Structured Behavior Analysis on Encrypted Traffic

João Meira
Instituto Superior Técnico
Lisboa, Portugal
joao.meira@ist.utl.pt

ABSTRACT
The main objective of this work is to study network attacks. By profiling the inherent network behavior patterns of maliciously used software tools, we can detect the techniques that these tools implement without needing to specifically detect the tool based on its specificities.

It is developed and proposed a network feature extraction tool dubbed NetGenes, which considers a vast number of conceptual and statistical network communication features exclusively based on metadata extracted from L1-4 (OSI-Layer 1 to OSI-Layer 4) protocols. NetGenes takes a network trace-file (PCAP, PCAPNG) as an input, and extracts features of three network objects (flows, talkers and hosts) which build off of each other, logically aggregating lower-level network object features beneath them, and also enabling the creation of new features.

Then, we study various threat classes, organizing them in a taxonomy-like manner and outlining their encompassed threats, attack techniques and tools that implement them.

Moreover, we create various rule sets based on the network objects extracted by NetGenes, for the "Port Scan" threat class.

Finally, we apply the previously created rule sets to the CIC-IDS-2017 dataset, providing valuable insight about how to best detect the "Port Scan" threat class and its encompassed variants in a direct transparent manner.

Author Keywords
Network Security, TCP/IP Feature Extraction, Encrypted Network Traffic Analysis, Network Threat Hunting.

ACM Classification Keywords
Information Systems, Information Systems Applications, Data Mining, Association Rules.

INTRODUCTION
It is developed and proposed a network feature extraction tool dubbed NetGenes, which considers a vast number of conceptual and statistical network communication features exclusively based on metadata extracted from L1-4 (OSI-Layer 1 to OSI-Layer 4) protocols. It takes a network trace-file as input and extracts relevant data from it.
Various rule sets based on the network objects extracted by NetGenes are then created for the "Port Scan" threat class.

Finally, we apply the previously created rule sets to the CIC-IDS-2017 dataset.

RELATED WORK
In a small article written by Kevin Sheu for Infosec Island [1], he describes NetFlows as not being comprehensive enough in terms of cybersecurity features. He argues that NetFlow only look into layer-1 to layer-4 (L1-4) data ("layer-3 and layer-4 data", quoted from the article, obviously assumes layer-1 and layer-2 is also contemplated, since ethernet frames and the most common layer-2 protocols be contemplated as a basis for layer-3 and layer-4 protocols) and, thus, are not enough to go deeper in the connections themselves and gather protocol-specific features. Note that the NetFlow concept discussed comprises both NetFlow v1-9 and IPFIX (IP Flow Information eXport, a.k.a. NetFlow v10). Moreover, Kevin refers to Zeek [2] (a.k.a. Bro) network metadata as a superior solution in terms of knowledge depth (consequently, feature depth).

After researching about several tools, Tranalyzer-2 is the best flow extraction tool that we could find in terms of considered network-object features (it considers host, talker and flow features). It can extract information on a lot of protocols of different layers and contains up to 98 different flow features [3] at the network/transport layer level. It encompasses talker features and host features, based on Tranalyzer-2's latest documentation and presented flow aggregation techniques (mainly using tawk scripting) which they present in their website [4]. By communicating with the Tranalyzer-2 team and testing their tool, we could verify that Tranalyzer-2 now extracts 105 flow features by default, rather than the 98 flow features mentioned by Haddadi et. al [3].

Reading Cisco's article about End-to-End Visibility [7], one can see how Cisco FirePOWER and FireSIGHT can leverage NetFlows to obtain network intelligence at the L1-L4 level. It allegedly generates two useful types of event from L4-7 protocols' data, and two other types of event which are more poorly related to the L4-7 stack. It uses Snort, a signature-based NIDPS solution, to generate "Intrusion" events. Additionally, it outputs "Threat and Security" events as well, which combine both endpoint-based and network-based features to correlate OS events with network events, further used to perform host/user behavior score ranking and, additionally, to throw "Intelligence" events which are useful for cybersecurity experts to make informed decisions. Furthermore, the "Malware" event is a type of event which is outputted through an in-depth study of files received by an endpoint

system. Moreover, the "Anomaly" event is very strongly correlated to what this thesis aims to achieve, by detecting threats and threat classes. Threat classes are a logic aggregation of threats which, on the other hand, are a logic aggregation of software solutions, including malware variants. Malware variants are detectable using IoCs and applying signature-based rules, threats and threat classes are detectable by combining higher-level network features and network behavior analysis to automatically detect malicious behavior, which allows obtaining IoCs for newly detected malware variants in automated ways with the study of threats and threat classes.

