On High-Speed Flow-based Intrusion Detection using Snort-compatible Signatures

Felix Erlacher *Student Member, IEEE* and Falko Dressler *Fellow, IEEE*

Abstract—Signature-based Network Intrusion Detection Systems (NIDS) have become state-of-the-art in modern network security solutions. However, most systems are not designed for modern high-speed network links. In the field of network monitoring, an alternative solution has become the choice for such high-speed networks. Flow-monitoring, typically based on the Internet Protocol Flow Information Export (IPFIX) standard, now goes well beyond collecting statistical information about network connections. Current solutions are even able to include selected parts of the payload in these Flows to be used in conjunction with NIDS. Recently, we extended this concept to application layer HTTP Flows. We now present our improved version of the IPFIX-based Signature-based Intrusion Detection System (FIXIDS). FIXIDS makes use of HTTP intrusion detection signatures from the popular Snort system and applies them to incoming IPFIX-conforming HTTP Flows. Our evaluation shows that FIXIDS can deal with four times higher network data rates without drops compared to Snort, while maintaining the same event detection rate. Furthermore, a substantial part of the data traffic can be outsourced to FIXIDS so that Snort can be relieved of a significant portion of rules and traffic. This increases both the detection rate and the data rate the overall security appliance can handle.

Index Terms—Network security, intrusion detection, flow monitoring, high-speed networks.

1 INTRODUCTION

TODAY'S computer networks, and most prominently the Internet, are constantly threatened by malware and other IT security attacks [1]. Network Intrusion Detection Systems (NIDS) provide a very efficient tool not only to detect such threats and attacks, but also to enforce usage policies to avoid internal misuse [2]. According to established taxonomies (e.g., [3]), NIDS can be categorized according to the used detection method: Anomaly-based NIDS use behavior-based methods by defining a model of normal network behavior and then detecting deviations to this model [4]. Knowledge-based systems, on the other hand, use a precise definition of the attack and match incoming traffic against this definition. The most widespread variants of knowledge-based systems are signature or rule-based NIDS. We concentrate on signature-based systems such as Snort [5], which allow a very precise description of attacks that are then identified using Deep Packet Inspection (DPI) methods. They offer a very high detection rate at the cost of comparably low performance.

In signature-based NIDS, a detection engine applies a rule-set to all received packets. The majority of state-of-the-art Snort rules contain patterns that are matched against the payload of the received packets. These patterns range from specific bytes to complex Regular Expressions (RegExes) matching not only individual packets but also payload in a packet flow. Because of these performance intensive pattern-matching operations, such systems can only cope with relatively low packet rates.

A different approach to network intrusion detection is to exploit Flow-based traffic data [6–8]. Here, a Flow denotes a set of elements containing aggregated information of a number of packets sharing the same properties. Typically, these properties are composed by the following quintuple: source/destination addresses, source/destination ports, and transport protocol. We capitalize the word Flow (as in IPFIX Flow) to emphasize the distinction from other meanings of the word flow. Compared to the standard DPI approaches, the analysis of Flow-based traffic data requires an additional step in which the data is aggregated into so-called Flow records with the advantage of significantly reducing the amount of data to be analyzed. Flow monitoring has become quite popular in a wide range of applications due to its ability to work in high-speed networks. The Internet Protocol Flow Information Export (IPFIX) protocol [9–11] has become the primary standard, which is used to aggregate packet information into Flow records for further post-processing. Flow records can be adapted to the application scenario by choosing appropriate Information Elements (IEs).

The fact that the Hypertext Transfer Protocol (HTTP) has become the dominant protocol in the Internet [12, 13] has motivated us to introduce IPFIX IEs to extend the concept of IP flows to HTTP related information [14]. As a result, a single HTTP dialog (request and response) can now be aggregated and exported as one IPFIX Flow record. Such a Flow record contains, in addition to the aforementioned IP quintuple, the most important HTTP header information as well as a configurable amount of HTTP payload data. Meanwhile, these HTTP related IEs have been standardized by the Internet Assigned Numbers Authority (IANA).[1] Commercial and Open Source network appliances have also started to include HTTP IEs into their set of exportable

• *F. Erlacher and F. Dressler are with the Heinz Nixdorf Institute and Department of Computer Science, Paderborn University, Germany.*
E-mail: {erlacher,dressler}@ccs-labs.org

Manuscript received January 5, 2019; revised October 7, 2019; accepted February 7, 2020.

1. https://www.iana.org/assignments/ipfix/ipfix.xhtml

Flow fields. Examples are Citrix,[2] Sonicwall,[3] Vermont,[4] and Ntop.[5] The format of the exported HTTP data fields is basically the same, even though not all are already using the standardized IEs.

With FIXIDS, we go one step beyond and came up with a novel Flow-based intrusion detection method. We use HTTP related Snort signatures and apply them to IPFIX Flows containing HTTP information. To the best of our knowledge, this is the first time that HTTP payload-based IEs are directly exploited for signature-based intrusion detection. By relying on Snort signatures, we ensure that community validated signatures are available for the developed system. The usage of Flow data guarantees high network throughput because less data has to be analyzed when searching for signatures. As we will show in this article, FIXIDS analyzes HTTP signatures substantially faster than Snort while maintaining essentially the same detection rate.

In comparison to DPI, Flow-based intrusion detection offers the additional advantage of supporting parallelization of work by separating the task of fetching packets and aggregating Flows from the task of analyzing Flows. This separation is not possible with DPI-based intrusion detection where both tasks have to be done in a tightly coupled fashion.

Clearly, FIXIDS can not replace DPI-based systems such as Snort but instead works in tandem with such systems. Conceptually, FIXIDS is best suited for high data rate scenarios where network traffic is already aggregated in form of IPFIX Flows. Thus, in many realistic scenarios, FIXIDS can analyze HTTP traffic using all HTTP rules, while Snort analyzes all remaining traffic using the remaining rules. This leads to a significant speed-up because Snort processes both fewer rules and less traffic.

In the evaluation, we confirm that by using aggregated Flow data the performance of signature-based intrusion detection can be raised significantly. Our solution is also compatible with encrypted network traffic and privacy concerns. To support encrypted traffic, we propose to use TLS interception proxies [15], the standard for most commercial firewall systems. With respect to privacy concerns, our Flow-based solution requires much less data compared to DPI-based solutions and, thus, has the potential to provide better privacy protection [16, 17].

Our main contributions can be summarized as follows:

- We present our IPFIX-based Signature-based Intrusion Detection System concept as an extension of the work presented in [18] in Section 3. The pattern matching of FIXIDS has been parallelized leading to a significant rise in network throughput.
- We thoroughly evaluated the detection accuracy in Section 5. We kept the very high detection accuracy (>99%) while at the same time supporting more than three times as many rules and much higher network throughput rates.
- We also extensively assessed the performance of our FIXIDS system in Section 6. We show that our system, running on commodity hardware, is able to support line rates of 10 Gbit/s.

2. https://www.citrix.com/products/netscaler-adc/netscaler-data-sheet.html
3. https://www.sonicwall.com/en-us/products/firewalls/management-and-reporting/global-management-system
4. https://github.com/felixe/ccsVermont
5. http://www.ntop.org/products/netflow/nbox/

2 Related Work

We first review current efforts regarding the performance of NIDS and then look at related work on Flow-based NIDS.

Several approaches improve performance by executing performance intensive operations on special hardware. Vasiliadis et al. [19] proposed using a Graphics Processing Unit (GPU) to perform signature pattern matching. They were able to increase the processing throughput by a factor of two. Multiple publications use Field Programmable Gate Arrays (FPGAs) to parallelize performance intensive operations. Sommer et al. [20] use FPGA based Network Interface Controllers (NICs) to distribute the incoming packets to multiple parallel analysis threads. Similarly, the distribution of the traffic on multiple intrusion detection instances by using a system that monitors the load of the single instances has been proposed by Limmer and Dressler [7]. This approach should guarantee that no instance gets overloaded while other resources remain idle.

There have also been advances in software-based acceleration of NIDS. For example, Stylianopoulos et al. [21] proposed a parallelized pattern matching algorithm, which exploits cache locality and Intel's modern Single Instruction Mutliple Data (SIMD) instructions. They were able to reach a speed-up of more than a factor of two compared to the traditional Aho-Corasick algorithm. Furthermore, methods have been proposed to reduce the data that a NIDS has to analyze by intelligently filtering the network traffic while maintaining a high detection rate [8, 22, 23].

Intrusion detection and prevention on Flow data requires an additional processing step for aggregating network packets into Flow records. While this is mostly done on dedicated hardware (e.g., on Flow-monitoring enabled switches), this stage has also been subject of improvements: Pus et al. [24] use dedicated hardware and Fusco and Deri [25] use hardware features of modern NICs to bypass the kernel drivers.

