Journal of Computer Security 27 (2019) 507–520
DOI 10.3233/JCS-191286
IOS Press


# Using side channel TCP features for real-time detection of malware connections

George Stergiopoulos, Georgia Chronopoulou, Evangelos Bitsikas, Nikolaos Tsalis and Dimitris Gritzalis *
*Information Security and Critical Infrastructure Protection (INFOSEC) Laboratory, Department of Informatics, Athens University of Economics and Business, Athens, Greece*
*E-mails: geostergiop@aueb.gr, chronopoulou.georgia@gmail.com, vaggelisbtks@gmail.com, ntsalis@aueb.gr, dgrit@aueb.gr*

**Abstract.** During the past years, deep packet inspection has been prevalent in network intrusion detection systems. Most solutions employ complex algorithms to analyze the intended behaviour and underlying characteristics of packets and their payloads, in an effort to detect and prevent malicious users and software from communicating over business intranets and wider networks. Still, there are multiple issues that inhibit their success rate. Most signature-based security software is plagued by false positives and/or false negatives. On the other hand, behavioral-based solutions achieve better detection rates but need to analyze large amounts of traffic. In this article, we present a real-time network traffic monitoring system that implements machine learning over side channel characteristics of TCP network packets to distinguish normal from malicious TCP sessions, even when encryption is in place. We test in university networks and test multiple different types of traffic. We show that, our approach (i) requires notably less information to achieve similar (if not better) detection rates, (ii) works over encrypted traffic as well, and (iii) has notably low false positives and false negatives in everyday case study scenarios.

Keywords: Malware traffic, malware detection, machine learning, defacement, CART, botnet, reverse shells, trojan

## 1. Introduction

Detecting advanced and highly customized ongoing malicious attacks is an open area for research; both from academia and the industry. Constant updates on security solutions, even complex algorithms seem to fail to detect zero-day malware in most attack vectors, from web app hacking to typical buffer overflows and Distributed Denial of Service attacks (DDoS).

Deep packet inspection has been prevalent in network intrusion detection systems. Most solutions employ multiple, complex algorithms to analyze the intended behaviour and underlying characteristics of network connections, in an effort to detect and prevent malicious users and software from communicating over business intranets and wider networks.

Still, there are multiple issues that inhibit their success rate. The use of encryption in almost all connections nowadays (especially now that the EU's General Data Protection Regulation (GDPR) [12] is in effect) inhibits detection mechanisms, since encryption introduces high amounts of randomness and entropy on data transferred. To make matters worse, the GDPR will gradually enforce encrypted connections everywhere. Security solutions find it difficult to keep up with encrypted malware and most exhibit high rates of false negatives.

*Corresponding author. E-mail: dgrit@aueb.gr.

Another issue is the huge amount of data currently being transferred over modern networks. The sheer volume of packets makes it impossible to analyze all packets in existing connections. Instead, most solutions sample underlying connections and try to understand their nature by a small number of, often not so indicative, group of packet captures.

Steal attacks are another issue that modern network security measures have difficulty coping up with. These attacks are characterized by the ability to fine tune actions to avoid detection [4]. All aforementioned issues are open challenges in network security.

Side channels are a new type of attack that became popular during the past decade. They focus mostly on information disclosure through indirect information channels such as timing information, power analysis and electromagnetic leaks [22]. Concerning side channel attacks on encrypted traffic, previous research was successful from an attacker's perspective, mostly by exploiting SSH [32], VoIP [38] etc. In [5], researchers exploited timing characteristics of web applications to distinguish traffic and leak sensitive personal information of users.

### *1.1. Contribution*

We present a network traffic monitoring system that implements machine learning over network captures to distinguish normal from multiple types of malicious TCP/IP traffic, even when encryption is in place. Our machine learning approach uses side channel characteristics of TCP packets (such as payload size, ratios, time between multiple packet chains etc.) to actively monitor TCP sessions in real-time. A few similar efforts have been made for classification of some types of malicious traffic (e.g. encrypted malware traffic), yet existing methods rely mostly on complex feature selection and/or large datasets. Overall, the main contributions of this article are summarized as follows:

(1) We demonstrate an implementation that can assess real-time network connections with faster offline training and classification than other similar detection systems due to the smaller feature set. Minimum session samples are required to achieve convenient detection rates and low false positive/negative classification.
(2) Use of side channel features greatly reduces the size of traffic data that needs to be analyzed for training or detecting of various types of malicious traffic (not only encrypted malware), which seems to dominate current research projects.
(3) We manage to simultaneously detect multiple types of malicious traffic in real-time (unencrypted and encrypted malware traffic and/or shellcode connections, DDOS traffic attacks, etc.) using network packet attributes as features instead of complex features or deep packet inspection. Selected attributes mostly relate to side channel attacks (e.g. timing ratios, payload size ratios etc.).
(4) We still achieve the same overall detection rates as previously observed in [33] although now we analyze real-time network TCP sessions that are comprised by radically different types of traffic, instead of sampled sets of normal traffic.

Section 2 presents related work concerning malicious network traffic and similar classification approaches and argues about the differences with our presented system. Section 3 describes the datasets utilized in the current project and presents our data sanitization process. Section 4 presents the detection methodology implemented in the proposed system. Section 5 describes our experimental results, while Section 6 discusses further improvements and potential future work.

