An Architecture for Inline Anomaly Detection

Tammo Krueger, Christian Gehl, Konrad Rieck and Pavel Laskov*
Fraunhofer Institute FIRST
Intelligent Data Analysis, Berlin, Germany
{krutam,gehl,rieck,laskov}@first.fraunhofer.de

Abstract

In this paper we propose an intrusion prevention system (IPS) which operates inline and is capable to detect unknown attacks using anomaly detection methods. Incorporated in the framework of a packet filter each incoming packet is analyzed and – according to an internal connection state and a computed anomaly score – either delivered to the production system, redirected to a special hardened system or logged to a network sink for later analysis. Runtime measurements of an actual implementation prove that the performance overhead of the system is sufficient for inline processing. Accuracy measurements on real network data yield improvements especially in the number of false positives, which are reduced by a factor of five compared to a plain anomaly detector.

1. Introduction

Why are intrusion detection systems (IDS) still struggling for their acceptance in everyday practice? From the conceptional perspective, their main practical challenge is not how to detect attacks, but rather: how to act upon the detected events? From a technical perspective, the main challenge is wire speed. The entire decision-making has to match high traffic rates, which will show no signs of ever going down in the future. Putting these two requirements together, one realizes the need to develop *inline* intrusion prevention systems (IPS) – the ones that act as intelligent firewall-like black-boxes deployed in the main traffic path and deliver a "guaranteed protection" against all sorts of evil.

Departing from this science-fiction-like picture, one should note that among the two main classes of IDS – misuse and anomaly detection – inline methods for misuse detection have received considerable attention in recent work. Architectures for high-performance signature-based IDS with inline capabilities have been proposed using FPGAs [2, 5], network [3] or graphics processors [7, 18]. Signature-based systems, however, fail to provide adequate protection against novel attacks, such as zero-day exploits and rapid worm outbreaks, since signatures have to be supplied and installed before the detection of new attacks is possible.

Anomaly detection methods fill this gap and enable the detection of truly novel attacks. Previous work has addressed anomaly-based intrusion prevention for specific scenarios, such as web applications and clients [1, 17]. The main contribution of this article is, in contrast, a *service-independent* architecture in which decision-making is performed at the network layer in the TCP/IP stack whereas anomaly detection runs at the application layer. Using advanced packet content analysis techniques [19, 15], our system is applicable to *any* network services.

The goal of our system is not only to crank a high number of bytes per second, but also to investigate protection mechanisms feasible for integration with anomaly detection techniques. We attempt to balance the imperfect detection in anomaly-based IDS by a judicious choice of response mechanisms. Specifically, our design currently involves three actions chosen according to assigned anomaly scores: (1) forwarding to a production system, (2) redirection to a hardened system (*shadow system*), and (3) redirection to a monitored network sink (*forensic sink*). The system architecture and implementation details of these instruments are provided in Section 2, followed by the description of anomaly detection algorithms in Section 3 and the experimental evaluation in Section 4. A comparative discussion of the proposed architecture and alternative approaches is provided in Section 5.

At this stage in the development, certain shortcuts had to be taken in order to reduce the complexity of our system, for the sake of the first feasibility test. We have considered neither TCP re-assembly nor connection-level payload analysis and protocol parsing. These techniques could potentially improve the accuracy of our system, although they would require some re-thinking of response mechanisms. Another conscious decision was not to deploy any special accelera-

*Pavel Laskov is also affiliated with University Tübingen, Wilhelm-Schickard-Institute for Computer Science, Tübingen, Germany.

tion mechanisms for high-speed performance, such as network or graphics processors. Yet even the current "vanilla C" software implementation shows that the proposed architecture can fulfill performance requirements of medium-scale server systems. We focused on stateless services so that no synchronization mechanism between the production and the shadow system is necessary. Future work will address most of the currently taken compromises.

2. Architecture and implementation

The general architecture of our inline IPS follows the well known concept of a packet filter running in kernel mode and making decisions on individual packets. We enhance this concept with an anomaly detection unit running in user mode and supplying the packet filter with scores in real time. The packet filter maintains an internal *detection state* for TCP flows (not to be confused with a connection state) allowing it to keep track of some limited TCP context. The detection state also incorporates some dependencies on a history of previous anomaly scores for each flow. Based on the current anomaly score and the detection state, the filter decides whether a packet is forwarded to a production system, redirected to a hardened shadow system or dumped to a special log file, the so-called forensic sink.

