CYBERSANE

D3.2

Encrypted Network Traffic Analysis, Transformation and Normalization Techniques

Project number:	833683
Project acronym:	CyberSANE
Project title:	Cyber Security Incident Handling, Warning and Response System for the European Critical Infrastructures
Start date of the project:	1st September, 2019
Duration:	36 months
Programme:	H2020-SU-ICT-2018

Deliverable type:	Report
Deliverable reference number:	D3.2
Work package contributing to the deliverable:	WP3
Due date:	31 12 2020 – M16
Actual submission date:	31/01/2021, 03/08/2021

Responsible organisation:	ATOS
Editor:	José Francisco Ruiz
Dissemination level:	CO
Revision:	2

Abstract:	The present document covers an analysis of the encryption traffic as well as a study of some of the most common transformation and normalization techniques employed nowadays
Keywords:	Encrypted traffic, signatures, data normalization, data transformation, Elastic Common Schema
	The project CyberSANE has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 833683.

Editor

Jose Francisco Ruiz (ATOS)

Contributors (ordered according to beneficiary numbers)

Sergio Zamarripa López (S2)

Spyros PAPASTERGIOU, NicolaTAMBURINI, Andrea PRUCCOLI (MAG)

Eva Papadogiannaki, Manos Athanatos (FORTH)

Konstantinos Kontakis, Sofia Spanoudaki, Georgia Koutsouri, Marianna Manou Kaklamani (STS)

Saoulidis Harris (SID)

Revision History

Rev.	Date	Authors	Comment
1.2.1	03/12/20	José F. Ruiz, José Javier de Vicente (ATOS) Sergio Zamarripa López (S2) Eva Papadogiannaki, Manos Athanatos (FORTH) Konstantinos Kontakis, Sofia Spanoudaki, Georgia Koutsouri, Marianna Manou Kaklamani (STS) Saoulidis Harris (SID)	Final version to be submitted
2	03/08/21	José Javier de Vicente (ATOS), Eva Papadogiannaki (FORTH)	References have been reviewed: now the text has a direct link to the citation. This applies specially to pages 15,16 but also to pages 8, 9 (Section 1.1 and Section 1.2), page 10 (Section 1.2), page 13 (Section 1.3), pages 18, 19 (Sections 1.4.2 and 1.5), pages 22 and 23 (Section 1.5), pages 25 and 26 (Section 1.6). Section 1.3 has been updated to provide more details about the training of the datataset, more specifically, changes were made to section 1.3.2 to provide details on how the signatures have been generated (Pages 13 and 14) and updated references (Page 13). References of this section have also been reviewed (Pages 11 and 13). Section 1.4.3 has been enhanced and updated to clarify how the dataset is evaluated (Pages 17 and 18). Added section 1.6.3 Future work (Page 26).

Disclaimer

Executive Summary

Nowadays cybersecurity entails big importance in every aspect of our digital lives. Attacks are becoming increasingly sophisticated while the cost of keeping assets safe is raising since new, improved protection measures are required.

One way to deal with cyber threats is by utilizing encryption mechanisms. Encryption is widely used today to maintain information safe from unauthorized access but also protects data while in transit. In this document several encryption mechanisms are going to be described; thus, revealing the most relevant applications of encryption as far as network traffic is involved.

Apart from that, the document deepens into security incidents data transformation and normalization techniques. Currently, cybersecurity analysts rely on different solutions and tools provided by heterogeneous vendors with reference to security logging, monitoring and detection. The presence of distinct solutions in the field of cybersecurity makes the data correlation a necessity in order to make the most of it. However, proper correlation can only be achieved through the normalization of collected data. In other words, log transformation and normalization become essential when dealing with different log manufacturers. Otherwise, cybersecurity analyst won't be able to link information from various sources. Normalization is the way to make counter intelligence sharper, more intelligent and effective so, its importance cannot be praised enough in terms of saving both time and money.

The document describes several security incidents data transformation and normalization techniques. In addition, it shows how different solutions gather information, what data is collected in each case and presents an example of how data is normalized to the proposed output format.

Contents

List of Figures

List of Tables

Chapter 1 Encrypted Network Traffic Analysis

1.1 Introduction

The adoption of network encryption is rapidly growing. The 2019 Annual Report of Let's Encrypt[1] states that in just four years, global HTTPS page loads have increased from 39% to more than 80%[2]. In 2019, one year after TLS 1.3 been published as an RFC[3], IETF reports that its adoption is rapidly growing with a 30% of Chrome's Internet connections to negotiate TLS 1.3[4]. Even though network encryption is crucial for the protection of users and their privacy, it naturally introduces challenges for tools and mechanisms that perform deep packet inspection and rely heavily on the processing of packet payloads. Common applications of deep packet inspection are packet forwarding (Rizzo, Carbone and Catalli) and l7 filtering[5], while it is a vital operation in firewalls, intrusion detection and prevention systems[6][7][8]. Typical intrusion detection systems, such as Snort, inspect packet headers and payloads to report malicious or abnormal traffic behaviour. In encrypted packets[9] though, the only information that makes sense is (i) TLS handshake packets and (ii) TCP/IP packet headers, since 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"[10].

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. For example, Broadcom's Encrypted Traffic Management solution intercepts encrypted traffic to gain and offer visibility[11][12]. At another end, CISCO's Encrypted Traffic Analytics solution performs network analytics and machine learning to gain insight into threats in encrypted traffic without requiring decryption[13]. In fact, recently, machine learning techniques are widely used for traffic classification, network analytics and malware detection (Anderson and McGrew) (Lotfollahi, Siavoshani and Zade) (Rosner, Kadron and Bang). Others focus on the implementation of real-time traffic identification systems for encrypted networks (Papadogiannaki, Halevidis and Akritidis). 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 (M. Conti, L. V. Mancini and R. Spolaor) (Papadogiannaki, Halevidis and Akritidis). Packet headers contain information like IP addresses, port numbers and packet data sizes. Time-related features, such as flow duration and packet inter-arrival times, are also relevant in encrypted traffic analysis and can be easily computed. When

[1] https://letsencrypt.org
[2] https://www.abetterinternet.org/documents/2019-ISRG-Annual-Report-Desktop.pdf
[3] https://tools.ietf.org/html/rfc8446
[4] https://www.ietf.org/blog/tls13-adoption/
[5] http://l7-filter.sourceforge.net
[6] https://suricata-ids.org
[7] https://www.snort.org
[8] https://zeek.org
[9] With encrypted packets, we refer to TCP packets that are secured using the TLS protocol.
[10] https://www.snort.org/faq/readme-ssl
[11] https://www.broadcom.com/products/cyber-security/network/encrypted-traffic-management
[12] https://docs.broadcom.com/doc/the-importance-of-broad-cipher-suite-support
[13] https://www.cisco.com/c/en/us/solutions/enterprise-networks/enterprise-network-security/eta.html

properly combined, all these packet metadata, can offer valuable traffic insights (Anderson and McGrew).

In the context of CyberSANE and Task 3.3. “Encrypted Network Traffic Analysis”, FORTH proposes a methodology to generate signatures for intrusion detection using only packet metadata extracted from TCP/IP packet headers. The resulted signatures enable intrusion detection even in encrypted network traffic.

1.2 State-of-the-Art in Encrypted Traffic Analysis for Network Security

Popular network intrusion detection systems (NIDS) like Snort[14] and Suricata[15] utilize pattern matching and regular expressions matching algorithms to analyse network traffic. With the ever-increasing network speeds, the research community has put effort in improving the performance of NIDS using either commodity accelerators, such as GPUs (Vasiliadis, Antonatos and Polychronakis) (Vasiliadis, Polychronakis and Ioannidis, MIDeA: A Multi-Parallel Intrusion Detection Architecture)and parallel nodes (Paxson, Sommer and Weaver) (Vallentin, Sommer and Lee) or specialized hardware, such as TCAMs, ASICs and FPGAs (Meiners, Patel and Norige) (Sourdis and Pnevmatikatos). However, the majority of these works are based on methods that extract content from network packet payloads to match suspicious signatures. Traditional deep packet inspection is becoming insufficient for encrypted network traffic (e.g., SSL/TLS protocols).

BlindBox (Sherry, Lan and Popa) performs deep-packet inspection directly on the encrypted traffic, utilizing a new protocol and new encryption schemes. PrivDPI (Ning, Poh and Loh) reduces the setup delay of BlindBox and retains similar privacy guarantees. (Shone, Ngoc and Phai) propose a system that combines deep learning techniques to provide intrusion detection. (Tang, Mhamdi and McLernon) present a deep learning approach for flow-based anomaly detection in SDN environments, while (Niyaz, Sun and Javaid) utilize deep learning in order to detect DDoS attacks in such environments. (Anderson and McGrew) compare the properties of six different machine learning algorithms for encrypted malware traffic classification. Moreover, (Amoli, Hamalainen and David) present a real-time unsupervised NIDS, able to detect new and complex attacks within encrypted and plaintext communications. Kitsune is a NIDS, based on neural networks, and designed for the detection of abnormal patterns in network traffic (Mirsky, Doitshman and Elovici). It monitors the statistical patterns of recent network traffic and detects anomalous patterns. Moreover, (Rosner, Kadron and Bang) present a black-box approach for detecting and quantifying side-channel information leaks in TLS-encrypted network traffic. These techniques identify malicious events in the network, by examining the characteristics of the underlying traffic, using exclusively machine learning approaches. Many research and commercial solutions focus on inspection of encrypted network traffic mostly for network analytics (M. Conti, L. V. Mancini and R. Spolaor) (Lotfollahi, Siavoshani and Zade) (Taylor, Spolaor and Conti). OTTer (Papadogiannaki, Halevidis and Akritidis) is a scalable engine that identifies fine-grained user actions in OTT mobile applications even in encrypted network traffic. (Orsolic, Pevec and Suznjevic) use machine learning for the estimation of YouTube Quality of Experience (QoE). To test their approach, authors collect more than 1k different YouTube video traces under different bandwidth scenarios. (Mazha and Shafiq) investigate the Quality of Service (QoS) of video in HTTPS and QUIC protocols. The set of features that expose

[14] https://www.snort.org

[15] https://suricata-ids.org

usable information is based on (i) network and transport layer header information for TCP flows, and (ii) network layer features (based on inter-arrival time, packet sizes, packet/byte counts, throughput) for QUIC flows. CSI (Xu, Sen and Mao) infers mobile ABR video adaptation behaviour under HTTPS and QUIC using packet size and timing information. Finally, (Khokhar, Ehlinger and Barakat) put YouTube under experimentation and perform network traffic measurements for QoE estimation using network related features, as well. (Ghiëtte, Griffioen and Doerr) demonstrate that it is possible to utilize cipher suites and SSH version strings to generate unique fingerprints for bruteforcing tools used by an attacker.

Network middleboxes or client-side software that aim to inspect encrypted traffic can operate by acting as proxies. The common procedure is to terminate and decrypt the client-initiated TLS session, analyse the HTTP plaintext content, and then initiate a new TLS connection to the destination. (Goh, Zimmermann and Looi, Intrusion detection system for encrypted networks using secret-sharing schemes.) (Goh, Zimmermann and MarkLooi, Experimenting with an intrusion detection system for encrypted networks.) propose mirroring the traffic to a central intrusion detection system, which will be able to decrypt the traffic and perform deep packet inspection, yet, without any privacy preserving guarantees. As Symantec states "most cyber threats hide in SSL/TLS encryption" (which takes up to 70% of all network traffic)[16]. Symantec Proxies and SSL Visibility Appliance decrypt traffic to support infrastructure security and protect data privacy. More specifically, Symantec offers the Encrypted Traffic Management (ETM) tool[17] that provides visibility into encrypted traffic by decrypting part of it; however, this is a technique that could eventually cause privacy violations. Haystack enables network traffic inspection on Android mobile devices using a mobile application (namely "Lumen") that is distributable via the usual application stores. Haystack offers device-local and context-aware traffic inspection on commodity mobile devices. For full functionality even with encrypted network traffic, Haystack's application "Lumen" intercepts the encrypted network traffic via a local TLS proxy. The application prompts the user to install a self-signed Haystack CA certificate in the user CA certificate store at install time (Abbas Razaghpanah, Sundaresan and Kreibich).

Aiming to advance the state-of-the-art, FORTH proposes an automatic signature mining method for intrusion detection in encrypted network traffic. The majority of works that inspect encrypted network traffic exploits machine learning algorithms to examine the feasibility of identifying the nature of the traffic (e.g., for network analytics or network security). FORTH's methodology builds on these feasibility results, while at the same time focuses on establishing a procedure to effectively generate intrusion detection signatures in an automated manner.

1.3 Encrypted Traffic Signatures

After a careful examination of the literature and during our analysis, we observed that specific sequences of packet payload sizes can reveal discrete events that signify an intrusion attempt inside a system or a network. In this section, we describe our proposed signature language that is used to express such network traffic patterns.

First, we aim for an expressive but simple enough signature language to enable the automated signature mining. While the automatic generation of the signatures is an offline process, we aim also to support an efficient signature matching procedure at runtime on live

[16] https://docs.broadcom.com/doc/ssl-visibility-en
[17] https://www.broadcom.com/products/cyber-security/network/encrypted-traffic-management

network traffic. Also, we want to minimize the amount of state information that our intrusion detection engine requires to store per flow to effectively match traffic patterns across packets of the same flow. All the aforementioned requirements led us to build signatures using a simple format that can be applied on sequences of packet payload sizes. For the implementation of the intrusion detection engine, we will use an automaton, inspired by (Aho and Corasick). 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.

1.3.1 Signature Design and Representation

illustrates some signature examples that we extracted during our analysis. The proposed signature language uses a very simple format that is easy to follow. More specifically, when a network flow contains one or more sequences of network packet payload sizes which are combined with other network traffic characteristics (e.g., port numbers), an intrusion attempt event is reported. For instance, when (i) at least 7 different network flows, with the same source and destination IP addresses and the same destination port, contain a sequence of 4 packets with payload sizes 22, 976, 48, 16 bytes respectively, and (ii) the same network flows, contain a sequence of 4 packets with payload sizes 52, 68, 84, 84 bytes respectively, then our intrusion detection engine reports the existence of a password cracking attempt with the Hydra tool[18].

Intrusion attempt event	Penetration tool	Signature	Source ports
SSH password cracking	Hydra	22, 976, 48, 16	7
		52, 68, 84, 84	7
File/directory scanning	Dirbuster	608, 80	40
		155, 156	35
SQL injection	Sqlmap	194, 93	140

Table 1: Signature examples of intrusion attempts

Figure 1 shows how a sequence of packet payload sizes appears in time within a traffic capture. This figure illustrates a password cracking attempt with the “hydra” tool. We observe that the corresponding generated signatures from

[18] https://tools.kali.org/password-attacks/hydra

describe and express what Figure 1 shows. In detail, the signature for the password cracking intrusion attempt reports when at least 7 network flows (with same IP addresses and destination port and diversified by the source port) match the packet payload size sequences 22, 976, 48, 16 and 52, 68, 84, 84. The packet capture that is presented in Figure 1 contains 9 different network flows (with same IP addresses and destination port and diversified by the source port) and the two packet payload sequences: 22, 976, 48, 16 and 52, 68, 84, 84, respectively. Packet sequences in Figure 1 are presented in respect with the inter-packet arrival times in milliseconds. Each bullet colour represents one network flow (diversified by the source port):

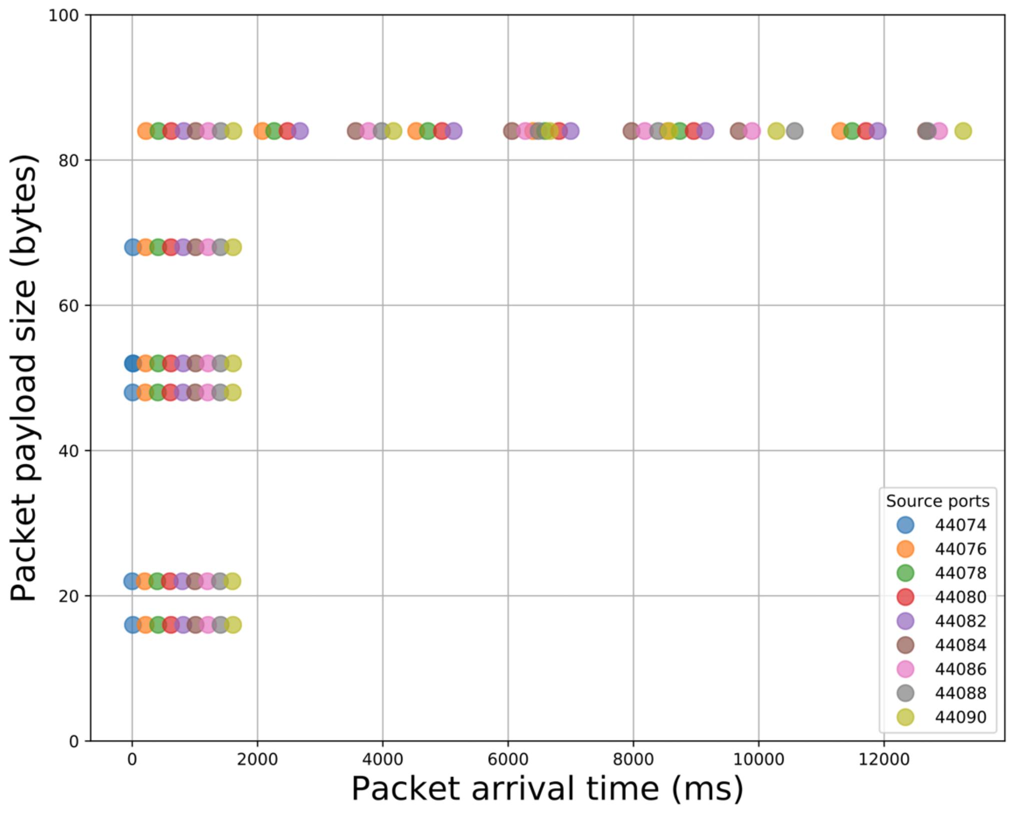


Figure 1 Illustration of (selected) packet payload size sequences within a traffic capture of a SSH password cracking attempt using the “hydra” tool.

