split each sender's data in a mailbox into training and testing sets and then train both kNN and SVM classifiers. For testing, we select random emails from other mailboxes and relabel them as known senders of the target mailbox to imitate spoofing attempts. This means that our testing set is comprised of 50% of legitimate emails and 50% of spoofed emails with a random percentage of correct traits of the target sender.

Note that to generate spoofed emails we do not rely on their textual content for feature extraction. Moreover, we adapt the transport features added by the recipient MTA to the recipient mailbox. As a result, the spoofed emails in our testing set are not different from real spear-phishing emails sent by an attacker, as no textual content is considered.

We measure the detection performance of our classifiers using the true-positive rate (TPR) and false-positive rate (FPR). In our setup, a true positive implies that a spoofed email has been correctly identified, while a false positive corresponds to a legitimate email wrongly being tagged as spoofed. Furthermore, we use a Receiver Operating Characteristic (ROC) curve to present both

evaluation metrics and calculate the area under the ROC curve (AUC) as a numerical aggregate of the classification performance [see 47].

Although an adversary with increasing capacity will affect the ability of the classifier to correctly identify deviations from a user profile, the information available to an attacker is constrained by threat scenarios that can occur in reality. In this work, we thus assume that the knowledge of an attacker can range from knowing nothing about the spoofed sender to having real examples of her emails. Accordingly, we model these attackers through a series of increasing adversarial setups and proceed to evaluate the performance of our approach in each scenario as depicted in Figure 2.6:




(a) Blind Spoofing (b) Known Domain (c) Known Sender

Fig. 2.6 Threat scenarios for increasing attacker capabilities based on the acquired knowledge about the spoofed sender: (a) the attacker has no information about the sender, (b) the attacker has access to emails received from the sender's domain and, (c) the attacker has access to one or more emails from the real sender.

(*a*) *Blind Spoofing:* In this scenario, the attacker (Mallory in Figure 2.6) tries to spoof a particular sender from which she does not have any information. The only available strategy for the attacker is to simply replace the `From` and `Return-Path` headers of the targeted email and try to guess the behavior, composition and transport features.

(*b*) *Known Domain:* In this scenario, the attacker has received or has access to one or more emails sent by a sender that belongs to the same email domain as the spoofed sender. The attacker can thus expect that some of their transport features are present in the emails received by the victim from the sender she wants to spoof.

(*c*) *Known Sender:* In this scenario the attacker has received or has access to one or more emails sent by the spoofed sender. As a result, several traits used for constructing the profile are available to the attacker and can be incorporated in her spoofed emails.

In the following, we describe how we learn a profile of each sender within a mailbox and assign the role of the victim to the owner of the mailbox. Then, based on the attack strategies described in each scenario and using the emails available in our dataset we build corresponding sets of spoofed emails for each sender and combine them with legitimate emails to evaluate our method.

### 2.3.3 Spoofed Email Detection

We proceed then to evaluate the performance of our approach in the threat scenarios defined in the previous section. In order to learn a profile for each sender we begin again by splitting all available emails into training and testing sets. For training, we consider all emails received up to a certain point in time. In the case of the kNN classifier one email from a sender in the training set suffices to make a decision about an incoming email from this origin address, while for the SVM classifier we require a minimum of 5 emails from a sender to include this class during training.

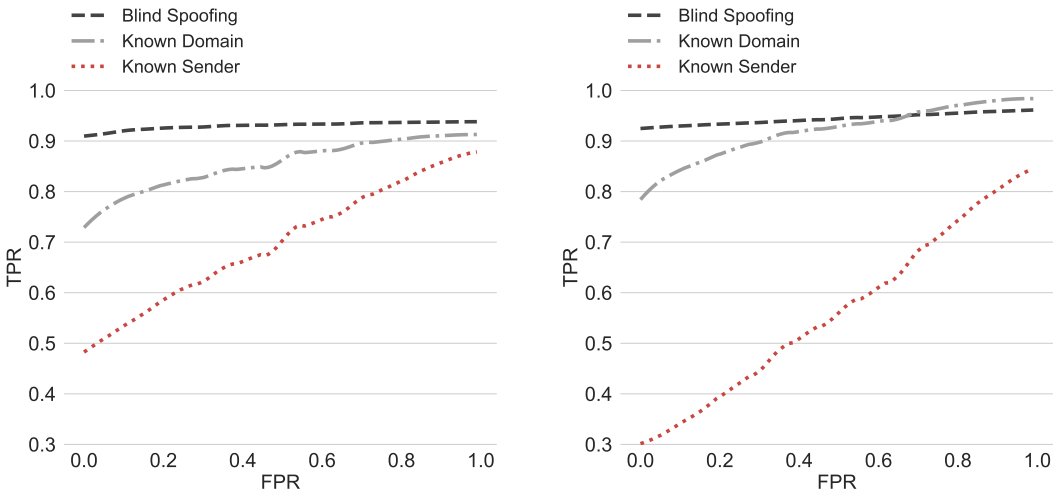
In order to tune the parameters of each classifier, we partition the training data into 5 splits and use training/validation partitions, such that the temporal order of emails is preserved—similar to a regular cross-validation. This enables us to simulate training with past data and generating predictions for data yet unseen. Note that although a mailbox or sender may not present enough emails for training, we still use these samples to generate test spoofed emails.

For the testing phase, we combine the test set of legitimate emails with a set of emails crafted according to the attacker strategies described in Section 2.3.2. In the case of a *blind spoofing* attack, we select a random set of emails sent to recipients at different domains than the victim and label them as the spoofed sender. Likewise, we evaluate the *known domain* attack by selecting emails sent from the domain of the spoofed sender by a different sender to other recipients. Finally, we select emails sent by the spoofed sender to different recipients to built the spoofed test set in the evaluation of the *known sender* attack.

During testing, we expect a legitimate email to be assigned to its true class by the classifier. On the contrary, a spoofed email should be assigned to any of the other classes, resulting in a mismatch between the sender address from which the email is sent and the output of the classifier. There exists thus a trade-off between the probability of detecting a spoofed email and the probability of wrongly highlighting a legitimate email as spoofed. The ROC curves depicted in Figure 2.7 show the trade-off between the false-positive rate and the false-positive rate for both classifiers.

Table 2.3 Detection performance of our approach in different threat scenarios.

| **Blind Spoofing** | | | | **Known Domain** | | | | **Known Sender** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **kNN** | | **SVM** | | **kNN** | | **SVM** | | **kNN** | | **SVM** | |
| FPR | TPR | FPR | TPR | FPR | TPR | FPR | TPR | FPR | TPR | FPR | TPR |
| 0.01% | 90.9% | 0.01% | 92.4% | 0.01% | 72.7% | 0.01% | 78.1% | 0.01% | 48.1% | 0.01% | 30.1% |
| 0.1% | 90.9% | 0.1% | 92.4% | 0.1% | 72.7% | 0.1% | 78.2% | 0.1% | 48.2% | 0.1% | 30.2% |
| 1% | 91.1% | 1% | 92.5% | 1% | 73.7% | 1% | 79.3% | 1% | 48.9% | 1% | 30.4% |
| 10% | 91.9% | 10% | 92.9% | 10% | 78.4% | 10% | 84.1% | 10% | 53.2% | 10% | 33.9% |

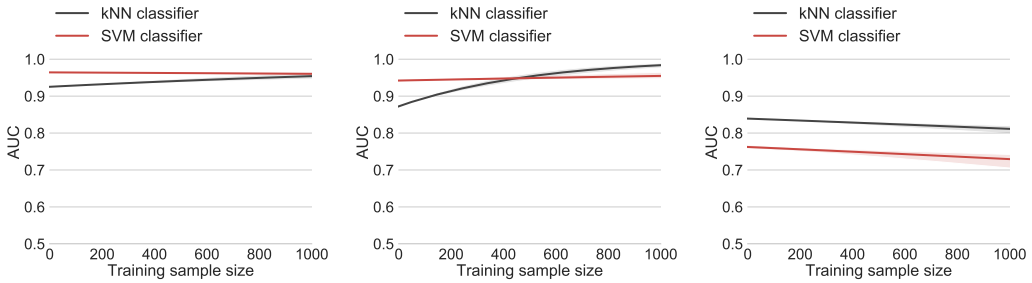



(a) kNN (k=10) (b) SVM

Fig. 2.7 ROC curves for the classification of legitimate emails versus emails spoofed by attackers with different levels of knowledge.

If the attacker lacks any knowledge about the spoofed sender, we observe that the kNN and SVM classifiers can identify a spoofed email with a true-positive rate of 90.9% and 92.4% respectively at a low false-positive rate of 0.01%. If the attacker has access to emails originating from the same domain, the performance decreases to 72.7% and 78.1% but the classifier is still able to effectively operate at the same low false-positive rate. In the worst-case scenario, the attacker has enough information to craft an email that resembles the learned profile of the spoofed sender, which causes the performance of the classifier to deteriorate considerably. Table 2.3 specifies numerically the detection achieved at 0.01%, 0.1%, 1% and 10% of false-positive rate for both classifiers in all scenarios.

As mentioned above, we set a lower threshold for the minimum number of emails required to train an SVM classifier. However, as shown in Figure 2.3 a larger number of emails above this threshold is available for many senders.




(a) Blind spoofing (b) Known domain (c) Known sender

Fig. 2.8 Area under the ROC curve as a function of the number of training emails used to learn each sender's individual profile.

Figure 2.8 shows in each scenario the relation between the number of emails from a sender used to train the classifier and the AUC averaged over all mailboxes and senders. As described in Section 2.3.1, sender profiles tend to be more compact with an increasing number of emails. However, this can affect the performance differently depending of the knowledge available to the attacker. For instance, in threat scenarios a) and b), emails are classified with an AUC over 0.85 with a small number of training samples. Spoofed emails lay here far enough from the sender profile, leading to a stable or increased performance when classes becomes more populated. In particular, the SVM classifier offers a better performance at a low number of available emails, while with an increasing training size, the kNN classifier surpasses the SVM.

On the contrary, in threat scenario c) the attacker is always able to craft an email that resembles the profile of the spoofed sender, while a larger number of training samples increases the variability of the sender profile. As each spoofed email lay very close or within the target class, it becomes more difficult for the classifier to correctly separate legimitate emails from spoofing attempts when the sample size increases. A possible approach in such a high risk scenario, is to operate the classifier at a higher FPR point and to retrain the model more often on a smaller sample of the most recent emails received from each sender.

Furthermore, the use of a linear SVM for classification allows us to study how the learning algorithm assigns different weights to each type of features according to its importance for the classification and how the importance of each group of features correlate with the importance of other groups. To this end, we first determine the distribution of the normalized SVM weights and group them by trait types.

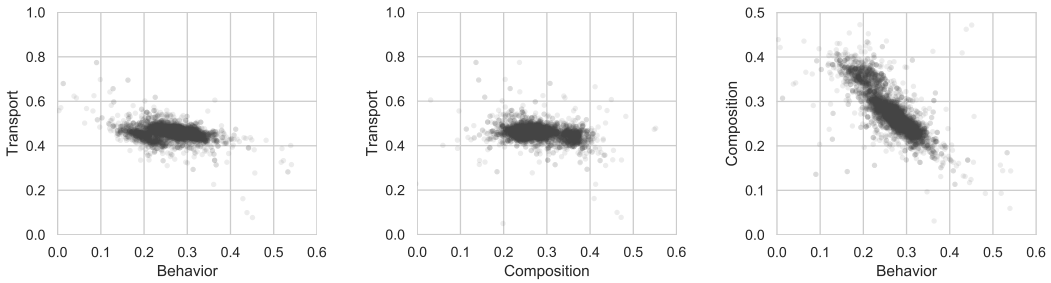



Fig. 2.9 Correlation between the linear SVM scores of the different groups of traits. Weights are assigned to each trait by the algorithm during training and indicate the influence of the trait in the decision of the classifier.

We can observe in Figure 2.10 that, in comparison with behavior and composition features, transport related features manifest both a smaller dispersion and a larger influence on the decision of the classifier. Moreover, the relations depicted in Figure 2.9 indicate that while there exists an inverse correlation between the influence of behavior and composition features as part of a sender's profile, transport features are mostly independent.

As a consequence, transport features have the most discriminative power and this is not influenced by the variance in importance of behavior and composition traits. At the same time, transport features are the most difficult traits to forge as even a skilled adversary is not able to fully control transport features without having access to the same delivery infrastructure of the sender.

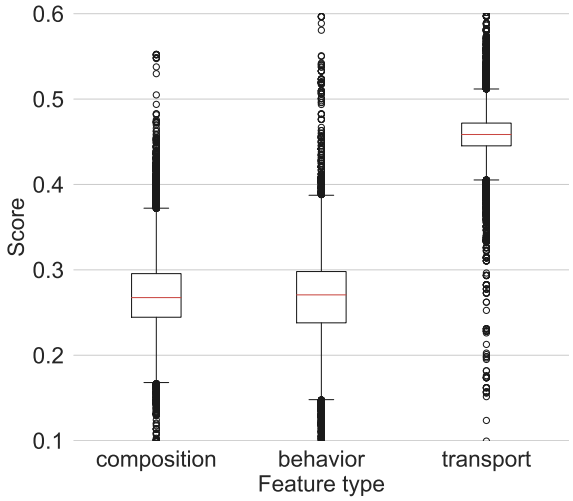



Fig. 2.10 Distribution of scores per group of traits as learnt by the linear SVM classifier during training.

### 2.3.4 Integration

Our proposed method represents not only an effective approach to detect spoofed targeted emails, as we have demonstrated in this section, but is also straightforward to integrate with any email client in a real deployment. To this end, feature extraction is performed locally at the mailbox of the recipient and the resulting feature vectors are fed to the learning algorithm. The classifier can be trained locally as well as on a remote cluster infrastructure without any loss of privacy as no information about the senders or their label mapping needs to be shared. The learned model is then used at the incoming mail transport agent or the client to make decisions about incoming emails.

Regarding the workflow of the recipient, Figure 2.11 illustrates how a prototype implementing our method can be operated through the graphical user interface of the email client. If an email does not match the profile of its sender it is labeled as spoofed and highlighted ❶ in the interface. The same occurs if the message is the first email received from an unknown address. In both cases links and attachments are removed from the email ❷ and an alert is shown to the recipient ❸. The user can then choose to load the removed attachments and/or links for further inspection and to label the email as trusted after proper verification ❹. If the email is labeled as trusted by the recipient, both links and attachments are downloaded and the email is marked as safe. As new emails arrive at the mailbox of the recipient it will be necessary to retrain the model. For this purpose, all emails not labeled as spoofed and also those manually labeled as trusted by the user will be included as training data and considered legitimate during retraining.

A special case can occur if a sender makes use of several aliases to send emails from a unique account. To avoid conflicts during testing, the recipient can link these addresses ❹ and a unique identifier will be assigned to the different addresses during training and testing.

## 2.4 Limitations

The evaluations in the previous section show that our method is capable of reliably discriminating thousands of senders and identifying spoofed emails if the attacker has limited knowledge of the email structure. Due to the problem setting of detecting spoofing at the receiving side, however, our approach has some inherent limitations which are discussed in the following.

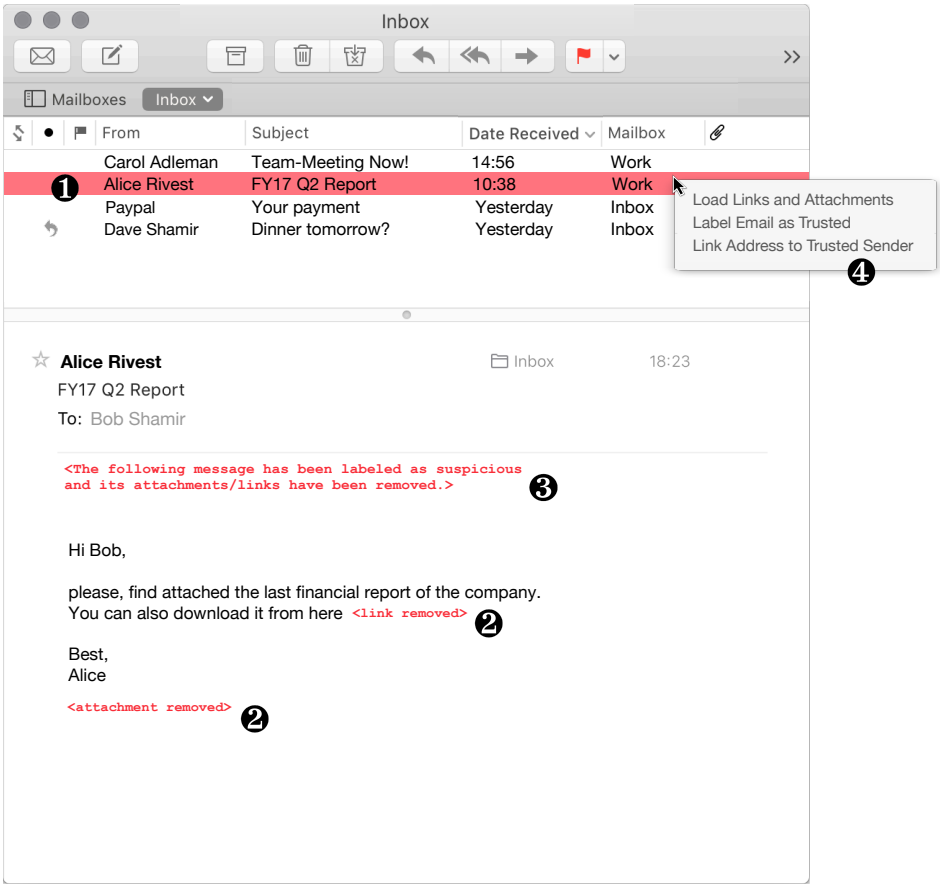



Fig. 2.11 Example of email client interface presented to the user when an email is detected as suspicious.

**Advanced forgery** Although spear-phishing and other targeted email attacks today focus on the forgery of visible features like the sender address, the subject and the content of an email to mimic trustworthy emails [11, 66], we likely have to deal with more advanced attacks in the near future. If current attacks are no longer successful because of increased user awareness and detection approaches like ours, attackers will adapt their techniques.

For our method, the best strategy for evasion is to forge as many features from the original sender as possible. An almost perfect forgery is thus a copy of an original mail including also its true transport features as observed by the recipient and enriched with some malicious content. However, the attacker needs to take care of several traits that our method inspects, such as timestamps, IP addresses in received headers and characteristics of the attachment. In the worst case, the attacker is able to forge all of these details and hence the only

indication of a spoofed email are minor inconsistencies between IP addresses and hostnames.

Our method fails in this scenario, as only a few features differ from the sender model. Nonetheless, the acquisition of emails from a sender and acquiring access to the senders delivery infrastructure to control the transport features, clearly raise the bar for conducting spear-phishing attacks. Therefore and with the current lack of alternative protection approaches, our approach is a valuable extension to current defenses.

**Privacy and feature extraction** We have implemented the feature extraction in a privacy-friendly way in that all sensitive information of sender, transport and recipients is only stored in an anonymized form by using a hash with random salt. Only these anonymized features are kept and used in the initial creation or retraining of the model. This makes it possible to implement the system for example in a security appliance which receives all feature vectors for analysis but does not store the mails. This also means, however, that the model cannot be simply extended with new features and retrained with old data, since the original mail as input for feature extraction is no longer available. Feature extraction is therefore performed locally in every case. Although this limits how anonymized data from different sources can be combined for analysis, the recipient's email information never leaves the local machine, avoiding privacy issues and possible attack vectors.

**Mislabeled data** The possibility of the training data containing spoofed emails should not be ignored. However and due to their very nature, the prevalence of spear-phishing emails can only be very low within all emails sent to a recipient. This problem, known as *label noise* [see 36], entails that training samples can be considered subjected to an additive noise during training with a probability of their labels being flipped. In our setup, however, such probability will be very low and the effect during testing of such infrequent examples, while existent, will be negligible.

## 2.5 Related Work

The detection of unwanted and malicious emails is a well-established problem in security research. Several methods have been devised in the last years that are related to our approach and which we briefly discuss in the following.

As non-targeted phishing mails are delivered in mass, they can be detected by spam traps or generic heuristics. Once detected, a variety of features can

be extracted [e.g., 48, 95, 148] and used to update blacklists, filter rules or reputation information of the web sites linked from the mail. By contrast, spear-phishing emails are sent only to a small group of recipients and are customized to look trustworthy, typically by spoofing a trusted sender. While malware analysis and link reputation still work to a lesser degree in this scenario, features depending on mass distribution are hidden from analysis. Recent strains of research have thus attempted to detect spoofed emails by generating models of trusted senders and comparing these learned models to the sender of an email.

For instance, several approaches exist that focus on the content of emails and the style in which they are written [e.g., 141, 44, 64]. The assumption behind these features is that the writing style of one sender differs significantly from another and that it is too hard for the attacker to write a mail in the same style as the sender she is trying to spoof. The implementation of such content-based features can be as simple as using a 5-gram tokenizer [93] but can also be more complex and include character distributions, atypical words or more advanced stylometric features [141, 44, 64]. In many cases, these stylometric features are used in combination with further behavioral features, such as the time of writing.

While these approaches potentially provide a good detection of spoofed emails, they present two problems. First, if text from the original sender is available from any source stylometric traits can be easy to forge and second such approaches require sufficient data to infer minor differences in stylometry and can be computationally expensive. As a consequence, previous work often operates with small datasets. For example, Lin et al. [93] conduct an evaluation with only 6 senders due to a lack of available data. Similarly, Duman et al. [44] discriminate only 215 senders in their experiments. Whether these techniques can be scaled to cover thousands of senders is unclear and thus the application of stylometric features for spear-phishing detection is still an open issue.

The problem of limited learning data is addressed by Stringhini et al. [141] who propose a detection approach that, while also relying on email content, is capable of analyzing larger datasets. However, their method requires a minimum of 1,000 emails per sender to be effective. Moreover, they position the defense at the sender's server and require to include emails from different mailboxes to build a reliable behavioral profile of a user. Such an approach is thus orthogonal to our method which operates at the recipient's side, who only requires the information contained in her own mailbox to build an effective defense. Furthermore, recipient related features are based on the idea that

different recipients have different risk to get spear-phishing mails. Such features are proposed by Amin [2] which determine the amount of information returned by a search engine about a recipient and how often a person has received malicious mails in the past. Unsurprisingly, the latter turns out to be a dominant feature, i.e., those senders who got attacked a lot in the past will probably also get attacked a lot in the future.

As in our work, infrastructure related features commonly include properties of the transport like the senders IP address or her geographic location [93, 64]. But also features of the used mail client belong in this category since a sender will usually use only a single or few email clients. Features related to the infrastructure are often similar for all senders in the same domain which can be used to increase model accuracy when only a few mails from a specific sender are available. Compared to stylometric features, infrastructural features do not model the actual author but only her environment. Therefore, it is impossible to detect a hacked account with these features. On the other hand infrastructural features need less training data to create a well-performing model. Thus, it might be useful to combine the strength of both approaches.

Structural based features, instead of content based features are the dominant ones in our evaluation. Such features were already used by Amin [2]. Contrary to this work, our approach makes use of a larger set of features from the mail client and from its transport and is based on distinguishing different senders based on these features instead of globally distinguishing all spear-phishing mails from all benign mails.

Finally, a method recently proposed by Ho et al. [70] focuses on the identification of credential phishing and is designed to identify attacks from unseen senders. Our approach is orthogonal to this work, as it addresses two of its main shortcomings:

a) Ho et al. [70] considers the problem of address spoofing irrelevant due to the availability of DKIM and DMARC. Our empirical analysis, however, shows that both techniques are not widely available in practice and thus alternative methods are needed to achieve a sufficient protection from spear-phishing. Furthermore, DKIM and DMARC need to be implemented at the sending side, which enables the attacker to choose a known sender with lacking support for this security feature.

b) The proposed method requires the victim to interact with the phishing email by clicking on a link. This poses a serious security risk and may

result in the victim's host being compromised before the attack is actually detected. Our approach does not require interaction and can block phishing attacks before they reach their victim, for example, by removing links and attachments from emails.