## 2. Related work

Mainstream approaches to detecting malicious traffic mostly rely on heuristic analysis of packets, payloads and session trends (like packets per min) along with botnet architecture [3,7,19]. Others rely on statistical analysis for classifying various types of traffic[8].

Our approach is similar to [7] and [6]. In [7], researchers utilize some of the same features as we do to extract information from the physical aspects of the network traffic. They too utilize machine learning but focus on OSI layer 7 features to distinguish between malicious and normal encrypted traffic. Thus, significant differences exist. The main differences of our work with [11] and [6] are the following: (a) We do not restrict our machine learning and detection system only to encrypted traffic but try to achieve similar (or even better) detection rates without distinguishing between different malicious traffic, (b) we provide full payload analysis per packet and in relation to previous packets sent, whereas researchers in [11] analyze tuples that check payload sizes for entire originator-responder sessions, (c) we minimize selected features by only using the ones that refer to side channel characteristics, while achieving better results, and (d) we do not aim to only understand and distinguish malicious encrypted traffic from malware but extend this to multiple types of both encrypted and unencrypted malicious traffic, ranging from defacing attacks, reverse shells, encrypted connections etc.

Cisco published a white paper concerning new advancements in detecting malicious traffic using similar side channel features [6]. Cisco utilizes similar types of data elements or telemetry that are independent of protocol details, such as the lengths and arrival times of messages within a flow. Their technology supports various Cisco routers and switches to perform detection of malicious traffic in network sessions that utilize the Transport Layer Security (TLS) protocol.

In summary, we manage similar performance and smaller datasets utilizing only five features on side channel characteristics, twenty two (22) features less than [11] and four (4) less than Cisco [6].

Authors in [27] also use malicious HTTPS traffic to train neural networks and sequence classification to build a system capable of detecting malware traffic over encrypted connections. Similarities with our work is that we use features to train a machine learning algorithm. The difference with our work is that: (a) we are able to detect multiple types of malicious traffic and not only encrypted malware traffic and (b) we utilize less data (and corresponding domain features) while achieving better and faster results, albeit not only on encrypted traffic but on a dataset consisting of 200K traffic samples of different malicious traffic flows.

Using CART decision algorithm instead of neural networks, we can achieve faster classification once the system is trained and have a more interpretable model to detect hidden interconnections of traffic features. On the other hand, neural networks might be more accurate (although our preliminary results do not support this), provided there is enough training data, although they can be prone to over-fitting as well; this is another reason why we tested other algorithms more suitable to unknown dataset characteristics.

Other approaches in analyzing encrypted HTTPS traffic are few [16,20]. Most of them focus on identifying target malware/botnet servers [20] or web servers contacted [39], instead of understanding malicious traffic of various types.

The following publications are worth mentioning although they differentiate and either utilize different technologies to achieve similar goals, or aim to analyze different aspects of network traffic albeit with similar algorithms. Authors in [13] and [39] utilize signal processing techniques (e.g., Principal Component Analysis (PCA)) to create aggregates of traffic and payload inspection data, in an effort to

detect anomalous changes to network flows [35]. They utilize a distance metric to understand network-change patterns in traffic. Lakhina et al. [17] modelled network flows as combinations of eigen flows to distinguish between short-lived traffic bursts, trends, noise, or normal traffic. Terrell et al. [36] grouped network traces into time-series and selected features, such as the entropy of the packet and port numbers, to detect traffic anomalies. While these approaches are based on models of malware behavior (not unlike signature-based intrusion detection), our approach seeks to identify important features on the physical characteristics of malicious network sessions and utilize them to train machine learning algorithms. This way, we increase the detection rate by (a) not relying on instances of malware traffic to understand future malware and (b) by creating a trained model that predicts the value of a network TCP sessions based on network values of several input (or independent variables). Our approach is non-parametric, therefore it does not rely on data belonging to a particular type of distribution. Also, it can utilize variables multiple times in different decision analyses, thus uncovering complex interdependencies between sets of variables [37].

The selected machine learning algorithm and relevant network features enhance malicious traffic detection in both encrypted and unencrypted traffic, ranging from a series of different malicious types such as botnets, defacement attacks, reverse shells, Trojans, etc. To our knowledge, no other prototype is able to accomplish this.

This article builds on our previous research [33]. The basic principals for creating the presented traffic monitoring system sprang from the aforementioned publication. Still, there are major differences on the size and type of experiments conducted. Specifically:

- Instead of using a split dataset to train and test multiple AI algorithms for detection rates, here we only use the best algorithms detected (namely, CART) to delve deeper into how detection rates apply on multiple amounts of data.
- Current experiments involve real-time capturing and detection of malicious traffic instead of training and classification on given datasets.
- the captured flow is split for each TCP connection and the prediction rates is a % result per session and not as a total outcome of current network sample.
- for optimization purposes, a CSV database is used for training and updating the system instead of MariaDB.

## 3. Datasets

Datasets of malicious network connections along with normal traffic were used in [33] to train and test the machine learning algorithm with the side-channel attributes (features) we selected. Here, we use the same datasets for preliminary training but then we move on to add more types of malicious network traffic, both handcrafted and publicly available. Most datasets used are public and contain traffic of real malware, defacing attacks, reverse shells and software exploitation attacks along with normal traffic. For reference, the following public datasets were used for training:

- FIRST 2015 dataset [14]. A collection of 4.4 GB pcap files containing normal as well malicious traffic. Traffic is composed from Reverse Shell shellcode connections, website defacing attacks, ransomware downloaded attack cryptolocker and a command and conquer exploit attack (C2) over SSL that takes over the victim machine.
- Milicenso dataset [23] containing normal and malware traffic for the Ponmocup Malware. It contains malicious traffic from a malware/trojan that connects the victim PC on a botnet.

