Head(er)Hunter: Fast Intrusion Detection using Packet Metadata Signatures

Eva Papadogiannaki*, Dimitris Deyannis*
FORTH-ICS
{epapado, deyannis}@ics.forth.gr
*Also with the Department of Computer Science, University of Crete, Greece

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

***Abstract*—More than 75% of the Internet traffic is now encrypted, while this percentage is constantly increasing. The majority of communications are secured using common encryption protocols such as SSL/TLS and IPsec to ensure security and protect the privacy of Internet users. Yet, encryption can be exploited to hide malicious activities.Traditionally, network traffic inspection is based on techniques like deep packet inspection (DPI). Common applications for DPI include but are not limited to firewalls, intrusion detection and prevention systems, L7 filtering and packet forwarding. The core functionality of such DPI implementations is based on pattern matching that enables searching for specific strings or regular expressions inside the packet contents. With the widespread adoption of network encryption though, DPI tools that rely on packet payload content are becoming less effective, demanding the development of more sophisticated techniques in order to adapt to current network encryption trends. In this work, we present HeaderHunter, a fast signature-based intrusion detection system even in encrypted network traffic. We generate signatures using only network packet metadata extracted from packet headers. Also, to cope with the ever increasing network speeds, we accelerate the inner computations of our proposed system using off-the-shelf GPUs.**

I. Introduction

The adoption of common encryption protocols – such as SSL/TSL and IPsec – is recently growing, aiming towards the protection of network communication channels. Statistics report that, currently, more than 75% of the Internet traffic is encrypted, while this percentage is constantly rising [1], [2]. Although this practice is crucial for the protection of end-users and services, it introduces new challenges for Internet traffic analysis and monitoring tools, which mostly rely on techniques like deep packet inspection (DPI). DPI is based on pattern matching, which enables searching for specific content inside the network traffic. Common applications for DPI include firewalls, intrusion detection and prevention systems, L7 filtering and packet forwarding. However, with the widespread adoption of network encryption protocols, solutions that rely on scanning packet payload contents are becoming less effective and new approaches have to be implemented in order to ensure their correct operation. Existing DPI solutions extract mostly coarse-grained information when operating on encrypted network traffic. Thus, it is important to adapt to current encryption trends to analyse fully encrypted network connections. An approach to inspect encrypted network traffic is the generation of signatures based on packet metadata, such as the packet timestamp, size and direction. These metadata can be usable even in encrypted traffic, since they can be easily extracted from packet headers. Recent related work has proven that revealing the traffic nature in encrypted communication channels is feasible using machine learning techniques either for network analytics, network security, privacy or fingerprinting. More specifically, such works focus on investigating which network related characteristics reveal the nature of the network traffic. As it occurs, we can divide the resulted network-related characteristics that make traffic classification feasible –even in encrypted networks– into three groups: (i) time-related, (ii) packet-related, and (iii) statistical characteristics [3], [4], [5]. Based on the outcomes of these works, we build signatures from the most descriptive and revealing packet metadata categories; the sequence of packet payload lengths in time and the packet direction. To cope with the ever increasing network speeds, we investigate the utilization of commodity hardware accelerators, such as GPUs, for high performance intrusion detection against network traffic. The benefits for such an implementation is the high processing throughput, as well as the low cost of powerful commodity high-end off-the-shelf GPUs (in contrast to expensive server setups). Since GPUs offer stream processing, real-time traffic inspection can be achieved [6], [7], [8], [9], [10]. This work can be divided into two parts – the offline and the online phase. In the offline phase, we firstly collect and process the ground truth samples for the signature generation. The ground-truth samples used for this work are collected from the publicly available dataset UNSW-NB 15 [11]. In the online phase, our system processes the input traffic and inspects the flows contained, reporting suspicious activity, based on the signatures.

The contributions of this work are the following: (i) we generate signatures for intrusion detection using strictly packet metadata extracted from packet headers, making our engine suitable for encrypted network traffic, (ii) we extend the most popular string searching algorithm, Aho-Corasick, to also support integers for packet metadata matching on GPUs, and (iii) we evaluate the signature quality as well as the pattern matching engine performance.