Flow-based NIDS inherently process much less data compared to DPI-based systems, so they typically have better throughput. Until now, Flow-based NIDS primarily relied on packet-header information – typically employing anomaly-based detection techniques. Thus, it is only possible to detect a subset of all possible network attacks [26, 27]. Prominent examples are Distributed Denial of Service (DDOS) attacks [28], scans [29], and Internet worms [30]. Deep flow inspection has been, for example, addressed by Liu et al. [31].

Because of the lack of payload-based IEs, only a few examples of Flow-based NIDS use knowledge-based methods. For example, honey-pot logs provide attack statistics that can be used to detect attacks in IPFIX Flow IEs [32]. Vizvary and Vykopal [33] use statistical properties gained from Netflow [11] data to detect brute-force attacks. More recently, a system called SSHCure has been presented by Hofstede and Hendriks [34] that detects SSH intrusions by identifying the different phases of such an attack in the statistical properties of Flows. Because they use only header information, all these systems offer rather static definitions of attacks. To the best of our knowledge, there is no Flow-based NIDS supporting user-definable signatures.

Hofstede et al. [35] analyze the impact of measurement or export artifacts in Flow data. They find several artifacts in real-world scenarios that have never occurred in their lab experiments. They conclude that it is important to understand what impact such artifacts may have on the specific application scenario. Most of the artifacts found relate to timing issues like inaccurate timestamps or imprecise Flow cache entry expiration. They will, however, not have an impact on the functionality of FIXIDS, as it only relies on correct aggregation of HTTP related fields. The other fields are not analyzed but still exported for statistical/informational reasons.

3 IPFIX-BASED SIGNATURE-BASED INTRUSION DETECTION SYSTEM

In the following, we introduce and explain our novel Flow-based NIDS FIXIDS. The goal is to be able to perform signature-based intrusion detection at high-speed while still being able to detect a very high number of events. By using IPFIX Flows, we can keep the amount of data to be analyzed relatively low. At the same time, combining HTTP IEs with signatures allows FIXIDS to detect essentially all HTTP related attacks.

3.1 Rules and Signatures

We use the same rules syntax as Snort so that we can take advantage of the various up-to-date and community validated databases of Snort signatures. Obviously, FIXIDS only supports HTTP-related rules and does not support all options that are possible within Snort signatures. Rules that are not supported are signaled and ignored during the initial parsing process. Among other capabilities, Snort searches for content patterns in the payload of the bypassing traffic. To be able to narrow the search space and speed up the pattern matching process, Snort offers the option to apply so called *content modifiers* to the pattern search. The idea is to restrict the search of *content* patterns to certain payload fields. The fact that most of the content modifiers in Snort restrict the pattern search to HTTP related fields shows again the importance of HTTP for intrusion detection. We exploit this by using Snort rules with HTTP content modifiers and apply them to the corresponding HTTP related IEs. Table 1 shows the currently supported content modifiers and the corresponding HTTP related IPFIX IE together with the IANA IE ID.

FIXIDS also supports the *uricontent* keyword, which is semantically equivalent to a content keyword with the http_uri content modifier. FIXIDS also accepts the *nocase* modifier, which enables case insensitive text search for the corresponding content pattern. The content patterns can be text and binary data (as hexadecimal numbers). FIXIDS also supports pattern descriptions as RegExes. Again, Snort supports using specific modifiers to be able to narrow down the RegEx pattern search space. Table 2 shows the modifiers supported by FIXIDS and explains their behavior.

Of course, FIXIDS also supports using user-generated rules, which can be customized for a certain application scenario.

Table 1
Snort content modifiers and their correspondig IPFIX IE

Content modifier		HTTP IE	IANA IE ID
http_method	→	httpRequestMethod	459
http_uri	→	httpRequestTarget	461
http_raw_uri	→	httpRequestTarget	461
http_stat_code	→	httpStatusCode	457
http_stat_msg	→	httpReasonPhrase	470

Table 2
Modifiers for pcre content definitions supported by FIXIDS

Modifier	Description
i	pcre pattern searches are by default case sensitive; this turns case insensitive pattern matching on
U, I	The pcre pattern search is applied to httpRequestTarget
M	The pcre pattern search is applied to httpRequest-Method
S	The pcre pattern search is applied to httpStatusCode
Y	The pcre pattern search is applied to httpReasonPhrase

3.2 Implementation

FIXIDS is implemented as part of the Open Source network monitoring toolkit Vermont. It is licensed under a GNU General Public License (GPL) and freely available.[6] Vermont is written in C and C++ and its functionality is divided among different modules, each having its own purpose. The FIXIDS component is implemented in its own module, accepting as input IPFIX Flows. A minimal working Vermont configuration including the FIXIDS functionality is sketched in Figure 1.

An external IPFIX exporter sends IPFIX Flows containing HTTP IEs to Vermont (currently, TCP, SCTP, UDP, and DTLS over UDP and SCTP are supported transport protocols). In Vermont, the IPFIX Collector module receives the Flows and hands them over to the FIXIDS module via a Flow buffer.

6. https://github.com/felixe/ccsVermont

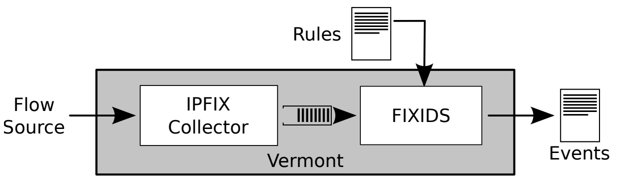

Figure 1. Minimal Vermont configuration with FIXIDS functionality

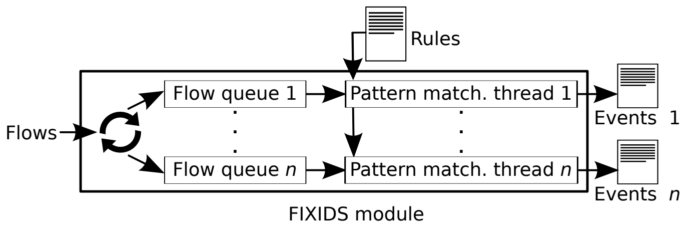

Figure 2. Sketch of the internals of the FIXIDS module

At startup, the FIXIDS module parses all signatures and options from the rules file given in the configuration. At run-time, it compares the IEs of every incoming Flow to the corresponding signature patterns of every rule.

In this work, we further introduce parallel pattern matching. A detailed sketch of the parallel behavior of the FIXIDS module is shown in Figure 2. The number of pattern matching threads (n) can be set in the configuration file and should be adapted to the number of cores available. Hereby it is important to keep in mind that every Vermont module already uses a dedicated CPU core. Because these CPU cores are under heavy usage, they should not be used for pattern matching.

The parallel processing of IPFIX Flows works the following way: Incoming Flows are distributed in a round-robin fashion to n FIFO queues. These queues are used as input by n pattern matching threads. Every thread continuously reads and removes a Flow from its dedicated input queue. Then it compares all rules from the rules file (which are already parsed and in memory) to this Flow. To compare the string patterns of a rule to the (UTF encoded) content of the corresponding IEs, FIXIDS uses the *strstr()* C string-compare function (and *strcasestr()* for case insensitive search). This function is written in an assembler and makes direct usage of CPU registers and, thus, outperforms other C and C++ string-compare functions. FIXIDS also supports perl compatible RegEx content patterns. To compare RegEx patterns to the HTTP IEs, we use the boost::regex library.[7] A rule might contain multiple patterns to compare (e.g., a "GET" in the httpRequestMethod IE and multiple patterns in the httpRequestTarget IE). If one of the matches fails, the execution of the remaining pattern matches of the current rule is aborted. If all patterns match, an alarm is triggered and the event information is written to the event file of this thread and pattern matching is continued with the remaining rules.

Every pattern matching thread uses the same input queue and the same event file over the whole runtime; this avoids race conditions or expensive access control mechanisms. The event files can be concatenated and sorted to achieve the same format as single-threaded FIXIDS event files.

Vermont can also be configured to read network packets from a NIC or a captured network trace (pcap file) and aggregate the packets to IPFIX Flows before handing these Flows to the FIXIDS module.

4 EXPERIMENT SETUP

In this section, we provide a short description of how we configured the used software parts for our experiments. We provide all configuration files used and other resources to reproduce the results on the author's homepage.[8]

4.1 Snort

Snort is the most widely used NIDS. During the evaluation experiments, we used Snort as a baseline both in functionality and performance. For the functional tests, we use Snort to produce the ground truth of events that are present in the test traces and compare the results with the detected events of FIXIDS. It is, however, not our goal to further investigate if the events produced by Snort are accurate (e.g., false-positives). We used Snort version 2.9.11.1 in IDS mode, with the default snort.conf configuration file, which is shipped with the installation archive.