- CTU13-1 dataset [9] containing Botnet Traffic of the Neris Botnet. All traffic is mostly encrypted botnet traffic, because the normal traffic that was captured at the same time is not public.
- PCAP file with PowerShell Empire (TCP 8081) and SSL wrapped C2 (TCP 445) traffic from CERT.SE [26].
- Metasploit Meterpreter reverse shells (TCP and encrypted connections) with various payloads.

The preparation of the dataset involved going through multiple sources of malicious TCP traffic and split them all up with recursive labeling. Each dataset provided network files with reports that document "malicious" IP addresses and what type of attacks they are deploying, so labeling the dataset for training the classifier was easy. Dataset traffic was included in pcap and pcapng files containing captured packets. Attacks captured by the malicious traffic are the following:

- Full website defacement attack, from injection to file manipulation (FrogSquad defacement).
- Webshell (PHP backdoor) on infected web server with multiple commands using cm0 backdoor.
- Spear Phising email attack. APT4711 spear phishing email to Krusty (192.168.0.54).
- Click Fraud attacks. Spambot connected to an HTTP CC to do some Click Fraud.
- IRC communication for spam and click-jacking. Neris botnet that run for 6.15 hours in a University network. The botnet used an HTTP based command & control (C&C) channel and not an IRC C&C channel as it was erroneously reported before.Send SPAM and perform click-fraud using some advertisement services.
- Malware traffic. The machine was successfully infected with POST requests. Malware connect to CnC server using a raw TCP connection.
- Encrypted malware traffic. HTTPS and SSH traffic from CTU-13.
- UDP and ICMP DDoS from CTU-13.
- Trickbot banking Trojan.Trickbot (Trojan.Trickybot) C2 over HTTPS.
- CERT.SE malware attack files included traffic from PowerShell scripts that were executed to give access to victim's computer. Scripts were created with publicly available tools (Metasploit and PowerShell Empire). Attackers also used tools to increase rights in the system, be able to move from machine to machine in the network, dump passwords and access systems and data.
- Meterpreter shell attacks from infected systems were used to create folders, dump passwords and manipulate the filesystem. Shells included /meterpreter/bind_tcp, /meterpreter/reverse_https and /meterpreter/reverse_tcp payloads from Kali linux's Metasploit Framework [15].

## 4. Detection methodology

### *4.1. Problem definition*

Given a TCP/IP network traffic flow, our system aims to sample and classify each connection as malicious (i.e. produced by malicious events such as a web attack or malware) or normal ("white" traffic). The end-system is comprised of two parts: (i) the traffic flow side channel feature selector and (ii) the network traffic classifier.

(1) Side channel feature selection: The first task is to choose correct, descriptive features of TCP traffic that do not refer to the content of a packet, but rather to the physical characteristics, such as time ratio between packet sending, size of payload etc.

(2) Traffic classification: The second task is to use the selected features to classify each TCP connection as malicious or normal. Each TCP session is sampled and captured on an individual .pcap file. Then, the trained CART classifier is executed on that data and classifies each session (i.e. pcap file) as malicious or not.

We do not aim to distinguish between types of malicious traffic. It is our belief that human interaction and digital forensics will always provide better solutions in dissecting security events. Instead, our system aims to warn against any potential malicious traffic for response teams to take action. Our classifier can easily be trained with new traffic in order to be up to date.

### *4.2. Feature selection*

In this subsection we discuss the features we selected to feed into our Machine-Learning algorithms and the rationale of the proposed system. We use features based on side channel characteristics of TCP traffic to analyze packet-to-packet sequences inside network sessions.

It is known that for any set of features, "*there will be a fundamental limit to the kind of determinations a NIDS can develop from them*" [31]. Choosing a correct set of features must always take into account the diversity of normal as well as malicious traffic. A good approach is to examine the invariance of features in diverse malicious traffic scenarios [31]. To this end, we opted to base our feature selection on previous publications [5,6,11] that utilized similar side-channel packet features for similar purposes. Authors in [11] and Cisco [6] made extensive tests and concluded in similar albeit quite larger feature sets than us. Authors in [5] had previously used a subset of features also found in [11] and [6], albeit for different purposes (i.e. to leak sensitive information from web application content).

Our intuition was that, the intersection of these features sets could minimize the features needed for the detection of malicious traffic, while at the same time achieve the same results. Also we believed that the same feature set could expand potential malicious traffic detection beyond encrypted malicious traffic; which was the focus of [11] and [6]. Thankfully, we found that these types of features are enough to identify malicious traffic. Since these features do not require complex aggregation of information, the runtime footprint is small and the system can be easily adapted to analyze traffic in real-time. Overall, we opt for five features on side channel characteristics, twenty two (22) features less than [11] and four (4) less than Cisco [6].

**Packet Size (Ps):** Every connection is defined by the packets exchanged between a sender and a receiver. Packet size is known to be good both for predicting the type of connection and protocols used [18]. For that reason this is a basic feature of our project.

**Payload Size (PAs):** It is a feature that defines a packet. The payload is the heart of any malicious traffic. In TCP, the payload is enclosed in the TCP Data Segment. Research has shown that side channel analysis of payload sizes can be used as a feature for information leakage [5].