Cisco Encrypted Traffic Analytics (ETA) solution is formed by both Cisco StealthWatch solution and the Enhanced NetFlow concept combined [17, 42, 43]. This solution allows analyzing network's encrypted traffic to understand the most of what is happening in the network based only on traffic metadata. As such, it can be used to detect threats in the network, without breaking users' privacies (decrypting and inspecting traffic) and without needing to parse diverse L5-7 protocols too deeply. Of all Cisco solutions, this one is the most closely related to the technical matters of this work. This solution, as well as this work, base themselves on the fact that even though not all data is intelligible, it is possible to extract a lot of threat intel from network traffic considering metadata only. By studying publicly available information about the Cisco ETA solution [8, 11, 12, 9, 10, 13], one can understand that it implements encrypted traffic analysis techniques (Cisco StealthWatch) which can be particularly applied to detect threats in the network, through the extracted and posteriorly enriched network information (enhanced netflows).

Cisco CTD provides in-depth defense against modern and advanced threats [14] which can bypass most detection mechanisms. For network-based detection, Cisco CTD uses NetFlows and, on top, Cisco StealthWatch and Cisco FireSIGHT (which uses Cisco FirePOWER as the knowledgeable backend module). It also uses an endpoint-based solution called Cisco AMP [15] (Advanced Malware Protection) for endpoint threat detection.

Cisco AMP [15] acts like an automated malware sandbox analysis mechanism capable of analyzing network packet data and detecting malicious incoming files using static and dynamic file analysis. In terms of file-related features, Cisco AMP integrates with Cisco Threat Grid [16, 17, 18] to obtain more than 700 behavioral indicators (indicator, in this context, refers to features, do not confuse with indicator of compromise) related to a file and automatically detecting and understanding malware captured in the endpoint, which is not our direct focus. However, it is relevant as a related application because endpoint-based detection systems also need to include network-based analysis capabilities.

Ongun et. al [5] used Bro connection logs to obtain network communication features. Later, they used CTU-13 datasets containing thirteen different botnet scenarios, each scenario using different botnets, techniques, and protocols. A feature representation that worked well in the authors' setting for classifying internal IP addresses is feature representation by time windows and port number. The authors also observed that feature representation depends on the amount of training data. Additionally, the authors mention that features extracted directly from raw data such as Zeek connection logs do not always result in the most optimal representation. They recommended that multiple feature representations apart from Zeek should be evaluated as future work. We agree with the authors in the sense that features extracted from Zeek [2] connection logs are not enough (standalone) to fulfill a full feature representation and, thus, recognize the consequent need of feature aggregation methods on top of Zeek's raw data to improve detection.

Gu et al. proposed BotSniffer [6], a botnet detection system which uses a detection approach that was able to identify C2 servers and the bots infected hosts in the networks. Their technique was predicated on the notion that bots belonging to similar botnets would probably indicate a spatial-temporal relationship and resemblance to each other due to the pre-programmed events associated with C2 botnets. They focus on protocols running over TCP by having diverse TCP flow features: number of upstream and downstream packets; size of the uplink and downlink transmission bytes; average length of the uplink and downlink data packets, maximum packet length, average packet variance, duration of the data stream and packets loaded in one stream. More specifically, the authors focus on two L7 protocols, IRC and HTTP, because these two protocols are very commonly used by bots to fetch or receive commands from a centralized C2 server. The authors used a custom dataset composed of diverse network traces, and some network logs recorded from an IRC tracker. Most of the traffic used for the dataset was generated by them in their university campus network. According to them, BotSniffer presented a high accuracy and low false positive ratio.