## 2.6 Summary

Spear-phishing attacks using spoofed emails are still one of the most effective vectors for infiltrating companies and organizations and, the main strategy put in place by actors with large resources to successfully initiate a targeted attack.

Although several anti-spoofing techniques, such as SPF, DKIM and DMARC, exist, their low adoption in practice makes it easy for adversaries to construct seemingly authentic emails. Moreover, users targeted by spear-phishing attacks have little options for fending off these threats, as other protection mechanisms, such as digital signatures or behavioral modelling [141], need to be deployed at the sending side of the communication. As a consequence, there is an urgent demand for detection methods that help to spot spear-phishing as a means of thwarting most targeted attacks before any target can be compromised.

In this chapter, we show that a sender leaves several traits in the structure of an email, resulting from her personal preferences, email client and infrastructure. Based on these traits, we present a detection method that is capable of learning profiles for senders and identifying impersonated emails without relying on their content or server-side implementations. In an empirical evaluation with over 17,000 senders, we demonstrate that this method can identify over 90% of spoofed emails with less than 1 false alarm in 10,000 emails, if the attacker has no knowledge of the sender's profile. If the attacker has access to emails from the same domain as the spoofed sender our method still attains a detection rate of 78% and thus raises the bar for an adversary to effectively complete a spoofing attack.

Although our approach cannot detect an attack by an adversary with vast resources, it provides a strong protection from attackers that are not able to obtain original emails from a specific sender. In practice, our approach thus provides a valuable tool for fending off spear-phishing attacks that would go unnoticed without a proper anti-spoofing detection.

With all that, however, effective detection represents only the first step in a comprehensive strategy against targeted attacks. Once an attempt to compromise a target has been blocked, the analyst will proceed to investigate

and characterize the threat in order to understand its implications and find possible ways to mitigate future attacks. In the next chapter, we will explore the role of malware as a source of threat intelligence and propose strategies that will allow the analyst to link new samples found during the investigation of an attack with existing known malicious code at a large scale.

# 03

# Analysis

In the previous chapter, we have discussed how e-mail represents one of the most common and effective vectors to compromise a victim in a targeted attack. To effectively achieve this goal, attackers include links to malicious sites or attach malicious code in such carefully crafted e-mails. Through a malware infection the attacker is able to obtain persistence in the system first and then move laterally to compromise the network. Therefore, in addition to e-mail attachments, malware can reach the target system through background downloads from a malicious website or directly through the execution in the browser. From the perspective of the security analyst, the binary code of the malware represents a major source of intelligence about the attacker and it can help attributing the targeted attack to a known actor if the piece of malicious code presents some similarity with previously studied samples.

As the vast majority of newly discovered malware samples are variations of existing malware, detecting similarities to known malware has shown to be a promising approach [see 161, 35, 58]. Identifying variations of code, however, is an involved task as small changes at the source code may already have drastic effects on compiled code: instructions may be reordered, branches may be inverted or the allocation of registers may change [see 43]. To make matters worse, such changes are often introduced deliberately by malware to evade detection.

Researchers dealing with the detection of malware have discovered that high-level properties of code, in particular *function call graphs*, offer a suitably robust representation to account for these variations [82, 71]. However, working with graph representations for binary code, in general, and malware, in particular, introduces a series of specific challenges. In the first place, it is not trivial to obtain a graph representation for binary code that is able to effectively capture the subtleties of code behavior. At the same time, taking advantage of modern machine learning algorithms for detection and classification of malicious code requires a proper representation that allows efficient learning on graphs. Moreover, many learning algorithms learn on feature spaces that sacrifice explainability for accuracy, resulting in output decisions being made in a black-box fashion and thus, standing in the way of the security analyst to understand why a piece of code has been labeled as malicious or classified as a specific type of malware. Therefore, in the best case, a graph representation for malware classification should be robust to low level code modifications while being expressive, be efficient to compute and allow for a good classification performance without scarifying explainability.

In this chapter we tackle several of these issues and propose a combination of approaches that allows an analyst to effectively perform malware triage. In summary, we make the following contributions:

- **Generic labeling of binary functions.** We present a generic labeling scheme for binary code that enables us to construct labeled function call graphs without information about function names.

- **Explicit embedding of call graphs.** We derive a feature map inspired by graph kernels that allows for embedding function call graphs in a vector space capturing structural relationships.

- **Learning of implicit embedding of call graphs.** We learn an alternative low dimensional representation for call graphs through a deep neural network architecture that allows us to embed latent variable models into feature spaces using discriminative information.

- **Structural triage of x86 malware.** Both vectorial representations of function call graphs enable us to classify x86 malware with high accuracy using different machine learning algorithms. First, in a explicit and high dimensional space and second, in a implicit and low dimensional space with even better accuracy.

The rest of this chapter is structured as follows: we introduce the problem of malware triage, its challenges and the opportunities offered by the structural analysis of binary code in Section 3.1. Next, we present our learning approach based on explicit call graph embeddings in Section 3.3 and discuss in Section 3.4 how deep neural networks can be trained to learn an alternative graph embedding that, under certain constraints, can improve the performance of classification algorithms in a lower dimensional space. In Section 3.5, we empirically compare both approaches on a well known malware dataset discuss their limitations in Section 3.6 and related work in Section 3.7 with Section 3.8 concluding the chapter.

## 3.1 Structural Malware Triage

Malicious code is often repurposed and, not in few cases, even generated through automated modular software. As a result most of the newly discovered samples are variations of existing malware, making malware triage an essential strategy to analyze and gather further intelligence about an attack. Accordingly, security researchers have long strived to develop approaches for the identification of similarities between previously unseen malware samples discovered in the wild and known malware families. The same holds true for malicious code used in targeted attacks. Threats actors behind persistent campaigns typically invest large resources in developing specialized modules [8], which might be used again or simultaneously in several operations.

However, the problem of identifying similar behavior in binary malicious code presents certain inherent challenges. In the first place, the analyst requires a suitable representation for binary code that allows to measure similarity in an adversarial setting. Such a representation should be able to expressively represent code while being robust to modifications. Second and, considering the sheer amount of new malware samples discovered everyday, similarity approaches for malware triage should be able to deal with very large datasets in an efficient manner.

We address these challenges through two complementary methods that build on a static structural representation for binary code and the use of machine learning techniques. In particular, our first approach builds on the ideas originally introduced by Gascon et al. [56] to detect malicious applications in the Android platform. In our generic setup for graph classification, we show how the combination of a graph kernel and a convenient embedding in an

equivalent explicit vector space can be successfully applied to the problem of malware triage. As we will see, a hash-value is calculated over each node in a function call graph and its direct neighboring nodes, allowing occurrences of graph substructures to be effectively and explicitly enumerated. Then, samples are embedded using an explicit map inspired by the neighborhood hash graph kernel introduced by Hido et al. [69]. The map is designed such that evaluating an inner product in the feature space is equivalent to computing the respective graph kernel. Finally, using a linear machine learning classifier on such explicit feature space allows us to, not only capture structural relationships in the binary code, but also to explain what functions in the binary present the most characteristic behavior of a family.

This representation is therefore designed with a focus on explainability, however, it does present a trade-off where the high dimensionality of its feature space imposes certain limitations in terms of scalability. As an alternative, we build on our function call graph representation and propose a complementary approach for graph embeddings that leverages a state-of-the-art neural network architecture specially suited to learn efficiently on large structured data. As we will see, by making a compromise on explainability, our method allows to achieve an increased classification performance in a low-dimensional space.

In the following, we describe in detail our generic method for extraction and labeling of malware call graphs. Then, we discuss our approach for designing an explicit feature space for call graphs and subsequently, our approach for learning a low-dimensional feature representation that improves the performance of machine learning classifiers for malware triage.

## 3.2 Call Graph Extraction and Labeling

We begin implementing our methods by designing a structured representation based on binary function call graphs and generated through static analysis. Although it should be possible to generate call graphs dynamically, a static approach enables the analyst to model functionality of the binary that may not be executed at runtime and specially, avoid the high computational cost of performing dynamic analysis at scale.

Therefore, the first step of our method requires each malware binary to be disassembled and the identification of the calling dependencies between functions. In addition, nodes of the function call graph are labeled to characterize their content conveniently by short bit sequences.

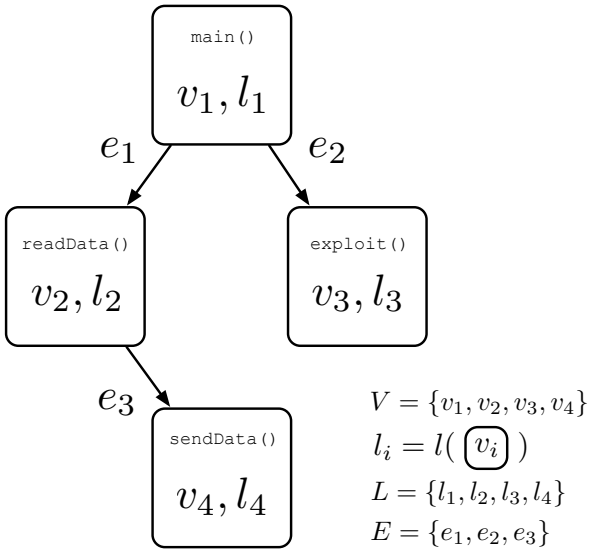



Fig. 3.1 Example of formal elements in a function call graph

Intuitively, the extracted function call graphs are directed graphs containing a node for each of the functions found in the binary and edges from callers to callees. Moreover, a *labeled* function call graph can be constructed, as shown in Figure 3.1 by attaching a label to each node. Formally, this graph can be represented as a 4-tuple $G = (V, E, L, \ell)$, where $V$ is a finite set of nodes and each node $v \in V$ is associated with one of the functions. $E \subseteq V \times V$ denotes the set of directed edges, where an edge from a node $v_1$ to a node $v_2$ indicates a call from the function represented by $v_1$ to the function represented by $v_2$. Finally, $L$ is the multiset of labels in the graph and $\ell : V \to L$ is a labeling function, which assigns a label to each node by considering properties of the function it represents.

The design of the labeling function $\ell$ is crucial for the success of our method. While in principle, a unique label could be assigned to each node, this would not allow the method to exploit properties shared between functions. By contrast, a suitable labeling function maps two nodes onto the same label if their functions share properties relevant to the detection task. Moreover, labeling must be robust against small changes in the code such as identifier renaming or branch inversion. To meet these requirements, we propose to label nodes according to the type of the instructions contained in their respective functions.

In particular, we rely on the specification of the intermediate language (IL) used by Radare2 and known as *Evaluable Strings Intermediate Language (ESIL)* [1]. ESIL is a low-level IL designed with the goal of evaluating and

emulating binary code for a wide range of architectures and whose instructions implicitly specify all side-effects.

Reviewing the ESIL specification, we define 38 distinct instruction categories based on their functionality as shown in Figure 3.2. Each node can thus be labeled using a 38-bit field, where each bit is associated with one of the categories.

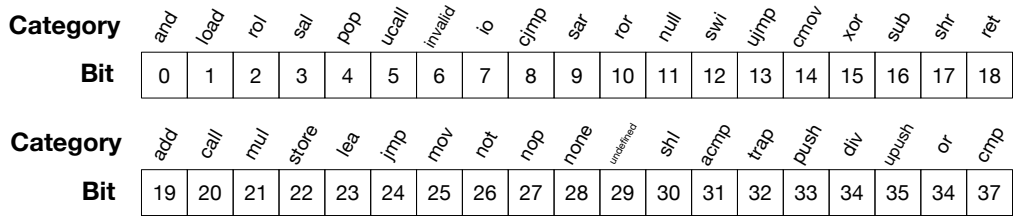



Fig. 3.2 ESIL instruction categories and their corresponding bit in the label assigned to each node.

Formally, the function $\ell$ can be defined as follows: We denote the set of categories by $\mathcal{C} = \{c_1, c_2, \cdots, c_m\}$ and the function associated with a node $v$ by $f_v$. The label $\ell(v)$ of a node $v \in V$ is then a bit field of length $m$, i.e., $\ell(v) = [b_1(v), b_2(v), \cdots, b_m(v)]$ where

$$b_c(v) = \begin{cases} 1 & \text{if } f_v \text{ contains an instruction from category } c \\ 0 & \text{otherwise.} \end{cases}$$

Consequently, the set of labels $L$ is given by a subset of all possible 38-bit sequences. As an example, Figure 3.3 shows the disassembled code of a function and the categories assigned to each of its instructions. Note that the function contains the `jg` and `je` instructions, which are used to conditionally jump to another address after a comparison. These instructions are part of a set of instructions denoted as `cjmp` and associated with the eight bit of the label. The eight bit is therefore set in the resulting function label.

## 3.3 Explicit Graph Embeddings for Malware

Based on our function call graph representation, we aim at designing an approach that allows us to train a machine learning algorithm for malware classification. We have two goals in particular. First, we want to obtain a numerical vector representation that captures the behavior of a function and its environment. Second, it should be possible to move from the decisions of

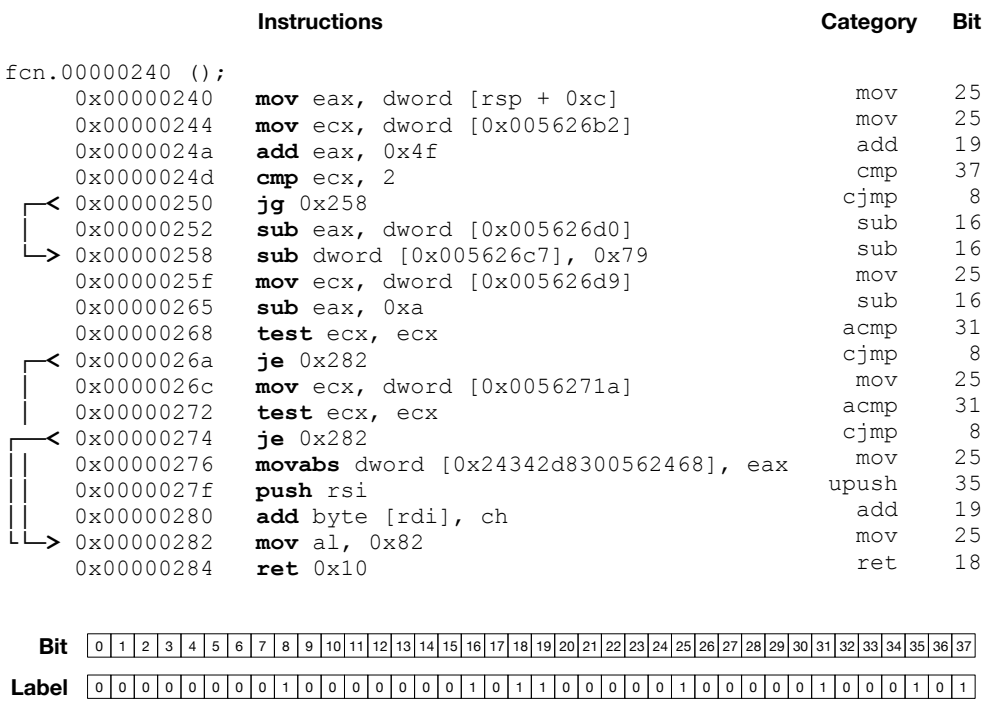



Fig. 3.3 Labeling example of a function from its code. Every opcode belongs to a category, which is a associated to a certain bit in the label.

the classifier back to the input feature space. This would allow the analyst to dig into the functioning of the classifier and better understand the behavior of a malware sample.

Therefore, we want to identify subgraphs of the function call graph representing code of a characteristic malware family. This is, however, not trivial, in particular since no polynomial time solution exists to test whether two graphs are isomorphic. In consequence, several solutions have been developed for inexact matching. Some of these methods rely on suboptimal strategies such as the graph edit distance [71] or the identification of maximum common subgraphs, while other ad hoc solutions propose the serialization of the graph structure [20] as a way to measure similarity. In most setups, this similarity metric is later used in a neighbor search to identify close candidates to a test sample.

Graph kernels have emerged as a solution to let kernel-based machine learning algorithms operate efficiently in the graph space. A graph kernel is, in short, a kernel function that computes an inner product on graphs. These kernels have been proposed at several occasions to address graph classification

problems in chemistry and bioinformatics, however, their applicability to static malware analysis remains largely unexplored.

In the following, we introduce the different steps of our approach to efficiently build an explicit embedding for function call graphs.

### 3.3.1 Hashing of Neighborhoods

Upon labeling nodes in the function call graph, each function is characterized by the instructions it contains. However, our method strives to model the composition of functions and thus the neighborhood of a function must be taken into account. To this end, for each node, we compute a *neighborhood hash* over all of its direct neighbors in the function call graph, a procedure inspired by the neighborhood hash graph kernel (NHGK) originally proposed by Hido and Kashima [69].

The NHGK is a so called decomposition kernel as defined by Haussler [67]. As such, it is a kernel operating over an enumerable set of subgraphs in a labeled graph. It has low computational complexity and high expressiveness of the graph structure, but its main advantage is that it is able to run in time linear in the number of nodes and can therefore process graphs with thousands of nodes such as the function call graphs of malware binaries.

The main idea behind the NHGK is to condense the information contained in a neighborhood into a single hash value. This value is calculated over the labels of a neighborhood and represents the distribution of the labels around a central node. It thus allows us to enumerate all neighborhood subgraphs in linear time without running an isomorphism test over all pairs of neighborhoods.

The computation of the hash for a given node $v$ and its set of adjacent nodes $V_v$ is defined by the operation

$$h(v) = r(\ell(v)) \oplus \left( \bigoplus_{z \in V_v} \ell(z) \right) \tag{3.1}$$

where $\oplus$ represents a bit-wise XOR on the binary labels and $r$ denotes a single-bit rotation to the left. This computation can be carried out in constant time for each node, more specifically in $\Theta(md)$ time where $d$ is the maximum outdegree and $m$ the length of the binary label.

The *neighborhood hash* of a complete graph $G$ denoted by $G_h = h(G) = (V, E, L_h, h(\cdot))$ is then obtained by calculating hashes for each node individually and replacing the original labels with the calculated hash values. This creates

an additional linear dependence of the computation time on the number of nodes in the graph. Furthermore, it can be noted that the hash values have the same length $m$ as the original label. However, they aggregate information spread across neighboring nodes. Moreover, the hash values are independent of the actual order in which children are processed, and thus sorting is not necessary.

Hido and Kashima also consider applying the neighborhood hash iteratively to aggregate information across neighbors up to a path length $p$. The neighborhood hash of order $p$ can then be defined recursively as $G^{(p+1)} = h(G^p)$. Choosing $p$ larger than one still allows to construct a valid decomposition kernel, however, higher values of $p$ also lead to an increased number of overlapping substructures.

Since we are particularly interested in designing an explicit representation of the kernel feature space that is easy to interpret by analysts, we thus fix $p = 1$ in order to limit the complexity of the feature space.

### 3.3.2 Feature Space Embedding

Some graph kernels are designed to operate only on unlabeled graphs or are unable to be evaluated on graphs with more than a few hundreds of nodes. Moreover, many of these kernels induce only an implicit feature space, which makes it impossible to determine the features predominantly responsible for the classification of a sample.

The use of graph kernels for the task of malware triage allows to abstract the code into a representation that enables learning its underlying structure. However, function call graphs have thousands of nodes and are characterized as directed labeled graphs. Therefore, it is necessary to apply a graph kernel that can not only deal with these specificities, but can also operate on a large number of nodes efficiently.

The neighborhood hash graph kernel function, evaluates the count of common identical substructures in two graphs, which after the hashing, is the number of shared node labels. Considering that several nodes can be labeled with the same hash, the kernel value can be represented as the size of the intersection of the multisets $L_h$ and $L'_h$ for two function call graphs $G_h$ and $G'_h$, that is,

$$K(G_h, G'_h) = |L_h \cap L'_h| \tag{3.2}$$

For the specific application of malware analysis, our goal is to find an explicit representation that is equivalent to that of the graph kernel. In this vector space, a linear SVM can be used to learn a model that is able to (a) classify samples into different families and (b) allows for an interpretation of its decisions. In order to achieve this, we abstain from using the implicit kernel function $K$, but instead embed every sample in a feature space whose inner product is equivalent to the graph kernel.

To this end, we start by considering the histogram of the multiset $L_h$ as $H = \{a_1, a_2, \cdots, a_N\}$, where $a_i \in \mathbb{N}$ indicates the occurrences of the $i$-th hash in $G_h$. The number of shared elements between two multisets can be calculated by sorting all elements of a certain type and counting the minimum number of elements of this type that are present in both multisets. This is known as the multiset intersection. If the size of the intersection of two histograms $H$ and $H'$ of length $N$ is defined as

$$S(H, H') = \sum_{i=1}^{N} \min(a_i, a'_i) \tag{3.3}$$

it becomes apparent that the kernel defined in Eq. (3.2) can be also phrased using the intersection of the histograms for two graphs $G_h$ and $G'_h$ as

$$K(G_h, G'_h) = S(H, H'). \tag{3.4}$$

Barla et al. [6] show that this histogram intersection can be indeed adopted in kernel-based methods and propose a feature mapping, such that $S$ is an inner product in the induced vector space. For this purpose, each histogram $H$ is mapped to a $P$-dimensional vector $\phi(H)$ as follows

$$\phi(H) = \left( \underbrace{\overbrace{1, \cdots, 1}^{a_1}, \overbrace{0, \cdots, 0}^{M-a_1}}_{\text{bin } 1}, \cdots, \underbrace{\overbrace{1, \cdots, 1}^{a_N}, \overbrace{0, \cdots, 0}^{M-a_N}}_{\text{bin } N} \right) \tag{3.5}$$

where $M$ is the maximum value of all bins in the dataset, $N$ is the number of bins in each histogram and $P = NM$ is the dimension of the vector.

In this representation, each bin $i$ of the histogram is associated with $M$ dimensions in the vector space. These dimensions are filled with 1's according to the value of $a_i$, whereas the remaining $M - a_i$ dimensions are set to 0. As a result, the sum of the $M$ dimensions associated with the $i$-th bin is equal to $a_i$ and moreover the sum of all dimensions in $\phi(H)$ is equal to the sum of all bins in the histogram.

By putting the different steps together, we can finally show that the inner product in the vector space induced by Eq. (3.5) indeed resembles the neighborhood hash graph kernel given in Eq. (3.2). That is, we have

$$K(G_h, G'_h) = S(H, H') = \langle \phi(H), \phi(H') \rangle. \tag{3.6}$$

The interested reader is referred to the original work of Barla et al. [6], which provides a more detailed analysis of histogram intersections and this mapping.

The mapping $\phi$ finally allows us to embed every call graph in a feature space, where a linear SVM can be used for efficiently learning with hundreds of thousands of graphs each containing thousands of nodes and edges.

### 3.3.3 Learning and Feature Analysis

As discussed in Section 3.1, malware plays a central role for the analyst when trying to attribute a targeted attack. Given a series of known malware families, where all samples in a family share a similar behavior, the problem of triaging new malware can be posed as a multiclass classification problem for $C$ classes and solved by means of a linear SVM, which learns a linear separation with a maximum margin [46] of the given classes in a one-vs-all fashion. Following this strategy, one linear classifier is fitted per class against the rest of $C - 1$ classes, allowing for computational efficiency and interpretability.