**Payload Ratio (Pr):** It refers to the ratio of the payload size to the total packet size. Malicious traffic can exhibit similar patterns concerning content ratios, so we opted to include this as a basic feature. The formula is shown below, where PAs refers to the payload size and Ps refers to the packet size:

$$Pr = \frac{PAs}{Ps} \tag{1}$$

**Ratio to Previous Packet (Rpp):** We noticed that, when malicious traffic flows inside the network, the packets are sequential and often exhibit specific trends in size. This can be used for fingerprinting

malicious traffic. By comparing two packets in a row that belong in the same session, we can get the ratio to the previous packet. The value defaults to 0 for the first packet of the session. The formula is shown below:

$$Rpp = \frac{Pp}{PPs} \tag{2}$$

where Pp refers to the current packet size and PPs to the previous packet size in the same session.

**Time Difference (Td):** The time difference between a packet and the previous packet of a session can be used to fingerprint malicious traffic. The value defaults to 0 for the first packet of a session. The formula is:

$$Td = Pt - PPt \tag{3}$$

Pt refers to Packet time and PPt to Previous Packet time. Both times refer to how long it took for a packet to be delivered.

**Flag:** A simple label that classifies the packet as either 0 or 1, where 0 stands for normal traffic and 1 for malware traffic.

### *4.3. Training the classifier*

Our proposed system utilizes offline training of a malicious traffic database that has stored packet traffic values from numerous malicious TCP connections. Each traffic flow contains a sequence of packets and corresponding sessions along with packet receiving time, packet receiving time, packet length and packet protocol type. We opted to use MLpack [28] library for our system.

Offline analysis (i.e. machine training) aims to extract traffic patterns of a given sniffed flow. The model used by the classifier is developed offline. The classifier used the CSV values for the training mode and exported an XML database that can be dynamically loaded and used in the prediction mode. This machine learning library uses Cython [30], Numpy [29] and Pandas [21].

From this point onward, each time our classifier detects new malicious TCP sessions or when we identify errors in classification, we update our training data with subsequent training executions (see Fig. 1).

### *4.4. Trials with existing tools*

During desktop research for the purpose of creating the real-time network traffic monitoring system, we conducted two sets of extensive tests on various tools and relevant libraries. We selected tools that are widely used on the field of network traffic monitoring. The first set of tests aimed at choosing the right tools and libraries for TCP session monitoring. We compared numerous tools based on their performance (speed of net flow capturing and resources needed) and on the types of output provided (should follow the ".pcap" form and not have a size limit).

Some of the selected tools for capturing live traffic were the following: Zeek (previously known as Bro) [25], Snort [2], Suricata [24] and Scapy [34]. All aforementioned tools are capable of sniffing network flows with negligible missing rates.

In the first set of trials, the most prevalent choice was Zeek [25]. Snort [2] is a very powerful tool with high probability of detecting an attack on networks up to 100 Mbps but the form of the output is

in binary form. On the other hand Suricata [24] is as powerful as Snort [2], since it uses a Snort rule-set to function. The issue here was that Suricata [24] needs lots of resources in order to operate properly and the outputs are mostly log files. On the contrary, Scapy [34], although capable of sniffing the flow of network connections and dump it on a ".pcap" file, it has serious issues concerning scalability when network sessions amount to dozens per second.

For our approach, we also needed a tool that can split TCP network traffic by session and provide different pcap files that contain different sessions per file. From the first round of trials, the tool that seemed superior for the tasks needed was Zeek [25]. This tool can split initial pcap files but, unfortunately, the splitted files have the form of raw data and can be easily processed only by hex editors. In the end, we opted to use PcapPlusPlus [29] only for its "PcapSplitter" module. For our needs, PcapSplitter was scalable enough, fast and could output TCP sessions in the form needed by our machine learning implementations.

In summary, we managed to build a sniffing engine of our owns using Libtins [10] and only use one tool for splitting. Tests showed that using more than one tool could affect in the functionality of our system.

### 4.5. *Real-time traffic classification*

Our system has three basic steps of operation, as depicted in Figs 1, 2 and 3:

(1) First, we load network traffic file (.PCAP extension file datasets) and extract all information needed by the presented machine-learning features above (e.g. such as time ration between sequential packets), for each packet from every captured malicious attacker session. This information is saved in a database with captured information on malicious network flows. We only keep the information needed from each packet in a TCP session and discard the rest (such as packet


Fig. 1. Workflow of the system from network capture to updating the scv database.


Fig. 2. Workflow of the system for training and exporting an xml database that is used for the prediction mode.

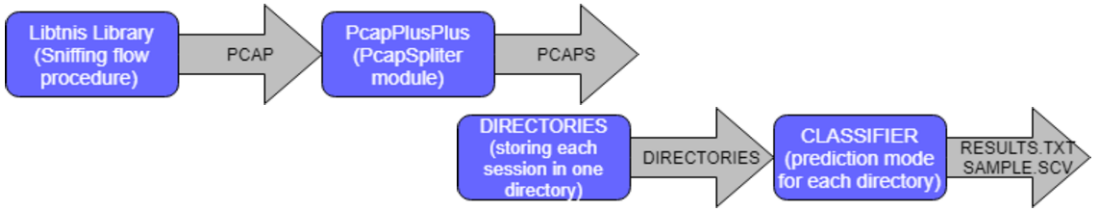


Fig. 3. Workflow of the system from network capture to exporting prediction for each TCP connection.

payloads or packet headers); namely we keep only information representing the attributes selected and presented above as features. Then we use this database to train the classifier and export (save) the result module to be used in the future as our prediction model and tool. The relevant process is depicted in Fig. 2. [33].