Despite BotSniffer's good results, Khan et. al [18] upholds that their detection strategy was widely concerned by experts in network traffic analysis because it does not depend on the botnet class to extract a common feature vector of the flow, which in theory compromises the definition of anomaly-based detection. We agree with Khan et. al and the referred experts that the proposed system does not use an anomaly-based approach, however it does use network behavior patterns to detect botnets, thus falling into the behavior-based detection category. In this work, we also analyze network behavioral patterns and use these to study and detect specific threats, which enables detecting new malware variants (tier-1 anomaly) and, even deeper, to study and detect threat classes, which enables detecting new threats (tier-2 anomaly); as such, this work falls into the behavior- and anomaly- based detection spectrum, independently of the usage of outlier and novel detection

algorithms. According to Khan et. al work [18], the main factors that determine the efficiency and accuracy of detection are the characteristics of the extraction and the classification strategy used. Among other things, these factors mainly encompass: the labelling taxonomy, the feature-sets, the type of labelling process and the used classification algorithms. We can see that this is true by the results we achieved using different types of feature-sets (ones that are commonly not used, flow-set based features), as well as acknowledging the fact that we could label each "Port Scan" event as its own thing (e.g., "Port Scan - closed-port probe") and other types of labelling that would drastically improve our flow classification results. It is all about what we are trying to detect.

PORT SCAN THREAT CLASS DEFINITION

Intent: Probe multiple ports of a given host, for a given L4 protocol.

Generic Attack Technique(s):

• Distributed Port Scan - multiple hosts probe multiple ports of a host.

• FTP Bounce Scan (-b) – this method allows an attacker to use a vulnerable FTP server as a proxy to port scan other hosts. This option is ideally used to target hosts in the same internal network as the FTP server, which will recognize it and accept packets coming from it, outputting responses that leak information about the port's state.

Specific Attack Technique(s):

• UDP Scan (-sU) - the attacker sends a UDP packet to each port. If the target responds with service data, the port is open. If the target does not respond, the port is either closed or filtered.

• TCP Connect Scan (-sT) - the attacker sends a TCP packet with the SYN flag bit set to each port. If the target responds with a SYN-ACK packet, the port is open and accepting requests: the attacker sends an ACK packet back; the target then responds with the service's specific data; then, the attacker sends a RST packet and closes the connection. If the target responds with a RST packet, the port is closed. Else, if the target does not respond, the port is filtered.

• TCP SYN Scan (-sS) - the attacker sends a TCP packet with the SYN flag bit set to each port. If the target responds with SYN-ACK, the port is open and accepting requests: the attacker sends a RST packet to close the connection. If the target responds with a RST packet, the port is closed. Else, if the target does not respond, the port is filtered.

• TCP ACK Scan (-sA) - the attacker sends a TCP packet with the ACK flag bit set to each port. If the target responds with a RST packet, the port is either open or closed, meaning that the port is unfiltered (not blocked by any firewall). Else, if the target does not respond or if it responds with certain ICMP error messages (ICMP Type 3; codes 0, 1, 2, 3, 9, 10 or 13), then the port is filtered.

• TCP Null Scan (-sN) - the attacker sends a TCP packet with no flag set to each port. If the target responds with a RST packet, the port is considered closed. Else, if the target does not respond, the port is either open or filtered. Finally, if the target responds with an ICMP "Destination Unreachable" error (ICMP Type 3; codes 0, 1, 2, 3, 9, 10 or 13) then the port is filtered.

• TCP Xmas Scan (-sX) - the attacker sends a TCP packet with the FIN, PSH and URG flag bits set to each port. If the target responds with a RST packet, the port is considered closed. Else, if the target does not respond, the port is either open or filtered. Finally, if the target responds with an ICMP "Destination Unreachable" error (ICMP Type 3; codes 0, 1, 2, 3, 9, 10 or 13) then the port is filtered.

• TCP FIN Scan (-sF) - the attacker sends a TCP packet with the FIN flag bit set to each port. If the target responds with a RST packet, the port is considered closed. Else, if the target does not respond, the port is either open or filtered. Finally, if the target responds with an ICMP "Destination Unreachable" error (ICMP Type 3; codes 0, 1, 2, 3, 9, 10 or 13) then the port is filtered.