1.3.2 *Signature Generation Methodology*

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 (Anderson and McGrew). In the paragraphs that follow, we present our methodology for the automatic signature generation. Also, Figure 2 illustrates the workflow of our methodology.

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, we discard retransmitted TCP packets, as well.

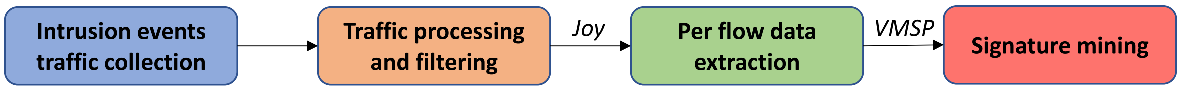

Intrusion events traffic collection
Traffic processing and filtering
Joy
Per flow data extraction
VMSP
Signature mining

Figure 2 Illustration of our methodology workflow.

Then, we use the joy tool[19] to extract per network flow data that are later used for the signature generation. More specifically, joy receives as input a packet capture with an intrusion event. Joy returns a JSON file with network flow related information, such as the sequence of lengths and arrival times of IP packets per network flow, DNS names, addresses, TTLs, HTTP header elements 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 time. This sequence of non-zero packet payload sizes is later used by the signature mining procedure. For the signature mining procedure, we chose to utilize a frequent sequential pattern mining technique.

Frequent sequential pattern mining techniques are used to 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 resulted sequences as potential signatures that can indicate an intrusion attempt. The resulted signatures are mined using the VMSP algorithm (Fournier-Viger, Wu and Gomariz). Finally, we select the maximal sequences that match to the ground truth information that we have. For instance, the time window of the intrusion attempt is close (in time) to the sequence's first occurrence inside the network traffic.

In detail, the process to generate the signatures, as presented in Figure 2, is the following:

1. For each packet capture, we note the label that characterizes the intrusion event.
2. We diversify the packet captures per intrusion attempt event.
3. We break each packet capture into 5-tuple network flows. Each network flow is hashed using the 5-tuple {source IP address, destination IP address, source port, destination port, protocol}. Each relevant network flow is now labelled with the corresponding intrusion event.
4. We parse each network flow with the joy tool, that exports the sequences of packet payload sizes contained. We keep only the non-zero TCP packets, as explained in Section 1.4.1.
5. We execute the VMSP algorithm to these sequences of packet payload sizes. The VMSP algorithm performs maximal sequential pattern mining and reports the longest sequences of packet payload sizes found.
6. We choose the most common longest sequences of packet payload sizes to build a signature that will describe the corresponding intrusion attempt event.

[19] https://github.com/cisco/joy

7. We repeat the process per intrusion event.

In Section 1.4.2, we explain in detail how we collect the ground-truth dataset that contains the packet captures that characterize a malicious activity. Each packet capture is labelled in the respecting packet capture. This ground-truth dataset is separated into two smaller sets randomly. The first dataset contains a 30% of the total packet captures and the second dataset contains the remaining 70%. The first dataset is used for analysis and signature generation (i.e., training dataset), while the second one is used for testing (i.e., testing dataset). In the literature, it is common to use the reverse proportions for analysis and testing (i.e., 30% for testing and 70% for training). Yet, we want to stress the effectiveness of our proposed methodology. In Section 1.4.3, we show that despite using a small dataset for signature generation, our methodology produces effective signatures. The evaluation of the signatures is performed in the testing dataset that contains the 70% of the packet captures.

As already discussed, we use the training dataset to create signatures that will describe a malicious activity. The malicious activities that our training dataset contains is presented in Table 2. Thus, the generated signatures describe the presented malicious activities (characterized by events 1-9). The evaluation of the signatures produced can be found in Section 1.4.3.

1.4 Signature Evaluation

In this section, we demonstrate the expressiveness of the proposed signature language by automatically generating pattern signatures for a set of intrusion events and evaluating their accuracy. We used 30% (randomly chosen) of the ground-truth packet captures as a reference for the signature generation, and the remaining 70% for the accuracy evaluation.

1.4.1 Traffic Processing

We divide the network traffic collected during the attacks into network flows. A network flow is characterized and represented by the standard 5-tuple that contains: (i) the source IP address, (ii) the source port number, (iii) the destination IP address, (iv) the destination port number and (v) the protocol (e.g., TCP). So, each network flow consists of packets that are defined by a certain 5-tuple.

Since the network traffic that we collect 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.

Furthermore, TCP natively provides numerous mechanisms to detect and bypass unpredictable network behaviour, including but not limited to packet loss and reordering methodologies. As already mentioned, 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.

1.4.2 Ground-truth Dataset Collection

For the ground-truth collection, 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[20] 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[21] is installed on the vulnerable machine and captures the incoming network traffic during the intrusion attempts performed by the malicious machine. Figure 3 illustrates the testbed setup.

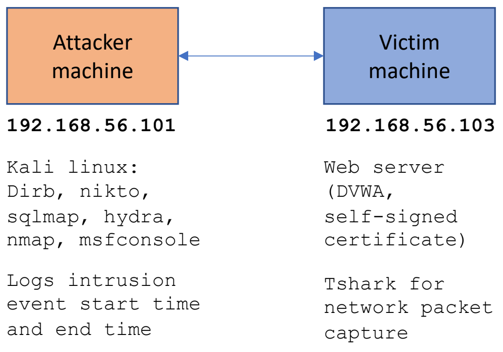

Attacker machine
Victim machine
192.168.56.101
192.168.56.103
Kali linux: Dirb, nikto, sqlmap, hydra, nmap, msfconsole
Logs intrusion event start time and end time
Web server (DVWA, self-signed certificate)
Tshark for network packet capture

Figure 3: Illustration of our testbed setup for traffic collection.

We choose some popular vulnerability scanners to evaluate our methodology. Some of the tools used for the intrusion events generation are DIRB[22], NIKTO[23], SQLMAP[24], HYDRA[25], NMAP[26], and METASPLOIT[27]. More specifically:

[20] http://www.dvwa.co.uk
[21] https://www.wireshark.org/docs/man-pages/tshark.html
[22] https://tools.kali.org/web-applications/dirb
[23] https://tools.kali.org/information-gathering/nikto
[24] http://sqlmap.org
[25] https://tools.kali.org/password-attacks/hydra
[26] https://nmap.org
[27] https://docs.rapid7.com/metasploit/

- DIRB is a web content scanner that looks for existing (and/or hidden) web objects. It basically works by launching a dictionary-based attack against a web server and analysing the response.
- NIKTO examines a web server to find potential problems and security vulnerabilities.
- SQLMAP is an open-source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers.
- HYDRA is a parallelized login cracker which supports numerous protocols to attack.
- NMAP is a free and open-source utility for network discovery and security auditing.
- METASPLOIT is a penetration testing software (we use the msfconsole, which is the metasploit framework console).

We perform numerous instances of attacks for different times and days within a one-month period. Table 2 presents the events generated.

Overall, we collected a set of over 120 packet captures. Each individual packet capture simulates an intrusion attempt as described in Table 2. For each packet capture, we log the start time and end time of each intrusion attempt event.

#	Tool name	Event
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

Table 2: Intrusion attempts to the vulnerable web server.

1.4.2.1 CyberSANE pilots

In the context of CyberSANE, FORTH will receive ground-truth data from data providers, such as LSE, to generate signatures for intrusion detection tailored to their systems. So far, FORTH has shared instructions and guidelines on how to generate abnormal traffic (e.g., using a penetration tool) in an isolated and protected environment and how to collect network traffic. In addition, FORTH has shared a list with the specific information that is required to effectively build signatures that will be used for intrusion detection in pilots' systems. The required information is the following:

- Infected and benign network traffic captures (suggested protocols: TCP, HTTPS, TLS)
- For each infected network traffic capture, the following annotations are required: (i) type/name of attack, (ii) attack start/end time, (iii) list of network flows involved in each attack

1.4.3 Signature Effectiveness

As already discussed in the previous section, we divide the packet traces by intrusion event. Then, we randomly select a 30% of each set to generate signatures, while the remaining 70% is used for evaluation.

Table 3 presents the resulting true positive and false discovery rates for each signature that corresponds to one of the events from Table 2. Each packet capture in the dataset contains only a single intrusion event type (event types: 1-9, Table 3). When a signature reports an intrusion event, we compare it to the actual intrusion event. For instance, when a signature reports that “web content scanning” probably occurs in a specific packet capture, we compare this report to the actual intrusion attempt 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.

Event #	True Positive Rate	False Discovery Rate (FDR)
1 (dirbuster)	100%	0%
2 (nikto)	100%	0%
3 (hydra)	100%	11%
4 (hydra)	100%	11%
5 (metasploit)	100%	11%
6 (metasploit)	100%	11%
7 (sqlmap)	100%	0%
8 (nmap)	100%	11%
9 (nmap)	100%	11%

Table 3: True Positive Rates and False Discovery Rates of the automated signature mining methodology. The percentages presented are extracted through the comparison of the results of our methodology to the ground-truth dataset.

The true positive rate 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 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%).

Besides the true positive rate, another metric that we believe is necessary for the evaluation of our methodology is the false discovery rate for each intrusion attempt event. Reporting intrusion attempt events using only the encrypted network traffic can easily become tricky, since the cross- validation is a challenging procedure. A network intrusion detection system must be able to report any traffic behaviour 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 3. False discovery rate is calculated as:

$$FDR = \frac{FP}{(TP + FP)}$$

where FP stands for False Positive and TP for True Positive. Our signatures generation methodology presents a maximum false discovery rate for some of the intrusion events.

Minimizing the false positives that an intrusion detection system presents is very important. Our signature generation methodology using the frequent pattern mining technique presents a perfect true positive rate, with an acceptable false discovery rate (up to 11%). At this point, we want to 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 though the granularity of the event is not fine-grained (the generated signature for the hydra tool cannot 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.

Finally, we use normal HTTPS traffic samples to measure the FDR for the signatures generated. The samples that we used for this experiment are publicly available in the Malware Capture Facility Project repository[28]. In this normal HTTPS traffic tested, the signatures that describe the events that previously reported an 11% FDR now result to 0% FDR.

1.5 Intrusion Detection Engine Implementation

A very efficient algorithm that popular signature-based intrusion detection systems use for pattern matching is the Aho-Corasick algorithm (Aho and Corasick). Pattern matching is the core operation of any deep packet inspection system, such as a network intrusion detection system. A deep packet inspection system dives into the network packet payloads in order to extract sequences of characters, namely strings. These strings are compared against well-known patterns that describe, for instance, the communication between a known botmaster with its bots. In our approach, we assume that the network traffic that should be inspected by our intrusion detection system contains encrypted payloads. Thus, we do not extract any payloads and we only process packet metadata. These packet metadata can be derived from the contents of network packet headers. For example, even in a TLS protected connection, the packet headers are not encrypted. As we have already mentioned, our methodology uses packet metadata like the packet payload sizes (i.e., data transmitted in the packet) and packet directions in order to generate signatures. We express the packet direction implicitly since a signature will match against one-directional network flows. A signature that we produce contains sequences of packet payload sizes. These sequences of packet payload sizes must be matched against the incoming network traffic in order to report an intrusion attempt event that is described by the corresponding signature. Yet, packet payload sizes are integers and cannot be expressed as strings. Thus, integrating signatures of packet metadata into a typical signature-based intrusion detection system that performs deep packet inspection in packet payloads, is not trivial. In the following paragraphs, we describe the implementation of our system.

1.5.1 *Efficient Automaton*

[28] https://www.stratosphereips.org/datasets-normal

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 (Aho and Corasick), we implement a finite state machine to efficiently match a set of patterns (i.e., signatures) against streams of network packets. We extend the Aho-Corasick algorithm to enable integer matching, instead of strings, similar to (Papadogiannaki, Deyannis and Ioannidis, Head (er) Hunter: Fast Intrusion Detection using Packet Metadata Signatures.) (Papadogiannaki, Halevidis and Akritidis).

The Aho-Corasick algorithm is a very efficient string searching algorithm that matches the items of a finite set of strings against an input stream. It is able to match a large volume of patterns simultaneously, so its complexity does not depend on the size of the pattern set. It constructs an automaton that performs transitions for each 8-bit ASCII character of the input text. For our approach, we replace the 8-bit characters with 16-bit values that represent the packet sizes. The algorithm builds a finite state machine, resembling a trie with added “failure” links between the trie nodes. When there is no remaining matching transition, we move through the state machine following the failure links, performing fast transitions to other branches of the trie that share a common prefix. In this way, we avoid the expensive back-tracking operation, so the algorithm allows the interleaving of a large number of concurrent searches, such as in the case of network connections, because the state can be preserved across input data that are observed at different points in time by storing a pointer to the current state of the automaton, with the state maintained for each connection. Backtracking is an operation very expensive since it requires the maintenance of 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.

To present our automaton’s characteristics, i.e., the automaton size and the compilation time, we generate signature sets out of varying packet sequences, each time increasing the number of signatures and the packet sequence length. Figure 4 presents the size of the automaton in regard to different signature sets. More specifically, we present the size of our automaton, using 500, 1K, 5K, 10K and 50k randomly generated patterns of sequence length 6, 8, 10 and 12 packets; for example, the automaton that is generated using 10,000 signatures, where each signature resembles a sequence of 10 packet sizes, is around 1.5 GB. Figure 5 presents the compilation time of the automaton based on the same signature sets. The compilation time of the automaton does not affect the end-to-end performance negatively, since the compilation happens offline and only once.

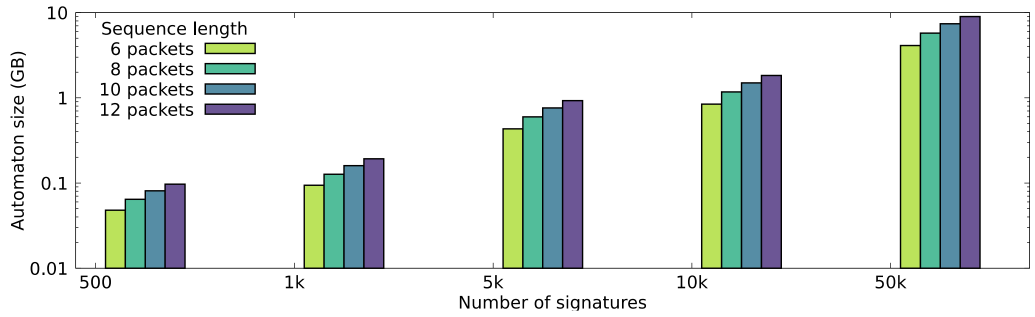


Figure 4: Automaton size.

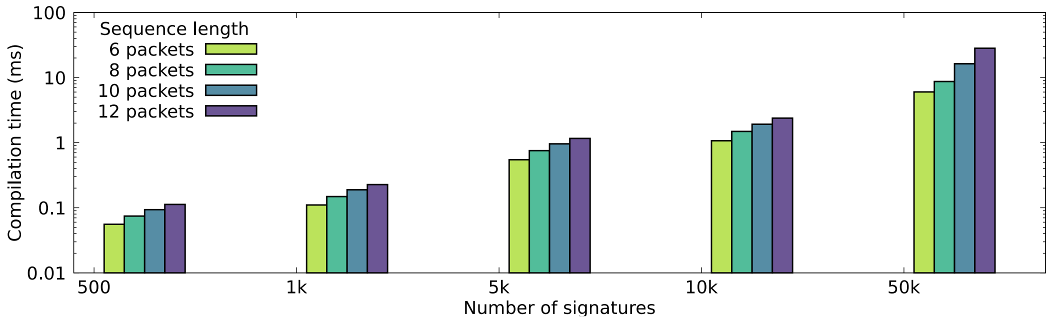


Figure 5: Automaton compilation time.

1.5.2 *Pattern Matching Engine*

To uniformly execute the pattern matching engine across every device in our testbed machine (i.e., the main processor Intel i7-8700K, a high-end discrete NVIDIA GTX 980 GPU and an Intel UHD Graphics 630 integrated GPU), we utilize the OpenCL framework. Our testbed system runs Arch Linux 4.19.34-1-lts and we use the Intel OpenCL 2.1 SDK for the Intel devices (i.e., the UHD Graphics 630 GPU and the Intel i7- 8700K CPU) and the OpenCL SDK from the NVIDIA CUDA Toolkit 10.2.120 for the NVIDIA GTX 980 GPU.

In OpenCL, an instance of a given code block and a thread that executes it is called work-item and a set of multiple work- items is called work-group. Different work-groups can run concurrently on different hardware cores. Typically, GPUs contain a significantly faster thread scheduler, thus it is recommended to spawn a large number of work-groups, since it hides the latency that is introduced by heavy memory transfers through the PCIe bus. While a group of threads waits for data consumption, another group can be scheduled for execution. On the other hand, CPUs perform more efficiently when the number of work-groups is close to the number of the available cores. When executing compute kernels on the discrete GPU, the first thing to consider is how to transfer the data to and from the device. Discrete, high-end GPUs have a dedicated memory space, physically independent from the main memory. To execute a task on the GPU, we must explicitly transfer the data between the host (i.e., DRAM) and the device (i.e., GPU DRAM). Data transfers are performed via DMA, so the host memory region should be page-locked to prevent any page swapping during the time that transfers take place. In OpenCL, a data buffer, which is required for the execution of a computing kernel, must be created and associated with a specific context. Different contexts cannot share data directly. Thus, we must explicitly copy the received network packets to a separate page-locked buffer that has been allocated from the context of the discrete GPU and can be moved towards its memory space via PCIe. Data transfers (host → device → host) and GPU execution are performed asynchronously, permitting a pipeline of computation and communication, something that significantly improves parallelism. Moreover, when the processing is performed on an integrated GPU, expensive data transfers are not required, since both devices have direct access to the host memory. To avoid redundant copies, we explicitly map the corresponding memory buffers between the CPU and the integrated GPU.