2.1. System architecture

The basic architecture of our system is presented in Figure 1. The packet filter is deployed on a central router or bridge to ensure that no packet bypasses its decision.
The filter delivers each incoming packet to a service specific connection handler, which requests a special trained anomaly detector (see Section 3 for details) to score its payload. Based on the anomaly score and its detection state machine – to be described in more detail in Section 2.2 – the packet is either delivered to the production system or redirected. The basic decision making mechanism proceeds as follows:

- If the anomaly score is below a pre-defined threshold, the packet is forwarded to the production system.

- If the first packet with payload receives a score exceeding the threshold, the flow is redirected to a *shadow system*. The shadow system provides the same services as a production system but is equipped with additional, possibly computationally expensive instrumentation for preventing a successful attack.

- If a subsequent packet receives a score above the threshold, the flow is redirected to a *forensic sink*. No action at the application layer is taken in the forensic sink, however, packets are acknowledged at a transport layer in order to receive as much malicious traffic as possible. The main purpose of a forensic sink is to obtain data for fine-tuning anomaly detection, e.g. by means of generating attack signatures, whitelisting or retraining.

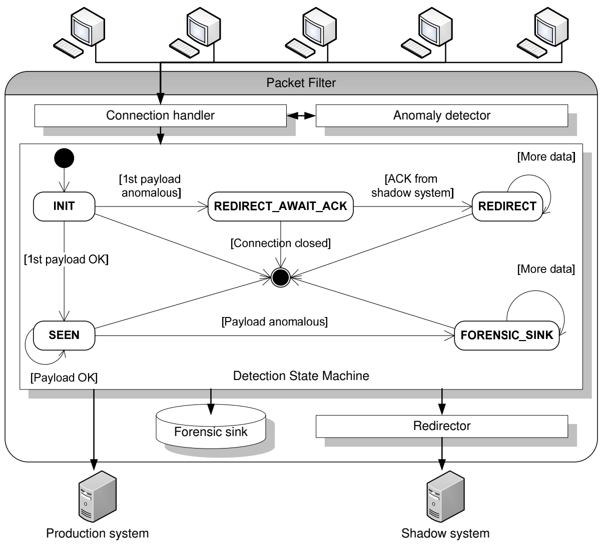

Figure 1. Overview of the inline anomaly detection architecture.

Once a flow is redirected, all packets are forwarded to the designated system regardless of their scores.

2.2 Detection states

The detection state machine lies at the core of the proposed inline anomaly detection architecture. It enables the system to accumulate information for a connection beyond that of a single packet. The states and transitions of the detection state machine are presented in Figure 1.
Each connection starts in the `INIT` state. After receiving the first packet with payload, the anomaly detector calculates an anomaly score and decides whether this packet is anomalous. If this packet is normal, the connection status is toggled to `SEEN` and the corresponding packet is delivered to the production system. If this packet is anomalous, the system redirects the packet to the shadow system.
The `SEEN` state is kept as long as all the following packets are tagged as normal by the anomaly detector. If any subsequent packet of the connection is anomalous, the detection state is changed to `FORENSIC_SINK` and from now on all packets of this connection are logged to the forensic sink. The packets are not delivered; solely an ACK for each packet is sent to harvest more data. Redirection to a

shadow system is no longer an option at this point, since some packets have already been delivered to the production system and cannot be recovered any more by the packet filter. The initial connection to the production system is terminated with a RST.

The redirection of a connection to the shadow system requires some structural overhead. The system has to tear down the initial connection to the production system and has to establish a new connection to the shadow system. Several parts of the redirected packets both from the client and from the server side (source or destination address and port, sequence or acknowledgement number) have to be adjusted accordingly. The main steps involved in the redirection are shown in Figure 2.

After resetting the initial connection our system establishes a new connection to the shadow system. Having received the SYN/ACK for the new connection, our system can translate all packets for this connection using the new sequence number obtained from the shadow system. The client side is not able to notice any difference in the transmission except for a small delay[1].

This redirection mechanism is reflected via two distinct states in the detection state machine: if the first packet with payload is anomalous, the corresponding connection is toggled to the `REDIRECT_AWAIT_ACK` state. The anomalous packet is dropped and our system establishes a new connection to the shadow system as described above. After the system receives the SYN/ACK of the shadow system, the corresponding connection is switched to the `REDIRECT` state, since the translation of the packets is now possible due to the received sequence number of the new connection.