This approach results in the algorithm learning $C$ classifiers. Each one with a decision function of the linear SVM $f_c$ for $c \in \{1, \ldots, C\}$ and given by

$$f_c(G_h) = \langle \phi(H), w_c \rangle + b_c, \tag{3.7}$$

where $w_c \in \mathbb{R}^P$ is the direction of the hyperplane for class $c$ and $b_c$ the offset from the origin of the vector space. In this setting, a function call graph $G_h$ is assigned to the class corresponding to the classifier with the highest confidence score. That is

$$\hat{y} = \underset{c \in \{1,\ldots,C\}}{\operatorname{argmax}} f_c(G_h) \tag{3.8}$$

In order to identify what substructures of $G_h$ contribute to this decision, it is necessary to reverse the expansion performed in Eq. (3.5). In particular, we compute an aggregated weight $\hat{w}_c^i$ for each bin $i$ of the histogram (corresponding to the $i$ hash value of the graph $G_h$). Formally, this is achieved for the $i$-th bin

of the histogram as follows

$$\hat{w}_c^i = \sum_{j=iM}^{(i+1)M} w_c^j. \tag{3.9}$$

The largest of these aggregated weights allows us to highlight those neighborhoods in a given graph $G_h$ that predominantly influence the decision and can be interpreted as typically belonging to that class. That is, if the weight $\hat{w}_c^i$ of the $i$-th bin is large, all nodes labeled with the corresponding hash value significantly contribute to the decision of the SVM and thus likely reflect the distinct functionality of the malware family represented by the corresponding class.

## 3.4 Learning Graph Embeddings for Malware Classification

In the previous section, we have discussed how our representation based on function call graphs allows us to triage malware samples by leveraging the structural relations in their binary code. Moreover, we discuss how to build an explicit feature space for learning on graphs which, in combination with a linear machine learning classifier, enables the analyst to recover the original input space and therefore better understand its output decisions.

This approach, however, presents a trade-off between explainability and complexity due to the fact that the resulting feature space is very high dimensional and needs to be expertly designed beforehand. Hence, in this section, we aim at designing an alternative method that can work at the other end of this trade-off, without sacrificing performance and also operating directly on graphs.

To this end, we propose a method based on deep neural networks that allow us to learn a low-dimensional representation for call graphs. In particular, we rely on Dai's *structure2vec* [37], an approach based on the idea of embedding latent variable models into feature spaces, and learning such feature spaces using discriminative information. As we will see, using this approach in combination with a *siamese network* configuration, let us build a trainable system that maps the function call graphs to a low-dimensional space where the distance between samples of the same family is small and large otherwise.

In the following, we explain in detail how we use these techniques to learn function call graph embeddings for malware classification and then, proceed to compare both approaches and their performance through different experiments in Section 3.5.

### 3.4.1 Graph Embedding Network

As argued by Dai et al. [37], kernel methods have achieved state-of-the-art performance when used in combination with standard machine learning classifiers. However, they suffer from certain limitations. For instance, the success of kernel methods on structure data relies on the expert design of the kernel function. As described in our explicit approach in Section 3.3, this class of kernels are designed around the idea of "bag of structures", where each structured data point is represented as a vector of counts of substructures. Therefore, the feature space defined by these kernels is fixed before learning with each dimension corresponding to a substructure, independent of the supervised learning problem and without allowing to take advantage of any discriminative information available. In addition, the number of substructures in structure data is typically large, as is the case of function call graphs in binaries, resulting this approach in very high dimensional spaces. Moreover, learning algorithms operating on pairwise kernel values require the kernel matrix to be computed in advance. Having this a square dependency with the number of samples, it is computational expensive for these methods to scale up to very large datasets.

Dai's *structure2vec* algorithm addresses to certain extent some of these issues by modeling each graph as a latent variable model and embedding the graphical model into a feature space which is learned by minimizing an empirical loss defined by label information.

In a similar fashion to the neighborhood hash approach described in Section 3.3.1, *structure2vec* aggregates node features recursively by following the topology of the graph. A $d$-dimensional feature vector $\mu_v$ for each node is initialized to zero and then updated through $I$ iterations. After all iterations the feature vector of each node contains information of its neighborhood up to a depth $I$. In contrast to the neighborhood hash algorithm, however, neighborhood information is aggregated through a non-linear mapping that operates iteratively on the original label $l(v)$.

Algorithm 1 describes in detail the steps to generate the mapping $\phi_W^{s2v}$. This mapping receives a call graph $g$ as input and is modeled as a neural network characterized by the set of parameters $W := \{W_1, W_2, W_3\}$. Accordingly, $W_1$

**Algorithm 1** Parameterized Graph Embedding

**Input:** $g = (V, E, L, \ell)$
**Output:** $\phi_W^{s2v}$

1: $\mu_v^{(0)} \leftarrow 0, \forall v \in V$
2: **for** $i = 1$ **to** $I$ **do**
3: **for** $v \in V$ **do**
4: $\mu_v^{i+1} = \sigma(W_1 l(v) + W_2(\sum_{z \in V_v} \mu_z^i))$
5: **end for**
6: **end for**
7: **return** $\phi_W^{s2v} := W_3(\sum_{v \in V} \mu_v^I)$

is a matrix with dimensions $l \times d$, being $l$ the size of the node label and $d$ the dimension of the final feature space. $W_2$ and $W_3$ have dimensions $d \times d$ and $\sigma(\cdot)$ is a rectifier linear unit, such that $ReLU(x) = \max\{0, x\}$.

In the following, we discuss the architecture and optimization setup that allow us to obtain the values for $W$ that minimize and maximize intraclass and interclass distances respectively in the resulting low-dimensional feature space.

### 3.4.2 Parameterization with Siamese Architecture

Based on an original idea from Baldi et al. [5], the *siamese network* architecture was proposed by Bromley et al. [14] in 1993 for verifying signatures on a pen-input tablet. It employs two identical neural networks whose inputs are used to compare two samples with one output that indicates the similarity between them. In 2005, Hadsell and Chopra [62, 26] introduced an approach that relies on this architecture for dimensionality reduction by learning an invariant mapping that leverages discriminative information from the input space.

Adapting their vector notation to our graph representation, we can formalize the problem of finding a function that maps function call graphs to a lower dimensional space as follows. Given a set of input graphs $\mathcal{G} = \{G_1, \ldots, G_n\}$ we aim at finding a parametric function $\phi_W : \mathcal{G} \longrightarrow \mathbb{R}^d$ with $d \ll P$, where $P$ is the dimension of the explicit feature vector $\phi(H)$ in equation 3.5. The mapping $\phi_W$ should allow for distance measures in the output space to approximate the relationships in the input space trough invariances to complex transformations and being able to generalize for graphs yet unseen. That is, function call graphs that belong to the same family should be mapped to nearby points in the output manifold or to distant points otherwise.

Taking the euclidean distance as the metric to be accordingly minimized or maximized in the output space, Hadsell et al. introduce the *contrastive loss* function, whose minimization can produce the mapping $\phi_W$ without reaching a collapsed solution. Being $D_W$ the parameterized distance function to be learned

$$D_W(G_1, G_2) = \|\phi_W(G_1) - \phi_W(G_2)\|_2 \tag{3.10}$$

the loss function is defined by

$$\begin{aligned} &L(W, Y, G_1, G_2) = \\ &(1-Y)\frac{1}{2}(D_W(G_1, G_2))^2 + (Y)\frac{1}{2}\{max(0, m - D_W(G_1, G_2))\}^2 \end{aligned} \tag{3.11}$$

where $m > 0$ is a radius around $\phi_W(G)$, so that only dissimilar pairs with a distance within this radius contribute to the loss function.

Because we want our approach to operate directly in the graph space at the input, we design a siamese architecture as shown in Figure 3.4, where the two copies of $G_W$ are *structure2vec* networks that share the same set of parameters $W$. The output of the siamese network is used as input for the loss function.

Then, in order to train the network, we define a *learning* set of graphs and pair each one of them will all the rest. The resulting pairs are then labeled $Y = 0$ if they belong to the same family or $Y = 1$ otherwise. The pairs of graphs are fed to the network and the contrastive loss is computed as a function of the expected label, updating the parameters of the network $W$ trough stochastic gradient.

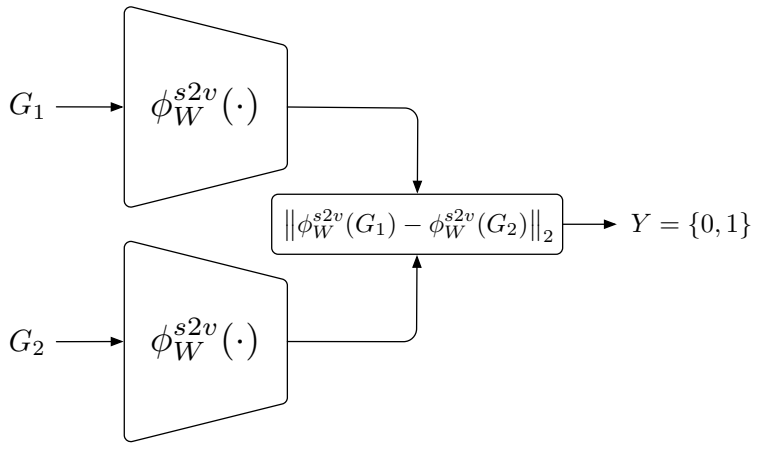



Fig. 3.4 Siamese architecture with *structure2vec* networks as function $\phi_W$.

Finally and once that the parameters that minimize the loss function are found, we can embed a new set of function call graphs by feeding each one of

them to the trained network $\phi_W^{s2v}$. The resulting representation can be used then to train and test a classifier as described in Section 3.4.

In the following, we evaluate both methods for explicit and learned embeddings for binary function call graphs on a well labeled malware dataset and compare their performance and their trade-offs for malware triage and novelty detection.

## 3.5 Evaluation

In this section, we proceed to evaluate how our representation based on function call graphs and the two explicit and learned graph embeddings can be used in combinations with a series of machine learning algorithms to perform malware triage in a well known and labeled dataset.

In particular, we explore how the analyst can cluster unknown malware samples in the absence of labels using our explicit embedding. Then, we evaluate how clustering results can be improved by labeling some samples and characterizing them through a learned embedding. Next, we proceed to evaluate the performance of several algorithms on both embeddings in a supervised classification setup, where labeled data is available to the analyst and finally, we tackle the problem of identifying an unseen sample as part of a known family or as the first member of a new class through anomaly detection.

In the following, we begin introducing our dataset and discussing how we generate function call graphs from the malware binaries. Then, we continue by creating different data splits that we use to train and test the classifiers and, in the case of our learned embedding, to learn the corresponding feature space.

### 3.5.1 Dataset

To demonstrate how our approaches can help at the task of malware triage, we focus on binaries that target the Windows platform and evaluate a series of machine learning algorithms on both embedded representations for function call graphs.

Therefore, we require a suitable dataset that includes malware from different families. Taking into account that the building of a proper malware dataset represents a research problem of its own, we rely in this work on a dataset that, not without certain constraints, includes high quality family labels assigned manually by analysts.

Table 3.1 Malware families in the Microsoft Malware Classification dataset

| ID | Family Name | Samples | Type |
|---|---|---|---|
| 1 | Ramnnit | 1541 | Worm |
| 2 | Lollipop | 2478 | Adware |
| 3 | Kelihos_ver3 | 2942 | Backdoor |
| 4 | Vundo | 475 | Trojan |
| 5 | Simda | 42 | Backdoor |
| 6 | Tracur | 751 | TrojanDownloader |
| 7 | Kelihos_ver1 | 398 | Backdoor |
| 8 | Obfuscator.ACY | 1228 | Obfuscated (various types) |
| 9 | Gatak | 1013 | Backdoor |

In the context of the *Microsoft Malware Classification Challenge* [128], Microsoft made available a dataset of nearly 0.5 terabytes containing the disassembly and bytecode of more than 20,000 malware samples. Table 3.1 lists the different families and their sample sizes together with the type of malware. In the first column, we assign an ID to each family that we will use to identify each class in the following experiments.

### 3.5.2 Generation of Function Call Graphs

In order to generate individual function call graphs for the binaries in the dataset, we rely on the Radare2 framework [1]. For this, we make use of Radare2 Python interface to analyze the provided bytecode files and identify the corresponding cross-references between functions.

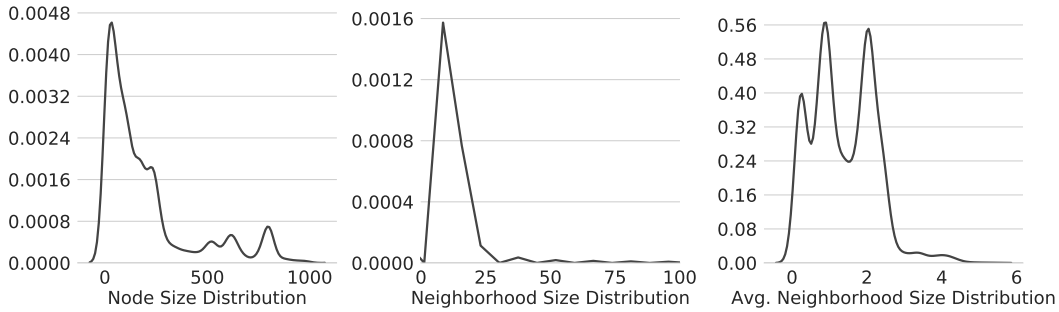



Fig. 3.5 Probability distributions of the number of nodes in function call graph, the number of nodes in a neighborhood in all graphs and the average size of a neighborhood in a graph.

The plots in Figure 3.5 depict the probability density function of several graph characteristics, what provide an idea of the shape and size of the function call graphs and their substructures. For instance, it can be observed how most

of the graphs contain less than 500 nodes and how the number of nodes in a neighborhood tends to be less than 25 for all graphs. If we consider each graph individually, the last plots shows how the average size of its neighborhoods stays under 4 nodes.

### 3.5.3 Function Call Graph Embeddings

In order to evaluate a series of machine learning classifiers on our representations and compare their performance, we split the dataset into `learning`, `validation`, `training` and `testing` sets.

As described in Section 3.4.2, we proceed to train a siamese network in order to obtain the parameters for the mapping $\phi_W^{s2v}$. To this end, we use the `learning` partition to train the neural network and the `validation` partition to select the combination of parameters that achieves the best performance on unseen data. Figure 3.6 shows the evolution of the loss on training and validation data in each epoch. It can be observed how further from the epoch where the best performance on validation data is obtained, the networks keeps improving the performance on training data but increasing the loss on validation data due to overfitting.

A usual concern is the need to gather enough data to train a neural network effectively. However, note that each input to the network is in this case a pair of graphs resulting from the combination of all graphs in the learning set. In particular the total number of pairs $\mathcal{P}$ that can be fed to the network is given by

$$\mathcal{P} = \frac{n!}{r!(n-r)!} \underset{r=2}{\rightarrow} \frac{n(n-1)}{2} \tag{3.12}$$

where $n$ is the number of graphs in the `learning` set, $r$ is the size of the unordered subsets (i.e. $r = 2$) and thus $n \ll \mathcal{P}$, resulting in a relatively large amount of input pairs.

Once we have determined the parameters for the mapping $\phi_W^{s2v}$ that minimize the distance between samples from the same family and maximize the distance between samples otherwise, we use this mapping to embed the `training` and `testing` sets. As discussed in Section 3.4.1, the mapping $\phi_W^{s2v}$ maps a graph to a vector of dimension $d$, being the dimensionality of the output space a parameter that can be found through cross-validation. In this work, we set $d$ to 1024, assuring thus that $d \ll P$, where $P$ is the final dimensionality of the neighborhood hash embedded feature space. Likewise, we follow the methodol-

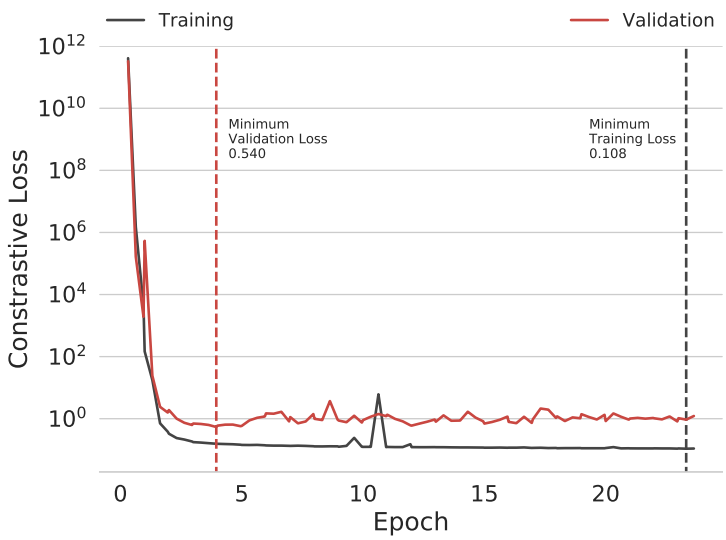



Fig. 3.6 Evolution of the training and validation loss per epoch.

ogy described in Section 3.3.2 and generate embeddings for the `training` and `testing` sets using the neighborhood hash.

Up to this point, training an testing sets of function call graphs are embedded in the feature spaces determined by both our neighborhood hash approach (NH) and *structure2vec*-siamese network approach (S2VSN). In the following we proceed to evaluate how each representation enables different algorithms to perform on clustering, classification and anomaly detection.

### 3.5.4 Clustering

In the absence of family labels, the analyst needs to rely on unsupervised methods to identify similarities between unseen samples while, in certain cases, it might be possible for the analyst to invest resources into initially labeling some data. In this section we begin thus by exploring how a clustering algorithm performs on our explicit graph representations when no labeled data is available. If enough labeled data exists, however, the analyst can improve the clustering performance through a semi-supervised approach by representing the data through a low-dimensional implicit embedding as described in Section 3.4.

In particular, we proceed first to obtain a 2-dimensional visualization of our data in both cases, which helps understanding how each embedding strategy shapes the training and testing sets. To this end, we make use of the *t-Distributed Stochastic Neighbor Embedding* (t-SNE) [96] algorithm, a technique

for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets.

Figure 3.7 shows the t-SNE manifold for each embedding in 2 dimensions. It can be observed how while the NH embedding already allows for part of certain classes to be separable, the S2VSN embedding manifold allows for complete classes to be clearly separable.

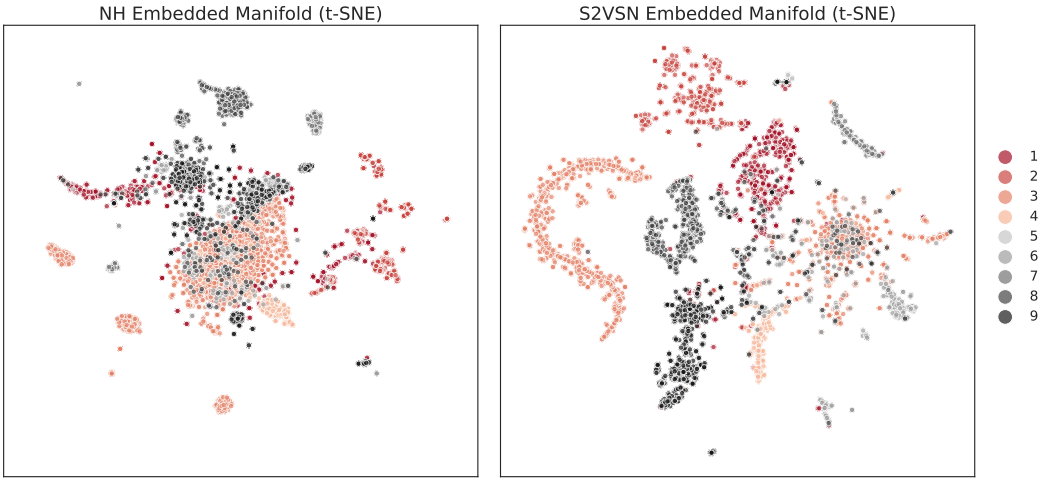



Fig. 3.7 t-SNE representation of `training` and `testing` NH and S2VSN embedded manifolds

Next, we proceed to evaluate the performance of the KMeans clustering algorithm on a set composed by our `training` and `test` partitions. Our algorithm selection its motivated by the fact that KMeans can both efficiently operate on a large number of samples and does not require input parameters other than the cluster size. Therefore, to measure the clustering performance on both embeddings, we make use of the performance metrics provided by the clustering performance evaluation module in the *Scikit-Learn* [112] machine learning toolbox. In particular, we proceed to evaluate KMeans on our embedded datasets and compute the following performance metrics (see [112, 129]) for different values of K:

- **Homogeneity**: The Homogeneity metric is bounded between 0 and 1, and indicates that the clusters of a clustering result only contain data points which are members of a single class.

- **Completeness**: A clustering result satisfies completeness, which is also ranged between 0 and 1, if all the data points that are members of a given class are assigned to the same cluster.

- **V-measure**: The V-measure is the harmonic mean between homogeneity and completeness and its score ranges between 0 and 1, where 1 indicates a perfectly complete labeling.

- **Adjusted Rand Index**: The ARI computes a similarity measure between two clusterings by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in the predicted and true clusterings. This metric takes values between -1 and 1, where random labelings have an ARI close to 0 and 1 indicates a labeling without errors.

- **Adjusted Mutual Info**: An AMI of 1 indicates that two clustering result are perfectly matched. When the labelings are independent, as is the case of random partitions, the expected AMI is around 0 on average and can therefore be negative.

- **Silhouette Coefficient**: This metric is defined for each sample and allows to evaluate a cluster without any ground truth. It indicates how well defined the clusters are and takes values between -1 and 1, where values near 0 indicate that clusters are overlapping and where negative values generally indicate that a sample has been assigned to the wrong cluster.

Figure 3.8 illustrates how the KMeans algorithm performs at clustering both embedded datasets given the cluster size $K$ as parameter. It can be observed how the S2VSN embedded set can be clustered in general with a better performance when the number of clusters is fixed to $K = 13$. Note that from all the performance metrics computed, the silhouette coefficient is calculated as an average of the silhouette coefficient of each sample and without the need to rely on ground truth. Therefore, in order to estimate the number of clusters present in an unlabeled malware dataset, the analyst can begin by computing the silhouette coefficient for different values of $K$ and selecting that value that achieves the best results. In this setup, while the maximum silhouette coefficient for the NH embedding would not result in similar good values for the rest of the metrics, the maximum of the silhouette coefficient for the S2VSN embedding is reached with $K = 12$, matching the results observed with the other metrics and close to the real number of 9 clusters in the dataset. Noticeably, both embeddings allow for the coefficient to remain positive on the range of number of clusters evaluated.

