Network Intrusion Detection in Encrypted Traffic

Eva Papadogiannaki
Technical University of Crete
14 PUBLICATIONS **167** CITATIONS
SEE PROFILE

Giorgos Tsirantonakis
Foundation for Research and Technology - Hellas
3 PUBLICATIONS **96** CITATIONS
SEE PROFILE

Sotiris Ioannidis
Technical University of Crete
299 PUBLICATIONS **7,173** CITATIONS
SEE PROFILE

Network Intrusion Detection in Encrypted Traffic

Eva Papadogiannaki*, Giorgos Tsirantonakis*
FORTH-ICS
{epapado, tsirant}@ics.forth.gr

Sotiris Ioannidis†
Technical University of Crete
sotiris@ece.tuc.gr
†Also with FORTH-ICS

***Abstract*—Traditional signature-based intrusion detection systems inspect packet headers and payloads to report any malicious or abnormal traffic behavior that is observed in the network. With the advent and rapid adoption of network encryption mechanisms, typical deep packet inspection systems that focus only on the processing of network packet payload contents are gradually becoming obsolete. Advancing intrusion detection tools to be also effective in encrypted networks is crucial. In this work, we propose a signature language indicating packet sequences. Signatures detect events of possible intrusions and malicious actions in encrypted networks using packet metadata. We demonstrate the effectiveness of this methodology using different tools for penetrating vulnerable web servers and a public dataset with traffic that originates from IoT malware. We implement the signature language and we integrate it into an intrusion detection system. Using our proposed methodology, the generated signatures can effectively and efficiently report intrusion attempts.**

I. Introduction

The adoption of network encryption in current communications is rising. Global HTTPS page loads have increased to more than 80%, in a time window of just four years [1]. After the publication of RFC [2], IETF reports that TLS 1.3 adoption is growing and more than 30% of Chrome's connections negotiate with this version of TLS [3]. Protecting user privacy with encrypted communications is fundamental, yet encryption introduces challenges for deep packet inspection (DPI) tools, which mostly inspect parts of packet payloads. DPI is the core operation of typical network monitoring applications, such as packet forwarding, L7 filtering [4], [5], firewalls and intrusion detection systems (NIDS) [6], [7]. A typical NIDS inspects packet headers and payloads to report malicious or abnormal traffic behavior. In encrypted packets[1] though, the only information that makes sense is (i) TLS handshake packets and (ii) TCP/IP packet headers (the data transmitted in packet payloads is encrypted). So, even popular intrusion detection systems seem to inadequately inspect encrypted connections. The SSL Readme page of Snort, for instance, reports that when inspecting port 443, "only the SSL handshake of each connection will be inspected" [8]. Yet, malware continues to be a crucial problem on the Internet [9] and network encryption makes it more difficult for malware detection systems to identify them. To overcome the challenges that network encryption introduces in the domain of network security, many works employ alternative techniques to identify the nature of the traffic. Recently, machine learning techniques are widely used for traffic classification, network analytics and malware detection [10], [11], [12], [13], [14]. Others focus on the implementation of real-time traffic identification systems for encrypted networks (such as in [15]). The majority of these works show that despite having encrypted payloads in network packets, we are still able to classify network traffic even in a fine-grained manner [15], [16], [17]. Packet headers contain information like IP addresses, port numbers and packet data sizes. Flow duration and packet inter-arrival times are time-related features that are relevant in encrypted traffic analysis and can be easily computed. When properly combined, packet metadata can offer valuable traffic insights [13]. In this paper, we examine the automatic generation of expressive and fine-grained signatures. The signatures are tailored for intrusion detection in encrypted networks and are constructed using sequences of packet payload sizes (Section II). We evaluate the effectiveness of the signatures, while we also modify a high-performance intrusion detection engine to support the matching of metadata-based signatures (Section III). In the evaluation, we use packet captures from several penetration tools that we produced in a controlled environment and packet captures from IoT malware that we found online in a public dataset. Yet, we expect that our methodology can be extended using more types of malicious network traffic, likewise.

The contributions of this work are the following:

1) We show how pattern mining algorithms automate the procedure of packet metadata signature generation (II).
2) We evaluate our methodology using a handful of packet captures and malicious activities (III).
3) We compare the effectiveness of our signatures against a popular NIDS in encrypted network traffic (III).

II. Encrypted Traffic Signatures

A thorough examination of the literature and our own analysis, led us to conclude that the inspection of sequences of incoming packet payload sizes can point to discrete events that possibly signify an intrusion attempt within a network [18], [15], [19]. Figure 1 shows examples of how discrete events in network traffic can be revealed only by observing their patters of sequences of packet payload sizes. With signatures, we refer to sequences of packet payload sizes within a network flow. A network flow is characterized by the typical 5-tuple {*source IP address, destination IP address, source port, destination port, protocol*}. This mean that a signature is unidirectional.

[1]With encrypted packets, we refer to TCP packets that are secured using the TLS protocol.

TABLE I

SIGNATURE EXAMPLES. EACH SIGNATURE CORRESPONDS TO A SEQUENCE OF PACKET PAYLOAD SIZES THAT MUST BE MATCHED AGAINST A NETWORK FLOW TO REPORT AN INTRUSION ATTEMPT EVENT.

Intrusion attempt event	**Pen. tool/Malware**	**Signature**
SSH Password cracking	Hydra	22, 976, 48, 16
File/directory scanning	Dirbuster	608, 80, 155, 156
Communication with C&C server	IRCbot	16, 23, 19, 13
Communication with C&C server	Muhstik	78, 15, 31, 47

Signatures are matched against incoming traffic anywhere in the network flow.