In the `REDIRECT` state each packet is translated as follows: if the packet originates from the client, the destination address and port is replaced with the respective values for the shadow system. The acknowledgment number is adjusted to match the sequence number of the new connection. If the packet originates from the shadow server, both the source address and port as the sequence number are changed to the corresponding values of the old connection.

After a connection is closed, the detection state machine reaches its final state. So regardless of the preceding state the corresponding connection information is freed in our system, after a connection ends.

2.3. Implementation

In order to perform anomaly detection inline, the system must be able to intervene, i.e. decide for every packet if it is delivered or not. The classical solution for this task is a packet filter. Unfortunately, the packet filter operates in the system's kernel space, which makes it difficult to develop new extensions. It is therefore desirable to have a mechanism for queueing selected packets to the user space, in order for a user process to decide the fate of a packet.

The de facto standard packet filter framework for the Linux system `Netfilter` offers exactly such a solution with the `libnetfilter_queue` library: all packets matching a specific `Netfilter` rule are queued to a user space process which receives the packet via a dedicated socket. `Netfilter` expects a packet to get a so-called verdict, which determines how the packet is processed by the packet filter (ACCEPT, DROP, REPEAT). The packet itself can also be modified by the user process down to the IP layer.

For the redirection of packets the `Netfilter` framework offers the DNAT target, which implements a destination network address translation mechanism. Unfortunately the DNAT target realizes just a "simple" redirect, which begins with the arrival of the first SYN packet and is then carried on automatically without the possibility of a user space process intervention. Since our redirection mechanism potentially kicks in only after we have received the first packet with payload, the DNAT target cannot be used for the implementation of our scheme.

Fortunately the `libnetfilter_queue` allows one to modify the packet content before reinjecting it into the network. Manipulating the address, port and sequence numbers as described in the previous section is straight forward. After the manipulation one has to recalculate both the IP and the TCP checksum, otherwise the injected packet would be dropped by the network.

The additional communication involved in the redirection and logging to the forensic sink (sending of RST/SYN/ACK packets) is implemented via the `libnet` library which provides an easy interface to packet creation and delivery.

To test our architecture we have implemented a prototype which can be configured to execute single actions from our scheme. We can thus measure the performance impact of each single action (forwarding, calculation of anomaly score, redirection, logging to forensic sink).

This prototype is deployed on a recent `Debian` system acting as a central router between a client system and the production / shadow system. Both the production and the shadow system are hosted on a recent `OpenBSD` system. We choose `Apache` as service and the corresponding testing tool `Flood`[2] as a client simulator. All systems are fresh installs and are not tuned for performance. The hardening of the shadow system is accomplished by using the `Systrace` [13] tool with the standard `Apache` policy file. The policy file needed manual customization in order to meet the special requirements of our system. All systems are hosted on a `VMware ESX Server 3` farm.

[1]Note that we could circumvent potential detection of this feature by randomly adding small delays to all connections.