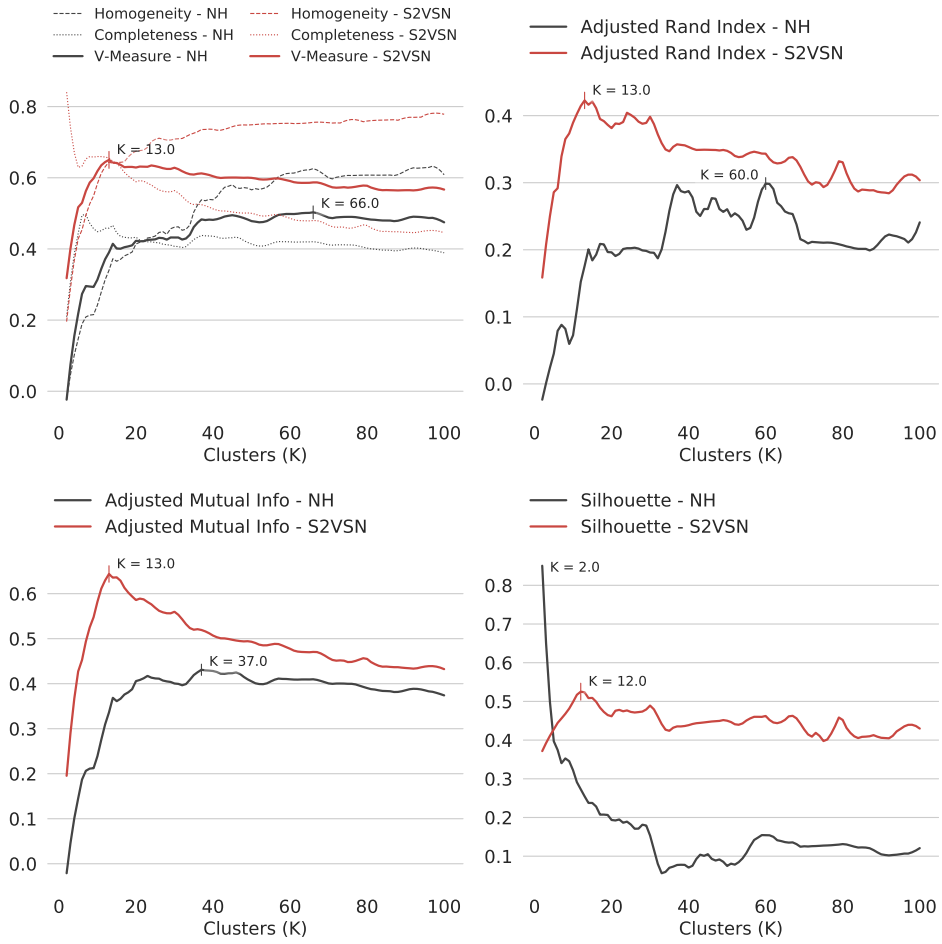



Fig. 3.8 Clustering metrics obtained with KMeans as a function of the cluster size K.

In the following, we evaluate how a series of learning algorithms perform in a supervised setup, both in a multiclass classification problem and an anomaly detection setup, where each individual class is considered as an outlier respect to the rest of the classes.

### 3.5.5 Multiclass Classification

In the most common scenario, the analyst observes a new sample and tries to find the closest known family in a supervised fashion. We evaluate thus four machine learning classifiers which can operate efficiently on large scale data in a multiclass classification setting.

To this end, we train and compare the algorithms *logistic regression (LR)*, *linear support vector machines (LSVM)*, *random forest (RF)* and *gradient boosted trees (XGB)*.

We find parameter values for all classifiers through cross-validation on the training set and compute the performance metrics for classification on the test set. Figure 3.9 illustrates the accuracy and the F1-score achieved by the different classifiers on the different embeddings and Table 3.2 shows in detail the average metric values and their standard deviation.

The accuracy indicates the total percentage of samples that are classified correctly during testing and the F1-score is defined as the harmonic mean of precision and recall. As our dataset present certain imbalance between classes, we compute the F1-score with both *micro* and *macro* averages. In the first case, the score is obtained by counting the total true positives, false negatives and false positives. In the second case, the score is first calculated for each class without taking into account the label imbalance and the unweighted mean of all classes is reported as final score.

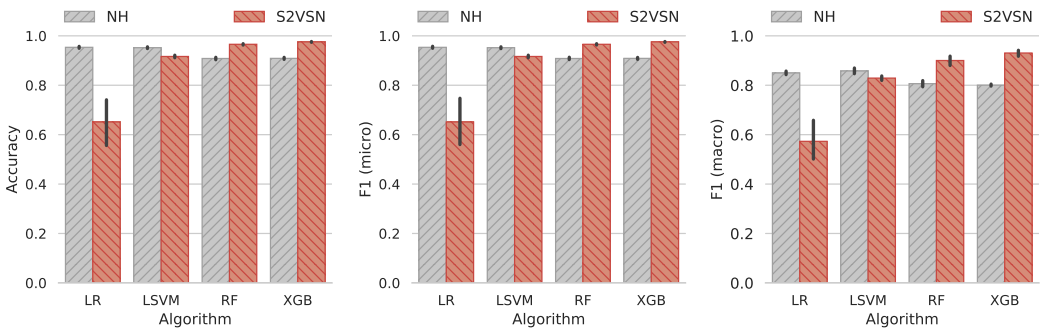



Fig. 3.9 Multiclass performance metrics for a classification algorithms in a multiclass classification setup.

Table 3.2 Average and standard deviation values of performance metrics for classifiers in Figure 3.9

| Algorithm | Embedding | Accuracy | F1 (micro) | F1 (macro) |
|---|---|---|---|---|
| LR | NH | 95% ± 1% | 95% ± 1% | 85% ± 1% |
| | S2VSN | 65% ± 16% | 65% ± 16% | 57% ± 14% |
| LSVM | NH | 95% ± 1% | 95% ± 1% | 86% ± 2% |
| | S2VSN | 92% ± 1% | 92% ± 1% | 83% ± 1% |
| RF | NH | 91% ± 1% | 91% ± 1% | 81% ± 2% |
| | S2VSN | 97% ± 1% | 97% ± 1% | 90% ± 3% |
| XGB | NH | 91% ± 1% | 91% ± 1% | 80% ± 1% |
| | S2VSN | **98% ± 0%** | **98% ± 0%** | **93% ± 2%** |

Both Figure 3.9 and Table 3.2 let us draw several conclusions. First and, as demonstrated by Gascon et al. [56], the explicit graph representation based on

the neighborhood hash kernel used to train different machine learning classifiers let us obtain a high classification performance for call graphs according to the accuracy and F1-score achieved for instance, by the logistic regression and linear SVM algorithms.

Nonetheless, the S2VSN embedding can improve performance in certain cases by allowing the algorithms to learn in a low-dimensional space. In particular, the performance of the random forest and gradient boosted trees algorithms improves when trained on the S2VSN embedded dataset. Furthermore, this representation allows for the XGB classifier to obtain the best performance overall.

We also observe how the performance decreases when the F1-score is computed using a *macro* average, indicating that the F1 score presents an uneven distribution across classes. To better understand how each family is characterized by each classifier in every case, we visualize in Figure 3.10 the confusion matrices for each algorithm and embedding.

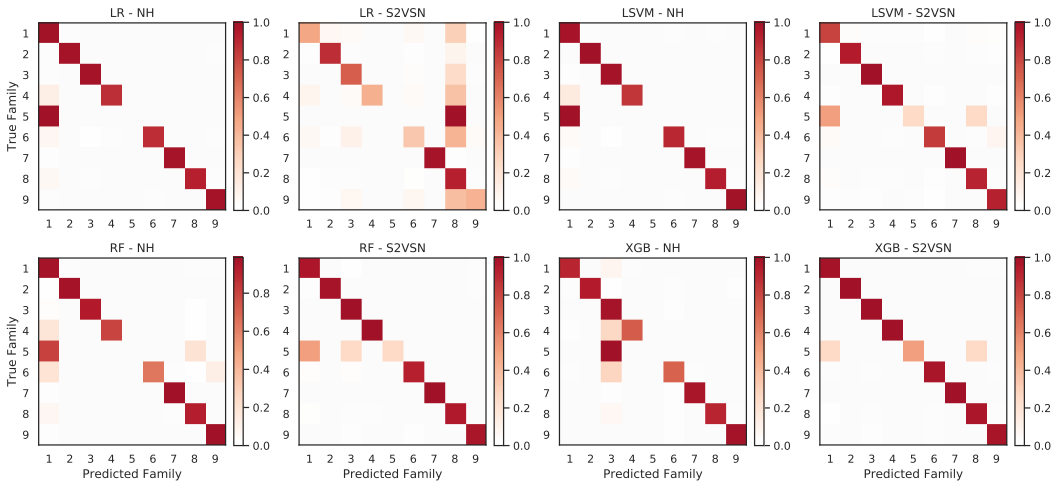



Fig. 3.10 Confusion matrices for each classification algorithm and embedding.

Although the results are consistent with those obtained for all algorithms and representations, as shown in Figure 3.9, certain significant patterns can be observed. Specifically, we see that class 5 is particularly difficult to identify regardless of the algorithm and representation used, resulting in a lower F1-score when computed with a *macro* average. Such result is however, consistent with the fact that class 5 contains the least number of labeled samples, as enumerated in Table 3.1.

In the following, we explore how the different classifiers can be used to identify samples of each class as a new family when such a family has not been observed before.

### 3.5.6 Anomaly Detection

We have discussed in the previous section how a malware analyst can assign a newly observed sample to a known family. However, it is not unusual for the analyst to observe a sample that belongs to an entirely new campaign and therefore a new family.

In such scenario, we would like the analyst to be able to identify the new sample as an outlier and use this outlier as the base to define a new family. To make this possible, we make use of the machine learning classifiers from the previous section in a setup where the score assigned to the most probable class is compared with a threshold that indicates if the decision made by the classifier has high confidence or the sample can be classified as an outlier from a new class.

Based again on the `training` and `testing` sets and the corresponding embeddings, we train the classifiers by leaving the outlier class out during learning and combining samples from all classes during testing. Figure 3.11 and Table 3.3 show the performance of the classifiers in this setup as a trade-off between the *outlier detection rate (ODR)* and the *inlier misdetection rate (IMR)*. That is, a sample from a known family is considered to be an inlier and a sample from an unknown family should be recognized as an outlier. Similar to the results obtained in the previous sections, we observe that the S2VSN embedding provides an advantage over the neighborhood hash embedding when used to train the random forest and gradient boosted trees algorithms. Results in Table 3.3 show, however, that if we limit the inlier misdetection rate to low values, the random forest classifier achieves the best performance in general.

Table 3.3 Outlier detection rates for specific values of the inlier misdetection rate in the trade-off curves depicted in Figure 3.11

| **Algorithm** | | LR | | LSVM | | RF | | XGB | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Embedding** | | NH | S2VSN | NH | S2VSN | NH | S2VSN | NH | S2VSN | |
| **IMR** | 0.01% | 37% | 7% | 23% | 11% | 27% | **65%** | 49% | 61% | **ODR** |
| | 0.1% | 37% | 7% | 23% | 12% | 24% | **65%** | 49% | 61% | |
| | 1% | 47% | 14% | 34% | 19% | 41% | **76%** | 63% | 74% | |
| | 10% | 65% | 37% | 57% | 37% | 55% | **89%** | 79% | **89%** | |

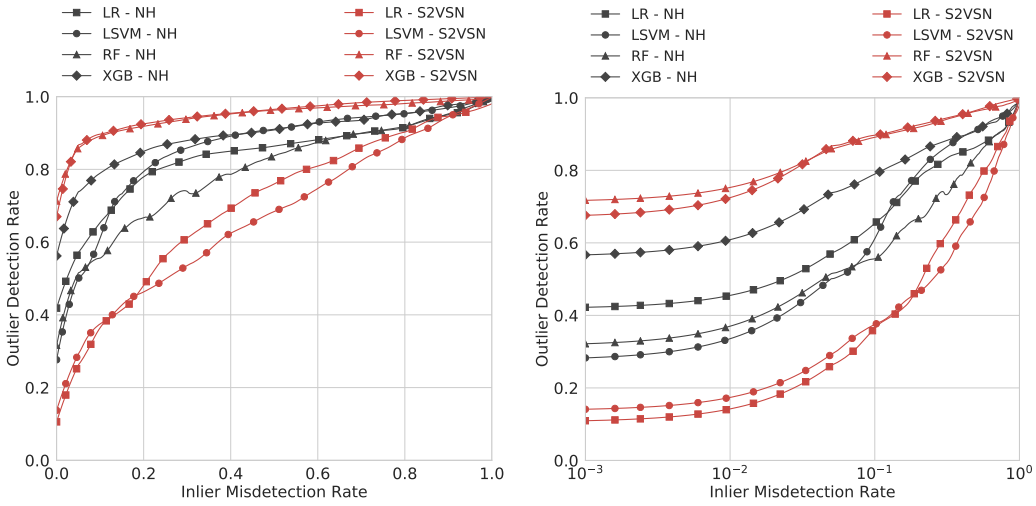



Fig. 3.11 Anomaly detection performance as a trade-off between the outlier detection rate and the inlier misdetection rate. Figure 3.11b shows the behavior of the curves in Figure 3.11a in logarithmic scale.

Curves in Figure 3.11 show the aggregated performance achieved by the classifiers for all classes. To understand in more detail how each one of the families in our dataset can be identified as an outlier in comparison with the rest of the families, we compute the same curves for each one of the families and calculate the *area under the curve (AUC)* as the performance metric for each class.

Figure 3.12 shows the AUC achieved by each classifier when trained on the dataset embedded using both proposed approaches and allow us to derive some conclusions. For instance, it can be observed that the average decrease in performance for the logistic regression and the linear SVM when trained on the S2VSN embedding is determined by large variations among classes. Moreover, we can also conclude that these two algorithms benefit from operating in a high-dimensional space, although not as much as the increase in performance achieved by the random forest and gradient boosted trees algorithms when trained in the low-dimensional space of the S2VSN embedding. Accordingly, this combination of embedding and classifiers offers the best and most consistent performance across all classes.

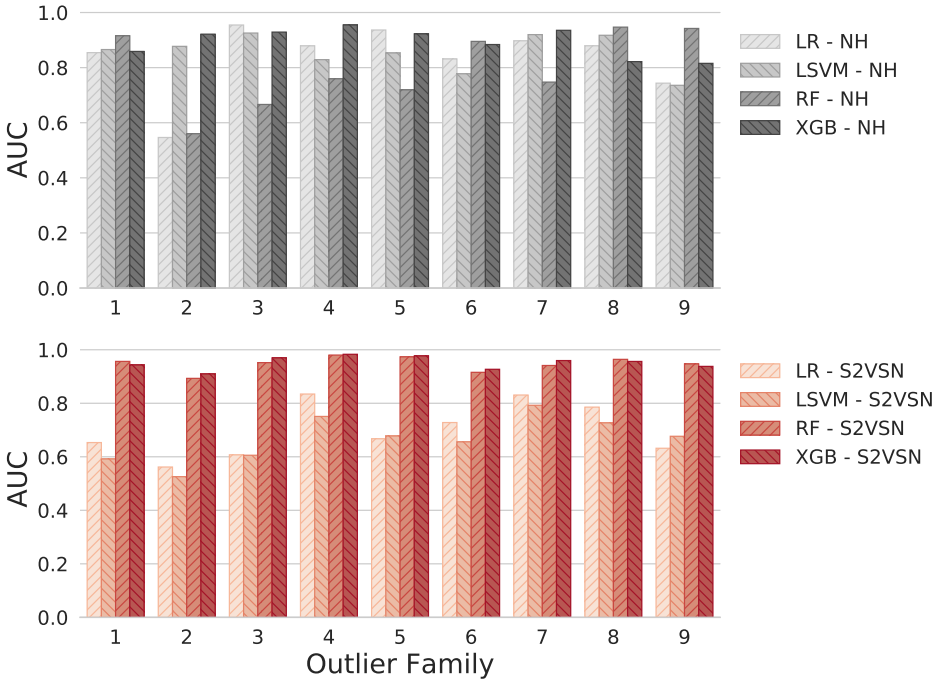



Fig. 3.12 AUC achieved by the different classifiers and embeddings at identifying each individual family as an outlier.

## 3.6 Limitations

The experiments from the previous section show how our structural representations can be used effectively for malware triage in different setups. However, our approaches are not free of certain limitations.

For instance, by analyzing the global structure of binary applications, our method is resilient towards typical local obfuscation techniques, such as instruction reordering, branch inversion or the renaming of libraries and identifiers. However, as a purely static method, it suffers from the inherent limitations of static code analysis. In particular, the construction of static call graphs is undecidable and thus the function call graphs processed by our method are typically over-approximations. In principle, this works towards the attacker's advantage, as the call graph can be obfuscated by adding unreachable calls. Moreover, function inlining can be used to hide the graph structure. While in the extreme case, this allows for the creation of malware with only a single function, this would both limit the functionality of the code and hint at a suspicious binary.

Attackers may also target the disassemble process to evade detection by our method. For example, invalid but unreachable bytecode sequences ("junk

code") can be deliberately inserted to hinder successful disassembling. Moreover, bytecode unpacked and loaded at runtime cannot be processed by the disassembler and thus can only be considered if our method is coupled with dynamic analysis techniques.

Finally, neural network architectures require a considerable amount of labeled data to be trained. However, as described in Section 3.5.2, the number of available samples can be combined to generate a much large number of pairs, as required for the input to the siamese architecture to tune the *structure2vec* network effectively.

## 3.7 Related Work

In the following, we discuss related work on structural code comparison in general and then proceed to discuss approaches specifically designed for malware classification with a focus on machine learning approaches that are specially linked to our work.

The analysis of malicious code and its structure have been a vivid area of security research in the last years. In particular, determining similar program code is an important problem encountered in several areas of security research, including the detection of malware [86, 136, 71, 3], software plagiarism [94, 106] and vulnerabilities [43, 156]. To this end, several methods to assess the structural similarity of code have been proposed. For example, Kruegel et al. [86] as well as Cesare and Xiang [21] present methods for polymorphic malware detection based on the comparison of control flow graphs. In particular, Kruegel et al. perform graph labeling to preserve instruction level information in a similar manner as performed by our graph representation. Unfortunately, both approaches are based on sets of control flow graphs and thus ignore the composition of functions entirely. We address this shortcoming by taking into account the function call graph.

Other researchers have also recognized function call graphs as a robust representation for code comparison. For example, Hu et al. [71] as well as Shang et al. [136] define similarity metrics for function call graphs, however, without considering the use of supervised learning techniques for automatic malware triage. The problem of clustering known malware into families has been considered by Kinable and Kostakis [80], who use approximations to graph edit-distances to cluster malware by call graph similarity. Efficiency is however, not a primary concern in this setting, whereas it is vital in malware

detection, a problem we address using an efficient linear time mapping in our explicit embedding and through a low-dimensional representation in our learned embedding.

Kernel functions for structured data have been pioneered by Haussler [67] and have been first applied over graphs by Kondor et al. [83]. Graph kernels have since then been applied mainly in bioinformatics [e.g., 12, 137] and chemical informatics [e.g., 116, 69]. Unfortunately, the high computational effort involved has prohibited many applications in the past. Researchers have therefore focused on developing efficient approximations of graph kernels. An overview of these approaches is given in [13].

Regardless of these efforts, graph kernels have found limited attention in malware detection to date. An exception is the work by Wagner et al. [150] and Anderson et al. [3] who use graph kernels to analyze execution traces obtained from dynamic analysis.

More recently and motivated by the extensive advances in the field of artificial neural networks, researchers have developed techniques that allow learning graph models by means of deep neural architectures and attempt to overcome the limitations of graph kernels. In this chapter, we build on the ideas of Dai et al. [37] to obtain vector representations from graphs through neural networks, allowing for learning directly on the graph space. Such an strategy is also used for instance in the security research context by Xu et al. [155], who aim at detecting whether two binary functions coming from different platforms are similar or not. In a similar vein, Narayanan et al. [110] propose *subgraph2vec*, an approach to obtain a vector representation from graphs through a neural architecture in an unsupervised fashion.

A second strain of research focuses on malware classification and we discuss here some works that share a similar scope or introduce tangential ideas to our approaches. In particular and due to the popularity of mobile devices in general, and of the Android operative system in particular, much of the recent research on detecting variants of known malware families has revolved around this platform. For example, Zhou et al. [161] as well as Hanna et al. [65] employ feature hashing on byte code sequences to measure code similarity. Furthermore, Crussel et al. [35] present a method called DNADroid, which compares program dependence graphs. Finally, RiskRanker by Grace et al. [58] compares function call graphs. Unfortunately, RiskRanker requires source and sink functions commonly linked to malicious behavior to be specified manually, thus requiring constant manual adaption to changing trends in malware development. In

contrast, the classifiers learned by our methods can be easily adapted by re-training on more recent malware data sets.

More recently, Rajasegaran et al. [115] introduce a method for detecting counterfeit mobile applications that employs a similar approach to ours. In their work, they make use of a siamese network and two convolutional neural networks to learn embeddings from images. In this case, they use the pretrained network VGGNet on images from icons of apps. The $n-1$ layer is used as an embedding for the image and then a distance metric is used to find similar looking icons. Other methods targeting the Android platform do rely, like ours, on graph learning. For example, Fan et al. [45] propose to construct static call graphs of sensitive API for Android malware classification. In their work, they characterize each family as a set of sensitive API subgraphs and measure the distance to each new sample as a function of the similarity between subgraphs. Computing this similarity is however expensive as it requires to build a matrix for each subgraph and calculating the normalized weighted sum of the cosine distances among nodes in the intersection of two subgraphs. In contrast, our function call graph representation and embeddings do not require the analyst to specify sensitive behaviors beforehand and allow to measure the similarity efficiently in the output vector space.

Closer in scope to our work, Vanderbruggen et al. [146, 145] focus on the classification of malicious x86 binaries through graph analysis. However, they rely on manually extracting features from each binary and the spectral properties of the interprocedural graph. This allows them to obtain a fixed size static representation of the binaries in different malware families. In contrast, both of the approaches that we propose can operate directly in the graph space.

Chen et al. [25] work instead with graphs resulting from the dynamic analysis of Windows executables. In particular, they build markov models over APIs to characterize data flow behavior of different software types including ramsomware. In contrast to our approach their method takes n-grams over API sequences and do feature selection through correlation and gain ratio to build the corresponding embeddings.

Also with a dynamic approach, Rosenberg et al. [131] address the problem of malware classification as a means to perform attribution of targeted attacks on an interesting APT dataset. However, in contrast with the sophistication of their malware data, they propose an approach based on building a bag-of-words model from the 50.000 most common words of Cuckoo sandbox reports. The resulting vectors are then used to train a feedforward neural

network. While the authors argue that dynamic analysis can reveal high level behaviors and results effective in a setup with a limited number of families, it is however very expensive to deploy at scale and hard to defend from evasive behaviors. Our approach addresses thus this problem through a robust static representation in combination with two complementary embeddings. An explicit embedding that allows for explainability while being efficient and a low-dimensional implicit embedding that leverages discriminative information to improve the performance of the classifier without sacrificing efficiency.

## 3.8 Summary

After the detection phase, the security analyst characterizes the threat in order to understand its implications and find possible ways to mitigate future attacks. Therefore, in this chapter, we have explored the role of malware as a source of intelligence and proposed strategies to establish a link between new samples found during the investigation of an attack and known malicious code.

In particular, we focus on the problem of assisting the analyst performing malware triage at scale as a means to understand the behavior of malicious samples and being able to attribute the attack to a known threat actor or, given the case, to an unknown attacker.

To this end, we propose to characterize malware through a structural representation based on function call graphs. This representation is robust against low level modifications and its expressiveness allows us to model the behavior of the binary code without loss of generality. Furthermore, and given the actual massive scale of the malware problem, we leverage modern machine learning algorithms that are able to deal efficiently with large amounts of data.

Based on our structural representation, we propose two complementary approaches to generate embeddings of function call graphs that let machine learning algorithms operate on numerical vector data. First, an explicit and high-dimensional embedding inspired by graph kernels that, thanks to its explainability, offers the analyst the possibility of understanding straightforwardly the decisions of a linear machine learning classifier. Second, an implicit low-dimensional embedding that is learnt from the data through artificial neural networks and that is based on the neural embedding approach *structure2vec* and optimized through a siamese architecture.

We address then the problem of malware triage from different perspectives and evaluate the performance of a series of machine learning algorithms when

trained on both types of embeddings in different setups. For instance, we first evaluate the quality of the clusters found by a clustering algorithm on our explicit graph representations when no labeled data is available. Then, we show how the analyst can improve these results through a semi-supervised approach by representing the data through a low-dimensional implicit embedding. Moreover, we evaluate a set of classifiers in a supervised setup in order to assign new unseen malware samples to known families and show how the explicit graph representation allows us to classify malware with 91% accuracy, a value that can be further improved up to 98% by compromising on explainability through the implicit embedding. Finally, we propose an approach based on anomaly detection that enables the analyst to identify if a new sample belongs to a new family and evaluate its performance in every case by considering each family as an outlier.