(2) Shown in Fig. 1, this process sniffs network traffic packets and sends them as a .pcap to the splitting tool to create captures of the flow for each TCP connection (session). Afterwards (displayed in Fig. 2) the trainer updating process is executed. The tool loads the new CSV database and sends values stored to our classifier. The classifier then re-trains to detect malicious traffic.

The above mentioned features and labelled traffic flows are used for training multiple algorithms. Deciding on a machine-learning algorithm was no trivial task. In [33], we opted to compare results between seven algorithms. Algorithms were selected as follows: We gathered all machine learning categories used in similar research [5,6,20] and detected their predictive model (e.g. decision tree, neural networks etc.). Then we opted to use the most efficient algorithms from each model area.

(3) The proposed monitoring system's main functionality is the prediction mechanism (see Fig. 3). This mode is a combination of existing tools and source code, all linked by an executable C++ software. The structure of this script is the following:

   (a) When initializing the script, it prints the network configuration information and the system awaits for input parameters (namely, the capturing interface and the number of packets to sample per session).
   (b) Then, the system starts the live capturing using Libtins [10] and stores the acquisition in a .pcap file.
   (c) Next, the initial .pcap file is splitted by PcapPlusPlus [29] to form different .pcaps per TCP connection. Each new file is stored in a separate directory with unique id number.
   (d) At this point the script generates the code that uses the mlpack library [23] to generate a "malicious traffic" prediction for each directory (and consequently for each TCP session). A "result.txt" file with the detection decision is printed inside the category, in the form of percentage of similarity (%).

Threads are used on this script in order to be scalable. The script responsible for the predictions requires a machine learning library (mlapck, namely) and is beeing reproduced by threads for each created directory. The creation of a thread occurs to a prediction of a TCP connection which can only be fulfilled by a comparison of the already classified values that are stored in a form of a xml database inside this script (see Fig. 2).

By gathering information from previous research [5,6,33], we present a small synopsis of machine learning approaches, which are suitable for this type of detection systems and which ones are suboptimal. We utilize an approach similar with [1] to summarize algorithms based on complexity (i.e. if the algorithm can understand complex or tailor-made attacks), Train time (i.e. time needed to train the classifier), speed of classification (i.e. how fast an algorithm classifies network samples), speed of learning (i.e. how fast an algorithm retrains itself with new network packets) and accuracy. See Table 1.

Our results agree with similar findings from [1], in that (i) SVM algorithms are efficient as detection engines for IDSes deployed in constrained networks, and (ii) neural networks are very efficient in detecting anomalies given that the algorithms are fed with adequate, not over-fitted, independent data from network sessions. Although CART is more vulnerable to tailored-made stealth attacks, still it seems that

Table 1
Comparison of algorithms

| Algorithm | Complexity | Train time | Speed of classification | Speed of learning | Accuracy |
|---|---|---|---|---|---|
| KNN | MEDIUM | FAST | $\ll$ 1 sec | FAST | 93% |
| CART | MEDIUM | FAST | $\ll$ 1 sec | FAST | 94% |
| Neural Networks | GOOD | SLOW | $\ll$ 1 sec | SLOW | 85% |
| Naive Bayes | BAD | FAST | $\ll$ 1 sec | FAST | 52% |

FAST = Function took less than 30 minutes
MEDIUM = Function took less than 90 minutes
SLOW = Function took more than 90 minutes
GOOD COMPLEXITY = Minimum false negatives in tailored, crafted attacks
MEDIUM COMPLEXITY = Some false negatives in tailored, crafted
BAD COMPLEXITY = Considerable false negatives in tailored, crafted

neural networks are prone to over-fitting if data are not selected properly and are much slower when multiple connections need to be analyzed almost real-time.

Using the CART decision algorithm instead of neural networks, we achieved faster classification of real-time TCP sessions once the system was trained and had a model to classify new traffic. On the other hand, neural networks might be more accurate, provided there is enough training data, although they can be prone to over-fitting as well; another reason why we tested other algorithms more suitable to unknown dataset characteristics. Preliminary tests showed that neural networks take a lot of time without having clear advantages over others neither in classification nor optimization.

## 5. Experimental results

The proposed system was tested on a Dell Inspiron 15-3537 (Intel Core i7-4500U, 8 GB RAM). Two types of detection experiments were conducted: (i) Monitoring mode of live traffic TCP connections (both malicious (Meterpreter malware) and normal) on the internal sub-network of a research laboratory in Athens University of Economics and Business, and (ii) offline classification of captured traffic, uploaded on our tool as a .PCAP file. Both experiments used random traffic from all types of datasets. The lab was comprised of thirteen (13) PCs. The second experiment involved classifying captured PCAP files of SSL-encrypted CnCs from other routers and PCs, that were afterwards sent to our tool for classification of samples. We fed the tool we traffic captured from other routers that involved machines infected with a PowerShell Empire payload that created an encrypted connection to an Empire server. The payload was triggered from the attack platform through Reflective DLL loading.