[2]http://httpd.apache.org/test/flood

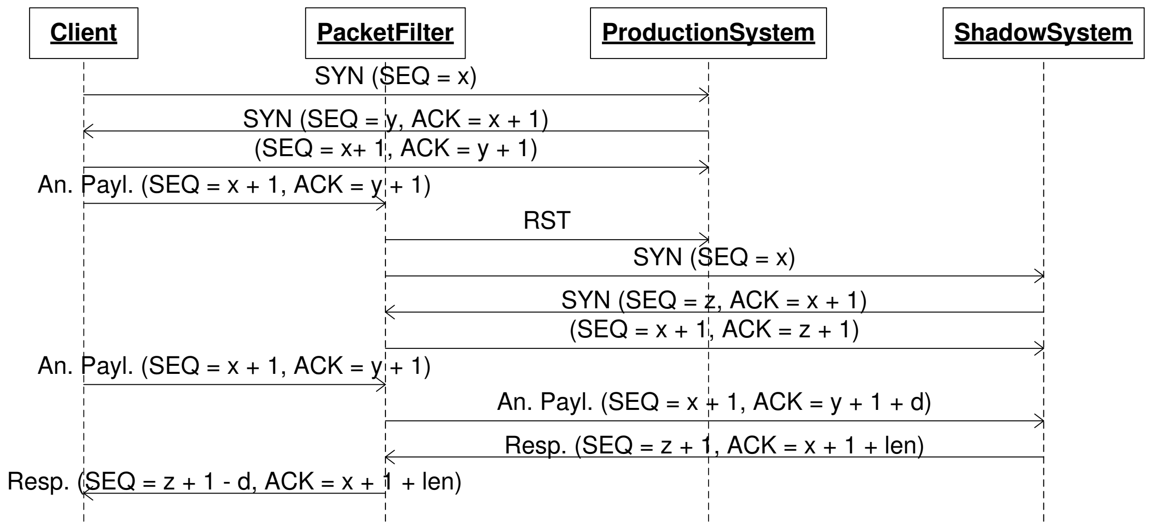
Client
PacketFilter
ProductionSystem
ShadowSystem
SYN (SEQ = x)
SYN (SEQ = y, ACK = x + 1)
(SEQ = x+ 1, ACK = y + 1)
An. Payl. (SEQ = x + 1, ACK = y + 1)
RST
SYN (SEQ = x)
SYN (SEQ = z, ACK = x + 1)
(SEQ = x + 1, ACK = z + 1)
An. Payl. (SEQ = x + 1, ACK = y + 1)
An. Payl. (SEQ = x + 1, ACK = y + 1 + d)
Resp. (SEQ = z + 1, ACK = x + 1 + len)
Resp. (SEQ = z + 1 - d, ACK = x + 1 + len)

Figure 2. Sequence diagram of the redirect process: If the first packet of a connection is marked as anomalous, the redirection process is triggered by the packet filter by resetting the connection to the production system and establishing a new connection to the shadow system. The packet filter has to memorize the difference in the sequence numbers (here $d = z - y$) and adjust the corresponding sequence numbers as shown in the lower part of the figure (An. Payload = anomalous payload).

3. Anomaly detection

Signature-based IDS provide effective defense against known threats, yet they fail to detect attacks if appropriate signatures are missing. Anomaly detection provides means for identification of unknown attacks and hence complements the capabilities of misuse detection systems. As a consequence anomaly detection methods and their application have been widely studied in the domain of intrusion detection, e.g., for identification of anomalous program behavior [4, 21, 10], packet headers [11, 12], network payloads [8, 22, 19, 15] and HTTP requests [9, 6]. All of these different approaches share the same concept – *anomalies are deviations from a model of normality* – and only differ in the considered data and the applied model of normality.

For our inline system we focus on packet payloads as basic data, since these can be efficiently obtained within the packet filter framework and include the majority of todays network attacks. To derive a model of normality from packet payloads, we pursue an approach proposed in [15]. Payloads are first mapped to a vector space, such that normality and deviation thereof can be expressed geometrically in terms of distances.

3.1. Embedding and similarity measures

A packet payload can be seen as a sequence of bytes, where each byte takes a value from 0 to 255. To map a payload to a vector, we move a sliding window of length n over its byte sequence. At each position we obtain a so called n-gram (a sub-sequence of length n). The set of all possible n-grams can be defined as

$$S = \{0, \ldots, 255\}^n.$$

Using the set S we can define an embedding of a byte sequence x into a vector space by counting the occurrences of all n-grams $s \in S$ contained in x. The corresponding embedding function ϕ is defined as

$$\phi(x) = (\phi_s(x))_{s \in S} \quad \text{with} \quad \phi_s(x) = \text{occ}(x, s),$$

where $\phi(x)$ is a vector with $|S|$ dimensions and $\text{occ}(x, s)$ is a function returning the frequencies for the n-gram s in x. Using this embedding function we can apply common vectorial distances over packet payloads to assess their similarity. Packets with similar payloads share the majority of n-grams and thus yield a low distance value in the vector space, while different payloads share only very few n-grams and obtain large distance values. Table 1 lists common distance functions expressed using the embedding function ϕ. The vector space induced by n-grams is high-dimensional, e.g., for $n = 4$ it contains over 4.2 billion dimensions. An explicit computation and comparison of vectors in such high-dimensional spaces is infeasible. The number of n-grams contained in a single payload, however, is linear in its length. This sparsity of the embedding function ϕ can be exploited to derive efficient methods for extraction and comparison of the resulting vectors [16].