In this section, we describe the signature language that expresses such network traffic patterns. Our goal is to develop a simple signature language that is expressive enough and enables automated mining. The automatic generation of the signatures is an offline process, the matching procedure happens at runtime on live network traffic. All the aforementioned requirements led us to build signatures using a simple format that can be applied on sequences of packet payload sizes. With this approach, for the implementation of the intrusion detection engine, we will use an automaton, inspired by Aho-Corasick [20]. Thus, we get the privilege of not having to maintain the previously observed packets for backtracking, for each incoming packet. Yet, we are expressive enough to identify the suspicious events that signatures indicate. Table I illustrates some signature examples that we construct during our analysis. The proposed format is easy to follow. An intrusion attempt event is reported, right after a network flow matches one or more signatures. For instance, when different network flows contain a sequence of 4 packets with payload sizes $22, 976, 48, 16$ bytes respectively then our intrusion detection engine reports a password cracking attempt originating from the Hydra tool [21].

The proposed signature language can be extended by adding regular expression support for additional expressiveness. Yet, we decide to keep the signature language complexity to a minimum in order to facilitate the automatic signature mining procedure. Since we inspect TCP packets, we have to take into consideration the retransmitted packets. Handling packet retransmissions via the signature can be risky due to the unpredictable nature of the network behavior. So, we want to avoid the risk of losing an intrusion attempt report only due to a not properly defined signature. So, we decide to normalize the network traffic by discarding retransmitted TCP packets in a filtering phase, before traffic inspection.

A. Signature Mining

To enable fast signature generation for detecting different intrusion attempts, we aim for automating the procedure. We extract the intrusion signatures from network packet traces using frequent sequential pattern mining. More specifically, from our ground-truth sample collection, we detect frequent packet payload size sequences that correspond to specific intrusion attempts. Unlike other works, our approach does not depend on network statistical measures for the encrypted traffic inspection [13], [22]. In the paragraphs that follow, we present our methodology for the automatic signature generation.

First, we process the traffic captures so as to keep only the network packets that are related to the malicious activity. All the remaining packets other than the malicious activity under examination are discarded. Similarly, as already discussed in §II, we discard retransmitted TCP packets, as well. Then, we use the `joy` tool [23] to extract per network flow data that are used for signature generation. More specifically, `joy` receives as input a packet capture that contains an intrusion event (§ III-B). `Joy` returns a JSON file with network flow related information, such as the sequence of packet sizes and arrival times, DNS names, HTTP header fields and others. For each network flow originated from the intrusion event under examination, we retrieve the sequence of non-zero packet payload sizes and the packet arrival times. This sequence of non-zero packet payload sizes is later used by the signature mining procedure. For the signature mining procedure we choose to utilize a frequent sequential pattern mining technique.

Such algorithms discover frequent sequential patterns that occur in sequence databases. Benefiting from such techniques, in our proposed methodology we choose to utilize a maximal sequential pattern mining algorithm. Maximal sequential pattern mining is used to extract the frequent *longest common sequences* of network packet payload sizes contained in traffic. Our methodology uses the resulting sequences as potential signatures that can indicate an intrusion attempt. The resulting signatures are mined using the VMSP algorithm [24], with minimum support 50%. Finally, we select the maximal sequences that match to the ground truth information that we have. For instance, if the time window of the intrusion attempt is close (in time) to the sequence's first occurrence inside the network traffic. The generated signatures, then, are used to report intrusion attempts on a test dataset. The training dataset is the traffic traces that we used to export the frequent longest common sequences and generate the signatures. This training dataset consists of the 30% of the total traffic traces that we collected, infected with different intrusion attempt events (e.g., password cracking). For the generation of the signatures we take into account the direction of the packets (i.e., incoming packets).

One of our design goals is to generate expressive signatures that will be able to detect intrusion attempt events in a fine-grained manner. After a manual examination of the signatures generated by the automatic methodology that we followed (§ II-A), we found out that the majority of resulted signatures consisted of short sequences of packet payload sizes (e.g., median sequence length was 3 packet payload sizes). Having signatures with short sequences of packet payload sizes results to a more compact automaton (i.e., less memory requirements). Yet, a short sequence can eventually result to high false positive rates, as well. A signature construction decision can significantly disturb the resulting rates of true positives and false positives. A short signature can lead to high true positive rates– yet, there is the cost of a high false positive rate. A large signature, with a more strict definition, can give satisfactory true positive rates and reduce false positives.

Figure 1(a) illustrates the sequences of packet payload sizes that appear within a network packet capture during a scanning attempt with the "dirbuster" tool. Examining the respective signature from the automatically generated signature set, we observe that the packet sequence that is depicted by the corresponding figure is the "$608, 80, 155, 156$". This packet payload size sequence is the longest common sequence in the dataset that was used for the signature generation (§ II-A). Yet, we speculate that having such a short sequence would probably lead to higher false positives when evaluated in a larger testing dataset. Additionally to the VMSP maximal sequential pattern mining algorithm, we use the CM-ClaSP algorithm to discover closed sequential patterns [25]. Closed sequential pattern mining produces the largest subsequences that are common sets of sequences. So in practice, the resulted number of maximal patterns are less than closed patterns (or all patterns). Following the same signature generation methodology, combining the two frequent pattern mining algorithms, we obtain the following signatures for the intrusion attempt event illustrated in Figure 1(a) (selected set of sequences):

```
608, 80, 155, 155, 152, 152
...
608, 80, 155, 155, 152, 158
...
608, 80, 155, 156, 153, 152
...
608, 80, 155, 156, 153, 158
...
```

Signatures that occur from this optimization are more difficult to match an irrelevant network flow that will lead to false positives. For instance, a network flow with the packet payload size sequence "608, 80, 155, 156, 90, 32, 60" (this sequence of packet payload sizes does not refer to a malicious event) would have matched against the short signature that was generated by our initial methodology (i.e., "$608, 80, 155, 156$"). However, if we combine the results of the two frequent pattern mining algorithms we will have a longer, more expressive signature set, which eventually will result to less false positives. Indeed, we noticed that this approach reduces some false positives from those of the initial methodology.