• TCP Idle Scan (-sI) - the attacker sends a SYN-ACK packet to a host, which will be dubbed "unaware host" because its technical name, "zombie", already associates to a completely different meaning in the botnet context. The unexpected SYN-ACK packet sent to the unaware host will be responded to with a RST packet sent back to the attacker, which has a certain IP ID associated with it. The attacker then sends a SYN packet to the target host with the source IP address spoofed with the IP of the unaware host, incrementing its IP ID by 1. On this moment, there are three possible scenarios: (A1) The target host responds to the unaware host with a SYN-ACK packet. Since the unaware host was not expecting the packet, it sends a RST packet to the target host, incrementing its IP ID by 1 again. (A2) The target host responds to the unaware host with a RST packet. The unaware host did not expect the packet, but since it isn't a packet that tries to initiate a connection (rather, abort it), the unaware host does not respond with any packet, thus not incrementing its own IP ID. (A3) The target host does not respond to the unaware host. As such, the unaware host does not receive any packet and, more importantly, it doesn't send a packet back, such as in scenario A2, thus not incrementing its IP ID. Continuation: Once any of the previous scenarios has taken place, the attacker will send a SYN-ACK packet to the unaware host, to which the unaware host will respond with a RST packet. The IP ID of the final RST packet will then be analyzed by the attacker for the existence of one of the following scenarios: (B1) The IP ID was incremented by 2 since the first packet received from the unaware host, which means that the target host responded with a SYN-ACK packet to the unaware host, so the probed port is open. (B2) The IP ID was only incremented by 1 since the first packet received from the unaware host, which means that the target host responded

with a RST packet or did not respond at all, since in both situations the unaware host does not create any response packet for the target host. As such, from the attacker's perspective, the probed port might be either closed (scenario A2) or filtered (scenario A3). The attacker then repeats this whole process for each port that he intends to scan.

• TCP Maimon Scan (-sM) - this technique is named after its discoverer, Uriel Maimon. It starts with the attacker sending a TCP packet with the FIN and ACK flag bits set to each port. According to the RFC-793 (TCP RFC), the host should generate a RST packet in response, independently of the fact of the port being open or closed. However, Uriel found out that many BSD-derived systems simply drop this packet if the port is open.

• TCP Custom Scan (--scanflags) - the attacker sends a TCP packet with a custom set of TCP flag bits set to each port. The analysis depends on the TCP flag set used, as this means different possible responses and interpretations. It can be used, for example, to find bypassable edge-cases for firewalls and IDSs.

• Service/Version Detection Scan (-sV). Probes open ports to determine service/version info, meaning that the flow will be fully initiated to allow sending test packets to try and detect the version of the probed service based on the responses.

• SCTP INIT Scan (-sY) - the attacker sends an SCTP INIT packet to each port of the target host. An SCTP INIT-ACK response packet indicates that the port is open and, in this case, the attacker aborts the connection right after. An SCTP ABORT response packet indicates that the port is closed and, if no response is received after several retransmissions, the port is marked as filtered.

• SCTP "COOKIE ECHO" Scan (-sZ) - the attacker sends an SCTP COOKIE ECHO packet to each port of the target host. If the target host doesn't respond, the port is either open or filtered. If the target host responds with an SCTP ABORT packet, then the port is closed.

Program Applicability: Any program that communicates over a network can eventually be used for network host discovery using a certain network protocol, given that the probed protocol is present on the probed machine. Given the latter, we will only consider a host discovery program as such if at least one of the following conditions are true:

• It supports sending and interpreting ARP probes for multiple hosts

• It supports sending and interpreting raw IP packets specifying the probed IP protocol number on the IP header for multiple hosts (IP protocol probes)

• It supports sending and interpreting TCP, UDP and ICMP probes (given their prevalence on today's networks) for multiple hosts

• Optionally, these programs can also support other much less adopted protocols such as SCTP. Also, the existence of any L5-7 protocol is irrelevant for this category.

Programs - <name> (<L1-4 protocols supported>):

• UnicornScan (TCP, UDP, ICMP)

• Nmap (ARP, raw IP, ICMP, UDP, TCP, SCTP)

• Ncat (UDP, TCP, SCTP)

• Hping3 (raw IP, ICMP, UDP, TCP)

• AngryIPScanner (ICMP, UDP, TCP)

• Masscan (ICMP, UDP, TCP)

• ZMap (ICMP, UDP, TCP)

References [19]-[23] were used as a basis for building this definition.