The only relevant changes made to the default Snort configuration are the following: We increased the queue sizes of the detection engine (max_queue_events: 1000, max_queue: 1000, log: 1000, and max_queued_bytes for the stream5_tcp preprocessor: 1.5 MByte). This was necessary because if the number of events per packet/message exceeds the queue size, they are not reported anymore by Snort. The queue changes were necessary to be able to realistically compare the outcome of the two tested NIDS as FIXIDS always reports all events. We also adapted the memory limits of the TCP reassembly engine (stream5) to be able to cope with very high network throughput speeds (memcap: 512 MByte, max_queued_bytes: 128 MByte, and max_queued_segs: 20000). To avoid Snort skipping packets with checksum errors, we turned this behavior off (*-k none* switch).

4.2 IPFIX-based Signature-based Intrusion Detection System (FIXIDS)

When talking about FIXIDS, we refer to the Vermont configuration with the FIXIDS module as depicted in Figure 1. In this configuration, the IPFIX Flow Collector module listens to a configurable port for incoming Flows from a Flow source. These Flows are handed over to the FIXIDS module via a Flow queue. The FIXIDS module analyzes the incoming Flows comparing its fields with the patterns from the signatures in the rule file. In all the experiments for this work, FIXIDS is configured to use four pattern matching threads. Finally, all the detected events are written to the file given in the configuration.

Our experiments revealed that this number is the main limitation for the speed-up achievable by multi-core systems. In our experiment, the system had six physical cores, of which one is busy with Operating System (OS) tasks (in our case mostly network related tasks), one is occupied by the Vermont IPFIX Collector module, and four are left for pattern matching. Using the hyperthreaded cores for parallelization did not lead to any performance increases. This means that the performance of FIXIDS will very likely increase on workstations with more physical cores.

4.3 Vermont Flow Probe

For some experiments, we also used Vermont as a Flow probe for aggregating packets to IPFIX Flows. Please note that in the configuration used in this work, Vermont is configured to export only Flows containing HTTP. The detection of HTTP traffic is done by checking for a valid HTTP header.

One of the configuration options is to select the IE fields of the Flow to be exported. Among the necessary HTTP IE fields that FIXIDS needs for intrusion detection is the HTTP Universal Resource Identifier (URI) field. The length of this field is configurable. An examination of the URI lengths of HTTP traffic captured at an HTTP proxy (cf. Section 4.8) over a week showed the following results: the average length is 99.8 Byte, the median length is 55 Byte, the maximum

7. https://www.boost.org/doc/libs/1_62_0/libs/regex
8. http://www.ccs-labs.org/~erlacher/resources

length is 2913 Byte, and the 85 % percentile is 143 Byte. This matches with the results of a similar investigation on HTTP header lengths done by Google and presented in the SPDY whitepaper.[9] According to these results, we configured Vermont to export the first 150 Byte of the HTTP URI. This also follows the findings in [14, 23] that the most relevant data for intrusion detection is found at the beginning of a HTTP stream.

Experiments not shown here demonstrated that the transport of IPFIX Flows between Vermont and FIXIDS works best over the PR-SCTP protocol [36]. Thus, we used PR-SCTP as the default transport protocol for IPFIX Flows with Vermont.

4.4 Nprobe Flow Probe

To assess FIXIDS also beyond the integration with Vermont using third party Flow probes, we also conducted experiments using Ntop's network probe Nprobe (Version 8.6) [37]. It is available as software or incorporated on dedicated hardware under the brand Nbox.[10] Using its HTTP plugin, it also exports HTTP related information in IPFIX Flows but uses proprietary enterprise specific IEs. We extended FIXIDS with a configuration option to also accept the enterprise specific HTTP related IEs of Nprobe. Because we are interested in IPFIX Flows containing HTTP fields, we configured Nprobe to only aggregate TCP flows on port 80.

Experiments not shown here demonstrated that the transport of IPFIX Flows between Nprobe and FIXIDS works best over the TCP protocol. Thus, we used TCP as the default transport protocol for IPFIX Flows with Nprobe.

4.5 Network Setup

For all of the experiments reported in this paper, we used one or more of the following three workstations: *Workstation 1* and *Workstation 2* are equipped with an Intel Xeon i7-3930K CPU and 32 GByte of RAM. Both workstations make use of an Intel 82599 10 Gbit/s dual port NIC. Workstation 1 and Workstation 2 are interconnected via a Cisco Catalyst 4506-E 10 Gbit/s switch. This way, we guarantee that the routing of the network traffic itself has no performance impact on the experiment results. Workstation 2 also has a dedicated 1 Gbit/s link to Workstation 3. *Workstation 3* is equipped with an Intel i7-7700K CPU and 32 GByte of RAM. All of the workstations are running Ubuntu 16.04 with kernel 4.4.0 as the OS.

4.6 Used Detection Rules

For both FIXIDS and Snort we used the same set of detection rules. We downloaded and used the most current Snort rules from three sources:[11]

- All Snort rules (snapshot 29111) as provided to Snort.org subscribers;
- the community rule-set from Snort.org; and
- all rules from the Emerging Threats rule-set.[12]

9. http://dev.chromium.org/spdy/spdy-whitepaper
10. http://www.ntop.org/products/netflow/nbox/
11. as of June 22nd, 2018
12. http://doc.emergingthreats.net/bin/view/Main/AllRulesets

From these rule-sets, we used all HTTP-related rules as supported by FIXIDS. The total number of rules used for the evaluation is 5540.

4.7 Attack Network Traffic

Choosing the traffic for evaluating a NIDS is a difficult task. Most publications use either real traffic from a live network or one of the publicly available network traces (e.g., [38, 39]). As only very few researchers have access to carrier grade networks, publicly available network traces are the first choice for such evaluations. But there are a few problems arising with these traces [40]. First, many of these traces are multiple years old, some even decades and, thus, do not contain recent attacks. Secondly, almost all of the publicly available traces do not contain any payload information for privacy reasons. This disqualifies most of them for the evaluation of signature-based NIDS. Furthermore, even if researchers would have realistic and up-to-date network traffic containing application payload at hand, it will not contain a big enough number of diverse attacks to prove that a NIDS has broad detection coverage.

A typical approach to this problem is to create your own attacks and mix them with benign, realistic traffic. Malicious traffic is crafted by hand [8] or by using attack frameworks like Metasploit[13] [41, 42]. However, this manual process is cumbersome and time intensive and, thus, the resulting traces still do not contain a sufficiently high number of unique attacks.

This problem has been addressed in the Generating Events for Signature Intrusion Detection Systems (GENESIDS) framework [43]. GENESIDS is a system that automatically generates HTTP attacks and, thus, allows for straightforward generation of network traces (or live traffic) where the number of different detectable events is precisely defined by the given attack configuration. One of the main advantages of GENESIDS is that it uses the Snort syntax as input format and, thus, the user can take advantage of thousands of up-to-date and realistic attack definitions.

GENESIDS creates one TCP flow per input rule. Every generated TCP flow contains one HTTP request containing all patterns of the given rule. Until now, no rule from all rule-sets that we downloaded uses HTTP content that has to be found in an HTTP response. Thus, all attacks generated by GENESIDS are located in an HTTP request. Additionally, GENESIDS adds a custom HTTP header to the request, which contains the unique rule sid so that we can assess if the generated attack triggered the corresponding event in a NIDS. Apart from a rule file, GENESIDS requires also a working HTTP server URI as input, which it uses to send the generated HTTP requests to. In our experiments, an Apache HTTP server (default configuration) has been used. This means that for most requests the server response will be a '404, Not Found'. This does not influence the detection result because, as stated before, all patterns to be detected are located in the HTTP request. In summary, GENESIDS allows us to test FIXIDS with an unprecedented variety of different attacks. GENESIDS can also be combined with traffic generators [44] to create mixed traffic traces and to maintain typical load patterns as background traffic.