III. Evaluation

In this section, we evaluate the effectiveness of the proposed signature language. The intrusion events that we examine come from activity originated by (i) penetration tools (i.e., *pentool-dataset*) and (ii) IoT malware (i.e., *iot-malware-dataset*). The traffic that characterises the network activity of penetration tools is collected by us in a controlled environment and the traffic that characterises the network activity of IoT malware is retrieved by a public repository [26]. For the *pentool-dataset*, we used 30% (randomly chosen) of the packet captures for the signature generation, and the remaining 70% for the evaluation. For the *iot-malware-dataset*, a big majority of the malware families that exist in the dataset are contained in a single packet capture file. Thus, we split the network flows into separate packet captures into (i) signature generation captures and (ii) evaluation captures. The process is straightforward, since there are files that contain the necessary information (e.g., which network flows contain a certain malware activity). Just like in the *pentool-dataset*, we use the 30% of the total packet captures for signature generation and the remaining 70% for the signature evaluation [2].

A. Traffic Processing and Filtering

We divide the network traffic collected during the attacks into network flows. As previously stated, a network flow is characterized by the standard 5-tuple {*source IP address, destination IP address, source port, destination port, protocol*}. Since the network traffic from the *pentool-dataset* is generated within a controlled and isolated environment and the IP address of both machines is known, we can presume that the resulted flows in a packet capture indicate the traffic generated by the malicious machine during each corresponding attack. Regarding the *iot-malware-dataset*, we use the available logs that describe each attack and are available in the repository [26]. The *iot-malware-dataset* contains unencrypted network traffic, containing protocols like HTTP [3]. The labels are produced after an analysis using the Zeek network security monitor [27].

In our methodology we discard retransmitted TCP packets, since such packets do not offer additional information to the flow. In addition, we assume that packet payloads are encrypted and thus, our approach proposes processing only packet metadata (e.g., packet payload size, packet direction). Packets that do not contain payload are also not processed (TCP ACK packets), since they do not provide any valuable information for our methodology.

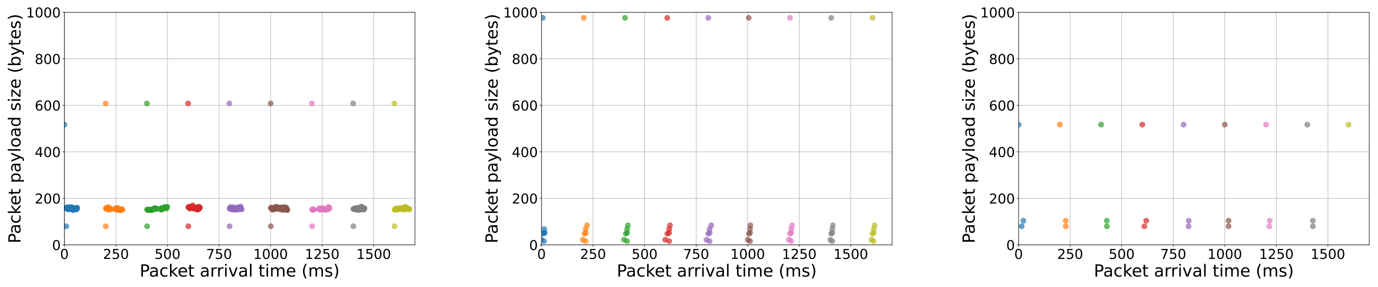
B. Testbed Setup for Traffic Collection

For the collection of the penetration tools dataset (i.e., *pentool-dataset*), we setup an environment with two virtual machines. The first machine runs Kali Linux and the second machine runs a vulnerable Ubuntu distribution with DVWA [28] installed using a self-signed certificate to enable HTTPS connections. The two machines are isolated from the network to ensure that no other machine is affected and a safe intercommunication between the two machines is established. The Kali Linux machine (IP address: 192.168.56.101) serves as the malicious entity that communicates with the vulnerable Ubuntu machine (IP address: 192.168.56.103) in order to perform various malicious activities (e.g., port scanning, file/directory scanning, password cracking, sql injection). The tshark tool is installed on the vulnerable machine and captures the incoming network traffic during the intrusion attempts performed by the malicious machine.

We choose some popular vulnerability scanners to evaluate our methodology. Some of the tools used for the intrusion

[2]While it is common to use 70% of the dataset for training and 30% for testing, we do the reverse to stress the effectiveness of our approach in cases of limited data.

[3]Even though the payloads are not encrypted, we follow the same methodology to produce the intrusion detection signatures using packet metadata.



(a) Scanning with "Dirbuster" tool. (b) SSH password cracking with "hydra" tool. (c) Login attempt with "hydra" tool.