Distance function	$d(x, z)$
Euclidean	$\sqrt{\sum_{s \in S} \lvert\phi_s(x) - \phi_s(z)\rvert^2}$
Manhattan	$\sum_{s \in S} \lvert\phi_s(x) - \phi_s(z)\rvert$
χ-Squared	$\sum_{s \in S} \frac{\lvert\phi_s(x) - \phi_s(z)\rvert^2}{\phi_s(x) + \phi_s(z)}$
Canberra	$\sum_{s \in S} \frac{\lvert\phi_s(x) - \phi_s(z)\rvert}{\phi_s(x) + \phi_s(z)}$

Table 1. Distance functions for payloads.

3.2. Anomaly score

An embedding of packet payloads in a vector space allows one to apply a large variety of anomaly detection techniques developed for vectorial data. Due to space constraints, we abstain from a review of relevant anomaly detection methods; after preliminary experimentation we have chosen a simple but very efficient centroid anomaly detection model, presented below.
The centroid model assumes that normal data lies in some compact region in a space whereas anomalies fall outside of this region. The model can be realized by the following simple algorithm:

1. *Training:* collect examples of normal data packets $X = \{x_1, \ldots, x_n\}$ and compute their mean $z = \frac{1}{n}\sum_{i=1}^{n} \phi(x_i)$. Computation of the mean can be also implemented using efficient primitives developed in [16].

2. *Validation:* determine the anomaly threshold t_{an}. To this end, collect an independent set of normal packets $\tilde{X} = \{x_1, \ldots, x_m\}$ and set t_{an} so that the ratio of packets x_i for which $d(z, x_i) > t_{\text{an}}$ is less than or equal to a pre-defined false-positive rate ν.

3. *Deployment*: for each incoming (possibly malicious) packet y, compute the anomaly score:

$$\text{score}(y) = \begin{cases} \text{normal}, & \text{if } d(z, y) \leq t_{\text{an}} \\ \text{anomaly}, & \text{otherwise} \end{cases}$$

A similar model has been used in a variety of recent work on intrusion detection, e.g. [15, 19, 20].

4. Experiments

In this section we analyze the runtime and the accuracy of our inline anomaly detection system. The runtime is measured using a stress test utility for performance analysis of the shadow system and the decision-making mechanisms. The accuracy of our system is evaluated by comparing it to a plain anomaly detection system on a real data set.

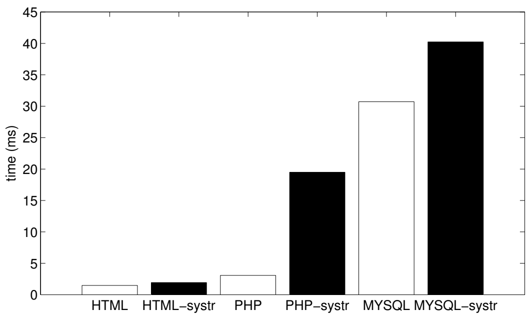

Figure 3. Median processing time of the different services with and without `Systrace`.

4.1. Runtime

For the first experiment we investigate the performance impact of our system on the network traffic. As mentioned in Section 2.3 we focus on `Apache` as web service and use the profile-driven HTTP load tester `Flood`. We simulate one client accessing a specific service 5000 times in a row to get a stable estimate of the runtime. To model a realistic behavior of modern HTTP servers, the `Apache` web server hosts three different types of services:

HTML returns just a static HTML page

PHP returns a dynamic, PHP generated page

MYSQL returns a dynamic, PHP generated page with values read from a MYSQL database.

All three services return the same amount of data. Figure 3 shows the median runtime of the different services on the production server (i.e. without running `Systrace`) and the shadow server (i.e. with running `Systrace`). Since runtime measurements are very noisy we have chosen a median instead of a mean to get a robust estimate of the average runtime. One can see that `Systrace` has almost no impact on the simple processing of an HTML file, however it increases the overhead significantly for both PHP and MYSQL requests. The first two columns of Table 2 show the corresponding runtime measurements for each service with and without `Systrace`.
Furthermore, we are interested in the runtime impact of our anomaly detection system. As described in Section 2.3 our prototype is capable of executing just one action for all incoming requests to keep the measurements focused on the impact of this single action. The right part of Table 2 shows the runtime for the following actions:

queue – each packet of a connection is copied to the user space process,

Type	normal	Systrace	queue	anomaly	sink	redirect-1st	redirect-next
HTML	1.47	1.94	1.59	2.05	1.64	235.63	1.62
PHP	3.08	19.49	3.16	3.59	3.36	238.25	3.13
MYSQL	30.71	40.24	31.00	31.09	30.72	242.32	30.75