The analyst is thus well equipped with the strategies proposed so far to put into place detection and analysis methods to block and characterize targeted attacks. However, given the current threat landscape, isolated efforts to thwart attack campaigns within companies and organizations are rendered mostly ineffective against actors with large dedicated resources. Therefore, in the next chapter, we propose a series of methods for collecting, sharing and correlating threat data and explore how the insights and threat intelligence generated by analysts can be effectively shared and consumed by the security community.

# 04

# Response

As we have discussed in the previous two chapters, the detection and analysis of attack campaigns is a daunting task: First, due to the focused operation of the campaigns, only few traces of the attackers are available for forensic investigation. Second, the employed malware often makes use of novel exploits and infiltration techniques. As a consequence, conventional security defenses such as intrusion detection systems and anti-virus scanners fail frequently to spot these type of threats, especially because detection patterns become available only with significant delay, if at all. It has become evident then, that isolated efforts to detect attack campaigns within companies and organizations are mostly ineffective against organized threat actors.

As a remedy, security research has started to explore means for collecting, sharing and analyzing threat information across organizations—evidence-based knowledge referred to as *threat intelligence* [e.g., 79, 50, 16, 111]. As part of this process, different exchange formats have been proposed to provide a standardized way for describing security incidents, forensic traces and observations related to attack campaigns. Examples of these formats are STIX [7], IODEF [39] and OpenIOC [98], which are gradually adopted by national and enterprise CERTs in combination with commercial and open source databases for storing knowledge about ongoing attacks such as *Alien Vault's Open Threat Exchange* [147] or the *Collective Intelligence Framework* [28].

However, collecting and sharing information alone is not sufficient for mitigating the threat of attack campaigns. Although such threat intelligence platforms enable searching for indicators of compromise that exactly match a query, the actual crux is to correlate the vast amount of available data and pinpoint similar characteristics of novel campaigns that can help eliminating existing infections as well as craft detection patterns more efficiently.

After introducing our approaches for detection of targeted emails in Chapter 2 and for triage of malware in Chapter 3, we present in this chapter MANTIS, a *threat intelligence* platform that enables the analyst to aggregate and correlate any threat data generated in the detection and analysis phases.

In particular, built on top of a unified representation that is based on attributed graphs, the platform is able to merge information from different exchange formats, solving the problem of analysing data contained in heterogeneous or overlapping standards. Furthermore, different threat objects that are typically analysed independently are correlated through a data type-agnostic representation. Such an approach allows unveiling high-level relations not visible within individual threat reports and linking unconventional patterns shared between seemingly unrelated attack campaigns.

At the core of our platform lies a novel graph-based similarity algorithm that allows discovering similarities between threat data objects at different levels of granularity. This analysis allows a security analyst to search the attributed graphs for threats related to individual observations—similar in spirit to a search engine. For example, given an object from a security incident, such as a suspicious file or an HTTP request, the platform can identify related nodes in the graphs and traverse them to the corresponding threat reports, ultimately returning information about the underlying attack campaign. In addition, MANTIS supports authoring reports for new incidents that can be used for searching and correlating existing information, as well as extending existing threat data with new insights.

We evaluate the utility of MANTIS as an information retrieval system for threat intelligence in a quantitative and qualitative fashion. To this end, we make use of a large data set of malware observed in the wild and collected by a security vendor at the end-point systems of different companies and organizations. We base our evaluation on the threat reports created during the analysis of such samples.

As a result, we show how given an object from a security incident, our platform is able to retrieve associated data to the corresponding malware with

a mean average precision of 80% in a set of 14,000 standardized threat objects. This means that 4 out of 5 results returned to the security analyst are relevant to her query. We further illustrate the performance of this analysis in two case studies based on threat intelligence from highly targeted attack campaigns: *Stuxnet*, the well-known joint endeavour of several west nations to sabotage Iran's nuclear program and *Regin*, a sophisticated espionage tool allegedly sponsored by a state-nation and distributed worldwide to selected individuals and organizations.

Therefore, while its analysis and query capabilities alone already provide a valuable tool for assessing the impact of security incidents, MANTIS complements the approaches introduced in Chapters 2 and 3, enabling a network-wide forensic examination for artifacts that have been detected and analysed.

To the best of our knowledge, MANTIS, available as an open-source project, is the first practical solution for performing similarity-based analysis of multi-format and structured data for threat intelligence. In summary, we make the following contributions in this chapter:

- **Unified representation of threat intelligence reports.** We present an open-source platform for threat intelligence that merges different standard exchange formats and provides a unified representation of threat reports as attributed graphs.

- **Similarity analysis of threats.** We introduce a similarity algorithm for attributed graphs that enables uncovering relations between threats at different levels of granularity.

- **Information retrieval for threat intelligence.** By incorporating the similarity analysis into our platform, we devise an information retrieval system that is capable of retrieving related reports given individual observations from security incidents.

The rest of this chapter is organized as follows: we discuss the concept of threat intelligence and its standards in Section 4.1. Then, we proceed to introduce our system for analysis and retrieval of threat data in Section 4.2, evaluate its effectiveness with real-world threat data in Section 4.4 and discuss its limitations in Section 4.5. Related work is discussed in Section 4.6 concluding the chapter.

## 4.1 Threat Intelligence

Companies and organizations dealing with sensitive data usually employ different security measures for protecting their infrastructure, including systematically monitoring network and host events. While this monitored data can be searched for security incidents on a regular basis, appropriate detection and search patterns are only available for known threats, leaving infrastructure vulnerable to novel and unknown attack campaigns. This situation, however, can be significantly changed if information about incidents is collected, shared and analyzed across organizations. Although this approach may not be sufficient for spotting extremely focused attacks, it enables hunting down threat actors that re-use or gradually evolve their techniques and strategies.

However, information regarding security incidents, related observations, and threat actors is very heterogeneous and difficult to transmit without a lack of context. In order to overcome this problem, different standard formats have been recently proposed to provide a structured representation of threat data that can be easily shared and processed. These standardised but diverse threat insights constitutes what has been known as *threat intelligence*. Examples of these standards are *IODEF*, developed by members of the IETF [39], *OpenIOC*, implemented by Mandiant in many of its products [98], and *STIX* with its associated family of formats, like *CyBOX* or *MAEC* [7]. In particular, the *STIX* standard is currently leading the adoption by national and enterprise CERTs. In the following, we briefly cover its design as an illustrative example of the structured representations implemented by all of the mentioned threat intelligence standards.

The STIX standard comprises a family of XML schemes whose development is driven by the security community under supervision of the MITRE Corporation. The individual STIX formats and constructs allow to describe numerous types of threat information in a structured way and for different use cases. For example, observations related to threats can be described as *Observables*, ranging from registry keys and file names to network addresses and strings in URLs. These Observables can be combined with logical operators to form *Indicators* that reflect and describe concrete threats. Other constructs include representations for *Incidents*, *Courses of Action*, *Attack Campaigns* and *Threat Actors*. A detailed description of the different constructs is provided in the STIX specification [7].

```
<stix:STIX_Package (...) id="package-37e">
  <stix:STIX_Header>
      <stix:Title>APT1</stix:Title>
      <stix:Description>
          This package contains the IOCs referenced
          in Appendix G of the APT1 report.
      </stix:Description>
  </stix:STIX_Header>
  <stix:Observables>
    <cybox:Observable id="Observable-9ba">
      <cybox:Object id="URI-9ba">
        <cybox:Properties type="URL">
          <URIObj:Value condition="contains">
              /mci.jpg
          </URIObj:Value>
        </cybox:Properties>
      </cybox:Object>
    </cybox:Observable>
    <cybox:Observable id="Observable-2b2">
      <cybox:Object id="File-2b2">
        <cybox:Properties type="File">
          <FileObj:Name>gdocs.exe</FileObj:Name>
          <FileObj:Extension>exe</FileObj:Extension>
          <FileObj:Size>261822</FileObj:Size>
          <FileObj:Attributed_List>
              <cybox:Object condition="contains">
                  v1.0  No Doubt to Hack You, Writed
                  by UglyGorilla, 06/29/2007
              </cybox:Object>
          </FileObj:Attributed_List>
        </cybox:Properties>
      </cybox:Object>
    </cybox:Observable>
  </stix:Observables>
```

Fig. 4.1 Exemplary STIX package for the "APT1" report by Mandiant [97]. Note that several identifiers and XML elements have been simplified for presentation.

As an example, let us consider the STIX package shown in Figure 4.1 which covers a tiny and simplified fragment of the indicators for the "APT1" campaign. This campaign was uncovered in February 2013 and comprised a series of targeted attacks against several companies and organizations [97]. Some common constructs of the STIX standard can be seen in the example: An

Observable matching the content of an URI (line 10–18), another Observable corresponding to a particular file (line 19–33), and an Indicator combining the two (line 36–61) that describes the malware family and references the underlying attack campaign. Note that although not included here, the original report in OpenIOC format covers over 3,000 Observables and 40 different Indicators for the attack campaign.

The use of threat intelligence standards allows to share and process a large amount of complex and enriched threat data in a standard and machine readable format. This has encouraged some companies with a large distributed infrastructure and a global view of the threat landscape to aggregate feeds that are made available to smaller organizations. However, the information received through these sources is highly heterogeneous and still needs to be put into context by the analyst. In our work, we aim at making this process much more efficient by providing a platform that integrates different standards into a unified representation and allows for exploring and searching structured threat data for relevant information.

## 4.2 The MANTIS Platform

As a last step for characterizing and understanding attack campaigns, we present MANTIS, a *threat intelligence* platform developed in collaboration with Siemens for storing, authoring and managing threat data. Its basic framework, implemented by Siemens CERT, offers support for several common threat intelligence standards, including STIX and OpenIOC, two of the standards with the largest adoption in the security community. As part of this thesis, we create a unified data model for MANTIS and design algorithms that extend its functionality, transforming the platorm into an information retrieval system for threat intelligence. To support its adoption and encourage further research, MANTIS is available as an open-source project[1] and readily applicable for experimenting with threat data at organizations and CERTs and the implementation of new importers for additional standards.

To provide a flexible and architecture-independent design, MANTIS is structured as a set of *Django* applications. Figure 4.2 shows a schematic view of its components. In the typical use case, the security analyst documents the findings of an investigation using the authoring interface, while at the same time accesses related information about already documented threats through

[1] MANTIS— https://github.com/siemens/django-mantis

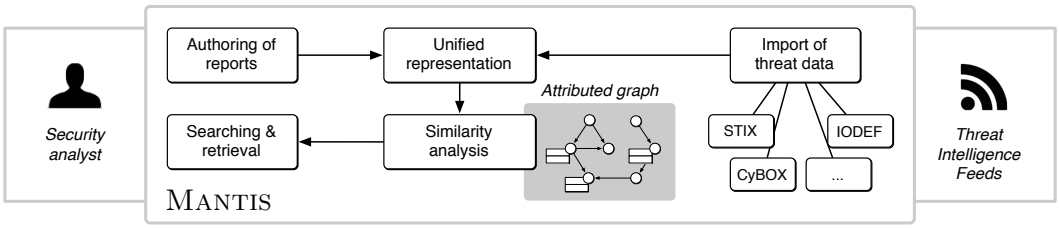



Fig. 4.2 Schematic overview of the MANTIS architecture.

the retrieval interface. Both interfaces provide different views for managing the creation and the collaborative maintenance of threat reports. Additionally, the platform supports receiving data feeds in different formats from other tools, organizations and security companies. The data contained in these feeds is jointly stored with authored reports and thereby enables an analyst to document her findings in the context of already known threats and attack campaigns.

We leave a more detailed description of the analyst workflow for Section 4.4.2 where, based on real data from a known attack campaign, we illustrate how the analyst can interact with the platform during an investigation.

### 4.2.1 Unified Data Model

To provide a joint view on the threat data collected, MANTIS expresses the different XML standards as directed graphs and links together constructs describing the same type of information.

Table 4.1 Example of flattened facts for an Observable.

| Id | Fact term (key) | Fact value |
|---|---|---|
| $f_1$ | Properties/File_Name | `gdocs.exe` |
| $f_2$ | Properties/File_Extension | `exe` |
| $f_3$ | Properties/Size_In_Bytes | `261822` |
| $f_4$ | Properties/File_Attributed_List/Object@cond... | `Contains` |
| $f_5$ | Properties/File_Attributed_List/Object | `v1.0 No Doub...` |

As a result, related data describing campaigns at different levels, such as generic attack strategies and concrete malicious payloads, are merged into a single view and can be accessed by simply traversing the edges of the graphs.

Formally, we define this directed graph as a tuple $G = (V, E, L)$, where each node $v \in V$ symbolizes a standard construct from an original XML document. Two nodes $v, u \in V$ are connected by a directed edge $(v, u) \in E$, if the construct corresponding to $u$ is either contained or referenced by the construct represented

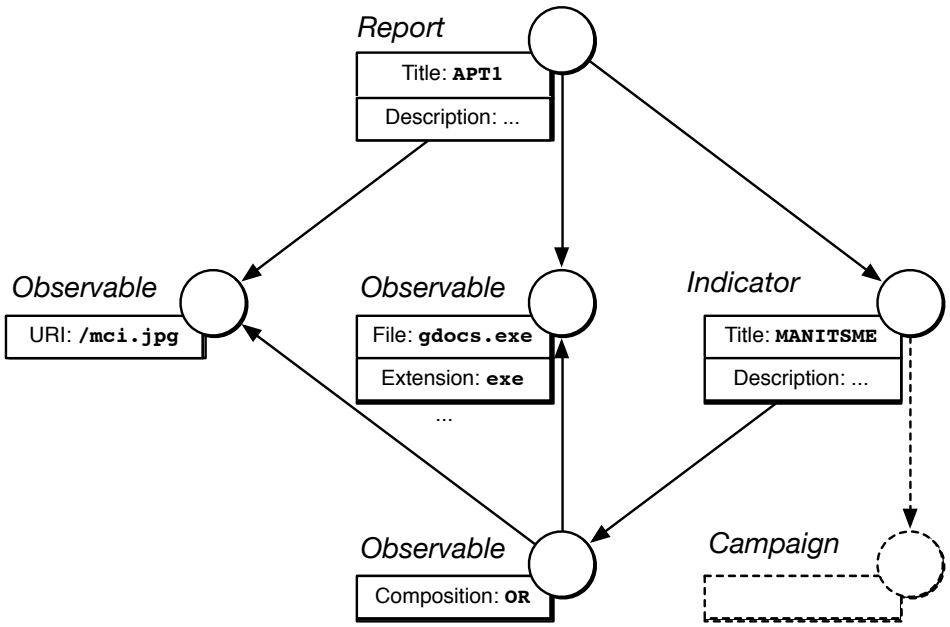



Fig. 4.3 Attributed graph for STIX package in Figure 4.1.

by $v$. Moreover, we attach a list of facts $l \in L$ to each node. This enables us to store unstructured data in the graph, assigning a set of attributes to each node. Each list $l \in L$ has the form $l = (f_1, f_2, \ldots, f_n)$ where a fact $f_i$ results from flattening the inner structure of a standard construct into facts of key-value pairs.

As an example of this unified representation, Figure 4.3 depicts the attributed graph that abstracts the relations between objects and data in the STIX report from Figure 4.1, including the two Observables, their composition and the corresponding Indicator. Note how several substructures have been flattened into facts, such as the title of the report or the URI pattern.

In addition Table 4.1 shows the complete list of flattened facts for the Observable at the center of Figure 4.3. Note that the flattening is conducted recursively and the fact terms are built using a hierarchical structure. This generic representation within the nodes of the graph will let us compare effectively different threat reports and traverse between objects even if their are of different type, such as from an observed URI pattern to the corresponding attack campaign.

Each fact value in the platform is stored exactly once and referenced from any object containing the fact. This de-duplication saves storage space and, more importantly, enables an efficient calculation of correlation based on fact

equality. Thus, the analyst can retrieve all nodes related to particular facts with a single query, for example to get a listing of all executable files with a size of 261,822 bytes. However, while equality-based searches already provide a powerful instrument for mining the collected threat data, it is obvious that more complex relations cannot be uncovered by focusing on exact fact matching alone. In the following, we introduce our analysis method which, as part of MANTIS, allows the analyst to perform similarity-based queries on top of its unified graph data model.

## 4.3 Threat Similarity Analysis

When working with threat intelligence, a security analyst investigating an incident begins by documenting any suspicious findings or the results of a more detailed analysis. Then, the analyst wonders if such an event has been observed in the past and specially, if relevant documentation about the incident already exists.

However, obtaining an answer to this question is far from trivial. Threat reports can be large and heterogeneous and contain data that, without being identical, are linked to related events. For example, consider the case of several Observables containing HTTP requests of similar URLs. While being slightly different in the URI or host name, such objects may be associated to the same threat actor and thus should be retrievable to help investigating the new incident.

As a consequence, the analyst requires a method that can identify and retrieve similar objects regardless of their structure, size or content given a query object. Therefore, we strive for a similarity-based search that is capable of identifying similar facts, nodes and subgraphs on top of the unified representation of MANTIS. In particular, we implement our approach in two steps: First, we draw on our unified representation and devise a method that enables non-exact matching based on *fingerprints* computed using the *simhash* algorithm. (Section 4.3.1). Second, we implement a retrieval system to efficiently identify all fingerprints similar to a given query (Section 4.3.2).

### 4.3.1 Simhash Fingerprinting

To measure the similarity between arbitrary objects in our representation, we make use of the bag-of-words concept from the information retrieval field [133]. In its original form, this model is intended for text documents in order to

obtain a numerical vector representation based on the words or phrases they contain. However, threat data is heterogeneous and may range from simple file names to code fragments and textual descriptions. Therefore, we employ *byte n-grams* to characterize the content of an object [152, 38]. This means that a fact $f$ is represented by all byte strings of length $n$ contained in the fact value. Similarly, a node $v$ is characterized by all $n$-grams contained in its associated facts $l$ and a subgraph rooted at a node $u$ is represented by the $n$-grams of all nodes reachable from $u$.

While the extracted $n$-grams provide a versatile and generic representation of the underlying content, they are not suitable for an efficient analysis, as they require variable-size storage and cannot be compared in constant time. For example, if new data introduced into the platform contained previously unseen $n$-grams, the existing vector representation of the bag-of-$n$-grams model should be recomputed for all objects to accommodate the new $n$-grams. As a remedy, we employ the *simhash* algorithm introduced by Charikar [23], an approximation technique that maps an arbitrary set of objects to a fixed-bit fingerprint.

The simhash algorithm ensures that although each object is represented by a hash of its $n$-grams, similar objects have similar fingerprints. More specifically, the design of the algorithm guarantees that the Hamming distance [63] of fingerprints computed from similar objects is small. This property allows us to articulate the problem of finding a similar construct in MANTIS given an input query and its fingerprint $F$ as the problem of finding those fingerprints that differ from $F$ in at most $b$ bits.

The algorithm proceeds as follows: First, each object is hashed to an $m$-bit value. Second, the bits at each position $i$ in the hash values are counted, where a 1-bit is interpreted as $+1$ and 0-bit as -1. Finally, the resulting $m$ count values are converted into an $m$-bit fingerprint by setting all positive counts to 1 and all negative counts to 0. In our setting we apply the simhash algorithm to compute $m$-bit fingerprints for the sets of $n$-grams associated with facts, nodes and subgraphs, where we set $n = 3$ and $m = 64$. Accordingly, the fingerprint $F_f$ of a fact $f$ is computed by

$$F_f = \text{simhash}(N(f)) \tag{4.1}$$

where $N$ is the set of $n$-grams contained in the fact value. Similarly, we compute the fingerprint $F_v$ of a node $v$ as

$$F_v = \text{simhash}\Big( \bigcup_{f \in l(v)} N(f) \Big) \tag{4.2}$$

where $l(v)$ is the list of facts associated with $v$, and arrive at the fingerprint $F_g$ of a subgraph rooted at a node $u$ by

$$F_g = \text{simhash}\Big( \bigcup_{v \in r(u)} \bigcup_{f \in l(v)} N(f) \Big) \tag{4.3}$$

where the auxiliary function $r(u)$ returns all nodes reachable from $u$. Figure 4.4 shows a complete example of this computation for a fact containing the value `/mci.jpg`.

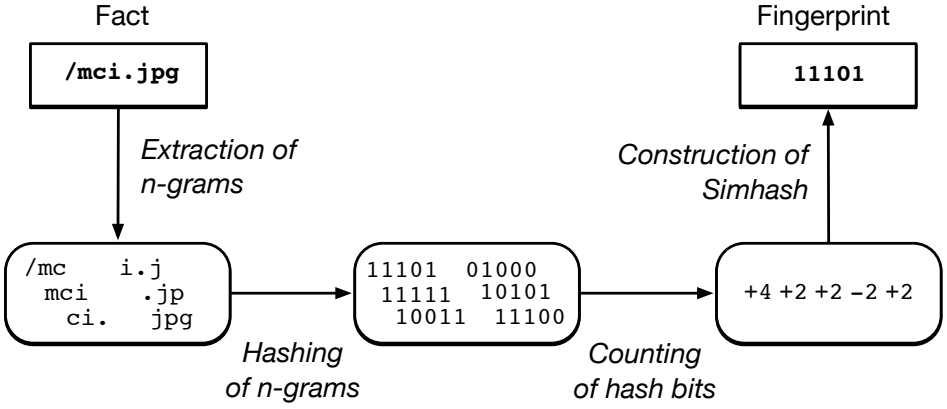



Fig. 4.4 Computation of the simhash fingerprint of a fact.

The value is first represented by a set of 3-grams and then mapped to a set of 5-bit hash values. These values are finally aggregated to form the fingerprint $F_f = 11101$.

Note that $n$-grams are agnostic to the type of each fact, what results in determining similarity at a lexical level. This means that, in the same way as a search engine works, our method is not limited to measuring the similarity between constructs of the same type (e.g. two IP addresses), but between all possible types. This comparison enables to find relations in cases where standards are incorrectly filled or the types of data are unknown. For instance, a construct including a fact that describes the name of a file can be matched to a report including a description where this file is mentioned.

### 4.3.2 Hamming Distance-based Queries

When a large number of threat reports is loaded into the system the number of constructs that needs to be analyzed can rapidly increase. For this reason, computing the Hamming distance between the query fingerprint and all queries in the platform can be computationally expensive.

As a remedy and to avoid precomputing the distance between all existing fingerprints at a maximum of $b$ bits we follow the strategy proposed by Manku et al. [100]. In their approach, an index contains a series of *buckets* where each *bucket* has associated an integer $p$ and a permutation of bits $\pi$. Each *bucket* is filled by first applying its permutation to all existing fingerprints and then sorting the resulting set of permuted fingerprints. Given a query fingerprint $F$ and an integer $b$, we identify all permuted fingerprints in each *bucket* whose top $p$ bits match the top $p$ bits of $\pi(F)$. From these fingerprints, the ones that differ at most $k$ bits from $\pi(F)$ are retrieved as result. Such approach can be completed in $O(p)$ and does not required the computation of a large distance matrix of fingerprints. For discussion on the optimal number of *buckets* and other implementation details, we refer the reader to the original description of the indexing approach introduced by Manku et al. [100].

For our particular application, we build three indexes: one for the fingerprints of individual facts, a second one for the fingerprints of nodes (i.e. individual constructs with their own semantics in the threat intelligence standard) and a third one for the fingerprints of subgraphs rooted at the different nodes. When a new report is imported into the system we first represent its data as an attributed graph. Then, we compute the fingerprints of its facts and constructs and add them to the corresponding index. When the analyst queries the system, the fingerprint of the query is computed and depending of its type, the results obtained from the corresponding index are retrieved. Moreover, retrieval results are sorted according to their Hamming distance and therefore their predicted relevance. This means that even in the case that a query returns a large list of results, the analyst can rapidly identify the most relevant entries and keep conducting a focused investigation. In the following, we proceed to illustrate in more detail the interaction of the analyst with the platform and to evaluate the efficacy of our approach using real-world threat data.