Fig. 1. Illustration of packet payload size sequences within a network traffic capture of (a) file scanning attempt using the "dirbuster" tool, (b) a ssh password cracking attempt using the "hydra"tool and (c) a login attempt to the web server using the "hydra"tool, during the first 1.7 seconds of the active network flows. Each bullet color represents a single network flow.

events generation are *DIRB* [29], *NIKTO* [30], *SQLMAP* [31], *HYDRA* [21], *NMAP* [32], and *METASPLOIT* [33]. We perform numerous instances of attacks for different times and days within a one-month period. DIRB is a web content scanner. NIKTO examines a web server to find potential problems and security vulnerabilities. SQLMAP is a penetration test tool that enables SQL injection exploitations. HYDRA is used for login cracking. NMAP is a utility that enables network security auditing. METASPLOIT is a penetration testing software (we use the `msfconsole`, which is the metasploit framework console). Table II presents the events generated. Overall, we collected a set of over 120 packet captures (half of these packet captures were generated using a different Kali Linux distribution, to ensure that the signatures are resilient across OS/application updates). Each individual packet capture simulates an intrusion attempt as described in Table II. For each packet capture, we log the start time and end time of each intrusion attempt event.

We do not discuss in detail the testbed setup of the IoT malware dataset (i.e., *iot-malware-dataset*) in this section, since a thorough description of the dataset is provided in the public repository [26] by the creators. The events that we examine are presented in Table III.

TABLE II
INTRUSION ATTEMPTS TO THE VULNERABLE WEB SERVER.

Event id	Tool/Malware name	Activity
1	Dirbuster	Web content scanning in victim machine
2	Nikto	Web server scanning in victim machine
3	Hydra	Admin login attempt to web server in victim machine
4	Hydra	Root login attempt to web server in victim machine
5	Metasploit	Directory scanning to web server in victim machine
6	Metasploit	File scanning to web server in victim machine
7	Sqlmap	SQL injection to web server in victim machine
8	Nmap	Detection of remote services version numbers
9	Nmap	OS detection, version detection, script scanning, traceroute

C. *Signature Effectiveness*

In this section, we evaluate the effectiveness of the signatures that are generated by our methodology. For the evaluation, we use 70% of the total packet traces that contain a single intrusion event (the remaining 30% packet captures were previously used for the signature generation).

TABLE III
MALICIOUS ACTIVITY PROCESSED FROM THE IOT-23 DATASET [26].

Event id	Malware name	Activity
10	Mirai	Connection of an infected device and a CC server
11	IRCbot	Connection of an infected device and a CC server
12	IRCbot	Attack from infected device to a host
13	Muhstik	Connection of an infected device and a CC server
14	Muhstik	Attack from infected device to a host

Each packet capture in the dataset contains only a single intrusion event type (Tables II and III). When a signature reports an intrusion event, we compare it to the actual event. For instance, when a signature reports that "web content scanning" probably occurs in a specific packet capture, we manually investigate the actual event. If the intrusion attempt event that happens in the specific packet capture is the same as the event that was reported, then we mark this report as correct. When an event is correctly reported, we increase the true positive counter. If the report is incorrect, we increase the false positive counter for the signature. The true positive rate (TPR) for each malicious activity is presented in Table IV. For the *pentool-dataset*, the TPR of our signature generation methodology is 100% individually for each event. This means that the signatures that are generated to report a specific intrusion attempt event can correctly identify the existence of this specific event. For instance, the signature that is generated for the identification of *event no.1* "web content scanning in victim machine" using the `dirbuster` tool, correctly reports the existence of such event in every packet trace that indeed contains such event (signature TPR for *event no.1*: 100%). For the *iot-malware-dataset*, the TPR fluctuates between 62% and 100%. Mostly, the signatures that cause a low TPR are exported by packet captures with limited network flows to contain a malicious activity. For instance, the IRCbot network flows that contain a communication with a CC server are 1530 (*event no.11* TPR: 100%) while the IRCbot network flows that contain an attack are 677 (*event no.12* TPR: 62%).

Besides the true positive rate, we measure the resulting true negative rates and the false discovery rates for each intrusion attempt event. The validation of results is challenging in encrypted network traffic. A network intrusion detection system must be able to report any traffic behavior that is suspicious, while it is equally important to not falsely report events that are not existent in the network. The false discovery rate of our methodology is reported in Table IV. False discovery rate is calculated as $FDR = FP/(TP + FP)$. In detail, events *no.1*, *no.2*, *no.7* and events *no.10–no.14* are expressed by signatures that perform very effectively, having high true positive rates and no false positives. Each signature that is generated by our methodology to identify each one of the three intrusion events (events no.1, no.2 and no.7) contains no less than two sequences of packet payload sizes. Thus, to report an event, a network flow must match each one of the sequences that are contained within the signature. This makes the three signatures (events no.1, no.2 and no.7) stronger than the remaining signatures that raise a number of false positives. For each one of the remaining tools (i.e., `hydra`, `metasploit`, `nmap`), our signature mining methodology produces a single signature. Each signature contains only a single sequence of packet payload sizes, which makes it easier for the signature to be matched against a network flow; something that could eventually present false positives. Moreover, we observed that the signatures that were generated by our methodology for two events of the same tool were identical. For instance, *event no.3* and *event no.4* (i.e., `hydra` tool) are described by the same sequence of packet payload sizes. Similarly, *event no.5* and *event no.6* (i.e., `metasploit` tool) share the same signature as well as *event no.8* and *event no.9* (i.e., `nmap` tool). As a result, different events of the same tool are reported simultaneously. For example, the false discovery rate is 11% for the signature of the *event no.3*, since the network traffic that is produced during an *event no.3* is falsely reported as *event no.4*, as well. Still, it is correctly reported as an *event no.3*. Minimizing the false positives that an intrusion detection system presents is very important. At this point, we highlight that the false discovery rate that is presented by some events (e.g., *event no.3*, *event no.5*) is negligible, if we consider that these signatures correctly report the existence of the tool and the traffic that it generates. Even thought the granularity of the event is not fine-grained (the generated signature for the `hydra` tool can not distinguish between events no.3 and 4), the signature is still able to correctly identify the existence of the traffic that the tool generates in a network. In addition, we use normal HTTPS traffic samples to measure the FDR for the signatures generated. These results are presented in Table IV under the column *FDR (normal)*. The samples that we used for this experiment are publicly available [34] (packet captures used: CTU-Normal-20 – CTU-Normal-32). Correctly, our signatures do not report any intrusion event in the normal traffic dataset, leading to 100% TNR and 0% FDR. .