NETGENES TOOL

The tool we developed, dubbed NetGenes, extracts features of the previous network objects:

• Packets - use packet metadata only, encompassing OSI layer 1 to OSI layer 4.

• Flows - aggregate packet features into flow features, considering the protocol stack. We consider two main protocol stacks: eth-eth-ipv4-udp and eth-eth-ipv4-tcp. TCP is implemented in the RFC way, meaning that we analyze TCP flags and the Sequence/Acknowledgment numbers to logically separate the incomplete 5-tuple TCP flow onto multiple 6-tuple flows.

• Talkers – aggregate flow features into talker features and create new talker-based flow-set features. We consider “eth-eth-ipv4” as the protocol stack for talkers and hosts, and we uniquely identify them using their IPv4.

• Hosts – aggregate talker features into host features and create new host-based talker-set features. We consider “eth-eth-ipv4” as the protocol stack for talkers and hosts, and we uniquely identify them using their IPv4.

Note about Host features: we now think that host features should aggregate flow features as well and, perhaps, substitute most talker features. This conclusion comes from the fact that these host features have not been as useful as talker features because the latter ones are flow aggregations and we can directly query them to understand the underlying flow sets, whereas hosts provide information about the underlying talker sets but there is a lot of lost information on the flow sets. As such, we consider that hosts should also focus on direct flow aggregation (flow sets). We think that implementing host-based flow-set features would be beneficial because it provides insight into each host individually and each of their flow sets, in a similar way that the talker does for each pair of talking hosts.

NetGenes is an unfinished prototype, as it will be for as long as every threat class's core feature is not implemented. Right now, it includes a lot of conceptual and statistical features on each network object which may not be at all relevant to detect any network attack by their core features, and it still does not include all the features that it needs to properly detect every threat class. These features are workable with Machine Learning, and have been designed to be worked with it as well (e.g., one-hot encoding of Boolean values), but successfully classifying threat class traffic is not as easy as splitting datasets in train and test datasets based on authors' labels and trying multiple classifiers and regressors, it's much more complicated than that to implement a generically efficient classifier.

We define core features as features that can successfully describe the core scenarios of a network attack (generically encompassed by its threat class), with either low possibilities of evasion or severely affecting the attack's effectiveness if not detected. The purpose of the NetGenes tool is to help us extract relevant information for detecting all the network attacks that we want to detect, which should be thought about by studying the threat classes that those attacks implement to extract the core features needed. As such, our long-term goal with this work is to continuously improve NetGenes towards encompassing more threat class core features, in all its extracted network objects. We also think that, by including non-core features that are useful for the detection of threat class instances, using statistical analysis and ML classifiers, we may be able to receive hints about what core features we should be looking for to implement in the tool. We recommend this as future work for more threat classes.

NetGenes supports the extraction of 146 flow features, 184 talker features and 262 host features, from network trace-files in the PCAP/PCAPNG format.

FLOW-SET BASED ANALYSIS

Until we thought about the Port Scan detection problem properly, we tried flow classification using ML algorithms, which was an improvement over packet- and signature-based detections for detecting new network attack instances. However, these methods can become outdated due to the fact that tools change overtime, and custom parameters can be given to these, altering the network traffic enough to be able to escape packet-based and signature-based detection, as well as flow feature analysis methods that do not solely focus on a threat class's core features. Additionally, neither CICFlowMeter nor other pure "flow" extraction tools can extract flow-set based features, which is why most researchers usually use flow-based features to feed ML models and study threat classes.

We recommend that researchers attempt to extract flow-set based features, such as talker- and host- based features, to not only improve their detection results but, more importantly, to find the core features of the threat class, to improve their results based only on those core features and to create their own rule sets to detect the threat class. The current state-of-the-art alternative is relying on black-box ML models working with multiple statistical flow-based features to attempt to model a whole threat class around those features. This may achieve great results because a ML model is capable of creating very complex rule sets within itself based on those types of features that are present on the training data, but the main problem is that it will most likely rely in features that do not truly define a threat class, so new implementations could result in undetected network attacks that completely drop said features (unless they happen to be core features). It is a common problem that a ML model will overfit around multiple non-core features based on the train dataset and achieves great results in the test datasets because of it when, in fact, the features used are completely irrelevant for the threat class itself but just happen to be a commonality within the train and test datasets. The previous problem is the reason why, for ML-based research for network traffic analysis, if we want to find relevant commonalities that lead us to understand the threat class itself, it is important to use broad train and test datasets with a preference for multiple tools. However, even then, it is difficult to truly understand the ML model's outputs and we cannot easily outline its limitations in detecting relevant instances of the threat class.