13. http://www.metasploit.com

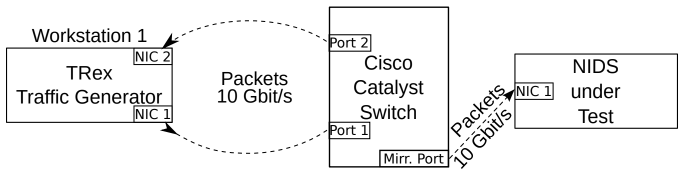

Figure 3. Sketch of TRex generating stateful network traffic

Table 3
Statistical properties of the SFR network traffic generated by TRex

Protocol	Packets	Bytes	CPS
TCP	61%	69%	2059
→ HTTP	32%	49%	1519
UDP	39%	31%	2004

4.8 Realistic Network Traffic

For most performance evaluation experiments, we use Cisco's TRex traffic generator (version 2.41).[14] It allows for stateful, timely, precise, and realistic high-speed traffic generation, and it is one of very few generators that supports the custom creation of application layer payload [45]. TRex ships with a set of realistic traffic pattern templates. These templates consist of a list of pcap network traces. For every pcap trace in the template file, there exists an entry stating how many Connections Per Second (CPS) TRex should generate with this trace. For every new connection, TRex uses a new IP src/dst pair from a pool of addresses stated in the configuration file. To control the throughput rate of the generated traffic, TRex offers a numerical multiplier switch. This number is multiplied with the CPS rate of every trace defined in the template.

Please note that there is a crucial difference between replaying the same traffic with different Packets Per Second (PPS) rates or replaying it with different CPS rates. With increasing PPS rates, the inter-packet time is reduced and the same traffic gets simply compressed in time. With increasing CPS rates, on the other hand, one connection (as defined in the TRex template) remains unchanged in time domain, but to increase the packet throughput, the same connection is repeated in shorter intervals. This has a significant impact on the performance of NIDS, because when increasing the packet throughput by increasing the CPS, more connections per time unit have to be kept in cache. This is not the case when increasing the packet throughput by increasing the PPS rate. Even more importantly, increasing the PPS rate is not what happens in real world networks with high network throughput rates. Here, the challenge is to cope with a huge amount of concurrent connections, which is exactly what we mimic when increasing the network throughput with higher CPS rates.

Figure 3 shows our evaluation setup using TRex as a traffic generator and the Cisco switch to copy the network traffic to our FIXIDS system for analysis. We will maintain this setup for the rest of the performance measurements. At startup, TRex reads all the pcap files from the traffic template and the corresponding CPS rates. It then starts with the first request packet from every pcap trace in the traffic template and sends it out on the configured egress NIC to a switch, which is connected via a mirroring port to the NIDS under test. The switch is configured to send a copy of every bypassing network packet to the mirroring port. The source and destination addresses of the packets generated by TRex are defined in the traffic template. The switch is also used to route packets with such destination addresses to the ingress NIC of the TRex device. As soon as the packet arrives at the ingress NIC, TRex answers with the response packet sent through the egress device, routed through the switch and arriving at the ingress device. This process is started multiple times per second per pcap trace depending on the configured CPS of the pcap trace.

The TRex traffic template that we use in the following experiments is a traffic mix defined by the french Telco provider SFR France[15] to represent typical Internet traffic. According to the TRex manual,[16] this template is also used by Cisco to benchmark their ASR1k/ISR-G2 routers. Table 3 shows the basic composition of this traffic. Not surprisingly, this traffic does not generate any events with Snort or with FIXIDS. For the remainder of this paper, we call this traffic "SFR traffic."

For the experiments in Section 6.1, we further require a network trace file containing mostly realistic traffic, but also some events to be able to assess the impact of packet/Flow losses on the event detection accuracy. We generated two different traces. For the first trace, we use a pcap file that we created by capturing 120 s of the SFR traffic generated at a speed of 1 Gbit/s. This trace contains almost 25 000 000 packets. We than added 500 random attacks generated by GENESIDS. The attacks are distributed evenly over the trace. Finally, we replicated this trace six times and concatenated the resulting traces. To avoid that repetitions of the same connection end up in the same Flow, we rewrote the IP addresses in every repetition using the tool *tcprewrite*. tcprewrite uses a deterministic way of changing IP addresses and, thus, retains TCP sessions between two hosts. When using Vermont as a Flow probe (with the same configuration as used in the experiments), it exports 1 485 000 Flows including HTTP IE fields. We call this trace the "SFR+500x6 trace."

The second trace was created by capturing the traffic going through an HTTP proxy used by a scientific work group for one week. This trace contains only HTTP traffic and consists of 2 200 000 packets. Again, we added 500 attacks. To have a comparable number of exported Flows we replicated and concatenated the trace 35 times. When using the same configuration as below, Vermont will export 1 494 000 Flows including HTTP IE fields. We call this trace the "PROXY+500x35 trace."

5 FUNCTIONAL EVALUATION

In the functional evaluation, we assess that the methods and algorithms implemented in FIXIDS work as expected and, thus, accurately and precisely detect a broad variety of events in the analyzed network traffic. We asses the general detection functionality by using GENESIDS to create attack

14. http://trex-tgn.cisco.com

15. https://www.sfr.fr
16. https://trex-tgn.cisco.com/trex/doc/trex_manual.html

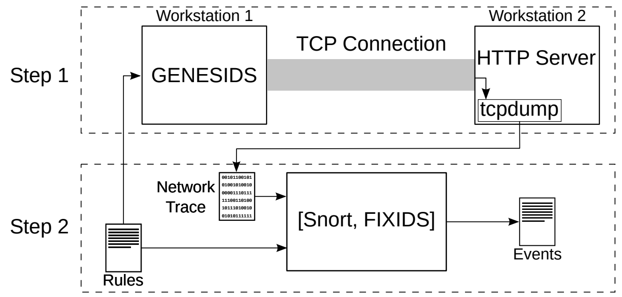

Figure 4. Sketch of the functional evaluation experiments

traffic. As input we used the rule-set described in Section 4.6 with 5540 different rules. The experiment setup is sketched in Figure 4.

In a first step, we configured GENESIDS, running on Workstation 1, to send all HTTP requests to an HTTP server located at Workstation 2. On Workstation 2, we captured all the HTTP traffic using the network traffic capturing tool tcpdump and saved the traffic to a pcap network trace. In a second step, we configured Snort and FIXIDS to analyze this network trace. We configured Snort to take as input the pcap network trace and use all signatures from our rule-set. The triggered events were written to a file and can then be compared to the events detected by FIXIDS. To be able to analyze this pcap file with FIXIDS, we configured Vermont as a Flow probe, reading the packets directly from the network trace, aggregating them to IPFIX Flows, and then handing them over to FIXIDS. Again, the events triggered by FIXIDS were written to a file.

For the evaluation, we compared the triggered events of Snort and FIXIDS. It is important to note that we did not simply check if Snort or FIXIDS triggered an event for a malicious HTTP packet, but we only denoted a true positive if the triggered event corresponds to the exact attack contained in the appertaining HTTP attack request. We accomplished this by comparing the port number and corresponding rule sid of the GENESIDS HTTP request with the rule sid and port number of the triggered event. This, of course, assumes that GENESIDS uses a unique port number for every HTTP request, which we verified beforehand.

To assess the detection robustness of FIXIDS, we repeated the experiment 100 times with changing attack traffic. The attack traffic generated by GENESIDS for the same rules differs between the individual runs. First, if a rule pattern is defined by a RegEx containing a subexpression that matches multiple characters (e.g., '.' matches any character, or \d matches any digit), the result will be a random string matching this subexpression and, thus, it will most likely differ in independent runs. Secondly, as GENESIDS generates events in the same order they appear in the rule file, we always randomly changed the order of the rules and, thus, the order of the generated HTTP attack requests also differs in every run.

Figure 5 shows the true positive events that have been detected during the experiment. GENESIDS created all 5540 attacks over all runs. Snort as well as FIXIDS have a reliably high true positive detection rate of more than 99 %. Snort, on average, detected 5489 true positives of the generated attacks with a maximum of 5494 detected events and a

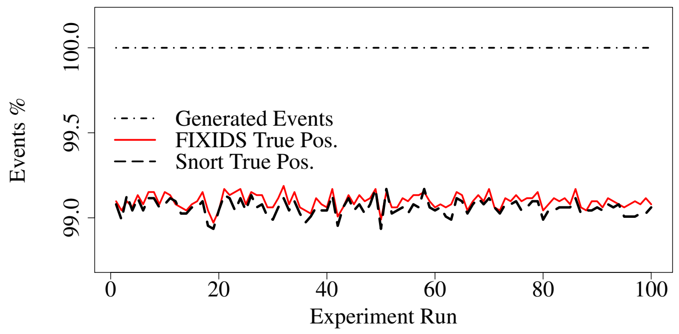

Figure 5. Detected true positive events by Snort and by FIXIDS in 100 different attack traces generated with GENESIDS

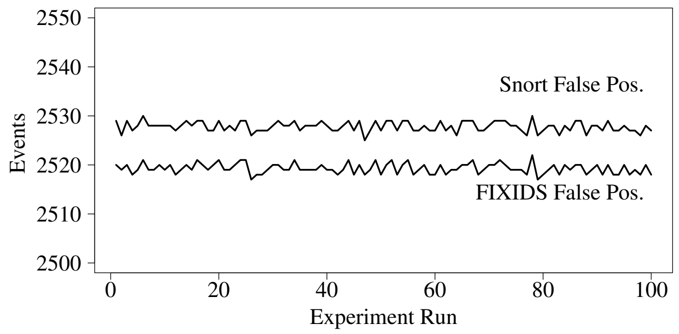

Figure 6. Detected false positive events by Snort and by FIXIDS in 100 different attack traces generated with GENESIDS

minimum of 5481. FIXIDS detected slightly more with an average of 5490 true positives of the generated attacks and a maximum of 5495 detected events and a minimum of 5483. These experiment results show that FIXIDS has essentially the same detection accuracy as Snort and, thus, its detection functionality can be considered equal to the state-of-the-art.