978-1-7281-6339-0/20/$31.00 ©2020 IEEE

II. Signature Generation

The first step to generate effective signatures is the analysis of a ground-truth dataset. For the purpose of this work, we utilize a publicly available dataset (i.e., [12]) that contains network traces originated from malicious activities and most importantly, these datasets are labeled and classified according to the attack type. As we discuss in the following paragraphs, this enables us to produce signatures able to report suspicious network flows in a fine-grained manner.

More specifically, in the dataset each record that represents a single attack contains a number of attributes. In this work, we build signatures based on packet-level information.As already mentioned, using a fraction of the attributes, we map every network flow to an attack type. More specifically, each network flow is characterized by the typical 5-tuple *"source IP address, source port, destination IP address, destination port, protocol"* combined by the two timestamps that indicate (i) the start and (ii) the termination of each connection. Then, we group these network flows by the attack type. Finally, a single record in our processed dataset is formatted as such: `''Attack Record 1: 1421927416, 1421927419, 175.45.176.2, 149.171.126.16, 23357, 80, TCP, Exploits''`, where `1421927416` is the first packet timestamp captured, `1421927419` is the last packet timestamp captured, `175.45.176.2` is the source IP address, `149. 171.126.16` is the destination IP address, `23357` is the source port number, `80` is the destination port number, `TCP` is the connection protocol and finally, `Exploits` is the attack category. Then, for each one of the records we extract the whole network packet trace from the traffic, similar to a network flow. Each flow, is characterized by the corresponding label that occurs from the ground-truth records (i.e., benign or malicious flow). After having divided the resulted network flows into the two categories, we divide the malicious network flows into two sets, the one that contains the malicious network flows that will be used for testing (60% randomly selected flows) and the one that contains the malicious network flows that will be used to produce our signatures (40% randomly selected flows).

A. Signature Specification

During our analysis, we observe that specific sequences of packet payload sizes reliably signify certain malicious activity. In this section we describe our proposed signature specification to express such patterns in network traffic. In this work, we aim for an expressive yet simple enough signature format to facilitate the automated generation of signatures. Another consideration for a practical system is to minimize the amount of state information that a packet inspection engine needs to maintain per flow in order to evaluate patterns across packet sequences in the same flow. The advantage of our solution is that it can be implemented with an automaton, without the need to retain previously observed packet sizes so as to support backtracking, and that it is expressive enough to capture the traffic metadata of interest. As already mentioned, we do not parse and we do not process any network packet payloads — we only focus on packet headers. This means, that the only information that we have available is contained inside a packet header. In addition, we aim to avoid adding any complexity in the signature specification procedure and, thus, we decide to utilize only the following fields for each network packet: (i) source IP address, (ii) source port, (iii) destination IP address, (iv) destination port, (v) protocol, (vi) payload size, (vii) packet timestamp. Afterwards, for each one of these network flows, we produce a sequence of metadata; (i) packet payload sizes and (ii) packet directions.

We introduce a regular-expression-like format to present the signatures (Table I). The sign of each number shows the direction of the packet when we produce signatures for bi-directional network connections. More specifically, a packet payload size with a negative sign indicates that the packet is received and not transmitted. Also, we support packet payload size ranges and repetitions. These techniques add extra expressiveness to the signatures. To build the signatures, we use a fraction of the network packet traces dataset. Having retransmitted packets is an unpredictable network behaviour, so we might lose a reporting solely due to a not properly defined upper bound in the repeat range of an expression. Thus, we choose to handle retransmitted TCP packets by discarding them in a packet filtering phase.

The network traffic ground-truth samples that we use to build our signatures provide fine-grained labels, which apart from classifying a network flow into the two categories *"benign"* and *"attack"*, also distinguish the attacks into categories (e.g., DoS and reconnaissance). Thus, we use this fine-grained information offered through the ground-truth dataset in order to build signatures that will further point to the most probable attack type, besides than solely classifying the network flow as malicious or not. In Section IV, we discuss about the quality of the generated signatures in respect to their attack type.