Table 2. Median processing time in milliseconds per request for different services and connection handling methods.

anomaly – the distance of each packet to a centroid is calculated and compared to t_{an},

sink – each packet is logged to the forensic sink,

redirect-1st – each connection is redirected as described in Section 2.2,

redirect-next – translation of sequence numbers and addresses/ports for redirection of subsequent packets.

Since the static HTML service gives us the least variation in the measurement (data not shown), we focus on this use case first. The impact of HTML-queue and HTML-sink is minimal compared to the normal processing time, just the HTML-anomaly action adds a small overhead. The HTML-redirect-1st action is extremely costly. But this comes as no surprise since the redirection of each of the connections involves the closing of the old connection to the production system and re-establishing a new connection to the shadow system. The runtime of HTML-redirect-next underlines, that the main part of the cost is caused by the re-establishment of the connection and the resending of the anomalous packet. Measurements of the actions for PHP and MYSQL have the same trend but show that the overhead remains the same with the increasing application-layer complexity of the service. Hence the relative overhead for the actions diminishes with the complexity of the service: While the anomaly detection induces a relative increase of 29% for plain HTML, this overhead reduces to 13% for PHP and is negligible for MYSQL.

In summary, both parts of our system, namely the shadow system represented by `Systrace` and the different actions, do not pose an overwhelming burden to the overall system performance. Only the redirection of a connection comes at a high cost.

4.2. Accuracy

For the evaluation of the accuracy of our system we assemble a real-live data set as follows: The data set contains roughly 150k unsanitized connections totaling to roughly 240k packets from ten consecutive days taken from two months of incoming HTTP traffic at our institute. We split the data set into three equal parts of 50k connections each for training, validation and testing of the anomaly detection algorithm. We inject 100 instances (470 connections totaling to 2960 packets) of 28 different attack classes taken from recent exploits in the `Metasploit` framework[3] and some `Nessus`[4] HTTP scans.

Training, validation and deployment are carried out as described in Section 3.2. The validation step with $\nu = 0.01$ is repeated for a variety of models: we test a total of four n-gram models ($n \in \{2, 3, 4, 5\}$) and four distances (Manhattan, Euclidean, Canberra, χ-Squared). The best validation accuracy, measured by the AUC$_{0.01}$ criterion (area under the ROC-curve with the FP-rate ≤ 0.01) is attained by the Canberra distance on 2-grams. This model is used in the remaining experiments.

On our test data set a total number of 3102 ($\sim 0.05\%$) packets with payload are redirected, 111 ($\sim 0.001\%$) packets with payload are logged to the forensic sink and 58,369 packets with payload are processed as normal. Therefore the runtime of the complete system would be dominated by the anomaly detection part.

We introduce the following ratios to look at some cases, which are of special interest for the evaluation of our IPS:

$$\text{broken} = \frac{\text{\# normal conn. in SINK}}{\text{\# all normal conn.}} = 0.0008$$

$$\text{jailed} = \frac{\text{\# attack conn. in REDIRECT}}{\text{\# all attack conn.}} = 0.9760$$

The "broken" rate corresponds to the fraction of normal connections incorrectly transferred to the sink. In our experiment it is less than 0.1%. Manual inspection of these connections reveals, that they are mainly caused by a special feature of the Googlebot, which sends the "Connection: close" part of a request as a separate packet, which gets tagged as anomalous by the anomaly detector. Furthermore we can find special browser anomalies. These cases could easily be exploited for setting up a whitelist of signatures or as special data set for the retraining of the anomaly detector. This would help to decrease the number of wrongly broken connections in the future.

[3]http://www.metasploit.com
[4]http://www.nessus.org

Type	True positive rate	False positive rate
AD	0.9939 ± 0.0030	0.0092 ± 0.0105
AD RED.	0.9952 ± 0.0022	0.0017 ± 0.0009

Table 3. Evaluation of intrusion detection with anomaly detection only (AD) versus the REDIRECT/SINK extension. The mean rates over 10 runs and the 95% confidence limits based on the standard error are given.