Figure 6 presents an illustration of the packet processing scheme in a hardware setup of a commodity machine that contains one main processor packed in the same die with an integrated GPU and one discrete high-end GPU. As previously explained, to process a network packet on a discrete GPU, the steps are the following:

i. the DMA transaction between the NIC and the main memory
ii. the transfer of the packets to the I/O bus that corresponds to the discrete GPU
iii. the DMA transaction to the memory space of the discrete GPU
iv. the execution of the OpenCL processing kernel and
v. the transfer of the results back to the host memory.

Due to the PCIe interconnect inability to quickly handle small data transfers, all data transfers are instructed to operate on large batches. The packet processing on an integrated GPU follows a shorter path, since the integrated GPU and CPU share the same physical memory space, which allows in-place data processing, resulting to lower execution latency.

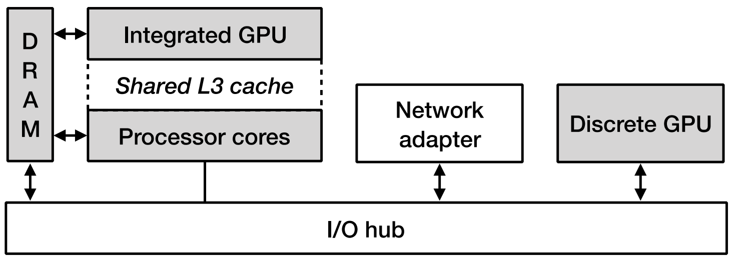


Figure 6: An illustration of the packet processing scheme in a hardware setup that contains one main processor packed in the same die with an integrated GPU and one discrete high-end GPU.

Memory accesses can be critical to the overall performance sustained by our application. GPUs execute code in a Single-Instruction-Multiple-Threads (SIMD) fashion, meaning that at each cycle multiple threads execute the same instruction. Moreover, they offer support for Single-Instruction-Multiple-Data (SIMD) execution when using vector data types (such as the ushort16 that is able to store 16 16-bit long values), since the vectorized code is translated to SIMD instructions (Shen, Fang and Sips). Furthermore, OpenCL offers the so-called local memory, which is a memory region that is shared between every work-item inside a work-group. This local memory is implemented as an on-chip memory on GPUs, which is much faster than the off-chip global memory. Hence, when we execute our engine on GPUs, we can utilize this local memory in order to improve the overall performance.

The overall architecture of our intrusion detection system is presented in Figure 7. The system utilizes one or several CPU worker-threads, assigning each one to a single input source (e.g., NIC).

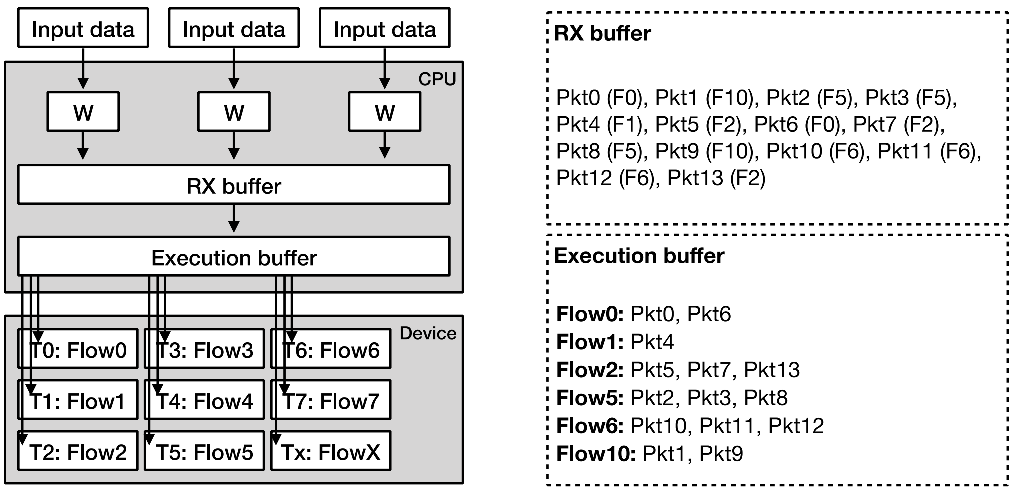


Figure 7: Overview of the packet processing architecture.

Once a CPU thread receives a network packet, it forwards it to a receive buffer, called RX batch (Figure 7). 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. In this way, we transform the input traffic to series of payloads with each series containing information of a single flow, ready to be processed by our pattern matching engine. Then, we transfer the execution batch to the device's memory address space. In the meantime, the receive buffer continues to accept 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 (Dobrescu, Egi and Argyraki) (Han, Jang and Park) (Vasiliadis, Polychronakis and Ioannidis, MIDeA: A Multi-Parallel Intrusion Detection Architecture) (Papadogiannaki, Koromilas and Vasiliadis), 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 (Figure 7). 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, 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.

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. Finally, in order to provide support for even further analysis, we keep a copy of the packet payload and metadata to the host's memory until their processing in the GPU has finished so their payloads can be examined in combination with the information provided by the engine.

1.5.3 Performance 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. The main processor is packed with an Intel UHD Graphics 630 integrated GPU. In our setup, we use Arch Linux with kernel version 4.19.34-1-lts. In addition, we use a NVIDIA GeForce GTX 980 GPU. During the micro-benchmarks, the pattern matching engine reads the traffic from memory. The performance results that are presented in Figure 8, Figure 9   Figure 10 display the median values occurring after 30 runs per configuration. In these figures, the colour-filled bars indicate the performance achieved by the pattern matching engine when the selection of (i) signatures and (ii) input, results to a computationally relaxed condition. In the figure, we present the most realistic scenario, where we have less than 10% malicious traffic. White-filled bars with borders indicate the performance achieved in a computationally loaded condition (i.e., 100% malicious traffic), which is the most worst-case scenario. We present the latency using different packet batch sizes. The discrete GPU introduces an almost stable latency across different batch sizes, close to 2ms. Executing on the integrated GPU results to higher latency records, up to 5ms. Executing on the main processor adds very low latency – especially for small batch sizes – making it ideal for real-time, latency-intolerant environments.

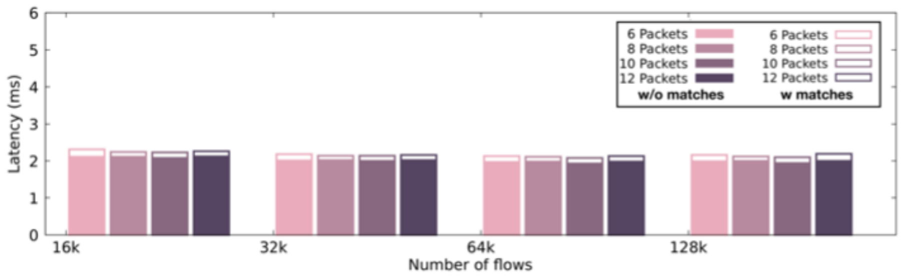


Figure 8: Latency of the pattern matching engine using the discrete GPU

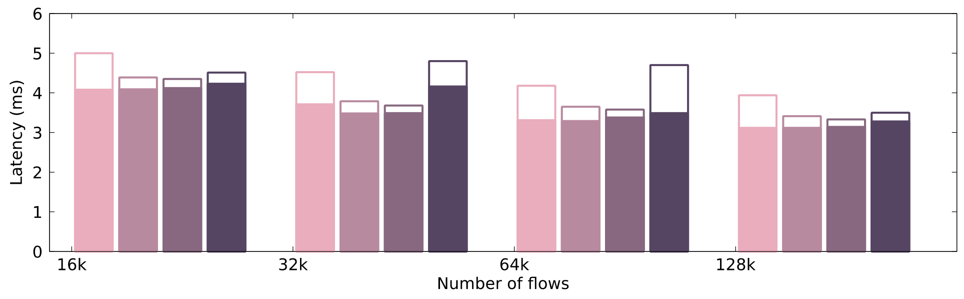


Figure 9: Latency of the pattern matching engine using the integrated GPU.

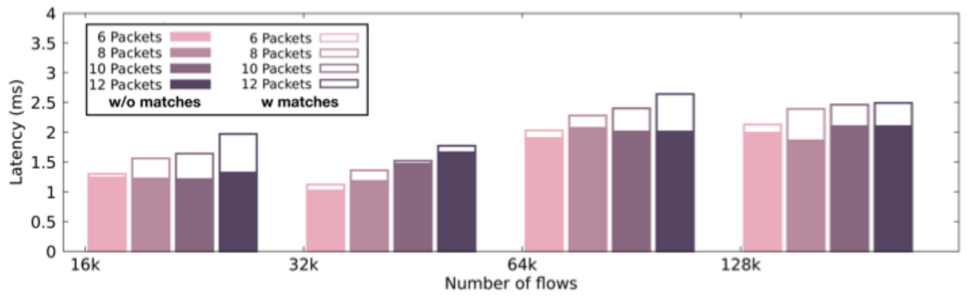


Figure 10: Latency of the pattern matching engine using the CPU.

1.6 Discussion

In this section, we discuss traffic analysis resistance techniques and we comment about the new version of TLS in respect to FORTH's proposed methodology.

1.6.1 Traffic Analysis Resistance

Features and characteristics of network traffic that present patterns after encryption (e.g., packet sizes and timing), can reveal information about the traffic's nature and contents. Padding packet sizes or transmitting packets at fixed timing intervals can obfuscate the behaviour of a communication mean for the preservation of privacy and the reduction of user information leakage. AnonRep (Zhai, Wolinsky and Chen) builds on top of anonymity and privacy guarantees for the case of reputation and voting systems. TARANET (Chen, Asoni and Perrig) employs packet mixing and splitting to achieve constant-rate transmission, providing anonymity at the network layer. (Frolov and Wustrow.) propose uTLS that enables tool maintainers to automatically mimic other popular TLS implementations to prevent censorship that originate from traffic analysis. Walkie-Talkie is a website fingerprinting defense approach that produces burst packet sequences that leak less information to the adversary. This makes sensitive and non-sensitive pages look the same (Wang and Goldberg). Vuvuzela (Hooff, Lazar and Zaharia) and Atom (Kwon, Corrigan-Gibbs and Devadas) are scalable systems that employ differential privacy to inject noise into observable metadata. Such techniques can circumvent the proposed methodology. Still, techniques like traffic morphing add substantial overhead to a system, making it impractical for cases, in which the privacy preservation is not a requirement.

1.6.2 TLS 1.3

Expecting the imminent adoption of TLS 1.3, we choose not to perform TLS certificate fingerprinting, like other relevant solutions[29]. The TLS 1.3 handshake is quite different from earlier versions of TLS, with a large portion of it getting encrypted (including certificates) (Kotzias, Razaghpanah and Amann). 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 and the inter-packet arrival time.

1.6.3 Future work

Signature generation is a time-consuming procedure that requires a huge amount of ground-truth data, constantly updated to keep up with modern attacks and maintained for different versions of software and operating systems. This work aims to offer contemporary signatures that will enable intrusion detection in encrypted networks and enrich the functionality of outdated, traditional intrusion detection tools that struggle to keep up with the increasing growth of network encryption in communication channels. During the life of the CyberSANE project, FORTH will keep on collecting data to produce more signatures, not only for penetration tools but also for malware that exist in the wild. The goal of this work is the generation of signatures that will enable intrusion detection in encrypted network packets.

[29] https://www.cisco.com/c/en/us/solutions/enterprise-networks/enterprise-network-security/eta.html

Chapter 2 Transformation and Normalization Techniques

2.1 Introduction

Nowadays cyber threats are becoming more and more sophisticated. Attackers have a wide variety of resources and information at their disposal and even, sometimes, time and money are no longer a problem. All these factors outline the importance of trusting security incidents data, which should be reliable, structured and easy to understand. Besides, organizations need to share information quickly and efficiently, given that incidents should be remediated as soon as possible. Here is where transformation and normalization techniques can make a difference: they assist analysts in various ways, such as:

- Allowing an easy share of incident data.
- Standardizing information gathered to make it more readable for humans.
- Formatting and structuring the information so it is more recognizable while discarding what is not needed.

To sum up, these techniques allow for a quicker and more coordinated answer when facing information security incidents.

2.2 State-of-the-art: transformation and normalization techniques

2.2.1 CEF

CEF or Common Event Format is known to be an auditing and logging file format developed by ArcSight. This standard is well suited to normalize output for log generating applications and devices while offering most important information. Its objective is to improve interoperability between security and network applications and devices. Therefore, data can be gathered and correlated with no effort. Amongst its main features are:

- Text-based format.
- Extensible.
- Possibility of supporting multiple device types.
- It does not determine ID for events generated by devices. Actually, devices and applications are responsible of this task.

The syntax for log records is made up of a standard prefix or header and a variable extension, while grouped in key-value pairs. In detail:

Prefix contains both date and hostname:

```
Jan 18 11:07:53 zurich message
```

Then, the variable extension of the message should include various fields separated by a pipe (“|”) with the following structure:

```
CEF:Version|Device Vendor|Device Product|Device Version|Signature
                    ID|Name|Severity|Extension
```

Where:

- Version (integer) or identifier of the CEF format. It can be used to better understand what the structure will be, i.e., what are the fields included in the register.
- Device vendor, product and version or string that helps identifying which is the device responsible for sending the report.
- Signature ID or identifier for each event. There cannot exist two equal identifiers. This field is quite important given that provides additional information about the type of event reported.
- Name: this field is a string which provides a description of what the event consists of.
- Severity field (integer) provides information about how critical an event is. It is evaluated in a scale which varies from 0 (least significant event) to 10 (more critical event).
- Extension: the field is a collection of a variable number of key and value pairs.

CEF format can be implemented on various types of devices:

- Cloud: in this case, the provider must implement the SmartConnector for ArcSight Common Event Format REST.
- On-premise: the device is required to implement the ArcSight Syslog SmartConnector.

Finally, CEF must be encoded in UTF-8 format. This means that:

- Spaces are valid.
- Any kind of pipe (|) used must be escaped with a backslash. This is to be done only in the header but not in the extension.
- Other symbols that should be escaped are backlash (\), equal sign (=) and multi-line symbols such as (\n) or (\r).

A real example of CEF would be the following:

```
Dec 18 20:37:08 <local0.info> 10.217.31.247 CEF:0|Citrix|NetScaler|NS10.0|APPFW|
APPFW_STARTURL|6|src=10.217.253.78 spt=53743 method=GET request=http://vpx247.ex
ample.net/FFC/login.html msg=Disallow Illegal URL. cn1=233 cn2=205 cs1=profile1 c
s2=PPE0 cs3=AjSZM26h2M+xL809pON6C8joebUA000 cs4=ALERT cs5=2012 act=blocked
```

2.2.2 STIX

STIX or Structured Threat Information eXpression is a collaborative, standard and structured language employed to represent and share cyber threat information including both incidents and threats.

Nowadays, any organization is expected to keep and update information about threats, that is, cyber threat intelligence. This usually includes different past attacks and vulnerabilities, actions that could potentially lead to recognize these attacks as well as mitigation strategies. Therefore, any company can make use of STIX to describe and store cyber threat intelligence information. STIX can assist organizations when sharing cyber threat intelligence with providers, partners or associates, resulting in a stronger net of information, more structured and organized. At the same time, STIX is currently used by analyst to recognize patterns that could potentially indicate threats.

STIX provides a way to structure cyber threat information and helps improving interoperability. Typical elements of the architecture joined by STIX include the following:

- Incidents or occurrences of malicious actions.
- Adversary TTPs (Tactics, Techniques and Procedures). Some examples of these would be malware, exploits or attack patterns.
- Cyber Threat actors, campaigns and initiatives.
- Indicators.

According to MITRE, all these elements are tied together with STIX architecture:

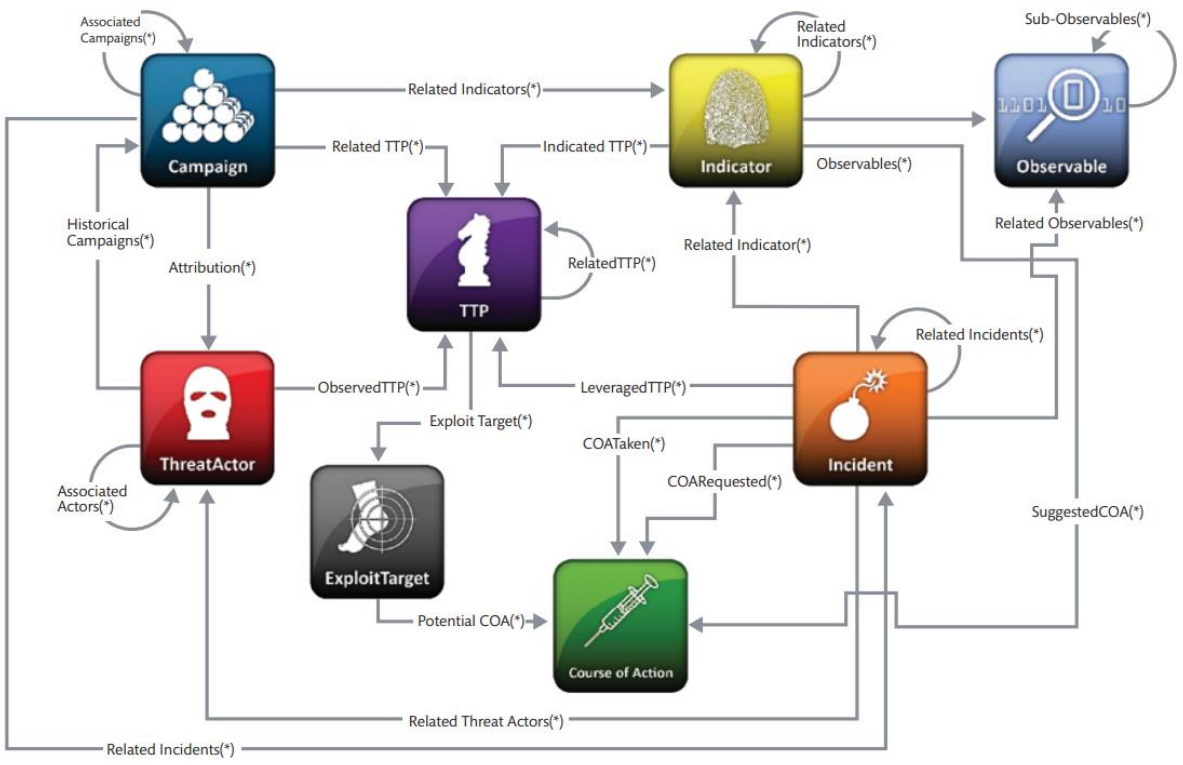


Figure 11: How STIX can relate cyber threat information (MITRE)

There are two main modes when using STIX:

1. Manually: this mode requires nothing more than an XML editor.

2. Programmatic: a bit more complex, it will be necessary to employ Python and Java bindings as well as some Python APIs.