TABLE I
Signature format.

Direction	Signature Specification
Src $\rightarrow$ Dst	`64-69,100,109{2-3},5`
Dst $\rightarrow$ Src	`0, 100, 30{1-3}, 5`
Src $\leftrightarrow$ Dst	`64-69,-0,100,-100,109,-30,109,...`

III. Implementation

The second part of our solution is the implementation of the intrusion detection engine. Our intrusion detection system is signature-based. This means that every report of suspicious network flow is produced only when one, or more, of the predefined signatures is found inside the network traffic. The main process of signature-based systems is pattern matching. Inspired by one of the most popular algorithms for fast string pattern matching, called Aho-Corasick [13], our extended implementation operates on packet sizes, which are represented as short integers, instead of the typical Aho-Corasick implementation that operates on packet contents, which are represented as strings.

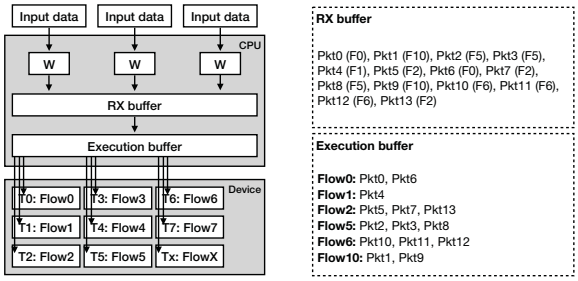

Fig. 1. Overview of the packet processing architecture. Each worker thread is assigned to a single input source, transferring the packets into the RX buffer. The RX buffer continuously receives packets, copying its contents to the execution buffer, where the packets are divided into flows, ordered and copied to the device's memory address space for processing. Each compute kernel thread is assigned to a single flow.

A. Automaton Creation

The Aho-Corasick algorithm is an efficient string searching algorithm that matches the elements of a finite set of strings against an input. It is able to match all pattern strings simultaneously, so its complexity does not depend on the size of the pattern set. It works by constructing an automaton that executes transitions for each 8-bit ASCII character of the input text. Since we aim to adapt the algorithm to match signatures that describe sequences of packet payload sizes, we replace the 8-bit characters with 16-bit values that represent packet sizes, similar to [4]. The algorithm builds a finite state machine, resembling a trie with added "failure" links between the trie nodes. These failure links are followed when there is no remaining matching transition and they offer fast transitions to other branches of the trie that share a common prefix, without the need for expensive backtracking using earlier inputs. This way, the algorithm allows the interleaving of a large number of concurrent searches, such as in the case of network connections, because the state of the matcher can be preserved across input data that are observed at different points in a time period by storing a pointer to the current state of the automaton, with the state maintained for each connection. Otherwise, backtracking would require the maintenance of expensive per-flow state for previously-seen packet payload sizes. In order to boost the resulted performance, we build a Deterministic Finite Automaton (DFA) by unrolling the failure links in advance, adding them as additional transitions directly to the appropriate node.