a) Comparison to Snort rules: As an additional evaluation step, we compare the signatures generated by our methodology to the most relevant Snort signatures. We download the latest version of the community Snort rules [35] and we extract the rules that match the same tools. More specifically, we have identified 101 *metasploit* rules, six *nmap* rules, one *dirbuster* and one *sqlmap* rule. For *hydra*, we identified eight Snort rules, which seem to target the Hydra malware and not the *hydra* tool – thus, we choose to exclude them from the evaluation. Finally, since there was no rule for *nikto*, we used one rule that was present in an older version of community snort rules. We executed Snort using the equivalent Snort rules against the same network packet captures that we used for the evaluation in Table IV. As presented in Table V, only the nmap rule that corresponds to the *event no.9* reported positively. Thus, it is apparent that Snort is not effective when performing against encrypted network traffic, in contrast to our methodology that matches packet metadata sequences and not packet contents. We do not execute Snort for the *iot-malware-dataset*, since it contains network packets that are not encrypted.

TABLE IV
RESULTING TRUE POSITIVE RATES (TPR), TRUE NEGATIVE RATES (TNR) AND FALSE DISCOVERY RATES (FDR) BY THE SIGNATURES EXAMINED.

Event id	TPR (in-dataset)	TNR (normal)	FDR (in-dataset)	FDR (normal)
1 (Dirbuster)	100%	100%	0%	0%
2 (Nikto)	100%	100%	0%	0%
3 (Hydra)	100%	100%	11%	0%
4 (Hydra)	100%	100%	11%	0%
5 (Metasploit)	100%	100%	11%	0%
6 (Metasploit)	100%	100%	11%	0%
7 (Sqlmap)	100%	100%	0%	0%
8 (Nmap)	100%	100%	11%	0%
9 (Nmap)	100%	100%	11%	0%
10 (Mirai)	100%	100%	0%	0%
11 (IRCbot)	100%	100%	0%	0%
12 (IRCbot)	62%	100%	0%	0%
13 (Muhstik)	100%	100%	0%	0%
14 (Muhstik)	100%	100%	0%	0%

TABLE V
COMPARISON OF THE EFFECTIVENESS OF THE RULES THAT ARE GENERATED BY OUR METHODOLOGY TO THE EFFECTIVENESS OF THE CORRESPONDING RULES THAT ARE USED BY SNORT.

Event id	Our TPR	Snort's TPR
1 (Dirbuster)	100%	0%
2 (Nikto)	100%	0%
3 (Hydra)	100%	–
4 (Hydra)	100%	–
5 (Metasploit)	100%	0%
6 (Metasploit)	100%	0%
7 (Sqlmap)	100%	0%
8 (Nmap)	100%	0%
9 (Nmap)	100%	80%

D. Integration with a Network Intrusion Detection System

The choice of the pattern matching algorithm is crucial for efficiently matching large data streams against multiple patterns. Inspired by the Aho-Corasick string matching algorithm [20], we implement a finite state machine to efficiently match a set of patterns (i.e., signatures) against streams of network packets. The majority of traditional signature-based network intrusion systems uses Aho-Corasick, since it offers simultaneous multi-pattern matching and is suitable for acceleration using hardware architectures that enable parallel processing [36], [37]. We extend the Aho-Corasick algorithm to enable integer matching – except for strings, similar to [15], [38], [19]. After several performance micro-benchmarks, our intrusion detection system for encrypted traffic inspection fluctuates between 69 – 85Gbps processing throughput, when the mean incoming traffic rate is 100Gbps (1000 signatures of varying lengths used). More specifically, when the incoming traffic causes high processing load conditions (i.e., when every network flow is malicious and matched against signatures), the intrusion detection system performs an average of 69Gbps, while when the incoming traffic causes moderate processing load conditions (i.e., when 10% of the network flows are malicious and matched against signature), the intrusion detection system performs an average of 85Gbps.

IV. Related Work

Popular signature-based network intrusion detection systems (such as Snort [6] and Bro [27]) include pattern matching and regular expressions matching algorithms for traffic inspection. However, Snort is not capable of encrypted traffic inspection from the network level. Traditional deep packet inspection tools are able to inspect encrypted traffic after it gets decrypted on the server side or in middleboxes used as proxies to intercept the encrypted traffic [18]. For instance, the SSL Readme page of Snort reports that when inspecting port 443, “only the SSL handshake of each connection will be inspected” [8] – leaving the encrypted packets uninspected. Also, Snort and Suricata rules provide support for packet payload size matching (rule option *dsize* [39], [40]). Yet, to the best of our knowledge, such tools do not match *sequences* of packet payload sizes, in contrast to our methodology.