In both trials, we experimented with limited amounts of traffic to understand detection rates in small samples. Previous research [33] has proven that detection rates are pretty robust (88%–96%) for samples above 50,000 packets. Still, most real-world, silent attacks will only create a network traffic capture in the size of 3,000–8,000 TCP packets or even less. For this reason, we had to re-write the entire tool in C++, implement a real-time network sniffer/wrapper for capturing sessions and re-train the classifier using the new implementation with new samples.

Classification results are depicted at Table 2. Column 1 depicts the type of experiment conducted. Captions correspond to multiple, different TCP connections in the same type of malware / traffic. Column 2 shows the size in packets of each TCP session tested when conducted the relevant experiment type. Column 3 provides the malicious traffic detection rates achieved for the each experiment and relevant

Table 2
Detection rates – all tests

| Traffic name-type | Size in packets | Malicious rate before recent re-training | Malicious rate after recent re-training |
|---|---|---|---|
| Metasploit meterpreter caption 1 | 6362 | 60.61% | 98.33% |
| Metasploit meterpreter caption 2 | 6436 | 60.63% | 98.17% |
| Metasploit meterpreter caption 3 | 1078 | 61.04% | 98.98% |
| Metasploit meterpreter caption 4 | 946 | 62.90% | 99.26% |
| Metasploit meterpreter caption 5 | 920 | 64.13% | 99.35% |
| Metasploit meterpreter caption 6 | 850 | 69.41% | 99.05% |
| Metasploit meterpreter caption 7 | 827 | 71.46% | 99.27% |
| Metasploit meterpreter caption 8 | 1741 | 47.79% | 96.09% |
| Empire powershell malicious ssl traffic caption 1 | 992 | 55.24% | 99.10% |
| Empire powershell malicious ssl traffic caption 2 | 809 | 52.78% | 98.27% |
| Empire powershell malicious ssl traffic caption 3 | 279 | 44.09% | 96.42% |
| Empire powershell malicious ssl traffic caption 4 | 32 | 34.38% | 65.63% |
| Metasploit-ms017-010-win7x64 exploit caption(*) | 310 | 35.48% | 75.16% |
| Downloading an Ubuntu iso file (white traffic)(**) | 45853 | 0.002% | not required(***) |
| Random Browsing Caption 1 (white traffic) | 2445 | 11.411% | not required |
| Random Browsing Caption 2 (white traffic) | 2087 | 3.54% | not required |
| Random Browsing Caption 3 (white traffic) | 528 | 3.03% | not required |
| Skype call (white traffic) | 472 | 5.29% | not required |
| Skype call (white traffic) | 387 | 2.58% | not required |
| Youtube Caption 1 (white traffic) | 17627 | 3.03% | not required |
| Youtube Caption 2 (white traffic) | 3298 | 2.94% | not required |
| Wikipedia browsing (white traffic) | 2230 | 5.87% | not required |
| Wikipedia browsing (white traffic) | 2304 | 3.21% | not required |

(*)not included in re-training
(**)the rate is nearly zero (= 0.00218088%)
(***)"not required" means that percentages were very good and did not become better after re-training the classifier.

sessions, with the default training using the aforementioned datasets (see Section 3). Lastly, Column 4 depicts the same results after re-training the tool with short (smaller) TCP malicious sessions.

We re-trained the classifier with numerous small, malicious TCP sessions to experiment on whether size differences in training samples affect the detection rates. To our surprise, they do. The default training of the tool involved large TCP sessions with tens of thousands of packets. That default training still detected short, silent malicious connections, albeit without being 100% positive (detection rates ranged from ∼44% to ∼72%). By re-training the classifier with random malicious sessions but this time use ***both*** small and large sessions, resulted in striking differences in detecting silent malicious TCP sessions.

We should note here that the training traffic was not the same as the detection traffic. Captions presented in Table 2 were not included in training and were only used as real-world attacks to see if we would be able to detect them. We can see from Fig. 4 that malicious shells have common characteristics that allow us to properly detect them if proper training over normal (white) user traffic is performed. The presented differences in detection were mostly affected by the normal traffic, rather than adding more malicious traffic to the trained classifier.

Extended experiments on offline traffic taken from the datasets yields the following results, as also presented in [33]:

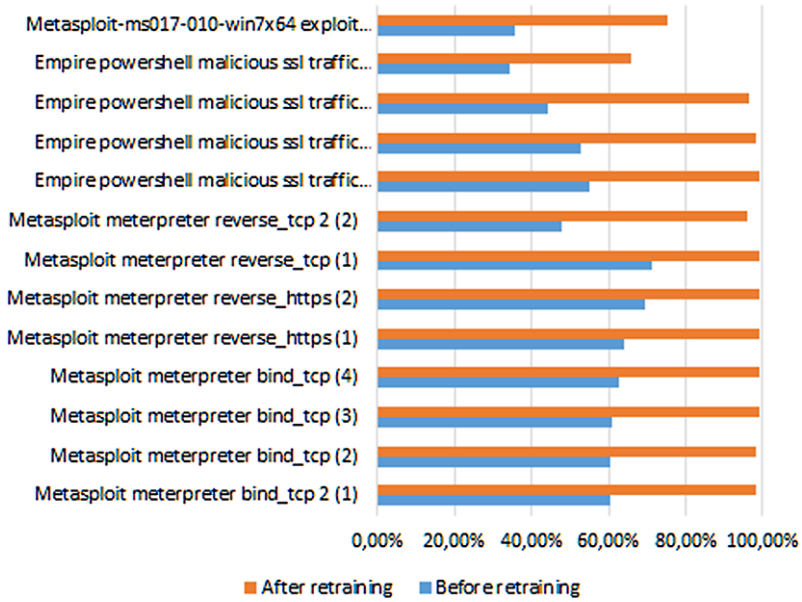


Fig. 4. Detection differences after retraining the classifier with new normal and malicious traffic.