B. Packet Processing Parallelization

The overall architecture of our intrusion detection system is presented in Figure 1. The system utilizes one or several CPU worker threads, assigning each one to a singe input source (e.g., NIC). Once a CPU thread receives a network packet, it forwards it to a receive buffer, called RX batch (Fig. 1). At this point, the receive buffer is filled with packets belonging to one or several TCP flows. When the buffer is full, our system generates an execution batch with the traffic contained in the receive buffer. The execution batch contains the payload sizes of the received network packets, divided and ordered by the corresponding flows. We transform the input traffic to series of payload sizes with each series containing information of a single flow, ready to be processed by our pattern matching engine. This adds an extra level of efficiency since a network packet (with a typical minimum size of 64 bytes) is now represented as a short integer (2 bytes). Then, we transfer the execution batch to the device's memory address space. In the meantime, the receive buffer continues to accepts incoming packets, avoiding packet losses. We implement the pattern matching engine of our system as an OpenCL compute kernel. Unlike other relevant works that follow a packet-per-thread processing approach [8],we follow a flow-per-thread approach. This means that each thread reads at least one network flow from the execution batch and then performs the processing (Fig. 1). Whenever a batch of packets is received and forward for TCP flow ordering and processing by the device, new packets are copied to another batch in a pipeline fashion. Moreover, in order to fully utilize the SIMD capabilities of the hardware, we represent the payload sizes in the execution buffer as unsigned short integers. In this way, we are able to access the data using the `ushort16` vector data type, as described above, in a row-major order, being able to fetch information for 16 packets at once. During the processing, the pattern matching kernel uses one `ushort` value as input, representing one payload size, at each step in order to traverse the automaton, as described in Section III-A. If a signature is identified, the engine reports the suspicious TCP flow identifier, packed with the packets that matched the signature – using the first and the last packet contained in the signature, together with the signature identifier. We encode this information using four `ushort` values for each TCP flow that is identified as suspicious. In this way we minimize the amount of data that need to be transferred back from the device to the host's DRAM. Moreover, in cases where an execution batch does not contain any suspicious flows, the engine does not need to perform any other memory transfers except for initially transferring the data for processing.

IV. Evaluation

A. Signature Quality

In this section we demonstrate the expressiveness and effectiveness of the proposed signature specification by generating pattern signatures for a set of malicious activities, evaluating their accuracy. We use 40% randomly chosen flows from the ground-truth dataset as a reference, and the remaining 60% for the evaluation. In order for the TCP protocol to deliver data reliably, it provides a set of mechanisms to detect and avoid unpredictable network behavior, like packet loss, duplication or reordering. In our methodology, we choose to discard packets that do not offer substantial information to the flow, like re-transmitted packets, focusing entirely on processing packet metadata. Tables II and III show the resulting true positive rates (TPR) for different attack types found into the

ground truth dataset. Each traffic trace in the test dataset contains a combination of malicious and benign traffic (labeled). When a signature reports malicious activity we compare it with the actual category of the flow (malicious or benign). If the activity is correctly reported as malicious, then the TP counter is increased. Otherwise, we have a false positive (FP). The true positive rates, as presented in Tables II and III show the effectiveness of each signature according to the signature's length. More specifically, the reported true positive rates indicate that the signature length can significantly affect the signature effectiveness. For instance, short signatures result to higher TPR. In some cases, however, a short signature (that results to a high TPR) is possible to introduce the trade-off of resulting to a high FDR, as well. The false discovery rates[1], as presented in Tables IV and V, present the percentage of signatures that falsely reported malicious activity. To measure the FDR, we search for the same signatures (as in the previous experiment) against benign traffic.

As it occurs from our results, the FDR rates for the "Exploits" category are low, especially for short signatures, resulting to false discovery rates below 1%. In the "Reconnaissance" category, short signatures result to unacceptable false discovery rates. However, signatures of 8, 10 or 12 packets per sequence, in the same category, result to balanced rates between true positives and false discovery. As part of our future work, we plan to include more packet metadata in the signature generation phase, except for packet payload sizes and direction. Building signatures that present a balanced behavior, both in terms of true and false positives, is probably the optimal solution, however it is not a trivial procedure.

In the same tables we present the results while using different kind of packet metadata; instead of only using the packet payload length as part of the signature, we also use the packet direction (i.e. source to destination, destination to source, and bi-directional). As it occurs, signatures that also indicate the direction result to high true positive rates. Apparently, signatures' effectiveness can be further improved to produce less false positives, while keeping the true positive rates high. Taking into account more packet metadata, except for packet payload length and direction, is one way to accomplish this. However, in order to implement a simple, flexible and efficient intrusion detection engine (in terms of performance), we choose to generate and test signatures using only the packet payload length and direction.

At this point, we want to stress the fact that the generated signatures are strongly affected by the ground-truth dataset used, which contains the network traffic traces with the malicious and benign activity. Being highly dependent on ground-truth datasets for signature generation, we plan to examine more and diverse datasets and test them against real network traffic environments in the future.