The "jailed" rate gives us an impression about the number of true attacks redirected to the shadow system. We see, that nearly all attacks are redirected, so we get an effective way of containing attacks.
Finally we compare our system with the REDIRECT/SINK extension against a plain anomaly detector, which inspects the packets and decides for each packet on its own, whether to drop it or not. This basic anomaly detector lacks our extension and therefore cannot incorporate knowledge about the connection state and is not capable of redirecting packets to a hardened shadow system.
For the evaluation of our system we count every connection, that is redirected to the shadow system as a right decision. The rationale behind this is, that the shadow system answers good traffic just the same as the production system, whereas attack traffic is detected on the shadow system and will do no harm. On the other hand normal traffic redirected to the forensic sink counts as failure, while attacks in the sink contribute to the true positives.
The results of this evaluation are denoted in Table 3. Our system improves all categories and also shows a much smaller standard error, indicating a more solid decision process of the detector. Remarkably the Achilles' heel of anomaly detection, namely the false positive rate, is lowered by a factor of five by our system compared to the plain anomaly detector.

5. Related work

The most widely known open-source intrusion prevention system is Snort inline[5]. It provides inline interaction between a firewall and a Snort engine by using a packet queuing library `libipq` instead of the packet capture library `libpcap`. The decision-making in Snort inline is straightforward and is limited to packet dropping, using one of the three modes: normal (drop), silent (sdrop) and loud (reject).
Packet-based signature-matching suffers from a number of shortcomings. One of them is the inability to handle the context of application-level protocols; another is the susceptibility to evasion attacks in the sense of Newsham and Pracek [14]. These deficiencies have motivated de Bruijn et al. to design SafeCard, a Gigabit speed signature-based IPS with full TCP re-assembly [3]. SafeCard uses a specialized network interface card comprising a network processor and 8 stream processors for implementing various preprocessing stages. By using parallelization of rule matching SafeCard is able to attain a rate of 940 Mbit/s.
Another potential realization of parallelism can be made on a graphics processing unit (GPU) used in graphics cards in most of the modern PCs. Their support for SIMD instructions (single instruction multiple data) is ideally suited for signature matching algorithms when a single packet is to be tested against multiple signatures. By porting a classical Aho-Corasick algorithm to a GPU architecture, a throughput of 2.3 Gbit/s has been recently reported for the Gnort system [18]. It should be noted, however, that Gnort does not address prevention mechanisms beyond those offered by Snort.
Integration of anomaly detection techniques with intrusion prevention mechanisms has been studied in the context of web applications and clients. A reverse HTTP proxy proposed by Valeur et al. [17] uses scores obtained from an anomaly sensor [8] to forward requests to servers running under different privileges. Anagnostakis et al. [1] propose a "shadow honeypot" system in which suspicious HTTP traffic is internally redirected to instrumented applications. Our approach differs from these methods, in that our architecture is service-independent and redirects traffic at the transport layer. Thus, our architecture for inline anomaly detection allows for a generic and transparent integration with existing network infrastructures.

6. Conclusions and further work

Our architecture for inline intrusion detection has proven to be a useful system. By using anomaly detection it is capable of identifying even unknown attacks. The redirection mechanism is a good tool to cope with the burden of normal anomaly-based intrusion detection systems, namely producing a high number of false positives. Via the redirection to a special hardened shadow system we are able to significantly lower the number of false positives while maintaining the good properties of the anomaly-based approach.
Performance measurements have shown, that our system is ready for real-time performance. So our next step will be to expand our prototype implementation into a complete, self-contained system. The current implementation handles nearly all processing of the data in user space. It would be preferable to shift some parts of the system to the kernel space to avoid excessive copying of packets from the kernel to the user space. One simple extension would be to implement the REDIRECT and SINK mechanisms as custom `Netfilter` targets. Then all packets of a connection in

[5]http://snort-inline.sourceforge.net

the FORENSIC_SINK or REDIRECT state could be kept in the kernel space, since further inspection by the anomaly detector is not necessary. By putting the anomaly detection algorithm on a custom made FPGA or exploiting massive parallel execution on a graphics processors it would be even possible to shift the anomaly detection component in the kernel space.