- Datasets were split 70/30 and all ~9 GB of dataset traffic was utilized.
- CART gets a 96% best detection rate with 4% FP and 7% FN. Averages remain similar as in [33].

### 5.1. *Findings*

By viewing the detection rates, two things become obvious:

(1) Bigger samples have a clear advantage over small ones in detecting malicious traffic, and
(2) Training the tool only with large malicious TCP connections provides detection rates of 50%–60% for small sessions. As intuited, re-training the model with various small malicious sessions greatly improves detection rates for sessions below 10k packets.
(3) Training a machine-learning IDS with normal, everyday traffic is essential to achieve very high (>90%) detection rates. In most cases it is considered to be more useful than training with malicious traffic, if adequate attacks have been recorded.

Some of our most important findings were the following:

- The best detection rate achieved was about 97% on CART on full-scale mixed types of malicious data from live traffic analysis.
- Machine learning algorithms that utilize Decision Tree classifiers may be prone to packet crafting, if an attacker has access to the prediction model, parameters and the entire sample.
- Selected features describing side channel characteristics of TCP packets are adequate for building lightweight classification modules even for real-time intrusion detection that utilizes less traffic captures. We can achieve very good malicious traffic detection percentages without utilizing full-scale TLS and connection certification features.
- The use of side channel features significantly reduces the amount of white-traffic analysis and training period on normal network traffic. Instead of a few weeks time deployed by current solutions, effective training can be performed within a few hours.

Although the size of the sample is proportionate to the detection rate percentage, it is interesting to notice that detection rates do not follow a specific trend-line. In general, falling below 800 packet samples drops detection to levels below 50%, although in some cases, even 32 packets are enough for the session to score an interesting 35% for malicious traffic.

Another interesting finding is that downloading big files through normal sessions and software provides zero false positives. Big data transfers through legitimate traffic never score more than 0.02% in the tools detection algorithm.

## 6. Conclusions, findings and future work

In this paper, we presented a working prototype for large-scale enterprise networks using machine a decision tree algorithm for predicting whether each TCP connection that is being captured is malicious or normal. During our research we discovered that well-known traffic monitoring systems, like Zeek [25], Snort [2] and Suricata [24] do not work efficiently with session-based network traffic analyzers due to their immutable set up. The presented experiments adequately prove that side channel characteristics of TCP packets can be effectively used together with machine learning to detect most types of malicious traffic, even if wide differences exist on the types of ongoing attacks and to their corresponding traffic.

Some of our most important conclusions are the following:

- We detected specific, descriptive features describing side channel characteristics of TCP packets (such as packet size, delivery time ratios etc.) and built a real-time intrusion detection system able to run on real-world networks and detect ongoing malicious attacks to enhance network security.
- We conducted various experiments with multiple types of sample sizes and types of samples to understand the amount of analysis and network traffic that needs to be saved for detection. This enabled us to gain a better understanding of their network traffic and get robust alerts on security incidents without relying on error-prone IDPS pattern matching or heavy behavioral analytics.
- Instead of a few weeks time deployed by current solutions, effective training using the presented features can be performed within a few hours.

## References

[1] C. Alcaraz, L. Cazorla and G. Fernandez, Context-awareness using anomaly-based detectors for smart grid domains, in: *International Conference on Risks and Security of Internet and Systems*, Springer, 2014, pp. 17–34.
[2] J. Beale, A.R. Baker and J. Esler, *Snort: IDS and IPS Toolkit*, Syngress, 2007.
[3] J.R. Binkley and S. Singh, An algorithm for anomaly-based botnet detection, in: *SRUTI'06: Proceedings of the 2nd Conference on Steps to Reducing Unwanted Traffic on the Internet*, 2006.
[4] L. Cazorla, C. Alcaraz and J. Lopez, Cyber stealth attacks in critical information infrastructures, *IEEE Systems Journal* **12**(2) (2016), 1778–1792. doi:10.1109/JSYST.2015.2487684.
[5] S. Chen, R. Wang, X. Wang and K. Zhang, Side-channel leaks in web applications: A reality today, a challenge tomorrow, in: *2010 IEEE Symposium on Security and Privacy*, IEEE, 2010, pp. 191–206. doi:10.1109/SP.2010.20.
[6] Cisco, Encrypted Traffic Analytics, White paper, vol. Cisco. (20, 2018.
[7] E. Cooke, F. Jahanian and D. McPherson, The zombie roundup: Understanding, detecting, and disrupting botnets, *SRUTI* **5** (2005), 6–6.
[8] M. Crotti, M. Dusi, F. Gringoli and L. Salgarelli, Traffic classification through simple statistical fingerprinting, *ACM SIGCOMM Computer Communication Review* **37**(1) (2007), 5–16. doi:10.1145/1198255.1198257.
[9] Ctu-13 dataset, Ctu University, Czech Republic, 2011, https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-1/. Accessed: 2019-01-30.
[10] M. Fontanini, Libtins: Packet crafting and sniffing library, 2016.