B. Micro-benchmarks

For the performance evaluation of our implementation we use a commodity high-end machine. The hardware setup of our machine includes an Intel i7-8700K processor with 6 cores that operate at 3.7 GHz with hyper-threading enabled, providing us with 12 logical cores, configured with 32 GB RAM. In addition, we use a NVIDIA GeForce GTX 980 GPU. Our testbed system runs Arch Linux 4.19.34-1-lts and we use the OpenCL SDK from the NVIDIA CUDA Toolkit 10.2.120 for the NVIDIA GTX 980 GPU. At this stage, we only perform offline traffic processing, thus, for the performance evaluation we focus on micro-benchmarks in order to present the throughput and latency of the engine's execution.

[1]False discovery rate can be calculated as FDR = FP/(TP+FP)

TABLE II
TRUE POSITIVE RATE (TPR) SCORE OF SIGNATURES ("EXPLOITS").

	Packet Sequence Length				
Direction	**4**	**6**	**8**	**10**	**12**
Source → Destination	100%	93%	69%	63%	54%
Source ↔ Destination	100%	100%	97%	74%	61%

TABLE III
TRUE POSITIVE RATE (TPR) SCORE OF SIGNATURES ("RECONNAISSANCE").

	Packet Sequence Length				
Direction	**4**	**6**	**8**	**10**	**12**
Source → Destination	100%	89%	89%	89%	87%
Source ↔ Destination	100%	100%	89%	86%	82%

To present our automaton's compilation time, we generate signature sets out of varying packet sequences, each time increasing the number of signatures and the packet sequence length. Figure 2 presents the compilation time of the automaton, depending on the same signature sets. As expected, increasing the packet sequence length (even with only a difference of two packets) can significantly affect the automaton compilation time. The compilation time of the automaton does not affect the end-to-end performance negatively, since the compilation happens offline and only once. We extracted 1,500 signatures when the sequence size was 12 packets – yet, we present how our automaton construction technique performs (in terms of memory and processing time) for a larger number of signatures.

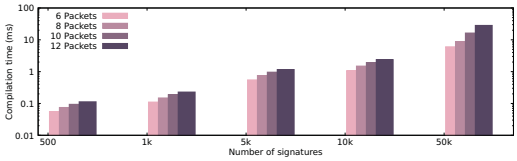

Fig. 2. Automaton compilation time using different combinations of patterns and pattern sizes.

Figure 3 presents the performance achieved by our pattern matching engine. More specifically, we show the throughput sustained by the discrete GTX 980 GPU executing the pattern matching engine of our intrusion detection solution. In Figure 3, the color-filled bars indicate the performance achieved by the pattern matching engine when the selection of (i) signatures and (ii) input results to a computationally loaded condition. In the figure, we present the worst-case scenario,

TABLE IV
FALSE DISCOVERY RATE (FDR) SCORE OF SIGNATURES ("EXPLOITS").

	Packet Sequence Length				
Direction	**4**	**6**	**8**	**10**	**12**
Source → Destination	0.8%	0.7%	0.6%	0.2%	0.1%
Source ↔ Destination	0.8%	0.8%	0.6%	0.2%	0%

TABLE V
FALSE DISCOVERY RATE (FDR) SCORE OF SIGNATURES ("RECONNAISSANCE").

	Packet Sequence Length				
Direction	**4**	**6**	**8**	**10**	**12**
Source → Destination	62%	0.9%	0.7%	0.3%	0%
Source ↔ Destination	63%	63%	0.11%	0.1%	0%

where we have full contamination of the traffic. White-filled bars with borders indicate the performance achieved in a computationally relaxed condition (i.e., less than 10% infected traffic), which is the most realistic scenario. We present the throughput using different packet batch sizes. In general, when we use discrete GPUs, increasing the batch size results in better sustainable throughput. Since typically, the discrete GPU and the CPU communicate over the PCIe bus and they do not share the same physical address space, it is preferable that all data transfers operate on fairly large chunks of data, due to the PCIe interconnect inability to handle small data transfers efficiently. The discrete GPU performs with up to 43Gbps throughput for 10% traffic infection and 37Gbps throughput for 100% malicious traces. The maximum processing latency never exceeded 2.5 msec. The results that are presented display the median values occurring after 30 runs per configuration.