The reason that both NIDS did not detect all of the generated attacks is because GENESIDS, in very rare cases, is not able to generate the attack as defined in the rule and, thus, the NIDS using these rules can not detect it. For example, if a rule defines multiple HTTP header patterns ending with \r\n\r\n, this would require GENESIDS to generate an illegal HTTP request. Nevertheless, the fraction of generated attacks that could not be detected is less than 1 % and, thus, negligible for our purposes.

Figure 6 shows the false positive events that have been detected during the experiment. Snort triggered on average of 2530 and FIXIDS triggered an average of 2519 false positive events over the 100 runs. Such a high number of false positives is not unusual, if rules are not adapted to the application scenario. The reason is the overlap of patterns in rules [46], causing multiple rules to trigger an alarm for the same packet. A closer look at the false positives in our experiment shows that 74 % of the false positives are triggered by only three rules. This implies that removing only few rules will dramatically reduce the false positive rate.

Finally, we compare the differences between the detected true positive events of Snort and FIXIDS. While almost all attacks were detected correctly by both NIDS, there were subtle differences in the detected events that we elaborate in the following. Please keep in mind that the following discussion only applies to a very small fraction of the events (at maximum 11 per run, i.e., 0.02 %). The influence on the detected events is negligible, but it helps to identify the detection differences between Snort and FIXIDS.

Over all 100 runs, Snort detected 624 events generated by 11 unique rules, that FIXIDS did not detect. In almost all cases (523 out of 624) the reason was that the URI stretched over more then 150 Byte. Thus, patterns that were located beyond that limit could not be detected by FIXIDS. This could be avoided by changing the IPFIX aggregation configuration to include more characters in the HTTP URI. For the remaining missed attacks, we identified the following reasons:

- One attack contains as HTTP method a “post”. As HTTP methods are case sensitive, the whole HTTP request is not recognized as valid HTTP by the Vermont IPFIX aggregation engine and, thus, the HTTP related fields in the IPFIX Flow are not filled. This attack was missed 100 times, once every run.
- One attack contains a double slash. Snort, in the standard configuration removes double slashes from the URI, and then this URI matches the RegEx. FIXIDS analyzes the URI with double slashes and, thus, it does not match the RegEx.

FIXIDS, on the other hand, detected 834 events over all 100 runs, generated by 65 unique rules that Snort did not detect (minimum six per run, maximum 11). Most of the reasons have to do with one of the Snort preprocessors, which prepare and arrange the data for faster analysis and to avoid so called evasion attacks.

- Certain attacks contain “\\”, which, in the default configuration, is converted to “//”. Thus, this pattern in the packet does not match the pattern in the rule anymore.
- Some attacks contain a “+” directly after the “/” at the beginning of the HTTP URI. The “+” sign is normally used to denote a whitespace. Because Snort does not expect a whitespace character at the beginning of the URI the normalization engine removes it before analysis.
- Some attacks contain a “http\ :” which is normalized by Snort to “http://”.
- The remaining attacks in this category all occurred less than 5 times in 100 runs. The reasons were mostly special characters (mostly whitespace characters), which are interpreted differently by Snort and by FIXIDS.

In summary, FIXIDS has essentially the same, very high detection rate (>99 %), as Snort on more than 5000 different, up-to-date attack patterns with a slightly lower false positive rate. This makes us confident that the detection functionality of FIXIDS is precise and reliable.

6 THROUGHPUT PERFORMANCE

In this section, we assess the performance of FIXIDS under high network traffic throughput. We first evaluate FIXIDS with two different traffic sets to show the baseline performance and to find possible bottlenecks of the analysis process and then assess the applicability of FIXIDS in combination with the third party IPFIX Flow probe Nprobe as well

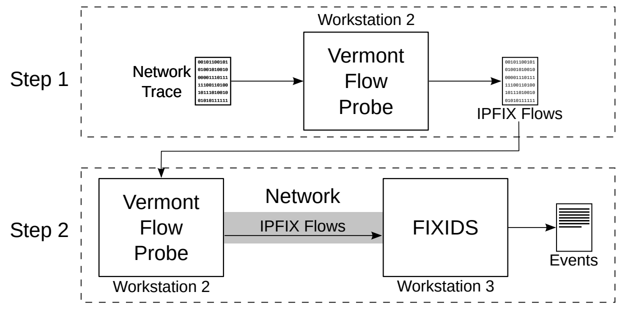

Figure 7. Sketch of the basic throughput experiments

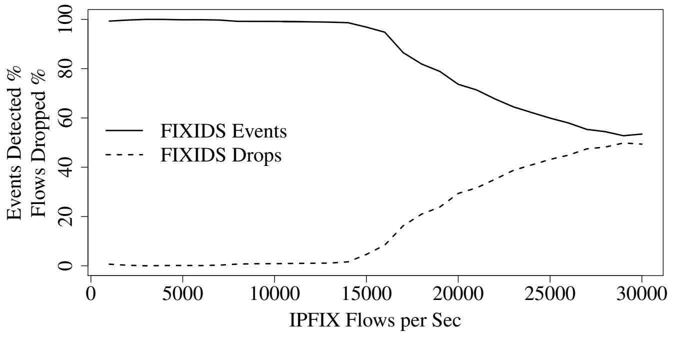

Figure 8. Flow throughput performance of FIXIDS analyzing the SFR+500x6 trace (average of ten runs)

as in a more realistic scenario. The main metric for the throughput performance of FIXIDS is (IPFIX) Flows/s. For a better comparison with Snort, we also converted this metric to Gbit/s, where possible.

6.1 Basic Throughput Experiments

In this first performance experiment, we assess how many IPFIX Flows/s FIXIDS can handle. To make sure to also evaluate the impact of different traffic properties we use the two realistic network traffic traces SFR+500x6 and PROXY+500x35 described in Section 4.8.

To avoid possible latencies introduced by replaying traffic, intermediate routing appliances, or Flow probes, we decided to subdivide the experiment into two steps as depicted in Figure 7. In the first step, we read the traffic trace directly with Vermont and export the IPFIX Flows, including the necessary HTTP IE fields, to a file. Because we only apply HTTP rules, only IPFIX Flows with HTTP IEs are exported. Vermont filters HTTP Flows by checking for a valid HTTP header. In the second step, we use Vermont to send the stored IPFIX Flows from Workstation 2 to a FIXIDS instance running on Workstation 3. FIXIDS is configured the same way as in the functional evaluation experiments (cf. Section 5) and as sketched in Figure 1, again using all 5540 rules. We repeat this step multiple times, always increasing the Flow throughput in steps of 1000 Flows/s from 1000 Flows/s to 30 000 Flows/s.

The results of this experiment are shown in Figures 8 and 9 for the SFR+500x6 trace and the PROXY+500x35 trace, respectively. Both plots show the average of ten runs. For better readability, 95 % confidence intervals are only plotted if above 2 %. With the SFR+500x6 trace, FIXIDS can cope with

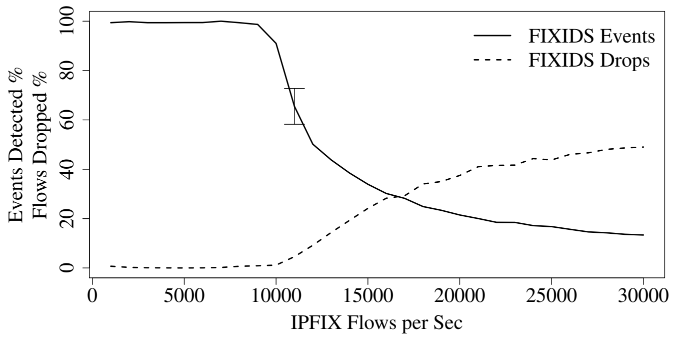

Figure 9. Flow throughput performance of FIXIDS analyzing the PROXY+500x35 trace (average of ten runs)

14 000 Flows/s before starting to drop data. As soon as Flows are dropped, the events contained in these Flows can not be detected anymore, thus, also the percentage of detected events starts dropping.