PORT SCAN RULES

"Port Scan" Host rules:

• (Unused rule) HR-1 – "Other hosts tried to access more than n network services of the host.": (bihost_bwd_biflow_n_unique_dst_ports>n)

"Port Scan" Talker rules:

• TR-1 – "Source host tried to access more than n network services of destination host, or destination host tried to access more than n network services of source host.": (bitalker_fwd_biflow_n_unique_dst_ports>n) | (bitalker_bwd_biflow_n_unique_dst_ports>n)

Default Flow rules:

• (Unused rule) FR-HR-Default – Filter flows for relevant backward uni-hosts: (bihost_bwd_id ==bihost_id)

• FR-TR-Default – Filter flows for relevant bi-talkers (dividable in forward and backward uni-talkers): (unitalker_id==unitalker_fwd_id) | (unitalker_id==unitalker_bwd_id)

"Port Scan" Flow rules:

• FR-1 – "Flow was initialized by an unacknowledged connection request. Either the initialization packet did not properly reach the destination host, or any host in-between the source host (exclusive) and the destination host (inclusive) dropped the packet. No connection was established.":
biflow_eth_ipv4_tcp_initiation_requested_connection==1

• FR-2 – “Flow was initialized in an incomplete manner, only completing a two-way handshake. In other words, source host requested a connection (syn1) and destination host acknowledged it (ack2)”, encompassing two connection possibilities: 1 – connection rejected, 2 – half-duplex connection established.”: biflow_eth_ipv4_tcp_initiation_two_way_handshake==1

• FR-2.1 – “The destination host rejected the connection (rst2-ack2).”: FR-2 & biflow_eth_ipv4_tcp_connection_rejected==1

• FR-2.2 – “A half-duplex connection was established, i.e., although the destination host accepted the connection request (syn2-ack2), the source host never acknowledged it (!ack3), as the third step of the three-way-handshake mandates.”:

FR-2 & biflow_eth_ipv4_tcp_connection_established_half_duplex==1

• FR-2.2.1 – “The source host established a half-duplex TCP connection, just to abort it afterwards.”:

(biflow_eth_ipv4_tcp_connection_established_half_duplex==1) &

(biflow_eth_ipv4_tcp_termination_abort==1) &

(biflow_fwd_eth_ipv4_tcp_n_active_rst_flags>0)

• FR-3 – “A full-duplex connection was established and there was only 1 packet (syn2-ack2) that was sent by the destination host, before the source host aborted the connection.”:

(biflow_eth_ipv4_tcp_connection_established_full_duplex==1) &

(biflow_bwd_n_packets==1) &

(biflow_eth_ipv4_tcp_termination_abort==1) &

(biflow_fwd_eth_ipv4_tcp_n_active_rst_flags>0)

We note that TR-1 source and destination hosts/ports are not based on packet direction, but on flow direction. Packet direction varies in a flow, so it would be a mistake to directly consider unique destination port counts if it was based in the packets, as you would capture both the source and the destination ports of the flow. As such, only after you have achieved a flow definition can you correctly define and extract talker features, and the same applies to flow-set based host features, such as the one presented in HR-1.

Results obtained using the host rules we defined, HR-1 and FR-HR-Default, are not presented, because the talker-based rules we defined, TR-1 and FR-TR-Default, were enough to achieve great results. Despite this, we further discuss this matter because the host rules can top the talker features when a single network attack is performed using multiple source IPs.

PORT SCAN RULE SET RESULTS AND ANALYSIS

We summarize each “Port Scan” flow rule set (considering “FR-TR-Default” applied with “TR-1 n=100”), applied to all CIC-IDS-2017 dataset days that consider a port scan (Thursday and Friday):

• A lack of flow rules, which captures every flow within the filtered talkers, has a precision of 69.947%. It has an F1-Score of 82.316%. It detects every Port Scan flow, but also wrongfully considers 34.780% of all Port Scan flows.