In addition, there are works that perform searchable encryption (e.g., BlindBox [41] and PrivDPI [42]). BlindBox performs DPI directly on the encrypted traffic, utilizing specific encryption schemes. Specifically, BlindBox performs searchable encryption with a technique that enables the inspection of certain keywords in encrypted traffic. It utilizes a middlebox that obtains the encrypted rules and a mechanism that decrypts the flow when a suspicious keyword is matched. PrivDPI reduces the setup delay of BlindBox and retains similar privacy guarantees. While both works are capable to perform DPI directly on encrypted packets as-is, without any decryption or editing, they add important performance overheads.

Kitsune [43] is a neural network NIDS, which detects abnormal patterns in network traffic by observing statistical patterns. Kitsune as a result, reports the anomalous patterns encountered. Authors in [44] propose the combination of different deep learning techniques for intrusion detection. Tang *et al.* [45] perform flow-based anomaly detection in SDN environments. Niyaz *et al.* [46] detect DDoS attacks in SDN environments using deep learning. In their work, Anderson *et al.* [13] compare the performance of six different machine learning algorithms based on their properties specifically for the use-case of encrypted malware traffic classification. Amoli *et al.* present a real-time NIDS, which is able to detect new attacks within encrypted communications [47] in an unsupervised manner. Rosner *et al.* [12] present a method to detect and quantify information leaks in TLS-encrypted network traffic, revealed by side-channel information and metadata. These techniques focus on the identification of the activities within the network traffic, using machine learning algorithms. Authors focus on the thorough analysis of the encrypted network traffic and test its feasibility, with techniques that are often not suitable for real-time traffic inspection. Papadogiannaki et al. [19] performed network intrusion detection with signatures of packet metadata sequences and tested their methodology against traffic traces containing DoS and Reconnaissance attacks. In this paper, we produce signatures using two data mining algorithms and we evaluate our methodology against popular penetration tools and a public IoT malware dataset. We do not compare the effectiveness of our signatures to works that use Machine Learning algorithms for numerous reasons: (i) we do not use the same datasets, so an effectiveness comparison would not be appropriate, (ii) in our work, we consider only packet payload sequences to build our signatures, while other ML works use diverse features, (iii) we do not offer another ML work to prove that it is feasible to do an analysis in encrypted communications [48], we offer an approach to mine intrusion detection signatures.

Besides network security, there are systems that focus on the inspection of encrypted network traffic specifically for use-cases of network analytics [11], [16], [49]. For instance, OTTer [15] is an engine that identifies user actions in mobile applications, such as Skype and Whatsapp, in encrypted network traffic by matching trains of packet metadata. Mazhar et al. [50] investigate if it is feasible to evaluate the Quality of Service of videos that are exchanged using the protocols HTTPS and QUIC. Based on their research, the features that expose the most usable information are retrieved from network and transport layer headers in TCP network flows, whereas in QUIC flows, the most import information are inter-arrival time, packet sizes, packet/byte counts and throughput. Orsolic et al. [51] estimate the Quality of Experience for YouTube using a machine learning technique. In CSI [52], authors aim to derive the video adaptation behavior under HTTPS and QUIC, again using information like packet size and timing. Ghiette *et al.* [53] demonstrate that it is possible to utilize cipher suites and SSH version strings to generate unique fingerprints for bruteforcing tools used by an attacker.

Acting as proxies and by intercepting the network traffic, network middleboxes or client-side software manage to inspect encrypted traffic. Goh et al. [54] propose the mirroring of the traffic to a centralised intrusion detection system, which decrypts the traffic and afterwards it attempts DPI. However, such systems raise privacy preserving guarantees. Our work is able to fully operate with encrypted network traffic, without the need of traffic interception, since we only process packet metadata patterns and not packet payload contents.

In this work, we propose a signature mining method for intrusion detection in encrypted network traffic. The majority of works that inspect encrypted network traffic turn to machine learning algorithms since it is possible to examine the *feasibility* of the identification of the traffic nature and the underlying activities for numerous use-cases (e.g., for network analytics or network security). Our work builds on these results, but our main concern is to establish a procedure to effectively generate intrusion detection signatures in an automated manner and integrate them in a real-world traffic monitoring system.

V. Discussion

a) Traffic Analysis Resistance: There are characteristics within a network flow that even with encrypted packets, they present patterns that often reveal the traffic nature and the underlying activities. Padding packet sizes or transmitting packets with a certain interval frequency can obfuscate the behaviour of a privacy preserving browser and reduce user information leakage. Techniques like [55], [56] can circumvent our methodology. Still, we find padding techniques impractical, since substantial overhead is added to a communication system.

b) TLS 1.3: Expecting the vast adoption of TLS 1.3, we do not perform TLS certificate fingerprinting, like other relevant solutions [57]. The TLS 1.3 handshake is quite different from earlier versions of TLS, with a large portion of it getting encrypted (including certificates) [58]. Thus, the introduction of TLS 1.3 encourages our proposed methodology, in which we simply search for sequences of packet metadata, like the packet payload size. Our methodology is not affected by the TLS 1.3 ClientHello Padding Extension [59] that enables padding in the TLS ClientHello messages to a desired size, since we do not process such packets.

c) Future Work: As future work, we plan to generate signatures and evaluate our methodology using traffic captures from other popular malware families. Also, we aim to examine (i) how we can approach intrusion detection for encrypted protocols that multiplex network flows (e.g., VPN) and (ii) how sensitive our signatures are to different network stack implementations of endpoints.