Table 4.2 Raw dataset indexed by MANTIS.

| Standard | Construct | Size |
|---|---|---|
| STIX | STIX Package | 2,621 |
| STIX | Observable | 7,282 |
| STIX | Indicator | 2,764 |
| CybOX | Observable | 255,941 |
| CybOX | DNSQueryObject | 2,583 |
| CybOX | FileObject | 12,334 |
| CybOX | ProcessObject | 17,914 |
| CybOX | SemaphoreObject | 244 |
| CybOX | WinMutexObject | 18,513 |
| CybOX | WinRegistryObject | 186,990 |
| CybOX | WinThreadObject | 22,347 |

## 4.4 Evaluation

In this section we illustrate through a use case how the analyst may interact with the MANTIS platform and then extend this example to a full quantitative and qualitative evaluation of our method for similarity-based searches. In particular, we first explore the performance of the system responses when every object and fact value is used as the input query introduced by the analyst. Second, we evaluate the results provided by the system in two specific scenarios. These involve threat data from the targeted and, therefore, more elusive *Stuxnet* and *Regin* attack campaigns.

### 4.4.1 Data Set

We consider for our evaluation a dataset of STIX packages automatically generated from malware samples collected in the wild by a security vendor in June 2015 at the end-point systems of different companies and organizations. The samples cover a wide range of malicious activity, including common botnets, backdoors and attack campaigns. Each sample is analyzed in a sandbox environment, where the results of the underlying static and dynamic analysis are automatically converted to CyBOX objects and grouped in STIX packages.

Based on results provided by VirusTotal [149], we assign a label to each STIX package according to the hash of the analysed binary. As the names assigned to different malware families by AV vendors vary, we use a majority voting strategy and select those reports with a consensus of more than 5 vendors. The resulting 2,621 STIX reports are then loaded into MANTIS for testing. Table 4.2 contains a summary of the constructs present in the original data.

Moreover, we take into considerations certain characteristics of the data that are relevant for the evaluation: First, we exclude all objects and facts that are unsuitable for a similarity search, such as local timestamps, identifiers and hash sums, reducing the size of the attributed graphs to 14,987 individual nodes. Note that although these types of objects are not included to evaluate the algorithm, they are still in the system and are thus, searchable. Second, if several objects in one or several STIX reports contain the same value, the importer stores this value only once in MANTIS. As a result, nodes in the unified representation contain only references to their values, saving storage space if a certain value occurs more than once. The de-duplication performed by our platform, results in a total of 46,015 unique facts being stored in the system for similarity analysis.

### 4.4.2 Analyst Workflow

In order to illustrate how the security analyst may interact with the platform and given the dataset present in the system, we load an additional set of 3 STIX reports containing several Observables and indicators of compromise of the attack campaign Taidoor [143]. This campaign has been active since at least March 2009. It has initially targeted the Taiwanese government but later extended its scope to further government agencies, corporate entities and think tanks [73]. In its typical attack scenario, the targets receive a spear-phishing email with a decoy document including legitimate content and a malicious file that when opened in a vulnerable system is silently installed. To obtain persistence, the Taidoor malware modifies the system's registry using registry entries from the file `~dfds3.reg` and then contacts the command-and-control server for further instructions.

In such setting, for instance, the analyst would perform dynamic analysis on the received file once the targeted email has been highlighted as suspicious by the detection approach introduced in Chapter 2. As a result of this analysis, let us consider for example that a request to the URL

```
http://211.234.117.132/ttgcy.php
```

is observed in the network traffic generated by the file and that the system hosting the HTTP server is located in an unusual geographical location, not commonly contacted by clients in the local network. The analyst decides that such behavior might be indicative of an attempted attack and decides to further inspect the issue. First, the authoring interface in MANTIS allows the analyst

to introduce all the information available about the suspicious HTTP requests. Then, this data is used to build a STIX Observable that is represented as an attributed graph and hashed according to the method described in sections 4.2.1 and 4.3.1. After incorporating the new data into the platform, the fingerprint resulting from hashing the Observable is used to retrieve and rank a list of relevant objects as described in section 4.3.2.

Table 4.3 Top results retrieved for an HTTP Observable of the Taidoor family.

| **Query: Observable** | |
|---|---|
| `HTTP GET to 211.234.117.132/ttgcy.php` | |
| **Returned results** | **Distance** |
| `HTTP GET to 211.234.117.132/klzvp.php` | 6 |
| `HTTP GET to 211.234.117.132/mtlxc.php` | 8 |
| `HTTP GET to 211.234.117.132/wobzz.php` | 10 |
| `STIX MAA Report Vobfus 1` | 21 |
| `STIX MAA Report Vobfus 2` | 32 |

Given the real-world dataset stored in the system, Table 4.3 shows the results returned for the query. For ease of presentation, we only show the HTTP URL in this listing and omit further details that are part of the HTTP request CyBOX object, such as the contacted hostname or the user agent. The first three entries, although not identical to the query, correspond to similar requests. By following the edges from the return nodes to the corresponding STIX packages, we are able to immediately identify the HTTP requests as traffic originating from the Taidoor campaign. The last two returned nodes are not related to our query, which can be directly concluded from the high Hamming distance to the query object. Note that we make use of 64-bit fingerprints and thus a distance of 20 bits already corresponds to a disagreement of 33%.

In addition to the capability of performing queries with threat objects of arbitrary size and complexity thanks to the unified representation introduced in section 4.2.1, MANTIS similarity-based search represents an advantage over standard threat intelligence platforms even for simpler queries. Systems such as CRITS [34], which operates with MongoDB as backend or ad-hoc solutions on top of SIEM platforms like *Splunk threat intelligence dashboards* [139], base their threat data correlations on exact matchings. Even for small strings, MANTIS simhash fingerprint allow us to retrieve not only objects containing the same fact but a ranked list of possible variations of the input query.

For instance, let us consider that, in the same investigation, the analyst finds the suspicious file `~dfds3.reg` in a host and, wanting to investigate more about it, issues a fact query with the name of the file. Based on the data loaded in the platform, Table 4.4 lists the returned results according to their similarity.

Table 4.4 Top results retrieved for a fact value of the Taidoor family.

| **Query: Fact Value** | |
|---|---|
| `~dfds3.reg` | |
| **Returned results** | **Distance** |
| `~dfds3.reg` | 0 |
| `~dfds3.reg` | 0 |
| `3fdata.reg` | 15 |
| `C:\DOCUME~1\acs\LOCALS~1\Temp\~dfds3.reg` | 25 |
| `~cnf.reg` | 27 |

The same file has been found in two different objects which are thus identical to the query and appear in the first place. The fourth result, although not identical, also contains the same file name and is therefore returned as result. By following the edges to its corresponding STIX package, we assert that it represents the same indicator of compromise from the Taidoor campaign.

Finally and although not exemplified here, the analyst can also query the system with a full threat report or a complete part of a larger report as explained in Section 4.3.2. The system will then retrieve those subgraphs similar to the query and rank them according to the hamming distance of their simhash fingerprints. To formally evaluate the performance of our method, we consider these different types of queries and analyze in the following sections how relevant the retrieved results are according to their class and the class of the query.

### 4.4.3 Quantitative Evaluation

From the perspective of the security analyst, our platform resembles the operation of an information retrieval system: an analyst enters a query and retrieves a list of relevant nodes from the attributed graphs. So in essence, MANTIS functions like a search engine and its performance will be as good as the relevance of the results retrieved. Accordingly, in order to evaluate its performance qualitatively we make use of a metric that is widely employed to assess the performance of search algorithms: the *mean average precision*

(MAP) [27]. The MAP averages the precision of a retrieval system over a set of queries $Q$ for different numbers $k$ of retrieved results. Formally, it is defined as

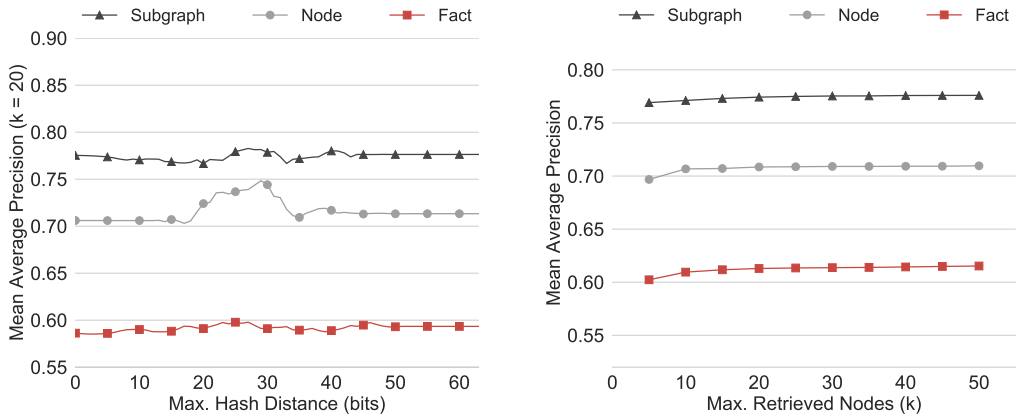
$$\mathrm{MAP}(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} \mathrm{Precision}(R_{jk}), \tag{4.4}$$

where $Q$ is the set of queries, $m_j$ the number of relevant nodes to the query $q_j \in Q$ and $R_{jk}$ the top retrieved nodes for the query $q_j$ up to the $k$-th relevant node. Moreover, we consider a node to be relevant, if it is associated with the same AV label as the object used as the query. For a single query, the average precision is the mean of the precision values obtained for the set of top $k$ documents. This average value is then averaged over all possible queries [27], in our case all available facts, nodes or subgraphs.

To understand the intuition behind this metric we consider again the example of a search engine. The performance of a query is better when more relevant results are returned on the first page of the search engine, that is, we get a high precision value for the top $k$ results [27]. Furthermore, the MAP score can be interpreted as the percentage of relevant objects in the returned results. For example, a MAP of 75% implies that 3 out of 4 returned results are relevant to the query.

We compute the MAP for our platform by considering all facts, all nodes or all subgraphs reachable from a node as queries to the system. To gain further insights into the similarity analysis, we repeat the queries with different number of retrieved objects $k$ and different numbers of bits to match between the fingerprints. The results of this experiment are presented in Figure 4.5, where the MAP is plotted for the different experimental setups.

We note that the quality of the returned results depends on the complexity of the query. If subgraphs are used as query, MANTIS is able to achieve a MAP value of 80%, such that 4 out of 5 returned results are relevant and constitute similar threats. If the analyst enters only a node or a fact as query, the MAP decreases. However, even when entering only single facts, our platform attains a MAP of at least 50%, thus providing retrieval results where every second result matters. Moreover, our platform reaches a good MAP already at 15 retrieved items (Figure 4.5b) which is a reasonable amount of information to display on the first results page of the search interface.




(a) MAP vs. maximum Hamming distance between fingerprints.

(b) MAP vs. maximum number of retrieved results.

Fig. 4.5 Mean average precision (MAP) for queries of different complexity.

As our dataset comprises a wide range of malware samples, we study how the diversity in the data affects the performance. Some samples, for instance, originate from small attack campaigns, while others are part of more common botnet and phishing activity. We evaluate then the results returned by the system when the queries belong to individual malware families. Figure 4.6 shows the amount of nodes and facts in each of the families. Note the logarithmic scale, that indicates a skewed distribution of samples per type of malware. Nonetheless and as shown in Figure 4.7, the unbalanced distribution has only a limited effect on the performance of our approach. Individual facts are retrieved with a MAP above 50% for most of the malware families, that is, every second returned result corresponds to the same malware family as the query. This is a remarkable result given that only individual facts, such as file names or URLs, are used to query the system.

Finally, scalability is another concern when designing an information retrieval system that is intended to accommodate large amounts of data. Figure 4.8a shows the evolution of the number of nodes and facts that need to be stored in the system per number of STIX reports imported. In both cases, a linear relation exist. As a result, we can expect our fingerprint indexes to also grow linearly with the number of imported reports. Moreover, every time the analyst introduces an individual fact or several facts as part of a construct, the fingerprint for each of them needs to be computed. As mentioned in Section 4.3.2, finding matching fingerprints for a query fingerprint $F$ can be completed in $O(p)$, but the time computation of the fingerprint for the

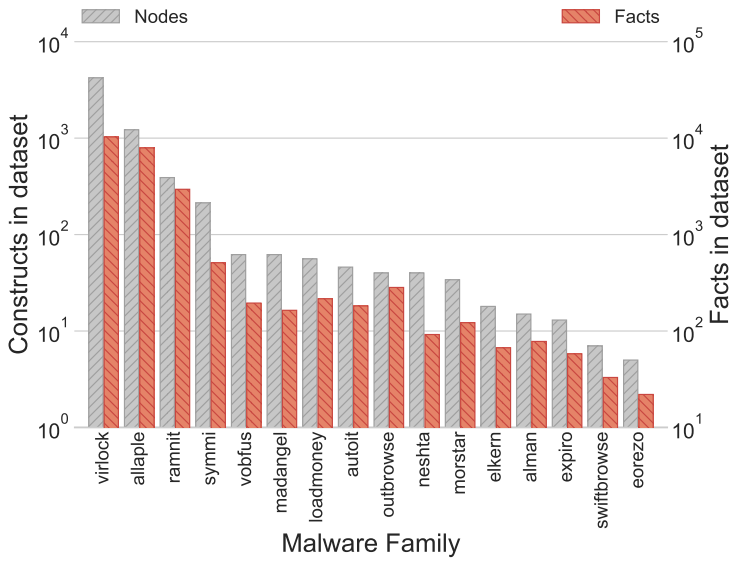



Fig. 4.6 Total number of constructs and facts per family.

query object is directly related to its size. Figure 4.8b shows how even for large subgraphs with more than 2000 $n$-grams, the simhash fingerprint can be computed in less than 20 milliseconds with a linear dependency to the number of $n$-grams.

### 4.4.4 Qualitative Evaluation

To evaluate our approach qualitatively, we consider a small set of STIX packages from the *Stuxnet* and *Regin* attack campaigns. Such highly targeted APTs are characterized by disparate indicators of compromise and are typically very elusive to identify. Stuxnet, for instance, which was initially discovered in 2010, is a sophisticated malware developed by west state-nations in order to sabotage the nuclear program of Iran. After remaining undetected for some time, its uncontrolled propagation through several attack vectors led to the identification of different variants in systems worldwide [32, 92]. The Regin trojan, on the other hand, is an advanced espionage tool that was used to surveil several companies and government entities including the European Council. Due to its stealth techniques, different variants of the malware remained unnoticed for several years until their discovery in 2011 [31, 91].

Thus, we evaluate the performance of our method when the analyst tries to retrieve such indicators from among more generic threat data. After loading a

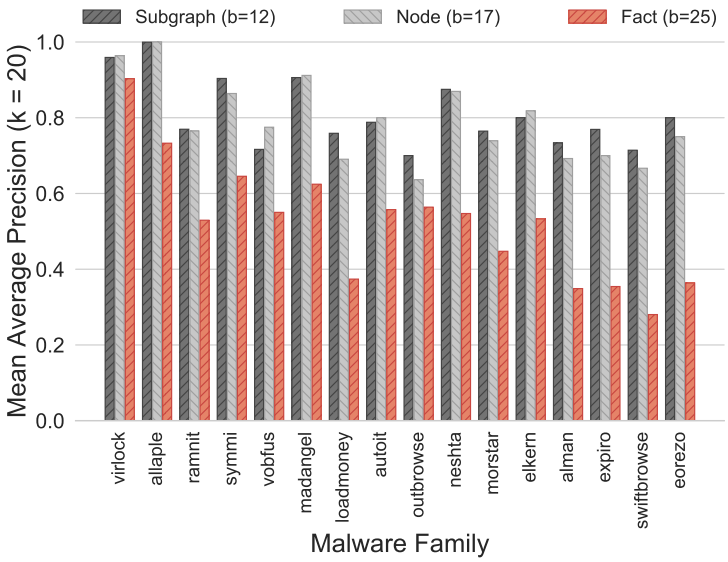


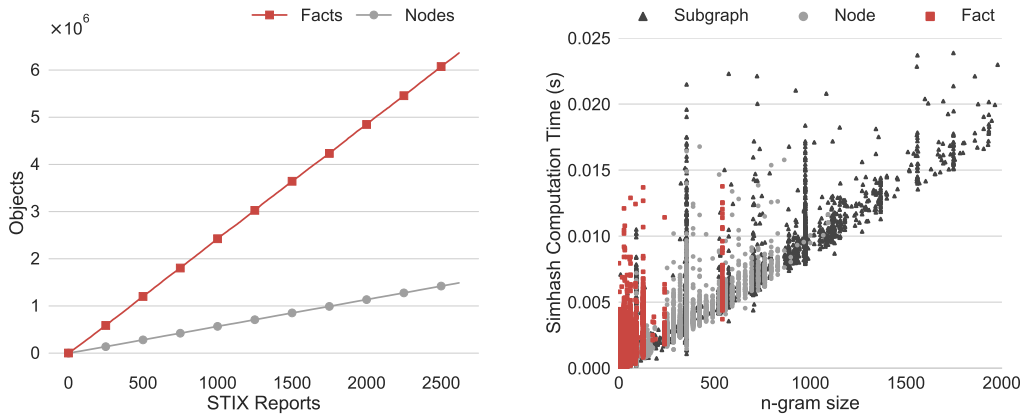

Fig. 4.7 MAP for each family with best $b$ and $k = 20$.

set of 31 and 10 STIX reports of the *Stuxnet* and *Regin* campaigns, respectively, we measure the mean average precision of the results when objects from these campaigns are used as queries. Additionally, we compare our method with the performance of searches based on exact fact matchings, as this type of retrieval strategy is the default approach used in threat intelligence engines and standard databases.

Table 4.5 Raw APT dataset indexed by MANTIS.

| | ***Stuxnet*** | ***Regin*** | **Total in Database** |
|---|---|---|---|
| STIX Reports | 31 | 10 | 2,662 |
| Subgraphs | 1,052 | 132 | 20,692 |
| Nodes | 557 | 76 | 15,620 |
| Facts | 4,785 | 1,395 | 52,195 |

Table 4.5 shows the number of APT reports and objects loaded into the system in relation to the total number of objects present in the platform. Thus, in Figure 4.9, each column indicates the MAP over all queries when similarity is measured through a specific type of object hash. The objects from the *Stuxnet* APT are retrieved with a MAP over 85% for subgraph-, node- and fact-based queries, while in the case of the Regin APT, fact-based queries allow to retrieve correct results with a MAP of 79%. Unlike in the previous case, the complexity of larger queries like subgraphs and nodes, do not compensate in average for the




(a) Number of constructs and facts created per number of STIX reports imported in MANTIS.

(b) Computation Time of the Simhash Fingerprint vs. the number of *n-grams* in the object.

Fig. 4.8 Scalability measurements respect to data size and fingerprint computation time.

small numbers of objects present in the database, making simpler fact queries more effective. Furthermore and as shown by the baseline performance, queries based on the similarity of objects in attributed graphs offer a more effective alternative than generic searches based on exact matchings of facts.

## 4.5 Limitations

The previous evaluation demonstrates the efficacy of MANTIS and our method to provide relevant similarity-based results for threat data queries. However, there exist certain limitations.

In the first place, the results of our platform are always bounded to the data present in the system when the query is issued. That is, an object cannot be retrieved, if it has not been imported into the system. Although an inherent limitation of every threat intelligence platform, this can be a disadvantage if a threat actor executes a targeted attack only once and without repurposing any part of its infrastructure or programming code. In such a situation, it is likely that the attack will not be documented and therefore never become part of a repository or feed of threat data. For such events where no correlation is possible, reactive solutions like intrusion detection or behavioral analysis can be more effective to prevent and thwart the attack. For example, we propose in this work a method for detection of spear-phishing emails as the first step of

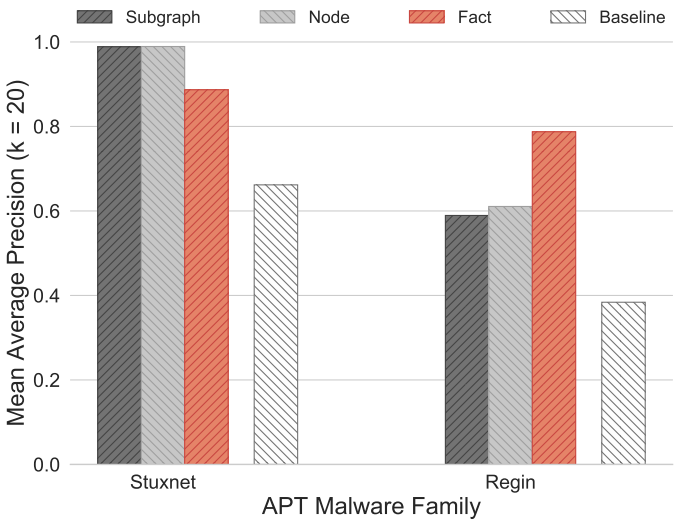



Fig. 4.9 MAP for query objects of APT families and comparison with baseline performance of standard search engines based on exact strings matching.

a comprehensive strategy against targeted attacks. Even if isolated, an email labeled as suspicious can lead to further investigation, enabling the analyst to document the incident in our threat intelligence platform.

Second, as other systems that aim at analyzing threat data, MANTIS is also subject to possible evasion attacks. For instance, a threat actor that targets an organization could use several types of attacks as part of a unique but large campaign. If the characteristics of such attacks are chosen to be different enough, it is possible that these events can not be linked to each other through our similarity analysis, even if each one of them is well documented.

Finally, our method compares $n$-gram from constructs and facts and, therefore determines similarity at a lexical level. Such a comparison can lead to false positives when unrelated objects share certain $n$-grams. Yet, as described in Section 4.3.1, this type of feature representation enables to correlate heterogeneous data even in situations where the standard is used incorrectly or when the type of the data is unknown like, for example, in the case of binary strings that are part of indicators of compromise.

## 4.6 Related Work

The body of work addressing threat intelligence issues has seen a surge in recent years thanks to the development of new sharing formats. Interestingly however, almost no previous work has been concerned with unifying and comparing

the data described through different standards, as we do in this chapter. Yet, several active and relevant areas exist that explore related research questions:

**Threat intelligence.** Although the security community has always been keen on sharing threat data for the sake of learning about new attacks and building better defenses, such pieces of individual information lack the necessary context to capture the complexity of the current threat landscape. As discussed by Barnum [7], the community-driven effort to design and extend the STIX format constitutes the most relevant and recent work to define a language that can represent such information in a structured and holistic way.

Being this a recent development, current academic research is yet trying to understand the ecosystem of threat intelligence data by creating taxonomies and models [16]. Most researchers recognize the benefits of these technologies but their focus still lies on the design and implementation of efficient sharing systems [135, 140, 84] and the privacy implications resulting from distributing sensitive security data across heterogeneous organizations [49, 68]. Moreover, practitioners acknowledge the potential improvements for situational awareness [50, 111] that comes from the sharing, storage and analysis of threat intelligence information but also the difficulty to ensure consistent interpretation without the need of the analyst. Kampanakis [79], for example, presents an analysis of all the standards under current development and points to the underestimated challenge of data collection and automatic analysis. This is exactly the field of operation of MANTIS.

Most approaches in this direction stem from non-academic initiatives and are being developed both by the security community and by commercial companies like Microsoft [57], which holds large amounts of security data from its customers. For instance, the open-source framework CRITS [34] presents some resemblance to MANTIS. In particular, bucket lists and relationships can be assigned to top level objects in order to identify campaigns and attributions. However, these assignments need to be done manually by the analyst whereas finding such correlations and matchings automatically is precisely the main goal of our method. Finally, Woods et al. [154] have recently proposed a system to infer similarity relationships and functional clusters of indicators using information about reporting patterns. Although close to our work in its goal and methodology, their approach relies on data not based on standardized open formats.