STIX can be labelled as transport-agnostic, i.e., neither structure nor serialization depend on a specific transport mechanism. When willing to share STIX objects, it is quite useful to use TAXII platform (Trusted Automated Exchange of Intelligence Information). TAXII has been tailored to transport STIX objects specifically.

STIX information model employs various types of data, some of them are:

- Boolean (true/false)
- External-reference (reference to an external content)
- Identifier (for a STIX Domain Object)
- String
- Timestamp
- List (sequence of values ordered)

2.2.3 IODEF

IODEF (Incident Object Description Exchange Format) defines a format employed to share computer security information amongst the actors involved, typically CSIRTs. Messages are presented in a human-readable way, i.e., not in machine format. It makes use of XML language and encodes information about networks, hosts and services running on systems. XML helps when defining a framework for data encoding due to its extensibility: various character encodings can be defined easily. IODEF aim is to improve communication between CSIRTs by means of sharing structured information about incidents. Therefore, some of its advantages can be summed up as:

1. Less resources are required to process incident data.
2. Less effort is needed to normalize security information.
3. A common format is provided to share information and incident handling.

It is necessary to bear in mind that:

- IODEF has been designed and was created as a transport model, i.e., it is not the best way to store data.
- Incidents can be defined in many ways. Although IODEF is credited to be flexible enough, it does not impose one strict incident format to be adopted. Incidents are quite different from each other so there would be useless to try to homogenise them. Flexibility to describe different kinds of incidents is vital.
- IODEF is compatible with IDMEF (Intrusion Detection Message Exchange Format), which has been developed for Intrusion Detection Systems (IDSs).

IODEF is composed of the following fields:

- IncidentID or incident identification number.
- AlternativeID: this ID should be used by other CSIRTs rather than the one which defined and labelled the incident.
- RelatedActivity: this field should include all IDs from other incidents related.
- DetectTime, that is, when was the incident first detected.
- StartTime and EndTime: both fields store information of when did the incident start and end.
- ReportTime: similar to the previous one, but this time when was the incident reported.

- Description of the event.
- Assessment (of the event).
- Method: this field should include all those techniques employed by the attacker.
- Contact information of any group which may be engaged in the incident.
- EventData: the aim of this field is to store a description of the events tied to the incident.
- History or log of the actions, events and noticeable things that happened while the incident was being managed.
- AdditionalData.

2.2.4 *Other options to normalize data*

The most commonly used, maybe, transformation technique to normalize values on a similar scale is the **min-max normalization**. It is preferred in occasions where the approximate upper and lower bounds of the data come with no -or only a few- outliers which are also distributed uniformly. This type of normalization techniques are also known as “scaling to a range” (Google Developers, 2020), since it involves the conversion of natural range values into a standard, fixed, and easily readable range of values (i.e., between 0 and 1). Regarding the cyber-security domain, prominent min-max normalizations have found application on transforming the input data of unsupervised deep learning approaches for network intrusion detection purposes (Alom & Taha, 2017), but also contributing to the identification and aggregation capabilities of Dark Web analysis tools like the BiSAL (Al-Rowaily, et al., 2015). One more popular variation of scaling lies to the statistical **z-score transformation** which is able to identify how a value varies from the mean in terms of standard deviations. This normalization technique is useful **for datasets with a few only outliers**, where a couple of works over the last few years have either integrated or implemented promising and novel techniques, respectively. (Gashteroodkhani, et al., 2019) presented a z-score time-time matrix solution for the protection of microgrids, while (Meira, et al., 2020) proposed the combination of z-score and min-max transformations as the medium to evaluate the performance of unsupervised techniques in cyber-attack anomaly detection.

Apart from the min-max and z-score normalization techniques, the **logarithmic transformations** are another area of normalizations which can be used to narrow down a wide range of dataset values by computing their log. This type of normalization is quite efficient in those cases where the data which have to be normalized are influenced by many independent factors and there are a few only values with many points. Due to the fact that the log scaling procedure greatly changes the distribution itself, all these data distributions are also known as power law distributions. The outcome of logarithmic transformations has been proved to improve the linear data transformation techniques met in zero-day attacks tools once they are combined with the appropriate anomaly detection techniques (Aleroud & Karabatis, 2013). Furthermore, this kind of transformations could be also used to normalize the events and alerts generated by the probes of specific Critical Infrastructures (Di Sarno, et al., 2016), enhancing in this way their de-facto security information and event management systems. On the other hand, there are cases where all values of a dataset, or the values coming from different tools, could be benefited by normalizing them using a **square-root transformation**. Typical use cases are met in variables engaged with Poison distributions (Bartlett, 1936; Freeman & Tukey, 1950) where several enhancements have been presented aiming to extend square-root applicability to more statistics data areas (Almeida, et al., 2000; Molina, et al., 2017). At this point, it is worth noticing that values with negative numbers have to be specially treated by adding a constant variable, sufficient enough to move the minimum possible value of the distribution above zero.

However, if the provided **data set includes extreme outliers**, **then** all of the aforementioned normalization techniques tend to suffer both in terms of performance and training stability. For these reasons, the application of a **feature clipping normalization technique** is usually preferred and conducted either before or after other normalization procedures. In generic, a feature clipping takes into account all the feature values belonging above or below a certain value, and reallocates them to a predefined and fixed value. Feature normalizations have been used before as the medium to predict cyber-security incidents of network-level malicious activities (Liu, et al., 2015), while an even recent study (Ferreira, et al., 2019) explored feature normalization techniques in the context of a ML-based insider threat detection tool, aiming to evaluate and improve its performance on different classifiers. Last but not least, (Box & Cox, 1964) presented another family of power transformations which were able to detect the optimal normalizing transformation for different variables by simply raising numbers to an exponent. **Box-Cox normalization techniques** have been widely adopted and proved their efficacy in various domains of interest as an alternative methodology in data cleansing (Osborne, 2010). On such occasions the quantitative analysis of data was quite inferior and impractical when it was undertaken by the most commonly used transformation techniques like the min-max, z-score, and square-root (Osborne, 2002).

2.3 Overview of Security Monitoring Devices

2.3.1 Devices

2.3.1.1 SiVi (SID)

SiVi is a human-interactive visual-based anomaly detection system that is capable of monitoring and promptly detecting several devastating forms of security attacks. The tool's novelty lies on the development of intuitively visualization graphs capable to offer a quick, reliable, and intuitively overview in the network. In comparison with other tools that offer a simple presentation of the traffic inside the network, SiVi uses pre-trained neural networks that can identify different cyber-attacks.

SiVi implements a series of data visualization techniques, including both standard visualization methods (graph lines, bars, columns, etc.) and advanced visualization graphs (activity gauge, dependency wheels, etc.) aiming at providing the administrator with a complete anomaly detection ecosystem. Tables with detailed information regarding the network status also offer a thorough status of the system. SiVi also implements a series of Machine Learning (ML) algorithms, realizing both supervised and unsupervised techniques in order to create security events and timely inform the CCI operator for security attacks with devastating results. The ML algorithms are periodically updated with new attack taxonomies offering a constantly growing layer of protection. SiVi constantly monitors the network, capturing and analysing the transmitted packets while seeking for inconsistencies and anomalies at the tactical and the operational layer of the CII environment. More specifically, SiVi combines two different functionalities that are described below.

1. **The first functionality** is the Security monitoring and analysis mechanism (an IDS tool) which means it combines different sensors in different layers. In the network layer, SiVi uses the Suricata as a sensor, and several custom Machine learning (ML) sensors for the monitor and analysis of different communication protocols, while in the host layer it leverages the functionalities of the OSSEC Server. Thus, it collects security logs from all these sensors and transforms them into security event (by

mapping them in a unified format, Appendix 1). Figure 12 presents a high-level architecture describing this process.

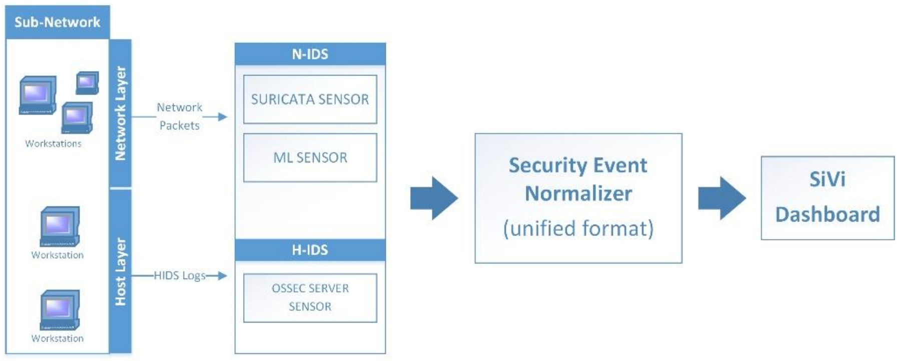


Figure 12: SiVi Security monitoring and analysis mechanism

In the network level the sensors (Suricata and ML sensors) receive network packets that are converted into network flows as an input. These flows are analysed and eventually each sensor results a security log to inform the tool operator. In the case of host layer sensors, the given input refers to logs coming from the hosts (e.g. syslog, eventlog, snort) while the output of the sensors refers to security logs (alerts in OSSEC terminology). All these security logs coming from different sensors will be translated into a unified security event (Appendix 1) to be easily interpretable from a correlation engine (not included in SiVi). Finally, the collected security events will be depicted on SiVi's dashboard.

2. **The second functionality** provides the Visual Analytics (Anomaly Detection). In this part, SiVi analyses network flows through pre-trained neural networks that can identify different cyber-attacks in order to detect anomalies to the network level and the communication between the network assets. The network flows are captured via a network monitoring sensor and are used as an input to pre-trained anomaly detection models to identify malicious flows. The results are depicted to the SiVi dashboard by allowing the users to create custom widgets by using different fields of the anomaly detection procedure. Figure 13 presents a high-level architecture describing the previous process.

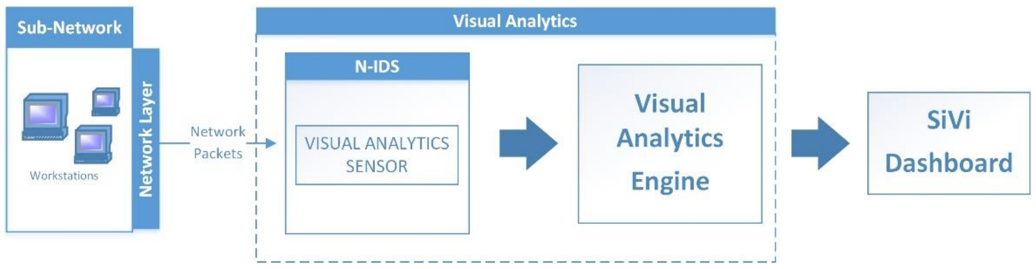


Figure 13: SiVi anomaly detection mechanism

SiVi contains a user friendly, uncongested dashboard providing useful information to the SiVi users. Its near real-time nature formulates SiVi as a tool capable to be used in everyday activities, since the integration with existing databases of attacks can classify the tool on the tactical level of the enterprise.

2.3.1.2 Encrypted Network Traffic Analysis (FORTH)

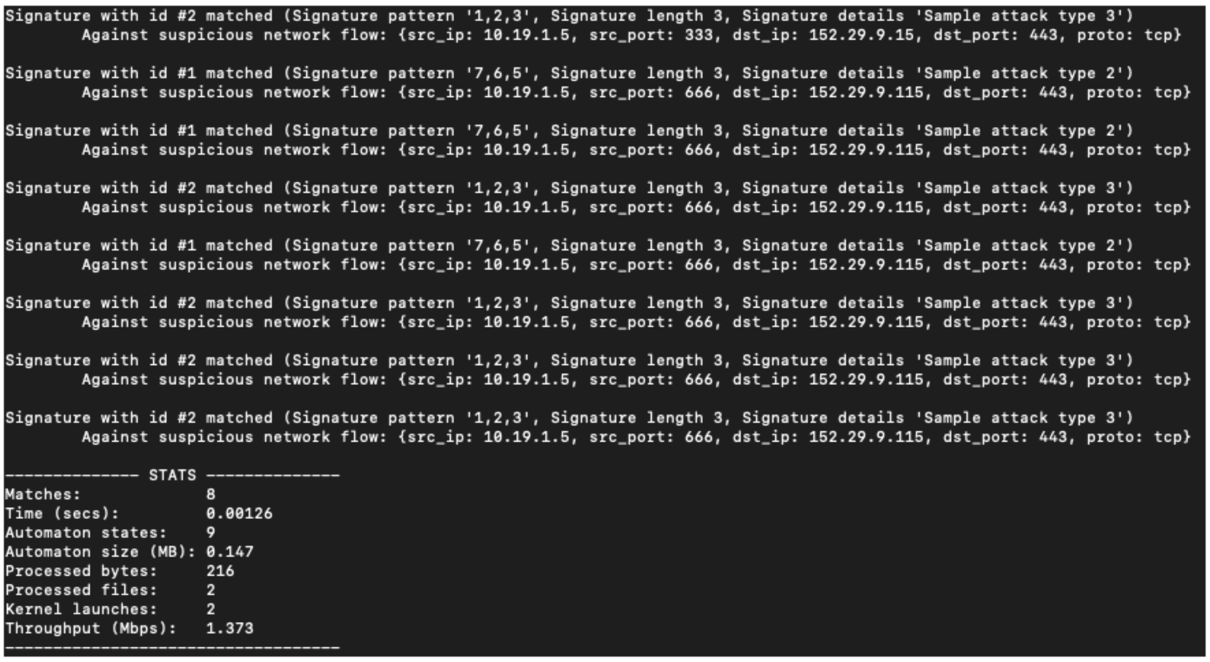
As it is stated in the description of Task 3.3, the goal of the tool is the analysis of encrypted network traffic for specifying patterns that signify the existence of suspicious and malicious activities online, taking also into account the outcomes of T3.1. In this task, existing network traffic, payload-independent classification techniques on identifying traffic patterns, which originate from packet metadata (such as frequent sequences of packet lengths and inter-arrival times) available in encrypted network flows, will be investigated and adopted. The resulted patterns will be processed in order to build a representation format that will assist the unified integration in T3.5. The outcome of this step will provide insight on the proper metadata handling and processing in order to produce network signatures that will be used in order to enrich current network inspection systems' functionality. The network signatures will be produced using a subset of a ground-truth malicious traffic dataset and will be then tested against the remaining traffic samples.

In the next paragraphs, we present the design of FORTH's Encrypted Network Traffic Analysis tool, which (i) is able to identify suspicious and malicious network activities even in encrypted networks and (ii) will serve as a LiveNet component in the CyberSANE architecture.

First and foremost, our tool's operation is divided into two distinct, asynchronous and independent parts.

1. The first part is the offline analysis operation, where malicious network traffic traces are analysed in order to extract meaningful information. More specifically, FORTH investigates (i) which network packet metadata are able to expose the nature of the traffic (e.g., a cyber-attack) and (ii) how we can handle, express and process these metadata to extract signatures that will indicate an attack or another network related event. This means that network traffic traces, which are known to contain malicious behaviour are analysed, and network packet metadata sequences that can indicate this behaviour are extracted.
2. Then, these packet metadata sequences are expressed into signatures using FORTH's proposed signature language. Examples of the proposed pattern language can be found in Section 1.

The output of the offline analysis part (i.e., signatures) is then delivered to the intrusion detection engine (second part). The intrusion detection engine monitors network traffic against the signature set and reports any matches that could signify malicious or suspicious behaviour (e.g., a cyber-incident). The reports are logged and printed as text in the standard output along with periodic statistics of the processing operation. An example of the intrusion detection engine's output is presented in Figure 14. The output of the intrusion detection engine can be verbose or not. If we wish to get fine-grained information about the signature matches, we choose the verbose output, which prints information about the signatures that matched and information about the network flows that triggered the matches and indicate a potential suspicious activity. More specifically, for each signature match, our tool reports the signature (i.e., contains the signature id, the signature pattern, the signature length (number of items per sequence) and signature details (e.g., attack type, name, CVE, URL reference, etc.) and the network flow 5-tuple (i.e., the source IP address, the source port, the destination IP address, the destination port and the protocol).



```
Signature with id #2 matched (Signature pattern '1,2,3', Signature length 3, Signature details 'Sample attack type 3')
        Against suspicious network flow: {src_ip: 10.19.1.5, src_port: 333, dst_ip: 152.29.9.15, dst_port: 443, proto: tcp}

Signature with id #1 matched (Signature pattern '7,6,5', Signature length 3, Signature details 'Sample attack type 2')
        Against suspicious network flow: {src_ip: 10.19.1.5, src_port: 666, dst_ip: 152.29.9.115, dst_port: 443, proto: tcp}

Signature with id #1 matched (Signature pattern '7,6,5', Signature length 3, Signature details 'Sample attack type 2')
        Against suspicious network flow: {src_ip: 10.19.1.5, src_port: 666, dst_ip: 152.29.9.115, dst_port: 443, proto: tcp}

Signature with id #2 matched (Signature pattern '1,2,3', Signature length 3, Signature details 'Sample attack type 3')
        Against suspicious network flow: {src_ip: 10.19.1.5, src_port: 666, dst_ip: 152.29.9.115, dst_port: 443, proto: tcp}

Signature with id #1 matched (Signature pattern '7,6,5', Signature length 3, Signature details 'Sample attack type 2')
        Against suspicious network flow: {src_ip: 10.19.1.5, src_port: 666, dst_ip: 152.29.9.115, dst_port: 443, proto: tcp}

Signature with id #2 matched (Signature pattern '1,2,3', Signature length 3, Signature details 'Sample attack type 3')
        Against suspicious network flow: {src_ip: 10.19.1.5, src_port: 666, dst_ip: 152.29.9.115, dst_port: 443, proto: tcp}

Signature with id #2 matched (Signature pattern '1,2,3', Signature length 3, Signature details 'Sample attack type 3')
        Against suspicious network flow: {src_ip: 10.19.1.5, src_port: 666, dst_ip: 152.29.9.115, dst_port: 443, proto: tcp}

Signature with id #2 matched (Signature pattern '1,2,3', Signature length 3, Signature details 'Sample attack type 3')
        Against suspicious network flow: {src_ip: 10.19.1.5, src_port: 666, dst_ip: 152.29.9.115, dst_port: 443, proto: tcp}

-------------- STATS --------------
Matches:             8
Time (secs):         0.00126
Automaton states:    9
Automaton size (MB): 0.147
Processed bytes:     216
Processed files:     2
Kernel launches:     2
Throughput (Mbps):   1.373
-----------------------------------
```

Figure 14: Output of the Intrusion Detection Engine

At its current form, FORTH's tool prints the results in the standard output as plain text.