VI. Conclusion

In this paper, we propose a methodology to mine signatures for intrusion detection in encrypted networks. We show that a simple signature language can be expressive and effective enough, achieving a processing throughput of up to 85Gbps that makes our intrusion detection engine appropriate for real-time inspection. Prior related work in the network security domain examined the *feasibility* of intrusion detection in encrypted networks using machine learning (offline processing).

In this work, we present a network intrusion detection system that operates with signatures that express sequences of metadata to enable traffic inspection even after the adoption of encryption. In network security applications, promptly reporting a correct event is vital. Current intrusion detection systems *must* become more effective with the conjunction of typical string pattern matching for packet payload content inspection and packet metadata matching. While our technique is inspired by traditional network inspection techniques, it solves a very contemporary and urging problem: the effective processing of encrypted packets when typical deep packet inspection techniques fail.

Acknowledgements

This work was supported by the projects CONCORDIA, CyberSANE and COLLABS funded by the European Commission under Grant Agreements No. 830927, No. 833683 and No. 871518. **This is a preprint version of the paper, which is accepted for publication in the proceedings of IEEE Conference on Dependable and Secure Computing (DSC) 2022 with DOI 10.1109/DSC54232.2022.9888942.**

References

[1] “A nonprofit Certificate Authority providing TLS certificates to 225 million websites.: 2019 Annual Report,” https://abetterinternet.org/documents/2019-ISRG-Annual-Report-Desktop.pdf, 2019, Accessed: 2020-10-29.

[2] “The Transport Layer Security (TLS) Protocol Version 1.3,” https://tools.ietf.org/html/rfc8446, 2018, Accessed: 2020-10-29.

[3] “TLS 1.3: One Year Later,” https://www.ietf.org/blog/tls13-adoption/, 2019, Accessed: 2020-10-29.

[4] L. Rizzo, M. Carbone, and G. Catalli, “Transparent acceleration of software packet forwarding using netmap,” in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 2471–2479.

[5] “Application Layer Packet Classifier for Linux,” Available: http://l7-filter.sourceforge.net/, 2009, Accessed on Oct. 7, 2020.

[6] “The Snort IDS/IPS,” Available: https://www.snort.org/, 2020, Accessed on Oct. 8, 2020.

[7] “Suricata Open Source IDS / IPS / NSM engine ,” Available: https://www.suricata-ids.org/, 2020, Accessed on Oct. 8, 2020.

[8] “Snort: README.ssl,” Available: https://www.snort.org/faq/readme-ssl, 2020, Accessed on Oct. 28, 2020.

[9] P. Kotzias, L. Bilge, P.-A. Vervier, and J. Caballero, “Mind your own business: A longitudinal study of threats and vulnerabilities in enterprises.” in *NDSS*, 2019.

[10] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, “Copperdroid: automatic reconstruction of android malware behaviors.” in *Ndss*, 2015.

[11] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, “Deep packet: A novel approach for encrypted traffic classification using deep learning,” *Soft Computing*, pp. 1–14, 2017.

[12] N. Rosner, I. B. Kadron, L. Bang, and T. Bultan, “Profit: Detecting and quantifying side channels in networked applications.” in *NDSS*, 2019.

[13] B. Anderson and D. McGrew, “Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity,” in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1723–1732.

[14] J. Liang, W. Guo, T. Luo, V. Honavar, G. Wang, and X. Xing, “Fare: Enabling fine-grained attack categorization under low-quality labeled data,” 2021.

[15] E. Papadogiannaki, C. Halevidis, P. Akritidis, and L. Koromilas, “Otter: A scalable high-resolution encrypted traffic identification engine,” in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 315–334.

[16] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, “Analyzing android encrypted network traffic to identify user actions,” *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 1, pp. 114–125, 2016.

[17] J. Bushart and C. Rossow, “Padding ain’t enough: Assessing the privacy guarantees of encrypted {DNS},” in *10th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 20)*, 2020.

[18] E. Papadogiannaki and S. Ioannidis, “A survey on encrypted network traffic analysis applications, techniques, and countermeasures,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–35, 2021.

[19] ——, “Acceleration of intrusion detection in encrypted network traffic using heterogeneous hardware,” *Sensors*, vol. 21, no. 4, p. 1140, 2021.

[20] A. V. Aho and M. J. Corasick, “Efficient string matching: an aid to bibliographic search,” *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[21] “Hydra Package Description,” https://tools.kali.org/password-attacks/hydra, 2020, Accessed: 2020-25-11.

[22] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, “Can’t you hear me knocking: Identification of user actions on android apps via traffic analysis,” in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*. ACM, 2015, pp. 297–304.

[23] “Joy: A package for capturing and analyzing network flow data and intraflow data, for network research, forensics, and security monitoring.” http://www.dvwa.co.uk, 2017, Accessed: 2020-10-11.

[24] P. Fournier-Viger, C.-W. Wu, A. Gomariz, and V. S. Tseng, “Vmsp: Efficient vertical mining of maximal sequential patterns,” in *Canadian conference on artificial intelligence*. Springer, 2014, pp. 83–94.

[25] P. Fournier-Viger, A. Gomariz, M. Campos, and R. Thomas, “Fast vertical mining of sequential patterns using co-occurrence information,” in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2014, pp. 40–52.

[26] “IoT-23 Dataset. A labeled dataset with malicious and benign IoT network traffic.” https://www.stratosphereips.org/datasets-iot23, 2021, Accessed: 2021-7-13.

[27] “The Zeek Network Security Monitor,” Available: https://www.zeek.org/, 2020, Accessed on Oct. 8, 2020.