With the PROXY+500x35 trace, FIXIDS starts dropping Flows a little earlier; at around 11 000 Flows/s we have a significant rise in Flow drops. At the same time also the detected event percentage drops significantly, but in a much more dramatic way. The reason for this is that a few Flows account for many events (because they also trigger many false positives). When taking a closer look, these Flows are the ones that are dropped first, and, thus, the percentage of detected events drops faster than the percentage of dropped Flows.

The difference in the throughput rate between the two traces is due to the difference in the HTTP traffic. The vast majority of rules uses as the first pattern to look for a "GET" in the HTTP request. If this pattern is found, FIXIDS will carry on and search for the next pattern. In many cases a more complex Perl Compatible Regular Expressions (PCRE) pattern check follows. In the PROXY+500x35 trace, the ratio between HTTP GET requests and other HTTP requests is 31 : 1, while with the SFR+500x6 trace this ratio is 7 : 3. This means that FIXIDS uses more PCRE pattern checks for the PROXY+500x35 trace than the SFR+500x6 trace because FIXIDS can continue with the next Flow if the first "GET" pattern does not match.

6.2 Third Party Flow Exporter Experiments

In the second experiment, we aim to assess how FIXIDS performs when receiving IPFIX Flows from a third party device. This will typically happen in a real environment where the IPFIX Flows are coming from an already in place Flow exporting device like a switch. The general experiment setup is the one described in Figure 3. Now, the NIDS under test are depicted in Figures 10 and 11.

We use the Nprobe Flow exporter to aggregate packets to IPFIX Flows before analyzing them with FIXIDS. We then compare the outcome of the experiment to the network throughput of Snort receiving the same traffic as packets from the switch. For both, FIXIDS and Snort, we apply all 5540 HTTP rules. Because only HTTP rules are applied to the traffic we also filter the traffic in front of the corresponding NIDS (by port) to only contain HTTP traffic.

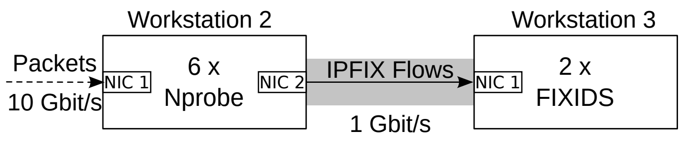

Figure 10. NIDS under test: 6 Nprobe instances aggregate the incoming packets to IPFIX Flows and send the flows to 2 FIXIDS instances

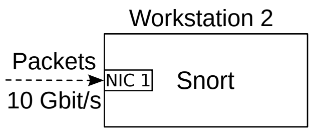

Figure 11. NIDS under test: Snort directly analyzes the incoming packets from the switch

An advantage of Nprobe is the capability to cluster multiple Nprobe instances on one NIC (using the *–cluster-id* switch). This allows to distribute the CPU heavy task of aggregating packets to IPFIX Flows to multiple cores. We made use of this and clustered six Nprobe instances. One Nprobe instance uses two cores, using six Nprobe instances proofed to guarantee full usage of our 12 hyperthreaded cores. Incoming packets are equally distributed according to their IP address and port among all instances of an Nprobe cluster.

We use the TRex stateful traffic generator on Workstation 1 to generate 5 min of SFR traffic including 100 random attacks per second. The attacks are randomly chosen from the set of attacks used in Section 5. The traffic generated by TRex on NICs 1 and 2 is routed through the Cisco switch. For the two following experiments, we configured the CPS multiplier of TRex to replay the traffic at a throughput of 0.5 Gbit/s up to 9.5 Gbit/s in steps of 0.5 Gbit/s. Please note that the throughput speed is measured at the TRex machine generating the traffic, both NIDS under test only analyzed the HTTP part of this traffic.

In the first experiment, our NIDS under test consists of six clustered Nprobe instances on Workstation 2 aggregating the incoming packets to IPFIX Flows including the necessary HTTP IE fields (cf. Figure 10). The resulting IPFIX Flows are sent via a 1 Gbit/s link to Workstation 3, where two FIXIDS instances were listening on different ports for incoming IPFIX Flows. Please note that the line rate of the 1 Gbit/s link

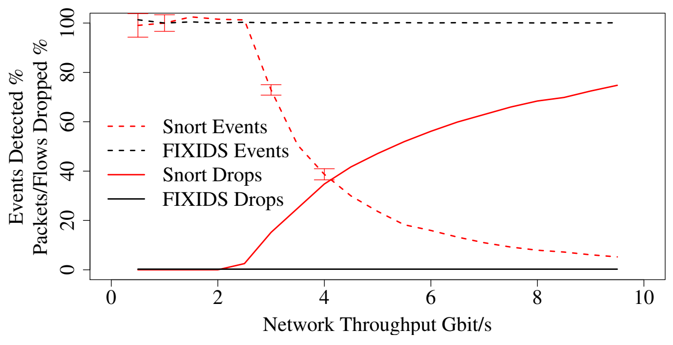

Figure 12. Event detection rate of Snort and FIXIDS analyzing the packets/IPFIX Flows of 5 min of SFR traffic including 100 events per second (average of ten runs)

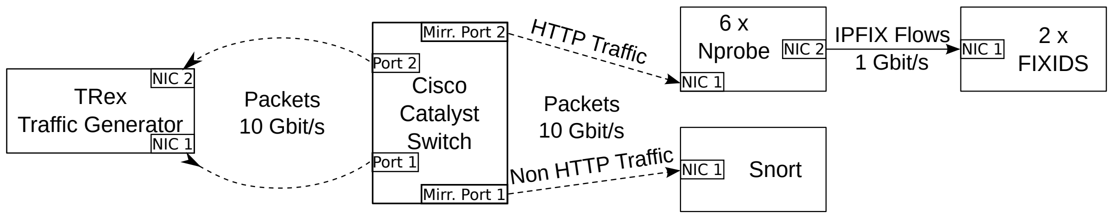

Figure 13. Realistic application scenario; FIXIDS analyzes all HTTP traffic and Snort analyzes all non HTTP traffic. This way Snort processes fewer rules and less traffic.

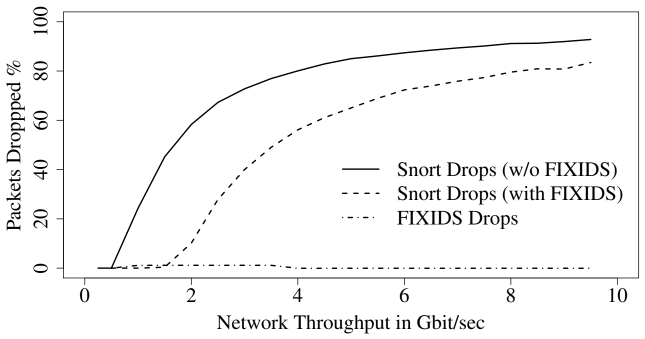

Figure 14. Packet drop rates of the realistic scenario were FIXIDS is deployed to reduce the load of Snort (average of ten runs)

used to transport IPFIX Flows from the Flow Exporter to the machine running FIXIDS was never fully used in our experiments. Each FIXIDS instance uses four cores for four pattern matching threads. Each of the three instances of Nprobe sends its IPFIX Flows to one instance of FIXIDS on Workstation 3. The results are shown in Figure 12, averaged over ten runs. Again, 95 % confidence intervals are only plotted if they are above 2 %. We repeated the experiment to compare the throughput of FIXIDS with the throughput of Snort. In this case, the NIDS under test is Snort, which was configured to analyze the incoming traffic on Workstation 2 and trigger an event if one of the rules matches.

Looking at the results in Figure 12, we see that Snort detects all events up to a data rate of 2.5 Gbit/s. FIXIDS on the other hand can cope with the highest rate of 9.5 Gbit/s without loosing a significant number of events. This is more than four times faster then Snort. One instance of FIXIDS had to cope with a maximum Flow rate of 11 000 Flows/s at 9.5 Gbit/s. This is still far below the drop rate for similar traffic from the basic throughput experiments in Section 6.1.

6.3 Realistic Application Scenario

In this experiment, we want to show how FIXIDS can be deployed in an existing IT security setup to reduce the load on a pre-deployed Snort NIDS. In contrast to the previous experiments, the goal of the measurements is not to assess the performance of FIXIDS alone, but to measure the network throughput increase of Snort, when FIXIDS is added to the scenario. The scenario is sketched in Figure 13. As a baseline (first experiment run), Snort receives all data traffic of interest from a switch and analyzes it. In this experiment, we apply all 5540 HTTP rules that are supported by FIXIDS plus 5714 non-HTTP rules from the sources described in Section 4.6. In total, this rule-set consists of 11254 rules. Again, we use TRex to generate realistic traffic using the SFR traffic template. We configured the CPS multiplier of TRex to replay the traffic at a throughput of 0.5 Gbit/s up to 9.5 Gbit/s in steps of 0.5 Gbit/s. This traffic does not contain any attacks.