• “FR-1”, which captures every flow with an unanswered requested connection (likely dropped), has a precision of 91.070%. Its low F1-Score (22.867%) reflects the fact that it only detects 13.075% of all Port Scan flows.

• “FR-2”, which captures every flow initiated with a two-way handshake, has a precision of 99.924%. It has an F1-Score of 92.671%, the highest F1-Score for a single flow rule, detecting 86.400% of all Port Scans.

• “FR-3”, which captures every flow that had a full-duplex connection that is later aborted by the source host, without the destination host ever sending another packet other than the three-way-handshake’s second packet, has a precision of 99.789%. Its low F1-Score of 0.410% reflects the fact that it only detects 0.205% of all Port Scan flows.

• “FR-2.1”, which captures every flow that was rejected, has a precision of 99.938%. it has an F1-Score of 92.458%, detecting 86.019% of all Port Scan flows.

• “FR-2.2”, which captures every flow that had a half-duplex connection, has a high precision, so most instances classified as a Port Scan with this rule set were, in fact, a Port Scan. Its low F1-Score is low reflects the fact that it only detects 0.380% of all Port Scan flows.

• “FR-2.2.1”, which captures every flow that had a half-duplex connection whose source host aborted it afterwards, seems to be the most specific to port scan situations, and its 100.000% precision indicates just that. In fact, there might be no other reason for a host to open a half-duplex connection and abort it afterwards unless it was simply checking if the port was accepting connections. Its low F1-Score of 0.419% reflects the fact that it only detects 0.210% of all Port Scan flows.

• “FR-1 | FR-2” has a precision of 98.663%. It has an F1-Score of 99.067%, detecting 99.474% of all Port Scan flows.

• “FR-1 | FR-3” has a precision of 91.193%. Its low F1-Score of 23.184% reflects the fact that it only detects 13.280% of all Port Scan flows.

• “FR-2 | FR-3” has a precision of 99.924%. It has an F1-Score of 92.789%, detecting 86.605% of all Port Scan flows.

• "FR-1 | FR-2 | FR-3" has a precision of 98.666%. With an F1-Score of 99.170%, it has the highest F1-Score among all rule sets, detecting 99.680% of all Port Scan flows.

PORT SCAN DETECTION RESULTS

Table 1 shows a comparison between our current work and three other works that have applied their detection mechanisms to the CIC-IDS-2017 dataset.

Works	Detected Port Scans	Friday's "Port Scan" Flow Classification Accuracy
Previous work (2018) [26]	1	99.73%
Singh et. al, ICAESMT-2019 (2019) [27]	1	99.9815%
Stiawan et. al, IEEE Access 8, 132911–132921 (2020) [28]	1	99.7%
Current work (2020)	13	99.953%

Table 1. Work Comparison in CIC-IDS-2017 Port Scan Detection.

Our work detected the 12 port scans that occurred Thursday and were incorrectly labeled in the dataset, of which 11 were referenced by the dataset authors [24,25], when they refer to Thursday's infiltration 2nd step in which 192.168.10.8 performs a port scan to "all other clients", and 1 port scan that was not referenced at all. NetGenes-generated data and the rules we employed were effective to spot these types of imprecisions in the dataset.

Furthermore, as other works, we detected the Port Scan that occurred on Friday, correctly detecting the only 2 hosts involved in this interaction. Even though our flow classification results were not as great as Singh et. al results [27], we tried hard to not flow fingerprint any flow, which is very hard to not do when working with Machine Learning unless manual feature selection is performed, due to issues with train and test datasets that do not allow accounting for many variants.

Additionally, this work's flow definition is different from the flow definition considered by most other works: namely, for Friday's TCP port scan flows, we are accounting a total of 158980 Friday TCP port scan flows extracted by NetGenes, while other works consider 158930 original port scan flows extracted by CICFlowMeter (158923 of which are TCP flows, 6 are marked as unidentified and 1 is the only correctly labeled UDP flow). Moreover, we also note that we did not detect the version detection scans, which would result in many rules that are essentially flow fingerprinting and would steer away from using core features only. If this traffic exists unencrypted, we should instead gather L5-7 features by default and assess these new features as indicators rather than performing flow fingerprinting, as at that abstraction level they could be core features. We propose that as future work.