[11] S. František, Detection of HTTPS malware traffic, 2017.
[12] General data protection regulation – gdpr, https://eur-lex.europa.eu/eli/reg/2016/679/oj. Accessed: 2019-06-10.
[13] G. Gu, P. Porras and V. Yegneswaran, Bothunter: Detecting malware infection through ids-driven dialog correlation, in: *Proceedings of the 16th USENIX Security Symposium (Security'07)*, 2007.
[14] Hands-on network forensics – training pcap dataset from first 2015, www.first.org/_assets/conf2015/networkforensics_virtualbox.zip. Accessed: 2019-01-30.
[15] D. Kennedy, J. O'gorman, D. Kearns and M. Aharoni, *Metasploit: The Penetration Tester's Guide*, No Starch Press, 2011.
[16] J. Kohout and T. Pevny, Automatic discovery of web servers hosting similar applications, in: *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015*, 2015, pp. 1310–1315.
[17] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E.D. Kolaczyk and N. Taft, Structural analysis of network traffic flows, *ACM SIGMETRICS Performance Evaluation Review* (2004).
[18] J. Liu, Y. Fu, J. Ming, Y. Ren, L. Sun and H. Xiong, Effective and real-time in-app activity analysis in encrypted Internet traffic streams, in: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2017, pp. 335–344. doi:10.1145/3097983.3098049.
[19] C. Livadas, R. Walsh, D. Lapsley and W.T. Strayer, Using machine learning techniques to identify botnet traffic, in: *Proceedings – Conference on Local Computer Networks, LCN*, 2006.
[20] J. Lokoč, J. Kohout, P. Čech, T. Skopal and T. Pevný, k-NN classification of malware in HTTPS traffic using the metric space approach, in: *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 131–145, 2016.
[21] W. McKinney and PyData Development Team, Pandas – powerful python data analysis toolkit, 2015.
[22] S. McLaughlin, C. Konstantinou, X. Wang, L. Davi, A.-R. Sadeghi, M. Maniatakos and R. Karri, The cybersecurity landscape in industrial control systems, *Proceedings of the IEEE* **104**(5) (2016), 1039–1057. doi:10.1109/JPROC.2015.2512235.
[23] Milicenso, ponmocup malware dataset (update 2012-10-07), http://security-research.dyndns.org/pub/botnet/ponmocup/analysis_2012-10-05/analysis.txt. Accessed: 2019-01-30.
[24] Open Information Security Foundation, Suricata: Intrusion detection system and intrusion prevention system, Stable release: 4.0.5 / July 18, 2018.
[25] V. Paxson, Zeek: Software network analysis framework, Stable release: 2.5.5 / August 28, 2018.
[26] Pcap file with powershell empire (tcp 8081) and ssl wrapped c2 (tcp 445) traffic from cert.se, https://drive.google.com/open?id=0B7pTM0QU5apSdnF0Znp1Tko0ams https://www.cert.se/2017/09/cert-se-tekniska-rad-med-anledning-av-det-aktuella-dataintrangsfallet-b-8322-16. Accessed: 2019-01-30.
[27] P. Prasse, G. Gruben, L. Machlika, T. Pevný, M. Sofka and T. Scheffer, Malware detection by https traffic analysis, 2017.
[28] C. Sanderson and R. Curtin, Armadillo: A template-based C++ library for linear algebra, *The Journal of Open Source Software* (2016).
[29] Seladb, PcapPlusPlus: Ultiplatform C++ network sniffing and packet parsing and crafting framework, Latest release: August 2018 Release (v18.08).
[30] K.W. Smith, *Cython: A Guide for Python Programmers*, O'Reilly Media, Inc., 2015.
[31] R. Sommer and V. Paxson, Outside the closed world: On using machine learning for network intrusion detection, in: *Proceedings – IEEE Symposium on Security and Privacy*, 2010.
[32] D. Song, Timing analysis of keystrokes and ssh timing attacks, in: *Proc. of 10th USENIX Security Symposium, 2001*, 2001.
[33] G. Stergiopoulos, A. Talavari, E. Bitsikas and D. Gritzalis, Automatic detection of various malicious traffic using side channel features on tcp packets, in: *Computer Security*, J. Lopez, J. Zhou and M. Soriano, eds, Springer International Publishing, Cham, 2018, pp. 346–362.
[34] J. Stretch, Scapy, Packetlife.net cheatsheets, 2017.
[35] C. Taylor and J. Alves-Foss, Nate – network analysis of anomalous traffic events, a low-cost approach, in: *Proceedings of the 2001 Workshop on New Security Paradigms – NSPW '01*, 2001, p. 89. doi:10.1145/508171.508186.
[36] J. Terrell, K. Jeffay, F.D. Smith, L. Zhang, H. Shen, Z. Zhu and A. Nobel, Multivariate SVD analyses for network anomaly detection, in: *(Poster) Proc. of ACM SIGCOMM*, 2005.
[37] R. Timofeev, Classification and regression trees (cart) theory and applications, Humboldt University, Berlin, 2004.
[38] C.V. Wright, L. Ballard, S.E. Coull, F. Monrose and G.M. Masson, Spot me if you can: Uncovering spoken phrases in encrypted voip conversations, in: *2008 IEEE Symposium on Security and Privacy (Sp 2008)*, IEEE, 2008, pp. 35–49. doi:10.1109/SP.2008.21.
[39] T.-F. Yen and M.K. Reiter, Traffic aggregation for malware detection, in: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2008, pp. 207–227. doi:10.1007/978-3-540-70542-0_11.