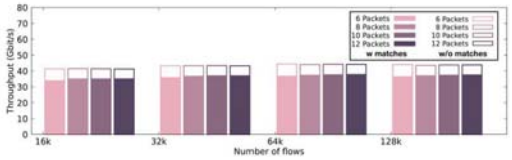

Fig. 3. Throughput of our pattern matching engine using a discrete GPU.

V. RELATED WORK

In this section we discuss the state-of-the-art in the domain of network traffic inspection and intrusion detection. Table VI presents a comparison between this and other related works. In the table, ticks signify that a specific work addresses the corresponding feature/approach, dashes signify that a specific work addresses the corresponding feature/approach and finally, circles signify that it is not clear whether the feature/approach is addressed – or to what extent. Unfortunately, we do not provide any comparison on the signature quality between these systems since we do not have access to such information.

Popular NIDS solutions like Snort [14] and Suricata [15] utilize pattern matching and regular expressions in order to analyse network traffic. The research community has also put effort in improving the performance of NIDS using either

TABLE VI
COMPARISON OF RELATED INTRUSION DETECTION SYSTEMS.

Works	*Encrypted Traffic*	*Online Inspection*	*Performance Efficiency*	*Details*
Snort [14], Suricata [15]	○	✓	○	Signature-/Anomaly-based, Payload inspection
Gnort [16], MIDeA [7]	–	✓	✓	Signature-based, Payload inspection, GPU
Kitsune [17]	✓	✓	–	Artificial Neural Network, No Payload inspection
HeaderHunter	✓	✓	✓	Signature-based, No Payload inspection, GPU

commodity hardware, such as GPUs [16], [18], [7] and parallel nodes [19], [20] or specialized hardware, such as TCAMs, ASICs and FPGAs [21], [22]. However, the majority of these works is able to process network traffic that is unencrypted, since they extract meaningful information from network packet payload content. More recently, many works focus on machine learning and deep learning techniques [23], [24]. Shone *et al.* [25] propose a system that combines deep learning techniques to provide intrusion detection. Tang *et al.* [26] present a deep learning approach for flow-based anomaly detection in SDN environments, while Niyaz *et al.* [27] utilize deep learning in order to detect DDoS attacks in such environments. Anderson *et al.* [28] compare the properties of six different machine learning algorithms for encrypted malware traffic classification. Moreover, Amoli *et al.* present a real-time unsupervised NIDS, able to detect new and complex attacks within encrypted and plaintext communications [29]. Kitsune [17] is a NIDS, based on neural networks, and designed for the detection of abnormal patterns in network traffic. It monitors the statistical patterns of recent network traffic and detects anomalous patterns. These techniques focus on identifying malicious behavior in the network, examining the characteristics of the underlying traffic, using exclusively machine learning approaches. BlindBox [30] performs deep-packet inspection directly on the encrypted traffic, utilizing a new protocol and new encryption schemes. Many other research and commercial solutions focus on inspection of encrypted network traffic mostly for network analytics [31], [3], [32]. OTTer [4] is a scalable engine that identifies fine-grained user actions in OTT mobile applications even in encrypted network traffic. Moreover, Rosner *et al.* [5] present a black-box approach for detecting and quantifying side-channel information leaks in TLS-encrypted network traffic.

All these works focus on machine learning techniques that signify the network-related characteristics that indicate this specific class. Unlike our work, the majority does not focus on providing a system implementation able to monitor and inspect network traffic in real time.