2.3.1.3 GLORIA (S2)

GLORIA is a platform developed with the aim of managing security incidents and cybersecurity threats by means of correlating events. It is based on SIEM (Security Information and Event Management) tools but goes one step beyond, providing extensive monitoring and data treatments in two ways:

1. Employing advanced intelligence correlation techniques to fight threats.
2. Using orchestration mechanisms to achieve higher efficiency for the IRTs.

Amongst reasons to employ GLORIA, there are the following:

- ✓ Massive security incident data processing with little effort.
- ✓ Ability to aggregate multiple sources, i.e., not being restricted by high number of inputs.
- ✓ Correlation can be performed either by Gloria or by the device which provides the information.
- ✓ Possibility to reduce number of alerts by means of employing previously aggregated information.

Gloria has been designed as part of an ecosystem or group of Spanish cybersecurity systems and could help security analysts in several ways. When integrated in this *team* of systems, Gloria can share security event information as well as gathered data. The tool can provide some functionalities such as:

- Security incident information gathering based on both NIDS and HIDS.
- IT and OT monitoring.

- Counter-intelligence to fight cyber threats. This is done by means of data correlation.
- Command shell to manage the platform available if needed.
- Reduced response time and reduction on the amount of time a human analyst is needed.

GLORIA has been designed to process events with a funnel model while in touch with other cyber security components. The idea is shown in Figure 15:

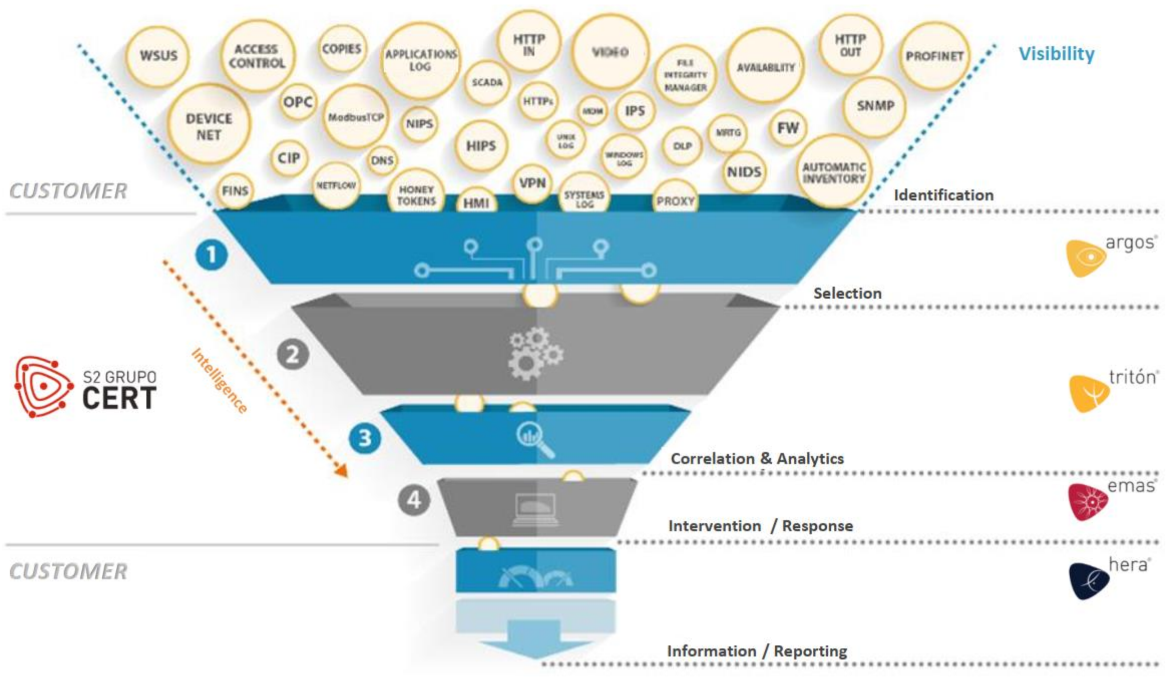


Figure 15: Event processing model (GLORIA)

Finally, GLORIA can identify and classify encrypted streams. To achieve that, it makes use of techniques to analyse encrypted network traffic. Sometimes, it also relies on other tools such as CARMEN (detection of APTs) and L-ADS (Live-Anomaly Detection System).

2.3.1.4 ATOS XL-SIEM

This tool is made up of two components:

1. XL-SIEM or Cross-Layer SIEM
2. L-ADS or Live-Anomaly Detection System

2.3.1.4.1 XL-SIEM

The objectives of a SIEM are various. Some of the most important could be the following:

- Gather and collect security events in real time
- Aggregate data from various and diverse sources
- Consolidate and correlate data
- Alert and report in concordance with regulatory compliance

Although different vendor's SIEMs can be designed with different architectures, its purposes are quite similar. With regard to data flow, XL-SIEM follows the following pattern:

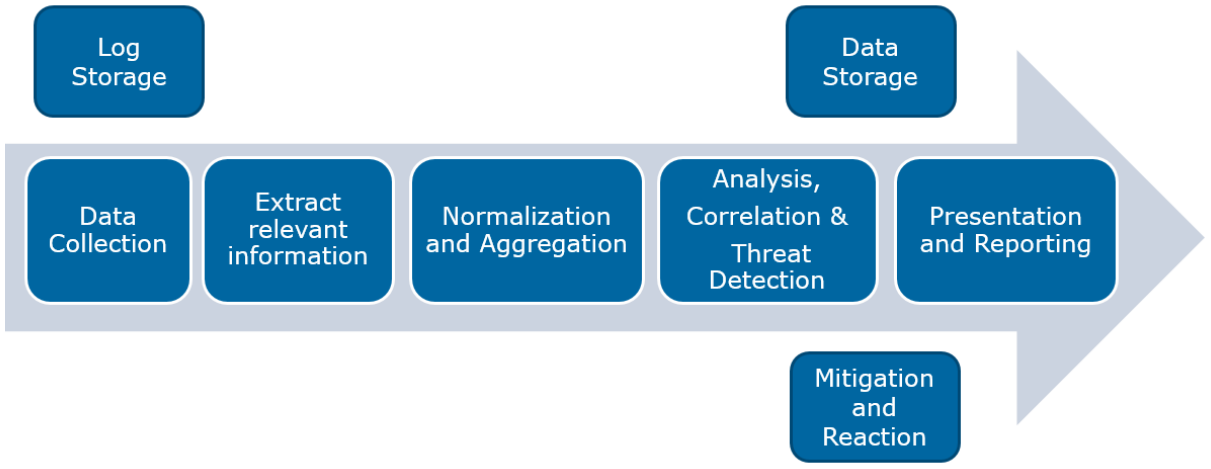

Log Storage
Data Storage
Data Collection
Extract relevant information
Normalization and Aggregation
Analysis, Correlation & Threat Detection
Presentation and Reporting
Mitigation and Reaction

Figure 16: Data flow for XL-SIEM

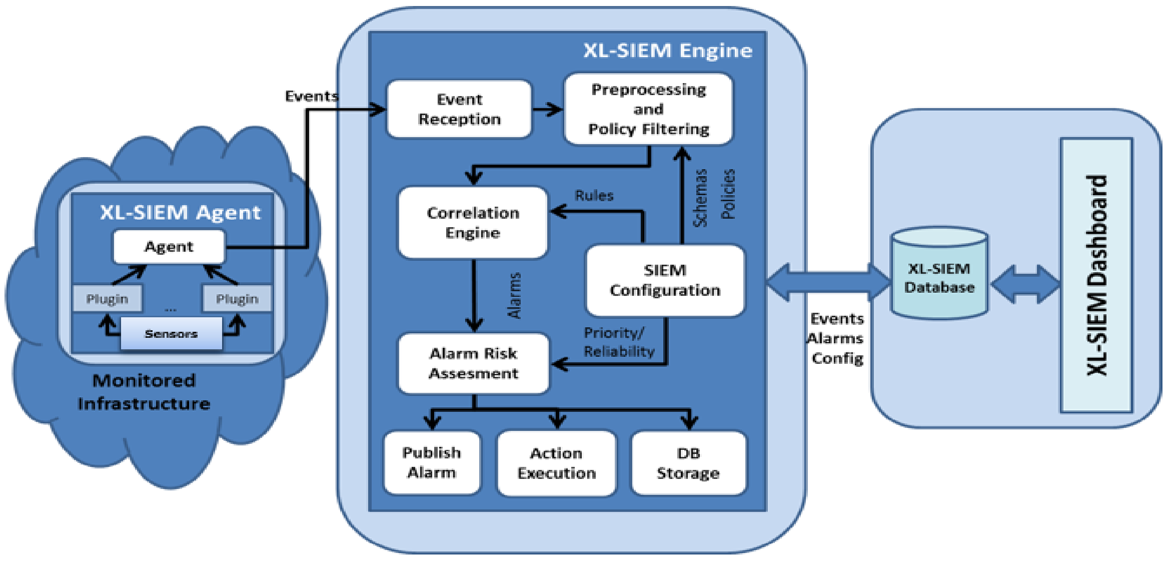


Figure 17: XL-SIEM Architecture

The SIEM is based on three components:

1) XL-SIEM Agent: responsible for the event collection, normalization and transfer to the XL-SIEM Engine for its processing.
2) XL-SIEM Engine: running on Apache Storm, it's responsible for the analysis and processing of the events collected by the XL-SIEM Agents, and the generation of alarms based on a predefined set of correlation rules or security directives.
3) XL-SIEM Dashboard: responsible for the visualization of data in the web graphical interface (graphical charts, alarms, security events, etc.).

Besides these components, the SIEM relies on the sensors to gather information. XL-SIEM works fine with the following sensors:

- Atos Net Tools (DNS Traffic Sensor): detection of botnets, DNS Amplification Attacks (DoS), Brute Force Attacks
- Atos L-ADS
- Snort / Suricata (NIDS): Detection of port scanning, brute force attacks, DoS, malware signatures in the traffic, Inspection of http traffic
- OSSEC (HIDS)
- Firewalls: NetFilter / Cisco.
- Snare (Windows): Unusual activity (brute force login attacks), Data tampering (privilege scalation), SQL injections
- Nagios: DoS (RAM, CPU% usage, nº processes…)
- Arpwatch: Detection of changes in MACs/IPs (not suitable for DHCP)
- Honeypots: Dionaea, Conpot (Modbus)
- SCADA / Modbus Attacks

Finally, an example of XL-SIEM input, in this case from Snort, would be the following log:

(Snort raw log)

```
07/07-17:04:36.504799 [**] [1:2018489:3] ET SCAN NMAP OS Detection
Probe [**] [Classification: Attempted Information Leak] [Priority:
2] {UDP} 5.225.218.36:49743 -> 212.34.151.211:40560
```

2.3.1.4.2 L-ADS

L-ADS is the Live-Anomaly Detection System developed by ATOS. Unlike other systems, which are based on predefined rules and patterns, L-ADS employs unsupervised machine learning to model patterns of normal traffic and identify abnormal network behaviour of devices based on the deviation from the normal operation model. The approach in this case is, then, NetFlow based.

Another important feature of L-ADS is that it has been tested with real, legitimate dataset. This approach was considered the most effective way to try the L-ADS. Results show a promising approach using multiple features, future work will include research to reduce false positives by incorporating more complex features.

Its architecture can be shown in Figure 18:

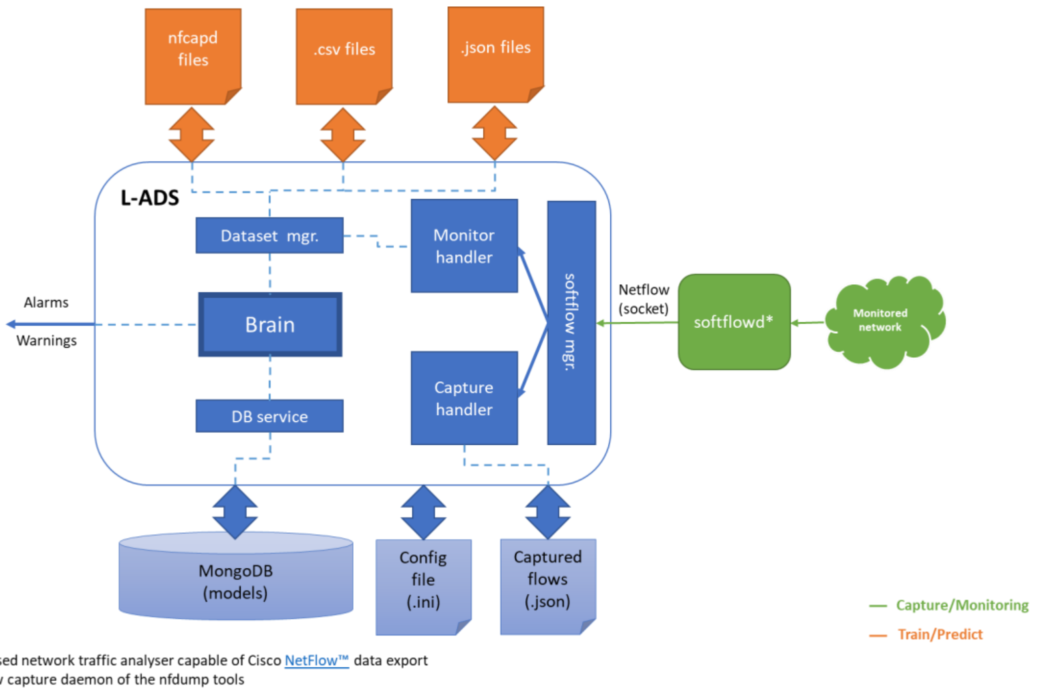


Figure 18: L-ADS architecture

The Anomaly Detection System can work in several different ways such as the following:

- ✓ clean: delete all models stored in the database.
- ✓ capture: store in .JSON files the NetFlow traffic received;
- ✓ train: generate the models and store them in the database.
- ✓ predict: test the models against text files.
- ✓ monitor: evaluate in real time the NetFlow traffic received.
- ✓ exit: close the application.

What data can L-ADS gather from NetFlow?

- Number or incoming/outgoing connections from, to or between servers running the applications;
- Size of the packets sent/received;
- Duration of the connections established between servers or between clients and servers;
- Source/destination IP addresses and ports of the connections;
- Information related to the protocol or application relevant for modelling its behaviour (e.g., URL or function invoked by the user) recovered from the application logs monitored.