[28] “Damn Vulnerable Web Application (DVWA),” https://github.com/cisco/joy, 2020, Accessed: 2020-10-11.

[29] “Dirbuster,” https://tools.kali.org/web-applications/dirbuster, 2020, Accessed: 2020-25-11.

[30] “Nikto,” https://tools.kali.org/information-gathering/nikto, 2020, Accessed: 2020-25-11.

[31] “Automatic SQL injection and database takeover tool,” http://sqlmap.org, 2020, Accessed: 2020-25-11.

[32] “Nmap Security Scanner,” https://nmap.org, 2020, Accessed: 2020-25-11.

[33] “Metasploit: The world’s most used penetration testing framework,” https://www.metasploit.com, 2020, Accessed: 2020-25-11.

[34] “Malware Capture Facility Project. Normal Datasets.” https://www.stratosphereips.org/datasets-normal, 2021, Accessed: 2021-2-25.

[35] “Snort Community rules 3100,” https://www.snort.org/downloads/community/snort3-community-rules.tar.gz, 2021, Accessed: 2021-2-25.

[36] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, “Gnort: High Performance Network Intrusion Detection Using Graphics Processors,” in *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection*, 2008.

[37] E. Papadogiannaki, L. Koromilas, G. Vasiliadis, and S. Ioannidis, “Efficient software packet processing on heterogeneous and asymmetric hardware architectures,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1593–1606, 2017.

[38] E. Papadogiannaki, D. Deyannis, and S. Ioannidis, “Head (er) hunter: Fast intrusion detection using packet metadata signatures,” in *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE, 2020, pp. 1–6.

[39] “Non-Payload Detection Rule Options: dsize,” http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node33.html#SECTION00467000000000000000, 2021, Accessed: 2021-2-25.

[40] “Payload Keywords: dsize,” https://suricata.readthedocs.io/en/suricata-6.0.0/rules/payload-keywords.html#dsize, 2021, Accessed: 2021-2-25.

[41] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, “Blindbox: Deep packet inspection over encrypted traffic,” *ACM SIGCOMM Computer communication review*, vol. 45, no. 4, pp. 213–226, 2015.

[42] J. Ning, G. S. Poh, J.-C. Loh, J. Chia, and E.-C. Chang, “Privdpi: Privacy-preserving encrypted traffic inspection with reusable obfuscated rules,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1657–1670.

[43] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, “Kitsune: an ensemble of autoencoders for online network intrusion detection,” *arXiv preprint arXiv:1802.09089*, 2018.

[44] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, “A deep learning approach to network intrusion detection,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, 2018.

[45] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, “Deep learning approach for network intrusion detection in software defined networking,” in *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. IEEE, 2016, pp. 258–263.

[46] Q. Niyaz, W. Sun, and A. Y. Javaid, “A deep learning based ddos detection system in software-defined networking (sdn),” *arXiv preprint arXiv:1611.07400*, 2016.

[47] P. V. Amoli, T. Hamalainen, G. David, M. Zolotukhin, and M. Mirzamohammad, “Unsupervised network intrusion detection systems for zero-day fast-spreading attacks and botnets,” *JDCTA (International Journal of Digital Content Technology and its Applications*, vol. 10, no. 2, pp. 1–13, 2016.

[48] C. J. Dietrich, C. Rossow, and N. Pohlmann, “Cocospot: Clustering and recognizing botnet command and control channels using traffic analysis,” *Computer Networks*, vol. 57, no. 2, pp. 475–486, 2013.

[49] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, “Robust smartphone app identification via encrypted network traffic analysis,” *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 63–78, 2017.

[50] M. H. Mazhar and Z. Shafiq, “Real-time video quality of experience monitoring for https and quic,” in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1331–1339.

[51] I. Orsolic, D. Pevec, M. Suznjevic, and L. Skorin-Kapov, “Youtube qoe estimation based on the analysis of encrypted network traffic using machine learning,” in *2016 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2016, pp. 1–6.

[52] S. Xu, S. Sen, and Z. M. Mao, “Csi: inferring mobile abr video adaptation behavior under https and quic,” in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[53] V. Ghiëtte, H. Griffioen, and C. Doerr, “Fingerprinting tooling used for {SSH} compromisation attempts,” in *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019)*, 2019, pp. 61–71.

[54] V. T. Goh, J. Zimmermann, and M. Looi, “Experimenting with an intrusion detection system for encrypted networks,” *International Journal of Business Intelligence and Data Mining*, vol. 5, no. 2, pp. 172–191, 2010.

[55] C. Chen, D. E. Asoni, A. Perrig, D. Barrera, G. Danezis, and C. Troncoso, “Taranet: Traffic-analysis resistant anonymity at the network layer,” *arXiv preprint arXiv:1802.08415*, 2018.

[56] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford, “Atom: Horizontally scaling strong anonymity,” in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 406–422.

[57] “Cisco Encrypted Traffic Analytics,” https://www.cisco.com/c/en/us/solutions/enterprise-networks/enterprise-network-security/eta.html, 2019, Accessed: 2020-20-11.

[58] P. Kotzias, A. Razaghpanah, J. Amann, K. G. Paterson, N. Vallina-Rodriguez, and J. Caballero, “Coming of age: A longitudinal study of tls deployment,” in *Proceedings of the Internet Measurement Conference 2018*, 2018, pp. 415–428.

[59] “A Transport Layer Security (TLS) ClientHello Padding Extension,” https://datatracker.ietf.org/doc/html/rfc7685, 2015, Accessed: 2020-7-17.