We have seen that most of the attacks end up being redirected to the shadow system. Since the redirection mechanism is costly it would be desirable to have a premature drop criterion for extremely anomalous connections. This could be achieved by e.g. having another threshold t_{drop}, which is bigger than t_{an} and gives a maximal anomaly score for a packet, i.e. a packet having an anomaly score bigger than t_{drop} and all subsequent packets of this connection are dropped by the system instantaneously.

In general, our approach demonstrates the advantages of integrating anomaly detection into an intrusion prevention system, which – in combination with classic signature-based defences – provides improved protection of network environments.

Acknowledgment

This work was supported by the German Bundesministerium für Bildung und Forschung (BMBF) under the project ReMIND (FKZ 01-IS07007A).

References

[1] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis. Detecting targeted attacks using shadow honeypots. In *Proc. of USENIX Security Symposium*, pages 129–144, 2005.

[2] M. Attig and J. Lockwood. A framework for rule processing in reconfigurable network systems. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)*, pages 225–234, 225.

[3] W. de Bruijn, A. Slowinska, K. van Reeuwijk, T. Hruby, L. Xu, and H. Bos. Safecard: a gigabit IPS on the network card. In *Recent Adances in Intrusion Detection (RAID)*, pages 311–330, 2006.

[4] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. A sense of self for unix processes. In *Proc. of IEEE Symposium on Security and Privacy*, pages 120–128, Oakland, CA, USA, 1996.

[5] J. M. Gonzalez, V. Paxson, and N. Weaver. Shunting: a hardware/software architecture for flexible, high-performance network intrusion prevention. In *Conference on Computer and Communications Security (CCS)*, pages 129 – 149, 2007.

[6] K. L. Ingham and H. Inoue. Comparing anomaly detection techniques for http. In *Recent Adances in Intrusion Detection (RAID)*, pages 42 – 62, 2007.

[7] N. Jacob and C. Brodley. Offloading IDS computation to the GPU. In *Proc. of Annual Computer Security Applications Conference (ACSAC)*, pages 371–380, 2006.

[8] C. Kruegel, T. Toth, and E. Kirda. Service specific anomaly detection for network intrusion detection. In *Proc. of ACM Symposium on Applied Computing*, pages 201–208, 2002.

[9] C. Kruegel, G. Vigna, and W. Robertson. A multi-model approach to the detection of web-based attacks. *Computer Networks*, 48(5), 2005.

[10] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In *Proc. of USENIX Security Symposium*, 1998.

[11] M. Mahoney and P. Chan. PHAD: Packet header anomaly detection for identifying hostile network traffic. Technical Report CS-2001-2, Florida Institute of Technology, 2001.

[12] M. Mahoney and P. Chan. Learning rules for anomaly detection of hostile network traffic. In *Proc. of International Conference on Data Mining (ICDM)*, 2003.

[13] N. Provos. Improving host security with system call policies. In *Proc. of USENIX Security Symposium*, pages 257–272, 2003.

[14] T. Ptacek and T. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc, 1998.

[15] K. Rieck and P. Laskov. Language models for detection of unknown attacks in network traffic. *Journal in Computer Virology*, 2(4):243–256, 2007.

[16] K. Rieck and P. Laskov. Linear-time computation of similarity measures for sequential data. *Journal of Machine Learning Research*, 9(Jan):23–48, 2008.

[17] F. Valeur, G. Vigna, C. Kruegel, and E. Kirda. An anomaly-driven reverse proxy for web applications. In *Proc. of the 2006 ACM symposium on Applied computing*, pages 361–368, 2006.

[18] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. Markatos, and S. Ioannidis. Gnort: High performance network intrusion detection using graphics processors. In *Recent Adances in Intrusion Detection (RAID)*, 2008.

[19] K. Wang, J. Parekh, and S. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Adances in Intrusion Detection (RAID)*, pages 226–248, 2006.

[20] K. Wang and S. Stolfo. Anomalous payload-based network intrusion detection. In *Recent Adances in Intrusion Detection (RAID)*, pages 203–222, 2004.

[21] C. Warrender, S. Forrest, and B. Perlmutter. Detecting intrusions using system calls: alternative data models. In *Proc. of IEEE Symposium on Security and Privacy*, pages 133–145, 1999.

[22] S. Zanero and S. Savaresi. Unsupervised learning techniques for an intrusion detection system. In *Proc. of ACM Symposium on Applied Computing*, 2004.