VI. Conclusions

Traditional traffic monitoring processes, such as intrusion detection and prevention tools, use deep packet inspection techniques that often focus on examining the packet payload contents to report certain activities, resulting to poor adaptiveness for encrypted network traffic. In this work, we propose an fast signature-based intrusion detection engine tailored for encrypted network traffic. The signatures are generated using only network packet metadata; an approach tailored for encrypted networks. As part of our future work, we plan to include other network packet metadata in the signature generation phase (besides packet payload sizes and direction) and investigate how they could reveal the encrypted network nature even in traffic analysis resistant implementations.

Acknowledgements

This work was supported by the projects CONCORDIA, Cyber-SANE, I-BIDAAS and SPIDER, funded by the European Commission under Grant Agreements No. 830927, No. 833683, No. 780787 and No. 833685. This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which maybe made of the information contained therein. This research work was also supported by the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Technology (GSRT), under the HFRI PhD Fellowship grant (GA. No. 2767).

References

[1] "Managing Encrypted Traffic with Symantec Solutions," https://www.symantec.com/content/dam/symantec/docs/solution-briefs/ssl-visibility-en.pdf, Accessed: 2019-07-11.

[2] "The rapid growth of SSL encryption:The Dark Side of SSL That Today's Enterprise Can't Ignore," https://www.fortinet.com/content/dam/fortinet/assets/white-papers/WP-The-Rapid-Growth-Of-SSL-Encryption.pdf, Accessed: 2019-07-11.

[3] 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.

[4] 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.

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

[6] G. Vasiliadis, L. Koromilas, M. Polychronakis, and S. Ioannidis, "{GASPP}: A gpu-accelerated stateful packet processing framework," in *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, 2014, pp. 321–332.

[7] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "MIDeA: A Multi-Parallel Intrusion Detection Architecture," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, 2011.

[8] 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.

[9] Y. Go, M. A. Jamshed, Y. Moon, C. Hwang, and K. Park, "Apunet: Revitalizing {GPU} as packet processing accelerator," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 83–96.

[10] G. Giakoumakis, E. Papadogiannaki, G. Vasiliadis, and S. Ioannidis, "Pythia: Scheduling of concurrent network packet processing applications on heterogeneous devices," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 145–149.

[11] "The UNSW-NB15 Dataset," https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/, Accessed: 2019-06-11.

[12] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 military communications and information systems conference (MilCIS)*. IEEE, 2015, pp. 1–6.

[13] 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.

[14] "The Snort IDS/IPS," Available: https://www.snort.org/, Accessed on Jul. 8, 2019.

[15] "Suricata Open Source IDS / IPS / NSM engine ," Available: https://www.suricata-ids.org/, Accessed on Jul. 8, 2019.

[16] 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.

[17] 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.

[18] G. Vasiliadis, M. Polychronakis, S. Antonatos, E. P. Markatos, and S. Ioannidis, "Regular Expression Matching on Graphics Hardware for Intrusion Detection," in *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, 2009.

[19] V. Paxson, R. Sommer, and N. Weaver, "An architecture for exploiting multi-core processors to parallelize network intrusion prevention," in *2007 IEEE Sarnoff Symposium*. IEEE, 2007, pp. 1–7.

[20] M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney, "The nids cluster: Scalable, stateful network intrusion detection on commodity hardware," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2007, pp. 107–126.

[21] I. Sourdis and D. Pnevmatikatos, "Pre-decoded cams for efficient and high-speed nids pattern matching," in *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2004, pp. 258–267.

[22] C. R. Meiners, J. Patel, E. Norige, E. Torng, and A. X. Liu, "Fast regular expression matching using small tcams for network intrusion detection and prevention systems," in *Proceedings of the 19th USENIX conference on Security*. USENIX Association, 2010, pp. 8–8.

[23] Z. Wang, "Deep learning-based intrusion detection with adversaries," *IEEE Access*, vol. 6, pp. 38 367–38 384, 2018.

[24] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, "Long short term memory recurrent neural network classifier for intrusion detection," in *2016 International Conference on Platform Technology and Service (PlatCon)*. IEEE, 2016, pp. 1–5.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.