**Information retrieval for security.** Tangential to our research is the field of information retrieval which covers a huge body of previous research and work. For brevity, we herein consider only previous research which like ours, makes use of information retrieval and data mining techniques for solving security problems. In particular, there exist several authors that deal with the question of how to efficiently detect and analyze new malware variants which have been submitted to application stores or analysis platforms by analyzing the output reports of their dynamic and static analysis [e.g., 59, 22, 76, 9]. Graziano et al. [59] present, for example, a method that can be used to identify new malware variants in samples that have been submitted to such an analysis platform. To this end, they cluster the samples based on binary similarity using ssdeep fuzzy hashes and code-based features which allow them to identify similar malware samples using machine learning techniques. Although we also address in our work the problem of identifying similar strains of malware, both in Chapter 3 and in this chapter, their scope is limited to types of malware, where the broader view achieved by our platform allow us to pinpoint disconnected elements from the same campaign.

Another line of research in security that has combined information retrieval techniques and analysis of structured data focuses on the identification of similar or even copied segments in code of large software projects [144, 132, 85]. In particular, Uddin et al. [144] demonstrate in their work that the simhash algorithm can help to detect similar code regions. While different in scope to our method, their approach also proves the effectiveness of the method proposed by Charikar as the basis to implement techniques that can identify similar entities in large repositories of data.

## 4.7 Summary

In this chapter we present MANTIS, a system that enables the authoring, collection and, most importantly, the analysis and correlation of threat intelligence data. To the best of our knowledge, MANTIS was at its original introduction, the first open-source platform to provide a unified representation of constructs from standard threat data that allows for assessing the similarity between heterogeneous reports at different levels of granularity regardless of their content, size or structure.

We evaluate the performance of MANTIS as an information retrieval system for threat intelligence in a series of experiments, where it enables an effective

retrieval of relevant threat data. For example, given a documented security incident with a malware family, the similarity search integrated in MANTIS allows to retrieve related objects with a mean average precision of over 80%. That is, 4 out of 5 returned results correspond to the same malware family as the query. Moreover, we demonstrate the ability of our system to handle a continuous stream of data in terms of growth and computation time of our similarity measure. Finally, we show with a simple but illustrative use case how MANTIS can be used to assist the analyst in her investigation of a security incident.

In summary, after a threat has been detected and characterized through further analysis as discussed in Chapters 2 and 3, the platform and methods introduced in this chapter represent an effective set of tools for the response phase, allowing the analyst to complete a holistic strategy against targeted attacks.

# 05

# Conclusions and Outlook

The ability to inflict significant damage to a country, its institutions or the members of civil society has been reserved in the past to nation-states or organizations with a military background. Today, however, and thanks to the unbounded reach of unsecure software, we are witnessing a global computer security arms race. Not only traditional actors are investing heavily on offensive computer capabilities, but a whole new legal and mostly unregulated industry is thriving by providing malicious software to law enforcement and intelligence agencies.

Although research in software security has yielded massive improvements during the last two decades,there is, unfortunately, no general change of paradigm in sight that can guarantee the implementation of secure programs. At the same time and while awareness about the security risks of software systems keeps increasing among users, social engineering techniques will continue to be effective against the general population. In addition to that, such risks will be exacerbated by connecting everybody and everything to the internet. Every year around 200 million new users get online [75] and networking capabilities are added to every piece of infrastructure, transforming de facto everything into a connected computer. As a result, the number of potential targets, the techniques for compromise and the opportunities for causing damage will continue to steadily increase.

Consequently, there is an acute need for assisting security experts with innovative technical solutions against security risks in general and, considering the greater challenge posed by motivated threat actors, against targeted attacks in particular.

In this thesis, we aim, thus, at providing a holistic solution against targeted attacks by focusing on specific problems faced during each one of the phases of a comprehensive defense strategy: detection, analysis and response. Respectively, we propose methods to detect spear-phishing emails, perform malware triage at scale and generate insights based on large scale threat intelligence.

With our work, we provide security practitioners with an effective set of tools for fending off advanced persistent threats. First, our approach to detect targeted emails without relying on their content renders the main vector exploited in targeted attacks unusable. While actors behind targeted attacks make use of several vectors for infection such as watering hole websites or trojanized software updates, as of 2017, 71% of organized groups tracked by Symantec relied on spear-phishing messages to compromise their victims [142]. Second, our techniques for malware triage enable the analyst to extend the capabilities of traditional sandboxing approaches to leverage the recent advances in machine learning classification and exploiting the structural attributes of binary code. Although the use of zero-day exploits is not prevalent in targeted campaigns, malicious code is still the main instrument for stealing, spying or sabotaging and therefore a key source of threat intelligence for defenders. Finally, our information-retrieval platform for threat intelligence enables the collection, authoring and correlation of threat data in order to link traces of ongoing attacks with existing information about known actors.

As a whole, the approaches proposed in this thesis can be easily adopted and implemented locally by the security team at any organization. For instance, as discussed in Chapter 2, content in suspicious emails can be blocked before reaching its target. These attachments or files obtained through sensitive links can be analyzed through the methods introduced in Chapter 3 and any information obtained can be correlated with additional threat intelligence as proposed in Chapter 4.

Against these measures, threat actors will be forced to put more effort into carrying out a successful attack. For instance, by obtaining email data to attempt evading our detection method, investing more into the implementation of malicious code to avoid attribution and developing new tactics to remain stealth. As a result, defenders will be better protected, also in the long term, as

the features proposed for detecting spear-phishing are based on metadata that is expected to remain necessary for email to function correctly. In addition, other messaging protocols without proper authentication measures could implement a similar approach, as long as there exists metadata that can be used to build a profile of each user. In the near future, such an approach could be used to protect users from social engineering attacks performed through chatbots with high language capabilities. Moreover, the approaches proposed for malware triage and correlation of threat intelligence can help heightening awareness among the security community leading to better coordination in defensive research and increased shared support for changes that complement technical solutions at the policy level.

From a methodological perspective, the solutions proposed in this thesis follow a similar technical schema. In order to exploit the inherent structure of data, corresponding abstractions are designed to let machine learning algorithms operate on the input problem space. Such an approach is extensible to other problems for which the identification of patterns in large pools of data can represent a strategic advantage, a usual situation in computer security in particular but also in computer science in general. Furthermore, such a technical schema allows for reusing the abstractions proposed in this thesis and applying similar data analysis approaches to different problems. We explore some of these ideas for future work in Section 5.2.

In the following, we contextualize and summarize the solutions and results proposed in this thesis. Then and to conclude our work, we outline different avenues for research, for which the ideas introduced in this dissertation can serve as stepping stones.

## 5.1 Summary of Results

Following the structure of our multifaceted proposition for defense against targeted attacks and organized along the phases of detection, analysis and response, the main results introduced in this thesis are thus three-fold. First, we propose a method to detect the attempt to compromise a target through the most common attack vector. Next, we present a series of analysis mechanisms based on the structural characterization of binary code that supports the analyst at understanding malicious code at scale and, finally, we introduce a platform that enables the response of the analyst in terms of authoring, sharing

and correlating threat intelligence. Each of these solutions is made possible by a series of particular results that we summarize as follows.

**Detection.** In this thesis, we have introduced a content-agnostic method for detecting spoofed emails as a proxy for thwarting spear-phishing attacks. We identify a series of behavior, composition and transport email features, that allow us to to define sender profiles and effectively characterize each email sender in the common absence of additional authentication mechanisms. Given the ability of a resourceful actor to create seemingly legit emails, we then propose to use these content-agnostic profiles as input for machine learning classifiers and identify a mismatch between their output and the sender of an email as a spoofing attempt.

Our experiments demonstrate that our approach can discriminate thousands of senders and identify spoofed emails with high accuracy. Moreover, we show that the traits of a sender profile are hard to guess and spoofing only succeeds if entire emails of a sender are available to the attacker (Chapter 2).

**Analysis.** We have proposed a representation for binary code based on function call graphs and a generic labeling scheme that enables the analyst to obtain a structural characterization of malicious code without information about function names and that is robust against certain obfuscation techniques, such as function renaming or instruction reordering.

Based on this representation we have introduced two graph embedding approaches that complement each other along the trade-off between explainability and accuracy. First, we have proposed an explicit high dimensional feature map inspired by the neighborhood hash graph kernel which allows for explainable decisions when used in combination with linear machine learning algorithms. Second, we have proposed an approach to learn a low dimensional feature map through a deep neural network architecture based on an adapted implementation of *structure2vec* and parametrized through a siamese network. This architecture allows to embed latent variable models into feature spaces using discriminative information, which in our problem space can be defined as the family or campaign of a malware sample.

With the help of a well known dataset of x68 malware, we evaluate the performance of a series of state-of-the-art machine learning algorithms for malware triage when fed with data embedded in both feature maps. To this end, we compare their results at clustering and classification, in unsupervised,

semi-supervised and supervised experiments. In particular, we illustrate how clustering results can be improved by learning a feature map when some labeling information is available. Moreover, we show that, in a multiclass setup, both representations allow to assign an unknown malware sample to its family with high performance. Likewise, we demonstrate how both representations allow for an effective identification of new strains of malware in an anomaly detection setup (Chapter 3).

**Response.** We have introduced MANTIS, an open-source platform for authoring, sharing and collecting threat intelligence and, in addition, devised a unified representation based on attributed graphs for competing threat intelligence standards that serves as the data model for our platform. Based on this representation, we have proposed a similarity algorithm for attributed graphs that enables uncovering relations between threats at different levels of granularity. Incorporated into our platform, MANTIS becomes an information retrieval system that is capable of retrieving related reports given individual observations from security incidents.

We evaluate how an analyst can effectively leverage our platform for threat intelligence in a series of quantitative and qualitative experiments and show how our platform helps investigating a security incident given certain traces of an ongoing attack. For instance, we illustrate the performance of this analysis in two case studies with the attack campaigns Stuxnet and Regin (Chapter 4).

## 5.2 Future Work

Given that no solution will ever provide perfect security in isolation, we believe that the methods proposed in this thesis raise the bar against targeted threats and will help security practitioners thwart, understand and prevent persistent attacks when used in combination with other security mechanisms and as part of an integral defense strategy. Furthermore, we are confident that many of the ideas introduced in this dissertation will foster new developments in the field and open new avenues for research. Some of these might include:

**Defense against targeted SEO Poisoning.** In very sophisticated cases, a targeted e-mail might not even contain a link or attachment but just a hint to a piece of information interesting to the victim (e.g. the name of a report, a person, an event, etc.). In this scenario, the attacker expects the victim to query a search engine for more details on the information. However, the attacker has

previously poisoned the results provided by the search engine for that specific query with one or more links to malicious sites. While our approach to identify targeted emails would also highlight such messages as suspicious, our method could be used in combination with language processing techniques to extract certain patterns that could be used as queries for identification of malicious sites and the further gathering of threat intelligence.

**Applications of structure-based code analysis.** Our approach for malware triage builds on graph classification concepts that have proven their utility for our specific problem. However, we expect these ideas to find additional applications in the security research field. For instance, in the context of malware analysis, our approach for structural analysis of code can be used for characterizing the behavior of malicious code during detection, as the explicit embedding can reveal what functions are the most characteristics in malware [see 56]. In the same vein, our ideas have already been useful for binary code attribution [18], where further research on targeted attacks could leverage our explicit representation to identify elements in code that characterize a certain actor. Likewise, these representations have also found applicability in the field of vulnerability discovery [e.g. 158, 157], where the embedded structure of binary code can help finding patterns that identify known types of errors in a program.

**Advanced analytics for large scale threat intelligence.** While threat intelligence standards are intended to ease sharing and analysis, most of the current discussion revolves around better exchange strategies. As a consequence, work on techniques for automatic analysis is almost non-existent and available tools mostly provide statistics and equality-based searches. Our work thus sets the ground for new research that in combination with modern data analysis techniques aims at capitalizing the structured design of the standards for a better insight on threats. While the adoption of standards is still increasing, security analysts need to take full advantage of the possibilities offered by these formats. Although still underused, these are certainly complex and allow to capture with a high degree of precision information regarding, not only threats, but also courses of actions or defense strategies. Furthermore, we observe a substantial lack of public threat intelligence resources. Without question, the number of threats continues to grow but most valuable and elaborated threat data is only available under subscription feeds provided by private

companies or held within organizations. This prevents small entities with little resources like NGOs or non-profits from accessing these data and improving their protection mechanisms. Therefore, we encourage national CERTs and other public organizations to publish their threat data by means of standardized formats.

# 0A

# Traits in Email Structure for Characterization of Senders

Tables A.1, A.2 and A.3 provide an overview of the different traits characterizing the behavior, composition and transport of emails, respectively.

Table A.1 List of behavior features.

| Identifier | Cardinality | Description | Examples |
|---|---|---|---|
| attachment-type | n | Type of attachment | attachment-type(image) |
| hdr-empty | n | Headers with empty values | hdr-empty(cc) |
| hdr-local-domain | n | Headers indicating local domains | hdr-local-domain(to:DO) |
| hdr-related-mails | n | Headers indicating relation to other emails | hdr-related-mails(subject:re) |
| hdr-count | n | Number of standard headers and their values | hdrcount(cc:1:2+) |
| hdr-x | n | Occurrences of non-standard headers | hdr-x(x-virus-scanned) |
| msgid | n | Structural description of Message-Id header | msgid(<A.A@H>) |
| reply-to | n | Hashed sender in Reply-To header | reply-to(xxx) |
| resent | 1 | Headers indicate redistribution | resent(1) |
| return-path | n | Sender in Return-Path header | return-path(full:same-as-from) |
| text-quoted | 1 | Ratio of quoted total text in main part | text-quoted(0.3) |
| frompart | n | 2-grams of From field | frompart(xxx:yyy) |
| from | n | Multiple senders in From header | from(full:*) |

Table A.2 List of composition features.

| Identifier | Cardinality | Description | Examples |
|---|---|---|---|
| base64 | n | Peculiarities of Base64 transfer encoding | base64(linelen(72)) |
| quoted-printable | n | Peculiarities of Quoted-Printable transfer encoding | quoted-printable(unencoded-ctrl) |
| 7bit | n | Peculiarities of 7bit transfer encoding | 7bit(7bit-contains-8bit) |
| 8bit | n | Peculiarities of 8bit transfer encoding | 8bit(long-line) |
| attachment-ext | n | Extension of the attachment | attachment-ext(doc) |
| attachment-mism | n | Mismatch of attachment type and extension | attachment-mism(doc\|zip) |
| attachment-sig | 1 | Signature of how the attachment is specified | attachment-sig(fTnT) |
| inline-ext | n | Extension of attachment when disposition inline | inline-ext(jpeg) |
| nodisposition-ext | n | Extension of attachment if no disposition is given | nodisposition-ext(jpeg) |
| boundary | n | Structural description of the MIME boundary | boundary(-=_H-H-H) |
| hdr-syntax | n | Syntactic format of headers | hdr-syntax(subject:q:ISO-8859-1) |
| hdr-pair | n | Pair-wise order of headers | hdr-pair(from:date) |
| part-hdr-pair | n | Pair-wise order of headers in MIME parts | part-hdr-pair(content-type:content-id) |
| ua | n | Simplified name of user agent | ua(outlook16) |
| preamble | n | Digest of the MIME preamble | preamble(c928c8bf) |
| mime | n | Peculiarities of MIME usage | mime(cd:inline+filename) |
| depth | 1 | Depth of the MIME structure | depth(2) |
| mime-warning | n | Minor problems in MIME structure | mime-warning(invalid-content-type) |
| mime-error | n | Major problems in MIME structure | mime-error(paramval-junk) |
| part-path | n | Path to MIME parts | part-path(alt(R).1:html) |
| part-size | n | Size of MIME parts | part-size(html:10:1000) |
| part-type | n | Type of MIME parts | part-type(image:base64) |

Table A.3 List of transport features.

| Identifier | Cardinality | Description | Examples |
|---|---|---|---|
| `dkim` | n | Results of DKIM validation | `dkim(1:valid), dkim(2:invalid)` |
| `rcvd` | 1 | Number of `Received` headers | `rccvd(13)` |
| `rcvd-pair` | n | Hashes of previous and current `Received` header | `rcvd-pair(xxx:yyy)` |
| `rcvd-mta` | n | Hashes of MTA features at given header position | `rcvd-mta(1:XXX)` |
| `rcvd-src` | n | Hashes of source features at given header position | `rcvd-src(2:xxx)` |
| `rcvd-tls` | n | Hashes of TLS features at given header position | `rcvd-tls(3:xxx)` |
| `rcvd-tocc` | n | Occurrences of `To` field in `Received` headers | `rcvd-tocc(to:x1)` |
| `hdrtz` | 1 | Path of time zones from `Received` headers | `hdrtz(-0200:+0800)` |
| `hdrtzcost` | 1 | Cost of transport based on the changes in time zones | `hdrtzcost(6)` |
| `srcip-asn` | 1 | ASN for source IP address of client | `srcip-asn(8881)` |
| `srcip-spf` | 1 | SPF result for source IP address of client | `srcip-spf(spf:Pass)` |

# References


[1] Sergi Alvarez. *Radare2*. URL: https://github.com/radare/radare2.

[2] Rohan Mahesh Amin. "Detecting Targeted Malicious Email Through Supervised Classification of Persistent Threat and Recipient Oriented Features". PhD thesis. George Washington University, 2010.

[3] Blake Anderson, Daniel Quist, Joshua Neil, Curtis Storlie, and Terran Lane. "Graph-based malware detection using dynamic analysis". In: *Journal in Computer Virology* (2011).

[4] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. "Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket". In: *Proc. of Network and Distributed System Security Symposium (NDSS)*. 2014.

[5] Pierre Baldi and Yves Chauvin. "Neural Networks for Fingerprint Recognition". In: *Neural Computation* (1993).

[6] Annalisa Barla, Francesca Odone, and Alessandro Verri. "Histogram intersection kernel for image classification". In: *Proc. of International Conference on Image Processing (ICIP)*. 2003.

[7] Sean Barnum. *Standardizing cyber threat intelligence information with the Structured Threat Information eXpression (STIX)*. Tech. rep. MITRE Corporation, 2014.

[8] Brian Bartholomew and Juan Andres Guerrero-Saade. "Wave Your False Flags! Deception Tactics Muddying Attribution in Targeted Attacks". In: *Virus Bulletin* October (2016).

[9] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. "Scalable, Behavior-Based Malware Clustering". In: *Proc. of Network and Distributed System Security Symposium (NDSS)*. 2009.

[10] Alina Beygelzimer, Sham Kakade, and John Langford. "Cover trees for nearest neighbor". In: *Proc. of International Conference on Machine Learning (ICML)*. 2006.

[11] Stevens Le Blond, Adina Uritesc, Cédric Gilbert, Zheng Leong Chua, Prateek Saxena, and Engin Kirda. "A Look at Targeted Attacks Through the Lense of an NGO". In: *Proc. of USENIX Security Symposium*. 2014.

[12] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. "Protein function prediction via graph kernels". In: *Bioinformatics* (2005).

[13] Karsten Michael Borgwardt. "Graph Kernels". PhD thesis. Ludwig Maximilian University of Munich, 2007.

[14] Jane Bromley, James W. Bentz, Léon Bottoy, Isabelle Guyon, Yann Lecun, Cliff Moore, Eduard Säckinger, and Roopak Shah. "Signature Verification Using a "Siamese" Time Delay Neural Network". In: *International Journal of Pattern Recognition and Artificial Intelligence* (1993).

[15] *BuildWith Technology Lookup*. 2017. URL: https://builtwith.com.

[16] Eric W. Burger, Michael D. Goodman, Panos Kampanakis, and Kevin A. Zhu. "Taxonomy Model for Cyber Threat Intelligence Information Exchange Technologies". In: *Proc. of ACM Workshop on Information Sharing & Security*. 2014.

[17] Juan Caballero, Chris Grier, Christian Kreibich, and Vern Paxson. "Measuring Pay-per-Install: The Commoditization of Malware Distribution". In: *Proc. of USENIX Security Symposium*. 2011.

[18] Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. "De-anonymizing Programmers via Code Stylometry". In: *Proc. of USENIX Security Symposium*. 2015.

[19] Deanna D. Caputo, Shari Lawrence Pfleeger, Jesse D. Freeman, and M. Eric Johnson. "Going spear phishing: Exploring embedded training and awareness". In: *IEEE Security & Privacy* 12.1 (2014).

[20] Silvio Cesare and Yang Xiang. "Classification of malware using structured control flow". In: *Proc. of 8th Australasian Symposium on Parallel and Distributed Computing*. 2010.

[21] Silvio Cesare and Yang Xiang. "Malware Variant Detection Using Similarity Search over Sets of Control Flow Graphs". In: *Proc. of International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 2011.

[22] Saurabh Chakradeo, Bradley Reaves, Patrick Traynor, and William Enck. "MAST: Triage for Market-scale Mobile Malware Analysis". In: *Proc. of ACM Conference on Security and Privacy in Wireless and Mobile Networks (WISEC)*. 2013.

[23] Moses S. Charikar. "Similarity estimation techniques from rounding algorithms". In: *Proc. of 34th annual ACM symposium on Theory of computing*. 2002.

[24] Ping Chen, Lieven Desmet, and Christophe Huygens. "A study on advanced persistent threats". In: *Proc of IFIP International Conference on Communications and Multimedia Security*. 2014.

[25] Zhi-Guo Chen, Ho-Seok Kang, Shang-Nan Yin, and Sung-Ryul Kim. "Automatic Ransomware Detection and Analysis Based on Dynamic API Calls Flow Graph". In: *Proc. of International Conference on Research in Adaptive and Convergent Systems (RACS)*. 2017.

[26] Sumit Chopra, Raia Hadsell, and Yann LeCun. "Learning a similiarty metric discriminatively, with application to face verification (CVPR)". In: *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*. 2005.

[27] D Manning Christopher, Raghavan Prabhakar, and Schutza Hinrich. "Introduction to information retrieval". In: *An Introduction To Information Retrieval* 151.177 (2008).

[28] *Collective Intelligence Framework*. 2016. URL: http://csirtgadgets.org/collective-intelligence-framework.

[29] Symantec Corporation. *Advanced Persistent Threats: A Symantec Perspective*. 2011.

[30] Symantec Corporation. *Butterfly: Corporate spies out for financial gain*. 2015.

[31] Symantec Corporation. *Regin: Top-tier espionage tool enables stealthy surveillance*. Symantec Security Response. 2015.

[32] Symantec Corporation. *Stuxnet 0.5: The Missing Link*. Symantec Security Response. 2013.

[33] Masashi Crete-Nishihata, Jakub Dalek, and Ronald Deibert. *Communities@ Risk: Targeted Digital Threats Against Civil Society*. Citizen Lab, Munk Centre for International Studies, University of Toronto, 2014.

[34] *Collaborative Research Into Threats*. 2016. URL: http://crits.github.io.

[35] Jonathan Crussell, Clint Gibler, and Hao Chen. "Attack of the Clones: Detecting Cloned Applications on Android Markets". In: *Proc. of European Symposium on Research in Computer Security (ESORICS)*. 2012.

[36] Neil D. Lawrence and Bernhard Schölkopf. "Estimating a Kernel Fisher Discriminant in the Presence of Label Noise". In: *Proc. of International Conference on Machine Learning (ICML)*. 2001.

[37] Hanjun Dai, Bo Dai, and Le Song. "Discriminative Embeddings of Latent Variable Models for Structured Data". In: *arXiv* (2016).

[38] Marc Damashek. "Gauging Similarity with $n$-Grams: Language-Independent Categorization of Text". In: *Science* 267.5199 (1995).

[39] Roman Danyliw, Jan Meijer, and Yuri Demchenko. *The Incident Object Description Exchange Format (IODEF)*. Tech. rep. IETF RFC 5070, 2007.