In our second experiment run we use FIXIDS to reduce the load on Snort. First, we remove all 5540 HTTP related rules from the Snort configuration and set up FIXIDS for the analysis for these rules. Now Snort has to apply only the 5714 non-HTTP rules. This implies that we can split the traffic and make Snort analyze non-HTTP traffic while FIXIDS analyzes all HTTP traffic. We filter the traffic by ports, using all ports that Snort defines as HTTP ports in its standard configuration. The difference in the switch configuration is that now the packets are copied to two mirroring ports, one linked to Snort and one linked to FIXIDS. The FIXIDS configuration is the same as in Section 6.2, using six Nprobe instances as Flow exporter and two FIXIDS instances on a second machine.

In our first baseline experiment run, we assess the network throughput of Snort analyzing all traffic with all 11254 rules. The results (average of ten experiment runs) are shown in Figure 14. The 95 % confidence intervals were always below 2 %, so we do not show them for better readability. The solid line shows that Snort can cope with about 0.5 Gbit/s of traffic and starts dropping packets at higher data rates. The reason that Snort can handle less network throughput than in earlier experiments is because of the higher number of rules it has to apply during the analysis.

The dashed line in Figure 14 shows the dropped packets by Snort for this second experiment. Now Snort can cope with 1.5 Gbit/s, meaning that with the help of FIXIDS the network throughput without drops of Snort has tripled. In theory, the drop rate of FIXIDS has to be added to the drop rate of Snort. However, in this case, the load on FIXIDS is so low that no additional drops occur. A look at the maximum Flow rate shows that at 9.5 Gbit/s the rate is at 8400 Flows/s per FIXIDS instance and, thus, far below the maximum Flow throughput. This also means that more rules could be added to FIXIDS without causing more drops.

7 Conclusion

We presented the improved version of the IPFIX-based Signature-based Intrusion Detection System (FIXIDS). FIXIDS provides precise event detection in high-speed networks by using Internet Protocol Flow Information Export (IPFIX) HTTP Flows for intrusion detection. It is the first signature-based Network Intrusion Detection System (NIDS) that completely operates on Flow information using the novel HTTP IPFIX Information Elements (IEs). Besides custom attack signatures, FIXIDS supports using signatures from the most widely used NIDS Snort. This ensures that thousands of community validated and up-to-date signatures are available for FIXIDS. By using Flows the amount of data to be analyzed is much less compared to traditional Deep Packet Inspection (DPI)-based NIDS.

In our evaluation, we have shown that FIXIDS has the same detection accuracy as Snort. On commodity hardware, FIXIDS can cope with up to 14 000 Flows/s without missing any event. Analyzing the same network traffic as Snort, FIXIDS can handle more than four times the network throughput of Snort while retaining the same detection rate. We also conducted an experiment where FIXIDS was deployed in a realistic scenario to mitigate high network loads for Snort. In this case we were able to triple the network throughput capability of Snort from 0.5 Gbit/s to 1.5 Gbit/s, while identifying HTTP based attacks even at 9.5 Gbit/s without drops on FIXIDS.

ACKNOWLEDGMENTS

This work has been supported in part by the German Research Foundation (DFG) under grant no. DR 639/20-1.

REFERENCES

[1] B. Stritter, F. C. Freiling, H. König, R. Rietz, S. Ullrich, A. von Gernler, F. Erlacher, and F. Dressler, "Cleaning up Web 2.0's Security Mess - at Least Partly," *IEEE Security & Privacy*, vol. 14, no. 2, pp. 48–57, Mar. 2016.

[2] R. Kemmerer and G. Vigna, "Intrusion Detection: A Brief History and Overview," *IEEE Computer, Special Issue on Security and Privacy*, pp. 27–30, Apr. 2002.

[3] H. Debar, M. Dacier, and A. Wespi, "Towards a Taxonomy of Intrusion-Detection Systems," *Elsevier Computer Networks*, vol. 31, no. 8, pp. 805–822, Apr. 1999.

[4] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network Anomaly Detection: Methods, Systems and Tools," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 303–336, Feb. 2014.

[5] M. Roesch, "Snort - Lightweight Intrusion Detection for Networks," in *13th USENIX Conference on System Administration (LISA 1999)*, Seattle, WA: USENIX Association, Nov. 1999, pp. 229–238.

[6] R. Hofstede, M. Jonker, A. Sperotto, and A. Pras, "Flow-Based Web Application Brute-Force Attack and Compromise Detection," *Journal of Network and Systems Management*, vol. 25, no. 4, pp. 735–758, Oct. 2017.

[7] T. Limmer and F. Dressler, "Adaptive Load Balancing for Parallel IDS on Multi-Core Systems using Prioritized Flows," in *IEEE International Conference on Computer Communication Networks (ICCCN 2011)*, Maui, HI: IEEE, Jul. 2011, pp. 1–8.

[8] F. Erlacher and F. Dressler, "High Performance Intrusion Detection Using HTTP-based Payload Aggregation," in *42nd IEEE Conference on Local Computer Networks (LCN 2017)*, Singapore, Singapore: IEEE, Oct. 2017, pp. 418–425.

[9] B. Claise, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information," IETF, RFC 5101, Jan. 2008.

[10] J. Quittek, S. Bryant, B. Claise, P. Aitken, and J. Meyer, "Information Model for IP Flow Information Export," IETF, RFC 5102, Jan. 2008.

[11] R. Hofstede, P. Celeda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow Monitoring Explained: From Packet Capture to Data Analysis with Netflow and IPFIX," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, Apr. 2014.

[12] B. Ager, N. Chatzis, A. Feldmann, N. Sarrar, S. Uhlig, and W. Willinger, "Anatomy of a Large European IXP," in *ACM SIGCOMM 2012*, Helsinki, Finland: ACM, Aug. 2012, pp. 163–174.

[13] P. Richter, N. Chatzis, G. Smaragdakis, A. Feldmann, and W. Willinger, "Distilling the Internet's Application Mix from Packet-Sampled Traffic," in *Passive and Active Measurement Conference (PAM 2015)*, New York City, NY: Springer, Mar. 2015.

[14] F. Erlacher, W. Estgfaeller, and F. Dressler, "Improving Network Monitoring Through Aggregation of HTTP/1.1 Dialogs in IPFIX," in *41st IEEE Conference on Local Computer Networks (LCN 2016)*, Dubai, United Arab Emirates: IEEE, Nov. 2016, pp. 543–546.

[15] F. Erlacher, S. Woertz, and F. Dressler, "A TLS Interception Proxy with Real-Time Libpcap Export," in *41st IEEE Conference on Local Computer Networks (LCN 2016), Demo Session*, Dubai, United Arab Emirates: IEEE, Nov. 2016.

[16] S. Degli Esposti, V. Pavone, and E. Santiago-Gomez, "Aligning Security and Privacy: The Case of Deep Packet Inspection," in *Surveillance, Privacy and Security: Citizens' Perspectives*, M. Friedwald, P. Burgess, J. Cas, R. Bellanova, and W. Peissl, Eds., Taylor and Francis, 2017, pp. 71–90.

[17] G. Bianchi, E. Boschi, D. I. Kaklamani, E. Koutsoloukas, G. V. Lioudakis, F. Oppedisano, M. Petraschek, F. Ricciato, and C. Schmoll, "Towards Privacy-Preserving Network Monitoring: Issues and Challenges," in *18th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2007)*, Athens, Greece: IEEE, Sep. 2007.

[18] F. Erlacher and F. Dressler, "FIXIDS: A High-Speed Signature-based Flow Intrusion Detection System," in *IEEE/IFIP Network Operations and Management Symposium (NOMS 2018)*, Taipei, Taiwan: IEEE, Apr. 2018.

[19] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, "Gnort: High Performance Network Intrusion Detection Using Graphics Processors," in *11th International Symposium on Recent Advances in Intrusion Detection (RAID 2008)*, vol. 5230, Cambridge, MA: Springer, Sep. 2008, pp. 116–134.

[20] R. Sommer, V. Paxson, and N. Weaver, "An architecture for exploiting multi-core processors to parallelize network intrusion prevention," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 10, pp. 1255–1279, Jul. 2009.

[21] C. Stylianopoulos, M. Almgren, O. Landsiedel, and M. Papatriantafilou, "Multiple Pattern Matching for Network Security Applications: Acceleration through Vectorization," in *46th International Conference on Parallel Processing (ICPP 2017)*, Bristol, United Kingdom: IEEE, Aug. 2017, pp. 472–482.