2.3.2 *Information Sources*

2.3.2.1 SiVi

The following table shows SiVi’s output format:

Field	Description	Example

source_IP	Source IP	10.0.0.2
source.Hostname	Hostname of the event source.	stable
source.MAC	Media Access Control (MAC) of the host for the event, if known.	N/A
source.Port	External or internal asset source port for the event.	0
source.LatestUpdate	The last time SiVi updated the asset properties.	N/A
source.UsernameDomain	Username and domain associated with the asset that generated the event.	N/A
source.AssetValue	Asset value of the asset source if within your asset inventory.	2
source.Location	If the host country of origin is known, displays the national flag of the event source.	N/A
source.Context	If the asset belongs to a user-defined group of entities, SiVi displays the contexts.	N/A
source.AssetGroup	When the host for the event source is an asset belonging to one or more of your asset groups, this field lists the asset group name or names.	N/A
source.Networks	When the host for the event source is an asset belonging to one or more of your networks, this field lists the networks.	Pvt_170
source.LoggedUsers	A list of any users who have been active on the asset, as detected by the asset scan, for example, with the username and user privilege (such as admin).	N/A
source.OtxIPReputation	(Yes/No) Whether or not IP Reputation identifies the IP address as suspicious.	No
source.Services_Service	List of services or applications detected on the source port.	No services available
source.Services_Port	Port used by the service or application.	-
source.Services_Protocol	Protocol used by the service or application.	-
destination.IP	Destination IP	10.0.0.5
destination.Hostname	Hostname of the event destination.	stable

destination.MAC	Media Access Control (MAC) of the host for the event, if known.	N/A
destination.Port	External or internal asset destination port for the event.	0
destination.LatestUpdate	The last time SiVi updated the asset properties.	N/A
destination.UsernameDomain	Username and domain associated with the asset that generated the event.	N/A
destination.AssetValue	Asset value of the asset destination if within your asset inventory.	2
destination.Location	If the host country of origin is known, displays the national flag of the event destination.	N/A
destination.Context	If the asset belongs to a user-defined group of entities, SiVi displays the contexts.	N/A
destination.AssetGroups	When the host for the event destination is an asset belonging to one or more of your asset groups, this field lists the asset group name or names.	N/A
destination.Networks	When the host for the event destination is an asset belonging to one or more of your networks, this field lists the networks.	Pvt_170
destination.LoggedUsers	A list of any users who have been active on the asset, as detected by the asset scan, for example, with the username and user privilege (such as admin).	N/A
destination.OtxIpReputation	(Yes/No) Whether or not IP Reputation identifies the IP address as suspicious.	No
destination.Services_Service	List of services or applications detected on the destination port.	No services available
destination.Services_Port	Port used by the service or application.	-
destination.Services_Protocol	Protocol used by the service or application.	-
event.event_type_id	ID assigned by SiVi to identify the event type.	5502
event.unique_event_id	Unique ID number assigned to the event by SiVi.	33b334vs-sdvs-asdf-335d-22daf467

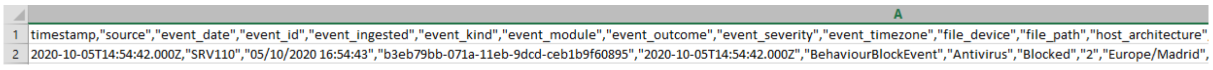
event.protocol	Protocol used for the source/destination of the event	TCP
event.category	Event taxonomy for the event.	Authenticatio n
event.subcategory	Subcategory of the event taxonomy type listed under Category.	Logout
event.data_source_nam e	Name of the external application or device that produced the event.	HIDS-Syslog
event.data_source_id	ID associated with the external application or device that produced the event.	7001
event.product_type	Product type of the event taxonomy, for example, Operating System or Server.	Operating System
event.additional_info	Security event additional information	N/A
event.priority	Priority ranking based on value of the event type. Each event type has a priority value, used in risk calculation.	1
event.reliability	Reliability ranking based on the reliability value of the event type. Each event type has a reliability value, which is used in risk calculation.	1
event.risk	Risk level of the event: Low = 0, Medium = 1, High > 1 Note: Risk calculation is based on this formula: Asset Value * Event Reliability * Event Priority / 25 = Risk If Asset Value = 3, Reliability = 2 and Priority = 2, the risk would be 3 * 2 * 2 / 25 = 0.48 (rounded down to 0)	4
event.otx_indicators	Number of indicators associated with an IP Reputation or OTX pulse event.	0
event.device_ip	IP address of the sensor that processed the event.	10.0.0.12
date	Date and time of the event (UTC)	2020-04-04 T17:00:00
raw_log	Raw log details of the event.	

filename	*Optional - Name of file associated with the event.	
username	*Optional - The username associated with the event.	root
password	*Optional - The password associated with the event.	
userdata1	*Optional - User-created log fields.	/var/log/auth.log
userdata2	*Optional - User-created log fields.	Log session closed.
userdata3	*Optional - User-created log fields.	pam, syslog
userdata4	*Optional - User-created log fields.	None
userdata5	*Optional - User-created log fields.	
userdata6	*Optional - User-created log fields.	
userdata7	*Optional - User-created log fields.	
userdata8	*Optional - User-created log fields.	
event.event_type_id	ID assigned by SiVi to identify the event type.	5502
event.unique_event_id	Unique ID number assigned to the event by SiVi.	33b334vs-sdvs-asdf-335d-22daf467
event.protocol	Protocol used for the source/destination of the event	TCP
event.category	Event taxonomy for the event.	Authentication

Table 4: Output format of SiVi

2.3.2.2 Encrypted Network Traffic Analysis

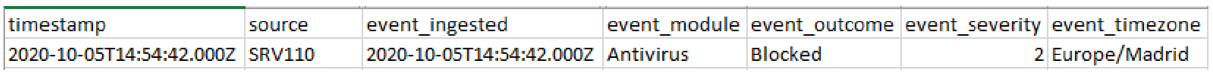
FORTH's Encrypted Network Analysis Tool produced an output which was extracted with the help of Graylog. Provided log was a CSV file which needed some processing. Raw log looks like:



```
timestamp,"source","event_date","event_id","event_ingested","event_kind","event_module","event_outcome","event_severity","event_timezone","file_device","file_path","host_architecture",
2020-10-05T14:54:42.000Z,"SRV110","05/10/2020 16:54:43","b3eb79bb-071a-11eb-9dcd-ceb1b9f60895","2020-10-05T14:54:42.000Z","BehaviourBlockEvent","Antivirus","Blocked","2","Europe/Madrid",
```

Figure 19: Raw log of Encrypted Network Analysis Tool

After some handling of the data, the format could be expressed in a more readable way:



timestamp	source	event_ingested	event_module	event_outcome	event_severity	event_timezone
2020-10-05T14:54:42.000Z	SRV110	2020-10-05T14:54:42.000Z	Antivirus	Blocked	2	Europe/Madrid

Figure 20: Raw log formatted

The following table shows an example of the output of Encrypted Network Analysis Tool once it has been processed:

Field	Example
timestamp	2020-10-05T14:54:42.000Z
source	SRV110
event_date	05/10/2020 16:54
event_id	b3eb79bb-071a-11eb-9dcd-ceb1b9f60895
event_ingested	2020-10-05T14:54:42.000Z
event_kind	BehaviourBlockEvent
event_module	Antivirus
event_outcome	Blocked
event_severity	2
event_timezone	Europe/Madrid
file_device	_
file_path	HKLM\SOFTWARE\MCAFEE\SYSTEMCORE\VSCORE\NVP\
host_architecture	Windows 2003 R2
host_hostname	LUBCS3
host_ip	172.22.3.3
host_mac	00505681ce2b
Message	<?xml version="1.0" encoding="UTF-8"?> <BehaviourBlockEvent><MachineInfo> <MachineName>LUBCS3</MachineName><AgentGUID>{df28668e-6205-11ea-0ca1-00505681ce2b}</AgentGUID>

	<IPAddress>172.22.3.3</IPAddress><OSName>Windows 2003 R2</OSName><UserName>NT AUTHORITY\SYSTEM</UserName><TimeZoneBias>-120</TimeZoneBias> <RawMACAddress>00505681ce2b</RawMACAddress> <ScannerSoftware ProductName="VirusScan Enterprise" ProductVersion="8.8" ProductFamily="TVD"> <EngineVersion>0</EngineVersion> <DATVersion>0</DATVersion> <ScannerType>OAS</ScannerType><TaskName>OAS</TaskName> <ProductFamily>TVD</ProductFamily> <ProductName>VirusScan Enterprise</ProductName> <ProductVersion>8.8</ProductVersion> <BlockedBehaviourInfo><EventID>1092</EventID><Severity>2</Severity> <GMTTime>2020-10-05T16:54:42</GMTTime> <UTCTime>2020-10-05T14:54:42</UTCTime> <RuleName>ProtecciÃ³n comÃºn estÃ¡ndar:Impedir la modificaciÃ³n de los archivos y las opciones de McAfee</RuleName> <ProcessName>C:\ARCHIVOS DE PROGRAMA\MCAFEE\AGENT\MACOMPATSVC.EXE</ProcessName> <FileName>HKLM\SOFTWARE\MCAFEE\SYSTEMCORE\VSCORE\NVP\</FileName> <Source>_</Source> <ActionsBlocked>5</ActionsBlocked> <szActionsBlocked>Escritura</szActionsBlocked> </BlockedBehaviourInfo></ScannerSoftware></BehaviourBlockEvent>#015
Observer_product	VirusScan Enterprise
Observer_type	Antivirus
Observer_vendor	TVD
Organization_id	TEST
Organization_name	TEST
User_name	NT AUTHORITY\SYSTEM

Table 5: Encrypted Network Analysis Tool output format

2.3.2.3 GLORIA

The following table sums up the proposed output format of GLORIA:

ECS field	Description
@timestamp	Date of detection in origin
agent_name	Custom name of agent

agent_type	Type of agent
destination_geo_city_name	Destination city name
destination_geo_country_iso_code	Destination country code
destination_geo_country_name	Destination country name
destination_ip	Destination IP for the connection
destination_port	Port of the destination
event_action	Action captured by the event
event_category	One of four ECS Categorization fields
event_ingested	Date when event was received
event_dataset	Name of the dataset
event_end	Date when event ended or activity was last observed
event_kind	High-level information of what type of data the event contains
event_module	Name of the module data is coming from
event_original	Original message description
event_outcome	Success or failure from the perspective of the entity that produced the event
event_severity	Numeric severity of the event according to event source
event_start	Date when the event started or when the activity was first observed
event_timezone	Time zone of the event (if not included in the timestamp) of device source of the alert
file_device	Device where file is located
file_hash_md5	MD5 hash of file
file_hash_sha256	SHA256 hash of file
file_name	Name of the file (including extension) without the directory
file_path	Path of the file
file_size	Size of the file (bytes)

host_geo_city_name	City name (of host)
host_geo_country_iso_code	Code of the country where host is located
host_geo_country_name	Country name
host_group_name	Group to which host belongs
host_hostname	Hostname of the host where alert is produced
host_ip	IP address of host where alert is produced
http_request_body_content	Full HTTP request body
message	Raw text message of entire event
network_transport	A name given to an application level protocol
observer_ip	IP address of the device that produces the alert
observer_name	Name of device which produces the alert
observer_product	Product model which generates the alert
observer_type	Kind of device which produces the alert
observer_vendor	Vendor of the device which produces the alert
organization_id	Unique ID for the organization client
organization_name	Organization name
process_args	Parameters received by a process being executed
process_executable	Name of executable
process_hash_md5	MD5 hash of executable file
process_hash_sha256	SHA256 of executable file
server_domain	FQDN name of server
source_geo_city_name	Name of city of origin
source_geo_country_iso_code	Code of country source of connection
source_geo_country_name	Name of country source of connection
source_hostname	Name of source of connection
source_ip	IP address of the source of the connection (IPv4 or IPv6)

source_port	Port of source of connection
threat_framework	Name of threat framework used to classify or categorize the threat
rule_id	ID or code established by signature manufacturer
rule_name	Signature established by manufacturer
url_full	URL of HTTP request
user_domain	Windows domain of the user
user_group_name	Name of the group to which the user belongs
user_name	Short name or login of the user

Table 6: Output format of GLORIA

2.3.2.4 XL-SIEM

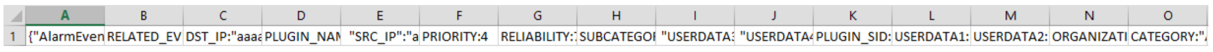
The SIEM of ATOS produces the following output:

Field	Description
Backlog_ID	Backlog Identificator
Category	Category of the event
Date	Date when event was produced or detected
Device	Device affected by the event
DST_IP	Destination IP
DST_IP_Hostname	Destination IP hostname
DST_Port	Puerto de destino
Event_ID	Event identification
Filename	Name of file related to the incident
Interface	Interface through which the event was detected
Log	Raw log of event
Organization	Name of organization engaged in detecting the event
Password	

Plugin_ID	Identification of the plugin which detected the event
Plugin_name	Name of the plugin that detected the event
Priority	Numeric priority of the event
Protocol	Protocol related to the event
Related_events	List of related events (if applicable)
Reliability	Numeric estimate of reliability of event detection
Risk	Numeric estimate of risk associated to event
SRC_IP	Source IP address
SRC_IP_Hostname	Source IP hostname
SRC_Port	Source port
Subcategory	Description of subcategory of event
Userdata1	*Optional
Userdata2	*Optional
Userdata3	*Optional
Userdata4	*Optional
Userdata5	*Optional
Userdata6	*Optional
Userdata7	*Optional
Userdata8	*Optional
Userdata9	*Optional

Table 7 - Output format of XL-SIEM

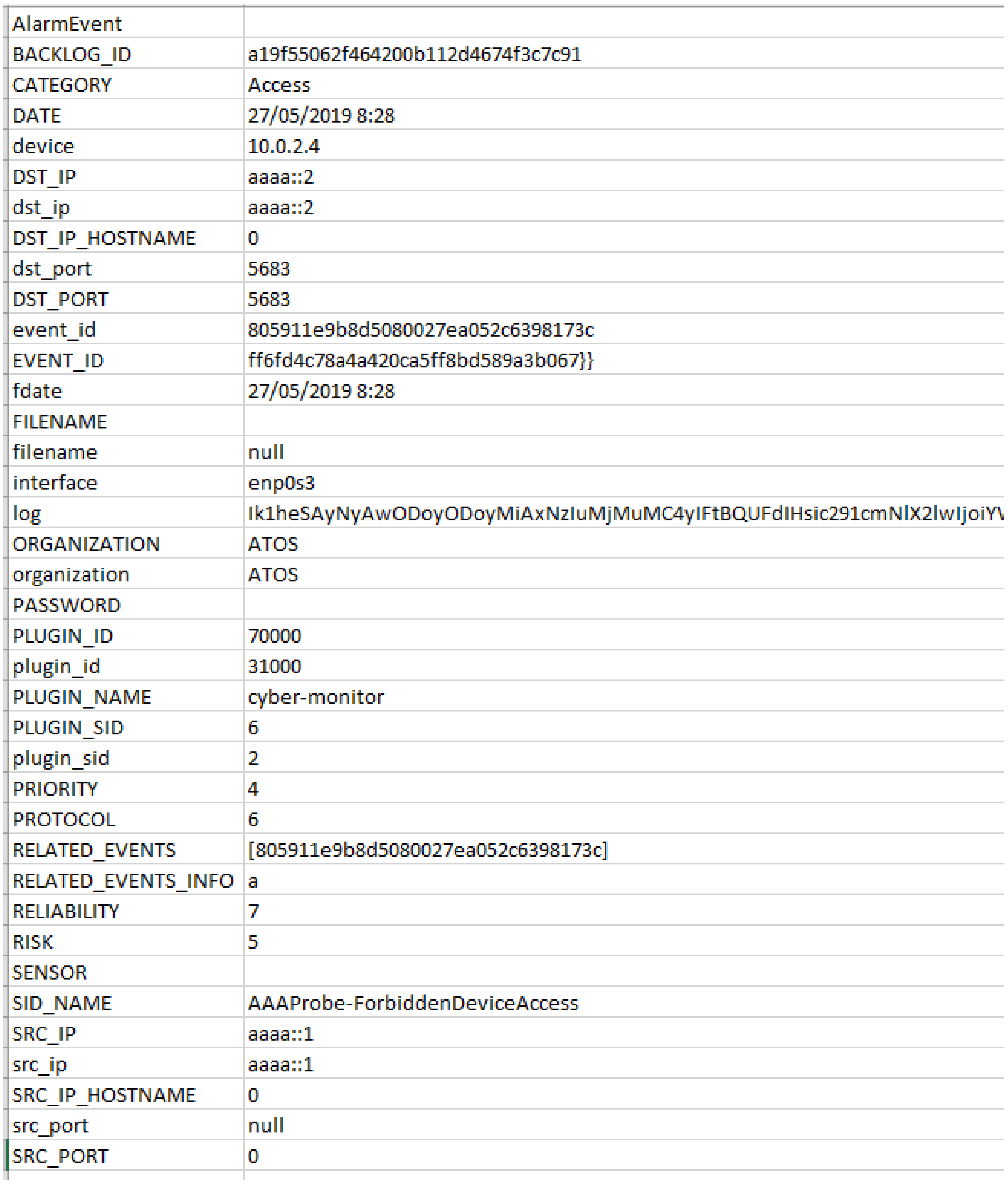
The format itself presents different event security information fields which must be filtered and normalized before mapping. First process is to translate the output to a more readable format:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	{"AlarmEven	RELATED_EV	DST_IP:"aaaa	PLUGIN_NAN	"SRC_IP":"a	PRIORITY:4	RELIABILITY:1	SUBCATEGOI	"USERDATA3	"USERDATA4	PLUGIN_SID:	USERDATA1:	USERDATA2:	ORGANIZATI	CATEGORY:"

Figure 21: Separate fields in output of XL-SIEM

It will be necessary to transpose fields to filter and know what could be discarded:



AlarmEvent	
BACKLOG_ID	a19f55062f464200b112d4674f3c7c91
CATEGORY	Access
DATE	27/05/2019 8:28
device	10.0.2.4
DST_IP	aaaa::2
dst_ip	aaaa::2
DST_IP_HOSTNAME	0
dst_port	5683
DST_PORT	5683
event_id	805911e9b8d5080027ea052c6398173c
EVENT_ID	ff6fd4c78a4a420ca5ff8bd589a3b067}}
fdate	27/05/2019 8:28
FILENAME	
filename	null
interface	enp0s3
log	Ik1heSAyNyAwODoyODoyMiAxNzIuMjMuMC4yIFtBQUFdIHsic291cmNlX2lwIjoiY
ORGANIZATION	ATOS
organization	ATOS
PASSWORD	
PLUGIN_ID	70000
plugin_id	31000
PLUGIN_NAME	cyber-monitor
PLUGIN_SID	6
plugin_sid	2
PRIORITY	4
PROTOCOL	6
RELATED_EVENTS	[805911e9b8d5080027ea052c6398173c]
RELATED_EVENTS_INFO	a
RELIABILITY	7
RISK	5
SENSOR	
SID_NAME	AAAProbe-ForbiddenDeviceAccess
SRC_IP	aaaa::1
src_ip	aaaa::1
SRC_IP_HOSTNAME	0
src_port	null
SRC_PORT	0

Figure 22: Final transformation of output of XL-SIEM

Once these steps have been completed, log is ready for the mapping procedure.

2.4 Overview of Architecture

2.4.1 Introduction

CyberSANE project makes use of several tools from very different manufacturers. Each one has its own set of characteristics and has been designed to accomplish a concrete mission with regard to event information gathering. Because of that, when comparing devices involved on monitoring tasks, information obtained may vary significantly from one device to another.