[40] Red en Defensa de los Derechos Digitales. *Gobierno Espía - Vigilancia sistemática a periodistas y defensores de derechos humanos en México*. Tech. rep. 2017. URL: https://r3d.mx/gobiernoespia/.

[41] Stephen Doherty, Jozsef Gegeny, Branko Spasojevic, and Jonell Baltazar. "Hidden Lynx–Professional Hackers for Hire". In: *Symantec Security Response Blog* (2013).

[42] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[43] Thomas Dullien and Rolf Rolles. "Graph-based comparison of executable objects". In: *Proc. of Symposium sur la Securite des Technologies de L'information et des communications*. 2005.

[44] Sevtap Duman, Kubra Kalkan Cakmakci, Manuel Egele, William Robertson, and Engin Kirda. "EmailProfiler: Spearphishing Filtering with Header and Stylometric Features of Emails". In: *Proc. of IEEE Computer Software and Applications Conference (COMPSAC)*. 2016.

[45] Ming Fan, Jun Liu, Xiapu Luo, Kai Chen, Zhenzhou Tian, Qinghua Zheng, and Ting Liu. "Android Malware Familial Classification and Representative Sample Selection via Frequent Subgraph Analysis". In: *IEEE Transactions on Information Forensics and Security (TIFS)* (2018).

[46] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. "LIBLINEAR: A library for large linear classification". In: *Journal of Machine Learning Research (JMLR)* 9.Aug (2008).

[47] Tom Fawcett. "An introduction to ROC analysis". In: *Pattern Recognition Letters* 27.8 (2006).

[48] Ian Fette, Norman Sadeh, and Anthony Tomasic. "Learning to Detect Phishing Emails". In: *Proc. of International World Wide Web Conference (WWW)*. 2007.

[49] Gina Fisk, Calvin Ardi, Neale Pickett, John Heidemann, Mike Fisk, and Christos Papadopoulos. "Privacy Principles for Sharing Cyber Security Data". In: *Proc. of IEEE International Workshop on Privacy Engineering*. 2015.

[50] Peter Fonash. *Using Automated Cyber Threat Exchange to Turn the Tide against DDOS*. RSA Conference. 2014.

[51] Ian D. Foster, Jon Larson, Max Masich, Alex C. Snoeren, Stefan Savage, and Kirill Levchenko. "Security by Any Other Name: On the Effectiveness of Provider Based Email Security". In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2015.

[52] Hugo Gascon, Bernd Grobauer, Thomas Schreck, Lukas Rist, Daniel Arp, and Konrad Rieck. "Mining Attributed Graphs for Threat Intelligence". In: *Proc. of ACM Conference on Data and Application Security and Privacy (CODASPY)*. 2017.

[53] Hugo Gascon, Sebastian Uellenbeck, Christopher Wolf, and Konrad Rieck. "Continuous Authentication on Mobile Devices by Analysis of Typing Motion Behavior". In: *Proc. of GI Conference "Sicherheit" (Sicherheit, Schutz und Verlässlichkeit)*. 2014.

[54] Hugo Gascon, Steffen Ullrich, Benjamin Stritter, and Konrad Rieck. "Reading Between the Lines: Content-Agnostic Detection of Spear-Phishing Emails". In: *Recent Adances in Intrusion Detection (RAID)*. 2018.

[55] Hugo Gascon, Christian Wressnegger, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. "Pulsar: Stateful black-box fuzzing of proprietary network protocols". In: *Proc. of International Conference on Security and Privacy in Communication Networks (SECURECOMM)*. 2015.

[56] Hugo Gascon, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. "Structural Detection of Android Malware using Embedded Call Graphs". In: *Proc. of ACM Workshop on Artificial Intelligence and Security (AISEC)*. 2013.

[57] Cristin Goodwin, J Paul Nicholas, Jerry Bryant, Kaja Ciglic, Aaron Kleiner, Cornelia Kutterer, Alison Massagli, Angela Mckay, Paul Mckitrick, Jan Neutze, Tyson Storch, and Kevin Sullivan. *A framework for cybersecurity information sharing and risk reduction*. Tech. rep. Microsoft Corporation, 2015.

[58] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. "RiskRanker: scalable and accurate zero-day android malware detection". In: *Proc. of International Conference on Mobile Systems, Applications, and Services (MOBISYS)*. 2012.

[59] Mariano Graziano, Davide Canali, Leyla Bilge, Andrea Lanzi, and Davide Balzarotti. "Needles in a Haystack: Mining Information from Public Dynamic Analysis Sandboxes for Malware Intelligence". In: *Proc. of USENIX Security Symposium*. 2015.

[60] Greathorn. *Spear Phishing Report*. 2017. URL: https://info.greathorn.com/2017-spear-phishing-report.

[61] Surbhi Gupta, Abhishek Singhal, and Akanksha Kapoor. "A literature survey on social engineering attacks: Phishing attack". In: *Proc. of IEEE International Conference on Computing, Communication and Automation (ICCCA)*. 2016.

[62] Raia Hadsell, Sumit Chopra, and Yann LeCun. "Dimensionality reduction by learning an invariant mapping". In: *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2006).

[63] Richard W. Hamming. "Error detecting and error correcting codes". In: *Bell System Technical Journal* 29.2 (1950).

[64] Fei Han and Yu Shen. "Accurate Spear Phishing Campaign Attribution and Early Detection". In: *Proc. of ACM Symposium on Applied Computing (SAC)*. 2016.

[65] Steve Hanna, Edward Wu, Saung Li, Charles Chen, Dawn Song, and Ling Huang. "Juxtapp: A Scalable System for Detecting Code Reuse Among Android Applications". In: *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*. 2012.

[66] Seth Hardy, Masashi Crete-Nishihata, Katharine Kleemola, Adam Senft, Byron Sonne, Greg Wiseman, Phillipa Gill, and Ronald J. Deibert. "Targeted Threat Index: Characterizing and Quantifying Politically-Motivated Targeted Malware". In: *Proc. of USENIX Security Symposium*. 2014.

[67] David Haussler. *Convolution kernels on discrete structures*. Tech. rep. UCSC-CRL-99-10. UC Santa Cruz, 1999.

[68] Jorge L Hernandez-Ardieta, Juan E Tapiador, and Guillermo Suarez-Tangil. "Information sharing models for cooperative cyber defence". In: *Proc. of IEEE International Conference on Cyber Conflict (CyCon)*. 2013.

[69] Shohei Hido and Hisashi Kashima. "A linear-time graph kernel". In: *Proc. of International Conference on Data Mining (ICDM)* (2009).

[70] Grant Ho, Aashish Sharma Mobin Javed, Vern Paxson, and David Wagner. "Detecting Credential Spearphishing Attacks in Enterprise Settings". In: *Proc. of USENIX Security Symposium*. 2017.

[71] Xin Hu, Tzi-cker Chiueh, and Kang G. Shin. "Large-scale malware indexing using function-call graphs". In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2009.

[72] Eric M. Hutchins, Michael J. Cloppert, and Rohan M. Amin. "Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains". In: *6th Annual International Conference on Information Warfare and Security* (2011).

[73] FireEye Inc. *Evasive Tactics: Taidoor*. 2016. URL: https://www.fireeye.com/blog/threat-research/2013/09/evasive-tactics-taidoor-3.html.

[74] Trend Micro Inc. *Spear-Phishing Email: Most Favored APT Attack Bait*. Tech. rep. Trend Micro Inc., 2012.

[75] International Telecommunication Union (ITU). *ICT Facts and Figures 2017*. 2018. URL: https://www.itu.int/en/ITU-D/Statistics/Pages/facts/default.aspx.

[76] Jiyong Jang, David Brumley, and Shobha Venkataraman. "BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis". In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2011.

[77] Thorsten Joachims. *Learning to classify text using support vector machines: Methods, theory and algorithms*. Kluwer Academic Publishers Norwell, 2002.

[78] Thorsten Joachims. *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*. Tech. rep. 23. LS VIII, University of Dortmund, 1997.

[79] Panos Kampanakis. "Security Automation and Threat Information-Sharing Options". In: *IEEE Security & Privacy* 12.5 (2014).

[80] Joris Kinable and Orestis Kostakis. "Malware classification based on call graph clustering". In: *Journal in Computer Virology* (2011).

[81] Darien Kindlund, Ned Moran, and Rob Rachwald. *WORLD WAR C: Understanding Nation-State Motives Behind Today's Advanced Cyber Attacks*. Tech. rep. FireEye Inc., 2014.

[82] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiao-yong Zhou, and XiaoFeng Wang. "Effective and Efficient Malware Detection at the End Host". In: *Proc. of USENIX Security Symposium*. 2009.

[83] Risi Imre Kondor and John Lafferty. "Diffusion kernels on graphs and other discrete input spaces". In: *Proc. of International Conference on Machine Learning (ICML)* (2002).

[84] Maciej Korczynski, Ali Hamieh, Jun Ho Huh, Henrik Holm, S Raj Rajagopalan, and Nina H Fefferman. "DIAMoND: Distributed Intrusion/Anomaly Monitoring for Nonparametric Detection". In: *Proc. the 24th International Conference on Computer Communications and Networks*. 2015.

[85] Jens Krinke. "Identifying Similar Code with Program Dependence Graphs". In: *Proc. of Working Conference on Reverse Engineering (WCRE)*. 2001.

[86] Christopher Kruegel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna. "Polymorphic Worm Detection Using Structural Information of Executables". In: *Recent Adances in Intrusion Detection (RAID)*. 2005.

[87] Tammo Krueger, Hugo Gascon, Nicole Kraemer, and Konrad Rieck. "Learning Stateful Models for Network Honeypots". In: *Proc. of ACM Workshop on Artificial Intelligence and Security (AISEC)*. 2012.

[88] Andreas Kurtz, Hugo Gascon, Tobias Becker, Konrad Rieck, and Felix Freiling. "Fingerprinting mobile devices using personalized configurations". In: *Proc. of Privacy Enhancing Technologies Symposium (PETS)*. 2016.

[89] Kaspersky Lab. *Advanced Threat Defense and Targeted Attack Risk Mitigation*. 2017.

[90] Kaspersky Lab. *Targeted cyberattacks logbook*. 2018. URL: https://apt.securelist.com/#!/threats.

[91] Kaspersky Lab. *The Regin Platform: Nation-State Ownage of GSM Networks*. 2014.

[92] Ralph Langner. "Stuxnet: Dissecting a Cyberwarfare Weapon". In: *IEEE Security and Privacy* 9.3 (2011).

[93] Eric Lin, John Aycock, and Mohammad Mannan. "Lightweight Client-Side Methods for Detecting Email Forgery". In: *Proc. of International Workshop on Information Security Applications (WISA)*. 2012.

[94] Chao Liu, Chen Chen, Jiawei Han, and Philip S. Yu. "GPLAG: detection of software plagiarism by program dependence graph analysis". In: *Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2006.

[95] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. "Learning to detect malicious URLs". In: *ACM Transactions on Intelligent Systems and Technology* 2.3 (2011).

[96] Laurens van der Maaten and Geoffrey Hinton. "Visualizing high-dimensional data using t-SNE". In: *Journal of Machine Learning Research (JMLR)* (2008).

[97] Mandiant. *APT1: Exposing one of China's cyber espionage units*. Tech. rep. Mandiant Intelligence Center, 2013.

[98] Mandiant. *Sophisticated Indicators for the Modern Threat Landscape: An Introduction to OpenIOC*. Tech. rep. Mandiant Whitepaper, 2013.

[99] Mandiant. *M-Trends 2017: A View from the Front Lines*. 2017.

[100] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. "Detecting near-duplicates for web crawling". In: *Proc. of International World Wide Web Conference (WWW)*. 2007.

[101] Bill Marczak and John Scott-Railton. *The Million Dollar Dissident*. Tech. rep. Munk School of Global Affairs' Citizen Lab, University of Toronto, 2016.

[102] Morgan Marquis-Boire, Claudio Guarnieri, and Ryan Gallagher. *Secret Malware In European Union Attack Linked to U.S. and British Intelligence*. 2014. URL: https://theintercept.com/2014/11/24/secret-regin-malware-belgacom-nsa-gchq.

[103] Morgan Marquis-Boire, Bill Marczak, Claudio Guarnieri, and John Scott-Railton. *For Their Eyes Only: The Commercialization of Digital Spying*. Tech. rep. Munk School of Global Affairs' Citizen Lab, University of Toronto, 2013.

[104] Morgan Marquis-Boire, Marion Marschalek, and Claudio Guarnieri. "Big game hunting: The peculiarities in nation-state malware research". In: *Black Hat, Las Vegas, NV, USA* (2015).

[105] Trend Micro. *Targeted Attacks*. 2018. URL: https://www.trendmicro.com/vinfo/us/security/definition/targeted-attacks.

[106] Jiang Ming, Meng Pan, and Debin Gao. "iBinHunt: Binary Hunting with Inter-procedural Control Flow". In: *Information Security and Cryptology (ICISC)*. 2012.

[107] Daniel Moore and and Rid Thomas. *Penquin's Moonlit Maze, The Dawn of Nation-State Digital Espionage*. Tech. rep. Kaspersky Lab, 2017. URL: https://securelist.com/penquins-moonlit-maze/77883/.

[108] Tatsuya Mori, Kazumichi Sato, Yousuke Takahashi, and Keisuke Ishibashi. "How is e-Mail Sender Authentication Used and Misused?" In: *Proc. of 8th Annual Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference (CEAS)*. 2011.

[109] University of Toronto Munk School of Global Affairs' Citizen Lab. *Malware Indicators*. 2017. URL: https://github.com/citizenlab/malware-indicators.

[110] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. "subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs". In: *arXiv* (2016).

[111] Mark Orlando. *Threat Intelligence is Dead. Long Live Threat Intelligence!* RSA Conference. 2015.

[112] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research (JMLR)* 12 (2011).

[113] Nicole Perlroth. *Researchers Find 25 Countries Using Surveillance Software*. 2013. URL: https://bits.blogs.nytimes.com/2013/03/13/researchers-find-25-countries-using-surveillance-software.

[114] Stephen Pritchard. *Espionage and industry in the internet era*. Financial Times. 2015. URL: https://www.ft.com/content/01714ea4-262e-11e5-bd83-71cb60e8f08c.

[115] Jathushan Rajasegaran and Suranga Seneviratne. "A Neural Embeddings Approach for Detecting Mobile Counterfeit Apps". In: *arXiv* (2018).

[116] Liva Ralaivola, Sanjay J Swamidass, Hiroto Saigo, and Pierre Baldi. "Graph kernels for chemical informatics". In: *Neural networks* 18.8 (2005).

[117] IBM Research. *PC Virus Timeline*. 2001. URL: https://web.archive.org/web/20121027045532/http://www.research.ibm.com:80/antivirus/timeline.htm.

[118] Ned Freed and Nathaniel Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. RFC 2045 (Draft Standard). RFC. Updated by RFCs 2184, 2231, 5335, 6532. Fremont, CA, USA: RFC Editor, Nov. 1996. URL: https://www.rfc-editor.org/rfc/rfc2045.txt.

[119] Ned Freed and Keith Moore. *MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations*. RFC 2231 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Nov. 1997. URL: https://www.rfc-editor.org/rfc/rfc2231.txt.

[120] Simon Josefsson. *The Base16, Base32, and Base64 Data Encodings*. RFC 4648 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Oct. 2006. URL: https://www.rfc-editor.org/rfc/rfc4648.txt.

[121] Jon Callas, Lutz Donnerhacke, Hal Finney, David Shaw, and Rodney Thayer. *OpenPGP Message Format*. RFC 4880 (Proposed Standard). RFC. Updated by RFC 5581. Fremont, CA, USA: RFC Editor, Nov. 2007. URL: https://www.rfc-editor.org/rfc/rfc4880.txt.

[122] Peter Resnick. *Internet Message Format*. RFC 5322 (Draft Standard). RFC. Updated by RFC 6854. Fremont, CA, USA: RFC Editor, Oct. 2008. URL: https://www.rfc-editor.org/rfc/rfc5322.txt.

[123] Blake Ramsdell and Sean Turner. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification*. RFC 5751 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Jan. 2010. URL: https://www.rfc-editor.org/rfc/rfc5751.txt.

[124] Dave Crocker, Tony Hansen, and Murray Kucherawy. *DomainKeys Identified Mail (DKIM) Signatures*. RFC 6376 (Internet Standard). RFC. Fremont, CA, USA: RFC Editor, Sept. 2011. URL: https://www.rfc-editor.org/rfc/rfc6376.txt.

[125] Scott Kitterman. *Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1*. RFC 7208 (Proposed Standard). RFC. Updated by RFC 7372. Fremont, CA, USA: RFC Editor, Apr. 2014. URL: https://www.rfc-editor.org/rfc/rfc7208.txt.

[126] Murray Kucherawy and Elizabeth Zwicky. *Domain-based Message Authentication, Reporting, and Conformance (DMARC)*. RFC 7489 (Informational). RFC. Fremont, CA, USA: RFC Editor, Mar. 2015. URL: https://www.rfc-editor.org/rfc/rfc7489.txt.

[127] Konrad Rieck, Christian Wressnegger, and Alexander Bikadorov. "Sally: A Tool for Embedding Strings in Vector Spaces". In: *Journal of Machine Learning Research (JMLR)* 13 (2012).

[128] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. "Microsoft Malware Classification Challenge". In: *arXiv* (2015).

[129] Andrew Rosenberg and Julia Hirschberg. "V-measure: A conditional entropy-based external cluster evaluation measure". In: *Proc. of joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*. 2007.

[130] Ishai Rosenberg, Guillaume Sicard, and Eli Omid David. "DeepAPT: Nation-State APT Attribution Using End-to-End Deep Neural Networks". In: *Proc. of International Conference on Artificial Neural Networks (ICANN)*. 2017.

[131] Ishai Rosenberg, Guillaume Sicard, and Eli Omid David. "End-to-End Deep Neural Networks and Transfer Learning for Automatic Analysis of Nation-State Malware". In: *Entropy* (2018).

[132] Andreas Sæbjørnsen, Jeremiah Willcock, Thomas Panas, Daniel Quinlan, and Zhendong Su. "Detecting Code Clones in Binary Executables". In: *Proc. of International Symposium on Software Testing and Analysis (ISSTA)*. 2009.

[133] Gerard Salton, Anita Wong, and Chung-Shu Yang. "A Vector Space Model for Automatic Indexing". In: *Communications of the ACM* 18.11 (1975).

[134] Secureworks. *Advanced Persistent Threats: Learn the ABCs of APTs - Part A*. 2016. URL: https://www.secureworks.com/blog/advanced-persistent-threats-apt-a.

[135] Oscar Serrano, Luc Dandurand, and Sarah Brown. "On the design of a cyber security data sharing system". In: *Proc. of ACM Workshop on Information Sharing & Collaborative Security (WISCS)*. 2014.

[136] Shanhu Shang, Ning Zheng, Jian Xu, Ming Xu, and Haiping Zhang. "Detecting malware variants via function-call graph similarity". In: *Proc. of International Conference on Malicious and Unwanted Software (MALWARE)*. 2010.

[137] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. "Efficient graphlet kernels for large graph comparison". In: *Artificial Intelligence and Statistics*. 2009.

[138] Aditya K. Sood and Richard Enbody. "Targeted cyber attacks, a superset of advanced persistent threats". In: *IEEE Security and Privacy* (2012).

[139] Splunk. *Splunk Threat Intelligence Dashboards*. 2016. URL: http://docs.splunk.com/Documentation/ES/4.2.0/User/ThreatIntelligence.

[140] Jessica Steinberger, Anna Sperotto, Mario Golling, and Harald Baier. "How to exchange security events? Overview and evaluation of formats and protocols". In: *Proc. of IEEE International Symposium on Integrated Network Management (IM)*. 2015.

[141] Gianluca Stringhini and Olivier Thonnard. "That Ain't You: Blocking Spearphishing Through Behavioral Modelling". In: *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*. 2015.

[142] Symantec Corporation. *2018 Internet security threat report*. Tech. rep. 2018. URL: https://www.symantec.com/content/dam/symantec/docs/reports/istr-23-2018-en.pdf.

[143] Trend Micro Threat Research Team. *The Taidoor Campaign. An In-Depth Analysis*. Tech. rep. Trend Micro, 2012.

[144] Md. Sharif Uddin, Chanchal K. Roy, Kevin A. Schneider, and Abram Hindle. "On the Effectiveness of Simhash for Detecting Near-Miss Clones in Large Scale Software Systems". In: *Proc. of Working Conference on Reverse Engineering (WCRE)*. 2011.

[145] Tristan Vanderbruggen. "Application of Deep-Learning to Compiler-Based Graphs". PhD thesis. University of Delaware, 2018.

[146] Tristan Vanderbruggen and John Cavazos. "Large-Scale Exploration of Feature Sets and Deep Learning Models to Classify Malicious Applications". In: *Resilience Week (RWS)*. 2017.

[147] Alien Vault. *Open Threat Exchange*. 2016. URL: https://www.alienvault.com/open-threat-exchange.

[148] Rakesh Verma, Narasimha Shashidhar, and Nabil Hossain. "Detecting Phishing Emails the Natural Language Way." In: *Proc. of European Symposium on Research in Computer Security (ESORICS)*. 2012.

[149] VirusTotal. URL: https://www.virustotal.com.

[150] Cynthia Wagner, Gerard Wagener, Radu State, and Thomas Engel. "Malware analysis with graph kernels and support vector machines". In: *Proc. of International Conference on Malicious and Unwanted Software (MALWARE)*. 2009.

[151] Jingguo Wang, Tejaswini Herath, Rui Chen, Arun Vishwanath, and H. Raghav Rao. "Research article phishing susceptibility: An investigation into the processing of a targeted spear phishing email". In: *IEEE Transactions on Professional Communication* 55.4 (2012).

[152] Ke Wang, Janak J. Parekh, and Salvatore J. Stolfo. "Anagram: A Content Anomaly Detector Resistant To Mimicry Attack". In: *Recent Adances in Intrusion Detection (RAID)*. 2006.

[153] Barry Wellman. "Physical Place and CyberPlace: The Rise of Personalized Networking". In: *International Journal of Urban and Regional Research* 25.2 (2001).

[154] Bronwyn Woods, Samuel J. Perl, and Brian Lindauer. "Data Mining for Efficient Collaborative Information Discovery Categories and Subject Descriptors". In: *Proc. of 2nd ACM Workshop on Information Sharing and Collaborative Security*. 2015.

[155] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. "Neural Network-based Graph Embedding for Cross-Platform Binary Code Search". In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2017.

[156] Fabian Yamaguchi, Markus Lottmann, and Konrad Rieck. "Generalized Vulnerability Extrapolation using Abstract Syntax Trees". In: *Proc. of Annual Computer Security Applications Conference (ACSAC)*. 2012.

[157] Fabian Yamaguchi, Alwin Maier, Hugo Gascon, and Konrad Rieck. "Automatic Inference of Search Patterns for Taint-Style Vulnerabilities". In: *Proc. of IEEE Symposium on Security and Privacy*. 2015.

[158] Fabian Yamaguchi, Christian Wressnegger, Hugo Gascon, and Konrad Rieck. "Chucky: Exposing Missing Checks in Source Code for Vulnerability Discovery". In: *Proc. of ACM Conference on Computer and Communications Security (CCS)*. 2013.

[159] Kim Zetter. *Countdown to Zero Day: Stuxnet and the launch of the world's first digital weapon*. Broadway books, 2014.

[160] Kim Zetter. *Inside the cunning, unprecedented hack of Ukraine's power grid*. Wired. 2016. URL: https://www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/.

[161] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. "Detecting repackaged smartphone applications in third-party android marketplaces". In: *Proc. of ACM Conference on Data and Application Security and Privacy (CODASPY)*. 2012.