[22] S. Kornexl, V. Paxson, H. Dreger, R. Sommer, and A. Feldmann, "Building a Time Machine for Efficient Recording and Retrieval of High-Volume Network Traffic," in *5th ACM SIGCOMM Conference on Internet Measurement (IMC 2005)*, Berkeley, CA: ACM, Oct. 2005, pp. 267–272.

[23] T. Limmer and F. Dressler, "Improving the Performance of Intrusion Detection using Dialog-based Payload Aggregation," in *30th IEEE Conference on Computer Communications (INFOCOM 2011), 14th IEEE Global Internet Symposium (GI 2011)*, Shanghai, China: IEEE, Apr. 2011, pp. 833–838.

[24] V. Pus, P. Velan, L. Kekely, J. Kovrenek, and P. Minavr'ik, "Hardware Accelerated Flow Measurement of 100 Gb Ethernet," in *14th International Symposium on Integrated Network Management (IM 2015)*, Ottawa, Canada: IEEE, May 2015, pp. 1147–1148.

[25] F. Fusco and L. Deri, "High Speed Network Traffic Analysis with Commodity Multi-core Systems," in *10th ACM Internet Measurement Conference (IMC 2010)*, Melbourne, Australia: ACM, Nov. 2010, pp. 218–224.

[26] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An Overview of IP Flow-based Intrusion Detection," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, pp. 343–356, Apr. 2010.

[27] M. F. Umer, M. Sher, and Y. Bi, "Flow-Based Intrusion Detection: Techniques and Challenges," *Computers & Security*, vol. 70, pp. 238–254, Sep. 2017.

[28] R. Hofstede, V. Bartos, A. Sperotto, and A. Pras, "Towards Real-time Intrusion Detection for NetFlow and IPFIX," in *9th International Conference on Network and Service Management (CNSM 2013)*, Zürich, Switzerland, Oct. 2013, pp. 227–234.

[29] M. Ring, D. Landes, and A. Hotho, "Detection of Slow Port Scans in Flow-based Network Traffic," *PLOS ONE*, vol. 13, no. 9, Sep. 2018.

[30] T. Dubendorfer, A. Wagner, and B. Plattner, "A Framework for Real-Rime Worm Attack Detection and Backbone Monitoring," in *First IEEE International Workshop on Critical Infrastructure Protection (IWCIP 2005)*, Darmstadt, Germany: IEEE, Nov. 2005, 10–pp.

[31] A. X. Liu, C. R. Meiners, E. Norige, and E. Torng, "High-Speed Application Protocol Parsing and Extraction for Deep Flow Inspection," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 32, no. 10, pp. 1864–1880, 2014.

[32] F. Dressler, W. Jaegers, and R. German, "Flow-based Worm Detection using Correlated Honeypot Logs," in *15. GI/ITG Fachtagung Kommunikation in Verteilten Systemen (KiVS 2007)*, T. Braun, G. Carle, and B. Stiller, Eds., Bern, Switzerland: VDE, Feb. 2007, pp. 181–186.
[33] M. Vizvary and J. Vykopal, "Flow-based Detection of RDP Brute-Force Attacks," in *7th International Conference on Security and Protection of Information, SPI 2013*, vol. 13, Glasgow, United Kingdom: ACM, Sep. 2013, pp. 131–138.
[34] R. Hofstede and L. Hendriks, "Unveiling SSHCure 3.0: Flow-based SSH Compromise Detection," in *International Conference on Networked Systems (NetSys 2015), Demo Session*, Cottbus, Germany, Mar. 2015.
[35] R. Hofstede, A. Pras, A. Sperotto, and G. D. Rodosek, "Flow-Based Compromise Detection: Lessons Learned," *IEEE Security & Privacy*, vol. 16, no. 1, pp. 82–89, Jan. 2018.
[36] R. Steward, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad, "Stream Control Transmission Protocol (SCTP) - Partial Reliability Extension," IETF, RFC 3758, May 2004.
[37] L. Deri, "nProbe: An Open Source NetFlow Probe for Gigabit Networks," in *TERENA Networking Conference (TNC 2003)*, Zagreb, Croatia, May 2003.
[38] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "Mawilab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking," in *6th ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT 2010)*, Philadelphia, PA: ACM, Nov. 2010.
[39] N. Moustafa and J. Slay, "The Evaluation of Network Anomaly Detection Systems: Statistical Analysis of the UNSW-NB15 Data Set and the Comparison with the KDD99 Data Set," *ACM Information Security Journal: A Global Perspective*, vol. 25, no. 1-3, pp. 18–31, Jan. 2016.
[40] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in *4th International Conference on Information Systems Security and Privacy (ICISSP 2018)*, Funchal, Portugal: INSTICC, Jan. 2018, pp. 108–116.
[41] T. Lukaseder, J. Fiedler, and F. Kargl, "Performance Evaluation in High-Speed Networks by the Example of Intrusion Detection Systems," in *11. DFN-Forum Kommunikationstechnologien*, Müller, Paul AND Neumair, Bernhard AND Reiser, Helmut AND Dreo Rodosek, Gabi, Ed., Bonn, Germany: GI, Jun. 2018, pp. 33–42.
[42] K. Nasr, A. Abou-El Kalam, and C. Fraboul, "Performance Analysis of Wireless Intrusion Detection Systems," in *5th International Conference on Internet and Distributed Computing Systems (IDCS 2012)*, Fujian, China: Springer, Nov. 2012, pp. 238–252.
[43] F. Erlacher and F. Dressler, "How to Test an IDS? GENESIDS: An Automated System for Generating Attack Traffic," in *ACM SIGCOMM 2018, Workshop on Traffic Measurements for Cybersecurity (WTMC 2018)*, Budapest, Hungary: ACM, Aug. 2018, pp. 46–51.
[44] ——, "Testing IDS using GENESIDS: Realistic Mixed Traffic Generation for IDS Evaluation," in *ACM SIGCOMM 2018, Demo Session*, Budapest, Hungary: ACM, Aug. 2018, pp. 153–155.
[45] M. Primorac, E. Bugnion, and K. Argyraki, "How to Measure the Killer Microsecond," in *Workshop on Kernel-Bypass Networks (KBNets 2017)*, Los Angeles, CA: ACM, Aug. 2017, pp. 37–42.
[46] F. Massicotte and Y. Labiche, "An Analysis of Signature Overlaps in Intrusion Detection Systems," in *41st International Conference on Dependable Systems & Networks (DSN 2011)*, Hong Kong, China: IEEE, Jun. 2011, pp. 109–120.

Felix Erlacher received his B.Sc. and M.Sc. in Computer Science from the University of Innsbruck, Austria in 2009 and 2012, respectively. He worked as a Ph.D. student at the Distributed Embedded Systems Group at the Heinz Nixdorf Institute and the Dept. of Computer Science, Paderborn University. In 2019 he received his Ph.D. degree from the Paderborn University. His research interest is on network security and high-speed network monitoring.

Falko Dressler (dressler@ccs-labs.org) is full professor of computer science and chair for Distributed Embedded Systems at the Heinz Nixdorf Institute and the Dept. of Computer Science, Paderborn University. He received his M.Sc. and Ph.D. degrees from the Dept. of Computer Science, University of Erlangen in 1998 and 2003, respectively. Dr. Dressler is associate editor-in-chief for Elsevier Computer Communications as well as an editor for journals such as IEEE/ACM Trans. on Networking, IEEE Trans. on Mobile Computing, IEEE Trans. on Network Science and Engineering, Elsevier Ad Hoc Networks, and Elsevier Nano Communication Networks. He has been guest editor of special issues in IEEE Journal on Selected Areas in Communications, IEEE Communications Magazine, Elsevier Ad Hoc Networks, and many others. He has been chairing conferences such as IEEE INFOCOM, ACM MobiSys, ACM MobiHoc, IEEE VNC, IEEE GLOBECOM, and many others. He authored the textbooks Self-Organization in Sensor and Actor Networks published by Wiley & Sons and Vehicular Networking published by Cambridge University Press. He has been an IEEE Distinguished Lecturer as well as an ACM Distinguished Speaker. Dr. Dressler is an IEEE Fellow as well as an ACM Distinguished Member, and member of the German computer science society (GI). He has been serving on the IEEE COMSOC Conference Council and the ACM SIGMOBILE Executive Committee. His research objectives include adaptive wireless networking (radio, visible light, molecular communications) and embedded system design (from microcontroller to Linux kernel) with applications in ad hoc and sensor networks, the Internet of Things, and cooperative autonomous driving systems.