After careful considerations, it was decided that the best approach to follow would be to normalize all incoming logs to a recognizable, standard output. This normalized output can then be handled with less effort, due to the fact that it has been previously filtered and adapted. For that purpose, it is important to first define the necessary architecture, that is, how inputs are going to be integrated in the CyberSANE platform.

2.4.2 Elastic Common Schema as standard output

No matter what kind of operation we may be performing (cyber threat intelligence, operation analytics, etc.) it is quite common to gather data from different sources and devices.

Elastic Common Schema or ECS[30] is an open source specification and framework which defines a **common set of fields** to store event data. This common data model implies a big advantage when trying to correlate data. Although one of its main objectives is to be integrated with ElasticSearch, its versatility as a highly customizable set of fields makes it quite suitable to be employed for other purposes. Therefore, ECS can be adapted to map any security event or log, no matter what source has produced it. ECS proposes different groups of fields called "field sets". These categories can be used at will. Field sets are available online[31].

With the advantage of being open source, this framework for data modelling can be used for free. While some people might expect it to be installed and running as a standalone component, the idea behind the project is more about being a guideline to achieve data normalization from different sources. ECS proposes a framework flexible to accept whatever input might be necessary. An important nuance is that ECS makes **no modification on the original data**. Another strength of ECS is about its integration with ElasticSearch, which comes naturally. Therefore, it is possible to retrieve information, engage with dashboards and queries or drop to a more granular view in a few clicks. Correlation of data and collaboration between partners require little effort thanks to ECS, given its powerful capabilities when adapting to different kinds of inputs produced by vendors or devices.

Last but not least, ECS can take advantage of machine learning. It is possible to create jobs just by means of the Elastic Stack and store results into ElasticSearch. Handling operations such as filtering, sorting or correlating anomalies are no longer painful because of the way ECS links security event information.

[30] https://www.elastic.co/guide/en/ecs/current/index.html
[31] https://www.elastic.co/guide/en/ecs/current/ecs-field-reference.html

2.4.3 *Normalization and transformation of event information into unified format*

Normalization and transformation operations are performed with one clear purpose: homogenise data. When working with different sources, vendors or providers, information can adopt different ways and come into distinct formats. However, the best way to make the most of it is by means of putting all the pieces together. Normalization can assist in this process, enabling to care for just one standard format. Simplification is key for a fast answer when dealing with a security event.

According to what Elastic indicates, a good way to fit into ECS format should include the following steps:

1. Review of fields in the original source. The idea would be to understand what information is provided in each log.
2. Filter and discard what is not necessary.
3. Define a final ECS output version to which all inputs should reasonably map.
4. Map events to the relevant ECS Core or Extended field.
5. Review all remaining information with the objective of trying to populate as many ECS fields as possible. Different field sets can help with this task.
6. Refinement.

2.4.3.1 Filter (Reducing irrelevant information)

First thing to do before normalizing or transforming data is to filter it. Usually, logs provide a lot of information but not everything is needed. Therefore, it is always advisable to get rid of data to spare. Filtering has a double-side beneficious effect:

1. It reduces complexity by avoiding non necessary information to be processed.
2. It helps saving time as there is no need to process every single piece of information provided by the logs.

Additionally, filtering makes subsequent stages more manageable: by means of reducing the amount of information to be processed it is quite possible to be focused on what matters.

With regard to CyberSANE, first step is to understand what information is provided by each one of the inputs. It was performed a review of fields from each one of the original sources, to grasp what might come in the log. This entails a big importance, since every device acquiring data will gather different kind of fields. Once this is completed, filtering can be performed:

- Input provided by GLORIA is quite similar to the standard ECS output format. GLORIA already proposes an ECS format style to store data. Only thing which remained to be done was to check what fields were being used to verify there was no important information missing.
- Input of SiVi provides various fields which need some mapping. For instance, information such as reliability or asset groups was not easily mapped with the ECS format. But, more importantly, some optional fields such as password (associated with the event) or *userdata* field can be discarded without hesitation.

- FORTH's Encrypted Network Traffic Analysis Tool provided a more straightforward log. In this case the point was not about filtering but making sure there was enough information. Log of device lacks some information about destination, but it comes fully provisioned as far as event fields are concerned.
- With regard to XL-SIEM of ATOS, it provides data about several of the most important event fields such as host hostname, interface, device, sensor, source IP and port or username. However, it is difficult to map some of the ECS event fields to the log provided.

2.4.3.2 Output format proposal

One of the biggest advantages of ECS is the enormous possibility of customization it provides. Therefore, output format can be designed to fulfil any specific need. After understanding what information does every one of the partners provide, it has been easier to know the most important fields to be considered for the output. Minimum output proposal should include the following information:

- Time when the event took place. A timestamp would be enough, better if it includes timezone.
- Information about both source and destination of the attack: IP address, hostname, MAC address (if available)
- Identification of the event.
- Severity of the event.
- Information about what device processed the event or produced the alert.
- Result of the alert (block, false positive, etc.)
- Name of the user associated with the event (if any).
- Any other information of interest such as rule that was triggered, protocol or organization name.

All this information is compiled in the following proposed ECS fields:

Field	Description
@Timestamp	Date of detection in origin
destination.ip	Destination IP for the connection
destination.port	Port of the destination
event.action	Action captured by the event
event.category	Taxonomy of the event
event.end	Date when the event ended or when the activity was last observed
event.id	Unique ID to describe the event
event.outcome	Success or a failure from the perspective of the entity that produced the event
event.severity	The numeric severity of the event according to your event source
event.timezone	Event's time zone (if not included in the timestamp) of device source of alert
event.type	Represents a categorization "sub-bucket", i.e., subcategory of event taxonomy

file.name	Name of the file associated with the event including the extension, without the directory
file.path	Path of the file
host.hostname	Hostname of the host (where alert is produced)
host.ip	Host IP address (IP of host where alert is produced)
host.mac	Host mac addresses
message	Raw text message of entire event (raw of log)
network.transport	A name given to an application level protocol (Protocol used for source/destination)
observer.ip	IP address of device which produces alert
observer.name	Name of device which produces alert
observer.type	Kind of device which produces the alert
organization.id	Unique identifier for the organization (client)
rule.name	Signature established by manufacturer / Rule used to detect the event
source.hostname	Name of source of connection
source.ip	IP address of the source (of the connection) (IPv4 or IPv6)
source.mac	MAC address of the source (of the connection)
source.port	Port of the source
user.domain	(Windows) domain of the user
user.name	Short name or login of the user

Table 8: Output format proposal

2.4.3.3 Map and transform (Change the formats)

Mapping is the process of making an input fit into an output. This process has different levels of complexity depending on how far input and output are. The more similar they are, the less work is to be done. However, when input organization differs from output's one, some operations are needed, and these may entail some complexity.

According to what was stated in the filtering stage, output of GLORIA poses the least difficulty to be mapped, given that the format is the same to the proposed output format (ECS in both cases). For the other logs, some transformation is needed to adapt what comes to output proposed format.

Mapping is a process done on two sides:

i. Manual: the analyst designs how an input field should match to the corresponding output one, that is, what is the best output matching for every input. This step should be improved with several iterations to populate as many output fields in the most accurate way as possible. Each iteration should deliver more fields populated.

ii. Automatic: it would be necessary to define some pipelines to perform the transformation in an automatized way. The main problem will be that any input will need its own, custom pipeline to be transformed into a suitable output.

2.4.3.4 Review and refinement

The last step of the normalization process would be a review and refinement of the mapping. The purpose is that the analyst verifies whether there are some input fields that are not matching properly so he can make the proper adjustments. Sometimes two input fields could be matched with a particular output field. The objective of refinement is to review these situations and try to resolve them to achieve the most accurate result possible. Refinement seeks to reduce mapping errors as well as any doubtful situation it may arise.

It is important to bear in mind that in most of the situations, the process of mapping will not be 100% accurate, this should happen only if the same format is used in both origin and destination and, then, mapping would not be necessary. Only through time, dedication and expertise the analyst will be capable of matching source log to destination accurately.

2.4.4 Architecture proposal

The purpose of the system will essentially be to transform various inputs into a standard output. There are several ways to achieve this objective. In this case, architecture of the system proposed was thought to be as simple as possible while capable of meeting the requirements. It will be necessary to have, at least, some components:

1. First component should contain various plugins. Plugins are the best way to read the input and accept different kinds of logs. Its main function is just to accommodate to whatever is provided. It is essential that the component is scalable, that is, accept new plugins if needed. The idea is to make the architecture to easily accept new log formats in case these are provided.
2. Transformation and normalization component. Normally, pipelines are the mechanism in charge of providing a standard output. They work similar to funnels and make use of some normalization techniques to adapt what comes from the tools to the proposed standard output format:
    a. Filtering input, i.e., discarding what may not be necessary.
    b. Mapping, that is, matching input fields to the correspondent field in ECS format.

It would be desirable that all components are integrated in a single module. Although each component is responsible for its own duties, having a single module may help to simplify, especially in terms of avoiding unnecessary communications between modules.

As previously stated, output format will be ECS. Thus, it must not be forgotten that it would be possible to bring information to ElasticSearch if needed.

2.4.5 Sequence diagram

Sequence diagram groups in a visual manner the way the architecture should behave. Figure 24 shows the sequence diagram:

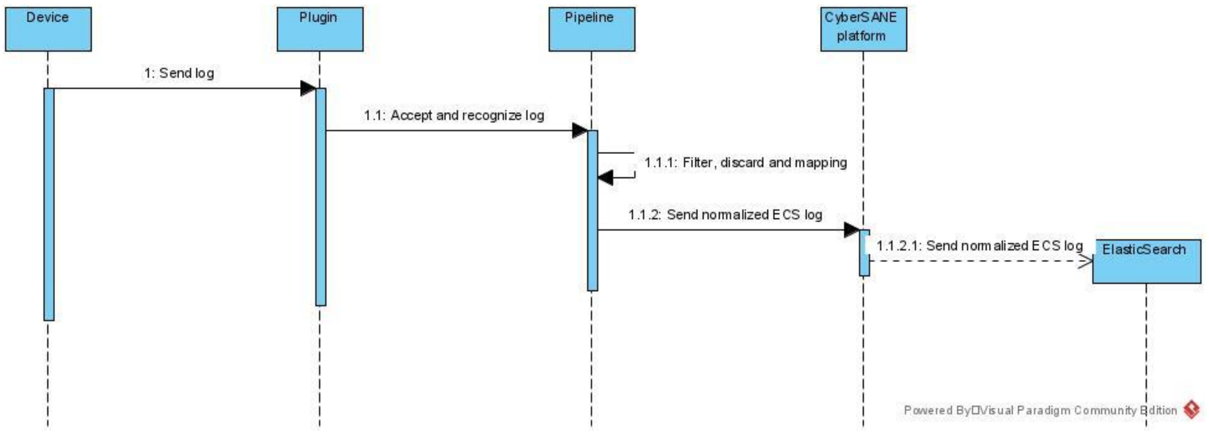

Device
Plugin
Pipeline
CyberSANE platform
1: Send log
1.1: Accept and recognize log
1.1.1: Filter, discard and mapping
1.1.2: Send normalized ECS log
1.1.2.1: Send normalized ECS log
ElasticSearch
Powered By Visual Paradigm Community Edition

Figure 23 - Sequence diagram

The most important things about sequence diagram are summed up in the following points:

- Plugin components are expected to accept raw logs from different devices. Plugins are designed to deal with different manufacturers and inputs. Architecture can be escalated just by adding whatever plugins may be necessary.
- Pipelines do the hard work of filtering logs, discarding whatever may be unnecessary and, finally, mapping to the selected ECS output format. Once the data has been given appropriate treatment, it is sent to the CyberSANE platform.
- CyberSANE will have all information standardized to ECS format. This will make easier to handle the data and provide new possibilities, such as providing information to ElasticSearch. ElasticSearch is one of the best ways to display complex information just with little effort.

2.4.6 *Normalization and data transformation incident-related examples*

2.4.6.1 Example 1 – Mapping GLORIA to ECS

In this example mapping between output of GLORIA to ECS format is going to be explained in detail. The advantage here is that GLORIA already provides a log in ECS.

An example of the output provided by GLORIA would be the following raw log:

@timestamp	*timestamp=1580908637.837166*
destination_ip	*dst=208.67.222.222:XX*
destination_port	*dst=x.x.x.x:53*
event_dataset	*security_event*
event_module	*IPS*

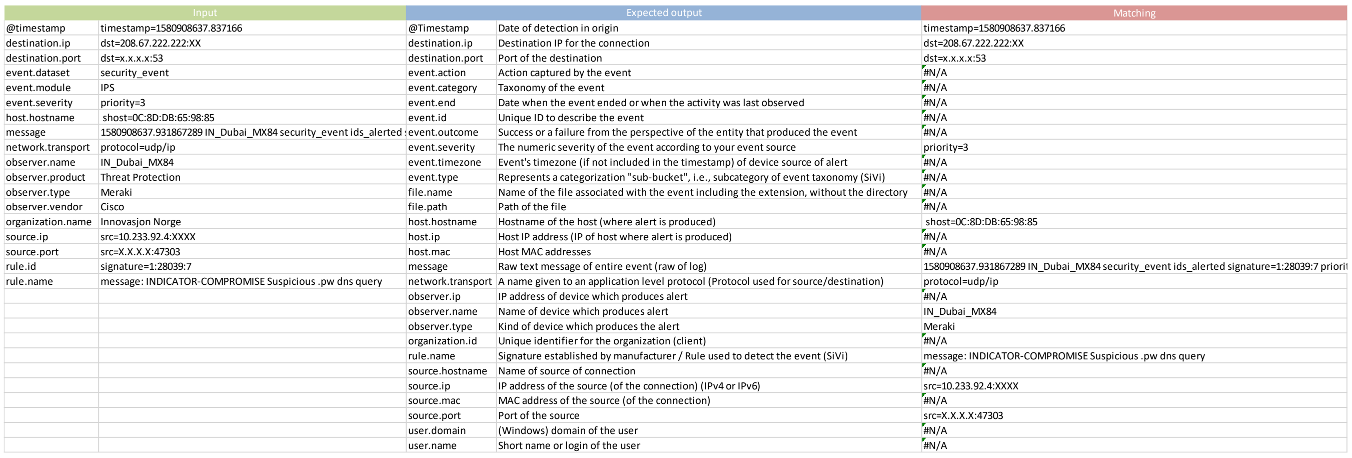
event_severity	*priority=3*
host_hostname	*shost=0C:8D:DB:65:98:85*
message	*1580908637.931867289 IN_Dubai_MX84 security_event ids_alerted signature=1:28039:7 priority=3 timestamp=1580908637.837166 shost=0C:8D:DB:65:98:85 direction=egress protocol=udp/ip src=10.233.92.4:47303 dst=208.67.222.222:53 message: INDICATOR-COMPROMISE Suspicious .pw dns query*
network_transport	*protocol=udp/ip*
observer_name	*IN_Dubai_MX84*
observer_product	*Threat Protection*
observer_type	*Meraki*
observer_vendor	*Cisco*
organization_name	*Innovasjon Norge*
source_ip	*src=10.233.92.4:XXXX*
source_port	*src=X.X.X.X:47303*
rule_id	*signature=1:28039:7*
rule_name	*message: INDICATOR-COMPROMISE Suspicious .pw dns query*

Table 9: Example of output of GLORIA

As the table shows, some of the ECS outputs are populated with security event information. For the time being, it is possible to know:

- Time and data when the event took place.
- Severity of the event.
- Source IP and port.
- Destination IP and port.
- MAC address of the host affected.
- Details of the product raising the alert.
- ID and name of the rule raised by the event.

Besides, GLORIA provides raw log of the event (see message field in the table). Once the input is in ECS format, it is easy to map with the expected output:



Input		Expected output		Matching
@timestamp	timestamp=1580908637.837166	@Timestamp	Date of detection in origin	timestamp=1580908637.837166
destination.ip	dst=208.67.222.222:XX	destination.ip	Destination IP for the connection	dst=208.67.222.222:XX
destination.port	dst=x.x.x.x:53	destination.port	Port of the destination	dst=x.x.x.x:53
event.dataset	security_event	event.action	Action captured by the event	#N/A
event.module	IPS	event.category	Taxonomy of the event	#N/A
event.severity	priority=3	event.end	Date when the event ended or when the activity was last observed	#N/A
host.hostname	shost=0C:8D:DB:65:98:85	event.id	Unique ID to describe the event	#N/A
message	1580908637.931867289 IN_Dubai_MX84 security_event ids_alerted	event.outcome	Success or a failure from the perspective of the entity that produced the event	#N/A
network.transport	protocol=udp/ip	event.severity	The numeric severity of the event according to your event source	priority=3
observer.name	IN_Dubai_MX84	event.timezone	Event's timezone (if not included in the timestamp) of device source of alert	#N/A
observer.product	Threat Protection	event.type	Represents a categorization "sub-bucket", i.e., subcategory of event taxonomy (SiVi)	#N/A
observer.type	Meraki	file.name	Name of the file associated with the event including the extension, without the directory	#N/A
observer.vendor	Cisco	file.path	Path of the file	#N/A
organization.name	Innovasjon Norge	host.hostname	Hostname of the host (where alert is produced)	shost=0C:8D:DB:65:98:85
source.ip	src=10.233.92.4:XXXX	host.ip	Host IP address (IP of host where alert is produced)	#N/A
source.port	src=X.X.X.X:47303	host.mac	Host MAC addresses	#N/A
rule.id	signature=1:28039:7	message	Raw text message of entire event (raw of log)	1580908637.931867289 IN_Dubai_MX84 security_event ids_alerted signature=1:28039:7 priorit
rule.name	message: INDICATOR-COMPROMISE Suspicious .pw dns query	network.transport	A name given to an application level protocol (Protocol used for source/destination)	protocol=udp/ip
		observer.ip	IP address of device which produces alert	#N/A
		observer.name	Name of device which produces alert	IN_Dubai_MX84
		observer.type	Kind of device which produces the alert	Meraki
		organization.id	Unique identifier for the organization (client)	#N/A
		rule.name	Signature established by manufacturer / Rule used to detect the event (SiVi)	message: INDICATOR-COMPROMISE Suspicious .pw dns query
		source.hostname	Name of source of connection	#N/A
		source.ip	IP address of the source (of the connection) (IPv4 or IPv6)	src=10.233.92.4:XXXX
		source.mac	MAC address of the source (of the connection)	#N/A
		source.port	Port of the source	src=X.X.X.X:47303
		user.domain	(Windows) domain of the user	#N/A
		user.name	Short name or login of the user	#N/A

Figure 24: Example of mapping of output

2.4.6.2 Example 2 – Mapping SiVi to ECS

In this example the transformation of the output format of SiVi into the ECS will be presented in details. A log of SiVi has the following format:

Proposed Fields	SiVi Fields	SiVi Field Example
@Timestamp	date	2020-04-04 T17:00:00
destination.ip	destination.IP	10.0.0.5
destination.port	destination.Port	5000
event.action	-	-
event.category	event.category	Authentication
event.end	-	-
event.id	event.unique_event_id	33b334vs-sdvs-asdf-335d-22daf467
event.outcome	-	-
event.severity	event.risk	4
event.timezone	-	-
event.type	event.subcategory	Logout
file.name	filename	
file.path	-	-
host.hostname	source.Hostname	stable
host.ip	event.device_ip	10.0.0.2
host.mac	source.MAC	00:1B:33:11:3B:A4
message	raw_log	
network.transport	event.protocol	TCP
observer.ip	event.device_ip	10.0.0.12
observer.name	event.data_source_name	HIDS-Syslog

observer.type	-	-
organization.id	-	-
rule.name	-	-
source.hostname	source.Hostname	stable
source.ip	source.IP	10.0.0.2
source.mac	source.MAC	00:1B:33:11:3B:A4
source.port	source.Port	4999
user.domain	-	-
user.name	username	root

Chapter 3 Summary and Conclusion

3.1 Intrusion detection in encrypted network traffic

With the advent and rapid adoption of network encryption mechanisms, typical deep packet inspection systems that monitor network packet payload contents are becoming less effective. Advancing intrusion detection tools to be also effective in encrypted networks is crucial. In the context of Task 3.3, FORTH examines the state-of-the-art in encrypted traffic analysis and proposes a methodology to automatically mine signatures for intrusion detection in encrypted networks. More specifically, FORTH:

- Reviews the state-of-the-art in the domain of encrypted traffic analysis for intrusion detection.
- Generates a ground-truth dataset with network traces that contain malicious activity.
- Processes the collected network traces and infers the most prevalent features contained in network packet headers.
- Analyses the exported data.
- Employs an algorithm for discovering sequential sequences of packet metadata inside a network flow.
- Concludes on a representation format and proposes a simple and mineable signature language.
- Extends the implementation of a deep packet inspection engine to support signatures for encrypted traffic.

3.2 Advantages of data normalization

As it has been described in this document, working with various sources can be quite challenging. So, when dealing with different kinds of inputs and formats, normalization becomes a necessity. The fact that solutions from several manufacturers are involved in the CyberSANE solution provides data which has to be standardized if is going to be exploited. Data normalization has helped through the process of standardization in several ways:

✓ Detaching what is necessary from what is dispensable, that is, reducing the amount of data to be processed and, therefore, time required.
✓ Funnelling data to the output.
✓ Producing an expected, structured output.
✓ Avoiding confusion and disruption with regard to data treatment.

However, sometimes normalization process is neither easy nor straightforward. Its complexity can be quite changeable because it is based on how the input arrives. Analysts need to devote some time on adjusting the way conversion must be done, typically by means on relying on the proper pipelines to do the hard work.

3.3 Architecture and ECS

A proper, conceived architecture should be the backbone of any project, and data normalization is no different in this point. It is essential to think carefully and design a solid architecture to make the most of data normalization.

As per the proposed architecture, the objectives had a double-side perspective:

1. Fulfil requirements of incident information normalization for every one of the sources involved in the process of data gathering.
2. Keep architecture as simple as possible and try to avoid unnecessary complexity.

When designing the architecture, these two goals have been kept in mind permanently. As per the results obtained, the objectives have been achieved.

With regard to Elastic Common Schema, choosing ECS as the standard output format was easy: a free framework for data modelling which is flexible, powerful enough and customizable. ECS provides versatility to keep things simple but, at the same time, fits with different input formats while allowing an easy mapping. It is always advisable to turn to free components which are, frequently, quite configurable and useful and provide no fewer functionality compared to the privative ones.

Chapter 4 List of Abbreviations

Abbreviation	Translation
Adversary TTP	Adversary Tactics, Techniques and Procedures
APT	Advanced Persistent Threat
ASIC	Application-Specific Integrated Circuit
CEF	Common Event Format
CSIRT	Computer Security Incident Response Team
DDoS	Distributed Denial of Service
ECS	Elastic Common Schema
FPGA	Field-Programmable Gate Array
IDS	Intrusion Detection System
IODEF	Incident Object Description Exchange Format
IRT	Incident Response Team
OSSEC	Open Source HIDS SECurity
SDN	Software Defined Network
SIEM	Security Information and Event Management
STIX	Structured Threat Information eXpression

TCAM	Ternary Content Addressable Memory

Chapter 5 Bibliography

[1]. Aho, Alfred V. and Margaret J. Corasick. "Efficient string matching: an aid to bibliographic search." *Communications of the ACM 18.6* (1975): 333-340.

[2]. Anderson, Blake and David McGrew. "Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity." *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017. 1723–1732.

[3]. Conti, Mauro, et al. "Can't you hear me knocking: Identification of user actions on android apps via traffic analysis." *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*. ACM, n.d. 297-304.

[4]. Fournier-Viger, Philippe, et al. "VMSP: Efficient vertical mining of maximal sequential patterns." *Canadian conference on artificial intelligence.* . Springer, n.d. 83–94.

[5]. Papadogiannaki, Eva, Dimitris Deyannis and and Sotiris Ioannidis. "Head (er) Hunter: Fast Intrusion Detection using Packet Metadata Signatures." *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD).* IEEE, 2020. 1-6.

[6]. Papadogiannaki, Eva, et al. "OTTer: A Scalable High-Resolution Encrypted Traffic Identification Engine." *nternational Symposium on Research in Attacks, Intrusions, and Defenses. (RAID).* Springer, 2018. 315–334.

[7]. Shen, Jie, et al. "Performance Traps in OpenCL for CPUs." *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing.* 2013.

[8]. Dobrescu, Mihai, et al. "RouteBricks: Exploiting Parallelism to Scale Software Routers." *22nd ACM Symposium on Operating Systems Principles.* 2009.

[9]. Han, Sangjin, et al. "PacketShader: a GPU-accelerated software router. ." *Proceedings of SIGCOMM.* 2010.

[10]. Papadogiannaki, Eva, et al. "Efficient software packet processing on heterogeneous and asymmetric hardware architectures." *IEEE/ACM Transactions on Networking 25* (2017): 1593–1606.

[11]. Vasiliadis, Giorgos, Michalis Polychronakis and Sotiris Ioannidis. "MIDeA: A Multi-Parallel Intrusion Detection Architecture." *Proceedings of the 18th ACM Conference on Computer and Communications Security.* ACM, 2011.

[12]. Fournier-Viger, Philippe, et al. "Fast vertical mining of sequential patterns using cooccurrence information." *Pacific-Asia Conference on Knowledge Dis- covery and Data Mining*. Springer, 2014.

[13]. Rizzo, Luigi, Marta Carbone and and Gaetano Catalli. "Transparent acceleration of software packet forwarding using netmap." *2012 Proceedings IEEE INFOCOM*. IEEE, 2012. 2471–2479.

[14]. Lotfollahi, Mohammad, et al. "Deep packet: A novel approach for encrypted traffic classification using deep learning." *Soft Computing* (2017): 1-14.

[15]. Nicolás Rosner, Ismet Burak Kadron, Lucas Bang, and Tevfik Bultan. n.d.

[16]. Rosner, Nicolás, et al. "Profit: Detecting and Quantifying Side Channels in Networked Applications." *NDSS*. 2019.

[17]. Vasiliadis, Giorgos, et al. "Gnort: High Performance Network Intrusion Detection Using Graphics Processors." *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection.* 2008.

[18]. Paxson, Vern, Robin Sommer and Nicholas Weaver. "An architecture for exploiting multi-core processors to parallelize network intrusion prevention." *2007 IEEE Sarnoff Symposium*. IEEE, 2007. 1-7.

[19]. Vallentin, Matthias, et al. "The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware." *International Workshop on Recent Advances in Intrusion Detection.* Springer, 2007. 107–126.

[20]. Meiners, Chad R, et al. "Fast regular expression matching using small TCAMs for network intrusion detection and prevention systems." *Proceedings of the 19th USENIX conference on Security*. USENIX Association, 2010.

[21]. Sourdis, Ioannis and Dionisios Pnevmatikatos. "Pre-decoded CAMs for efficient and high-speed NIDS pattern matching." *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2004. 258–267.

[22]. Sherry, Justine, et al. "Blindbox: Deep packet inspection over encrypted traffic." *ACM SIGCOMM Computer communication review 45, 4* 2015: 213–226.

[23]. Ning, Jianting, et al. "PrivDPI: Privacy-Preserving Encrypted Traffic Inspection with Reusable Obfuscated Rules." *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security.* ACM, n.d. 1657–1670.

[24]. Shone, Nathan, et al. "A deep learning approach to network intrusion detection." *IEEE Transactions on Emerging Topics in Computational Intelligence 2, 1* (2018): 41–50.

[25]. Tang, Tuan A, et al. "Deep learning approach for network intrusion detection in software defined networking." *016 International Con- ference on Wireless Networks and Mobile Communications (WINCOM)*. IEEE, 2016. 258–263.

[26]. Niyaz, Quamar, Weiqing Sun and Ahmad Y Javaid. "A deep learning based DDoS detection system in software-defined networking." *arXiv preprint arXiv:1611.07400* . 2016.

[27]. Amoli, ayam Vahdani, et al. "Unsupervised network intrusion detection systems for zero-day fast-spreading attacks and botnets." *International Journal of Digital Content Technology and its Applications 10, 2* (2016): 1–13.

[28]. Mirsky, Y., et al. "Kitsune: an ensemble of autoencoders for online network intrusion detection." 2018.

[29]. Conti, Mauro, et al. "Analyzing android encrypted network traffic to identify user actions. ." *EEE Transactions on Information Forensics and Security 11, 1* (2016): 114–125.

[30]. Taylor, Vincent F, et al. "Robust smartphone app identification via encrypted network traffic analysis. ." *EEE Transactions on Information Forensics and Security 13, 1 (2017)* (2017): 63-78.

[31]. Orsolic, Irena, et al. "Youtube QoE estimation based on the analysis of encrypted network traffic using machine learning." *IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2016. 1–6.

[32]. Mazha, M Hammad and Zubair Shafiq. "Real-time video quality of experience monitoring for https and quic." *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. 2018. 1331–1339.

[33]. Xu, Shichang, Subhabrata Sen and Z Morley Mao. "CSI: inferring mobile ABR video adaptation behavior under HTTPS and QUIC." *Proceedings of the Fifteenth European Conference on Computer Systems.* 2020. 1-16.

[34]. Khokhar, Muhammad Jawad, Thibaut Ehlinger and Chadi Barakat. "From Network Traffic Measurements to QoE for Internet Video." *2019 IFIP Networking Conference (IFIP Networking).* IEEE, 2019. 1-9.

[35]. Ghiëtte, Vincent, Harm Griffioen and Christian Doerr. "Fingerprinting tooling used for {SSH} compromisation attempts. ." *22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019).* 2019. 61-71.

[36]. Goh, Vik Tor, Jacob Zimmermann and MarkLooi. "Experimenting with an intrusion detection system for encrypted networks." *International Journal of Business Intelligence and Data Mining 5, 2* (2010): 172–191.

[37]. Goh, Vik Tor, Jacob Zimmermann and Mark Looi. "Intrusion detection system for encrypted networks using secret-sharing schemes." *International Journal of Cryptology Research (2010).* (2010).

[38]. Abbas Razaghpanah, Narseo Vallina-Rodriguez, et al. "Haystack: In situ mobile traffic analysis in user space." 2015.

[39]. Kotzias, Platon, et al. "Coming of age: A longitudinal study of tls deployment." *Proceedings of the Internet Measurement Conference 2018*. 2018. 415–428.

[40]. Zhai, Ennan, et al. "AnonRep: Towards Tracking-Resistant Anonymous Reputation." *NSDI*. 2016. 583–596.

[41]. Chen, Chen, et al. "TARANET: Traffic-Analysis Resistant Anonymity at the NETwork layer." 2018.

[42]. Frolov, Sergey and Eric Wustrow. "The use of TLS in Censorship Circumvention." *NDSS*. 2019.

[43]. Wang, Tao and Ian Goldberg. "Walkie-talkie: An efficient defense against passive website fingerprinting attacks." *26th {USENIX} Security Symposium ({USENIX } Security 17).* USENIX Association, 2017. 1375–1390.

[44]. Hooff, Jelle Van Den, et al. "Vuvuzela: Scalable private messaging resistant to traffic analysis." *Proceedings of the 25th Symposium on Operating Systems Principles.* ACM, 2015. 137–152.

[45]. Kwon, Albert, et al. "Atom: Horizontally scaling strong anonymity. ." *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017. 406–422

[46]. Common Event Format, ldapwiki. Available online: https://ldapwiki.com/wiki/Common%20Event%20Format.

[47]. CEF - Event Interoperability Standards. Artificial Intelligence and Big Data in Cyber Security. Available online: https://raffy.ch/blog/2007/07/18/cee-cef-event-interoperability-standards/

[48]. Understanding the Syslog - Common Event Format (CEF) forwarder mappings in ICDx. Available online: https://help.symantec.com/cs/ICDX_1.4/ICDX/v131903235_v133742888/Understanding-the-Syslog-Common-Event-Format-(CEF)-forwarder-mappings-in-ICDx?locale=EN_US

[49]. Common Event Format (CEF) Logging Support in the Application Firewall. Available online: https://support.citrix.com/article/CTX136146

[50]. HPE SecurityArcSightCommon EventFormat. Available online: https://www.secef.net/wp-content/uploads/sites/10/2017/04/CommonEventFormatv23.pdf

[51]. Structured Threat Information eXpression — STIX™A Structured Language for Cyber Threat Intelligence Information. Available online: https://makingsecuritymeasurable.mitre.org/docs/stix-intro-handout.pdf

[52]. What is STIX. Available online: https://searchsecurity.techtarget.com/definition/STIX-Structured-Threat-Information-eXpression

[53]. The Incident Object Description Exchange Format. Available online: https://tools.ietf.org/html/rfc5070#section-1

[54]. Elastic Common Schema – talking the same data language. Available online: https://www.siscale.com/elastic-common-schema-talking-the-same-data-language/

[55]. Aleroud, A. & Karabatis, G., 2013. Toward zero-day attack identification using linear data transformation techniques. Gaithersburg, Maryland, IEEE, pp. 159-168.

[56]. Almeida, J. F., Barbi, M. & do Vale, M., 2000. A proposal for a different chi-square function for Poisson distributions. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment,* 449(1-2), pp. 383-395.

[57]. Alom, M. & Taha, T., 2017. *Network intrusion detection for cyber security using unsupervised deep learning approaches.* Dayton, Ohio, IEEE, pp. 63-69.

[58]. Al-Rowaily, K., Abulaish, M., Haldar, N. & Al-Rubaian, M., 2015. BiSAL–A bilingual sentiment analysis lexicon to analyze Dark Web forums for cyber security. *Digital Investigation,* Volume 14, pp. 53-62.

[59]. Bartlett, M., 1936. The square root transformation in analysis of variance. *Supplement to the Journal of the Royal Statistical Society,* 3(1), pp. 68-78.

[60]. Box, G. & Cox, D., 1964. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological),* 26(2), pp. 211-243.

[61]. Di Sarno, C., Garofalo, A., Matteucci, I. & Vallini, M., 2016. A novel security information and event management system for enhancing cyber security in a hydroelectric dam. *International Journal of Critical Infrastructure Protection,* Volume 13, pp. 39-51.

[62]. Ferreira, P., Le, D. & Zincir-Heywood, N., 2019. *Exploring Feature Normalization and Temporal Information for Machine Learning Based Insider Threat Detection.* Halifax, Canada, IEEE, pp. 1-7.

[63]. Freeman, M. & Tukey, J., 1950. Transformations related to the angular and the square root. *The Annals of Mathematical Statistics,* pp. 607-611.

[64]. Gashteroodkhani, O. et al., 2019. A protection scheme for microgrids using time-time matrix z-score vector. *International Journal of Electrical Power & Energy Systems,* Volume 110, pp. 400-410.

[65]. Google Developers, 2020. *Data Preparation and Feature Engineering for Machine Learning - Normalization.* [Online] Available at: https://developers.google.com/machine-learning/data-prep/transform/normalization [Accessed 14 12 2020].

[66]. Liu, Y. et al., 2015. *Predicting cyber security incidents using feature-based characterization of network-level malicious activities.* Richardson, Texas, ACM, pp. 3-9.

[67]. Meira, J. et al., 2020. Performance evaluation of unsupervised techniques in cyber-attack anomaly detection. *Journal of Ambient Intelligence and Humanized Computing,* 11(11), pp. 4477-4489.

[68]. Molina, A., Natarajan, S. & Kersting, K., 2017. *Poisson Sum-Product Networks: A Deep Architecture for Tractable Multivariate Poisson Distributions.* San Francisco, California, AAAI, pp. 2357-2363.

[69]. Osborne, J., 2002. Notes on the use of data transformations. *Practical assessment, research, and evaluation,* 8(1), p. 6.

[70]. Osborne, J., 2010. Improving your data transformations: Applying the Box-Cox transformation. *Practical Assessment, Research, and Evaluation,* 15(1), p. 12.