Finally, even though the metrics we defined are important, it is more relevant to understand why we are getting such results. Understanding what our rules do is more important than the metrics they achieve in correctly classifying flows. Additionally, what we really want to do is to be able to safely state that a certain network attack has occurred, identify the attacker(s) and identify the victim(s), which can be performed using flow-set information (as we do with Port Scan's HR-1 and TR-1). Additionally, a set of flows filtered by high-precision rule sets indicates that they are malicious with a very high certainty (even if those are not the only malicious flows), and this information will help the analyst further narrow down the network traffic that they need to focus on to undertake a deeper network analysis.

CONCLUSION

We developed our own tool, dubbed NetGenes, to extract useful information from the packets captured inside a network trace-file. The extracted information is independent of encryption because only the packet metadata is used to generate it. This information is then hierarchically organized in three abstract network concepts, which we dubbed "network objects", responsible for logically aggregating traffic, each one with its own features. NetGenes provides a lot of conceptual and statistical network features, to arm an analyst or researcher with a readable feature format. This feature format enables a researcher to quickly acknowledge the fundamentals of the network communications captured inside the network trace-file, while each network-object features provide deeper insight on the network data.

By developing NetGenes, we can use a set of flow features that is lengthier than many flow extraction tools, including conceptual and statistical features, usable to study and handle the data in the most complete way that we possibly can (considering the multiple tool's development cycles this year). We also developed and extensively use the "Talker" network object as a flow aggregator, as well as the "Host" network object as a talker and flow aggregator, as well as their respective features. NetGenes is a tool that was inspired by CICFlowMeter (a tool made by researchers at the Canadian Institute for Cybersecurity, CIC), but at the moment includes slightly more flow features and adds the concept of flow-set based features on top, which is already present in tools that are used more by the network community like tranalyzer2 (which includes the "top

talkers" concept and possibility of flow aggregation scripts) and WireShark ("Conversations").

The Talker object proved to be an important concept to analyze network traffic. It provides a relevant context for flows, grouping them by source host and destination host, and allows filtering useful flows based on talker-based flow-set features.

Similarly, the Host object, also proved to be an important concept to analyze network traffic. It provides a relevant context for talkers and flows, grouping them by Host, and allows filtering useful flows based on host-based talker-set and flow-set features.

For the considered threat class ("Port Scan"), the flow classification results of this work were great, while talker classification results were perfectly accurate for every day.

ACKNOWLEDGMENTS

Thank you, Prof. Pedro Adão, for supervising my thesis and providing me the guidance I needed.

Thank you, Eng. Nuno Marques, for hearing me when I felt the need to randomly babble about my work, as well as helping me revise my thesis proposal.

Thanks, Portuguese National Cybersecurity Center, for providing me, from September 2019 to December 2019, a day-worth of my weekly working hours to have extra time to research and prepare my thesis proposal.

A final thank you to my family, my girlfriend, and my friends, for having always supported me and my wishes. Your support helped me keep the balance between my personal, academic and professional life, and gave me the strength to finish my thesis.

REFERENCES

1. Answering Tough Questions About Network Metadata and Zeek, http://www.infosecisland.com/blogview/25191-Answering-Tough-Questions-About-Network-Metadata-and-Zeek.html, last accessed 2019/11/27
2. Zeek GitHub, https://github.com/zeek/zeek, last accessed 2019/12/08
3. Haddadi, F., Zincir-Heywood, A.N.: Benchmarking the effect of flow exporters and protocol filters on botnet traffic classification. IEEE Systems journal 10(4), 1390–1401 (2014)
4. Tranalyzer Website, http://tranalyzer.com, last accessed 2019/12/02
5. Ongun, T., Sakharaov, T., Boboila, S., Oprea, A., Eliassi-Rad, T.: On designing machine learning models for malicious network traffic classification. arXiv preprint arXiv:1907.04846 (2019)
6. Gu, G., Zhang, J., Lee, W.: Botsniffer: Detecting botnet command and control channels in network traffic (2008)
7. Cisco VMDC Cloud Security 1.0 Design Guide, chap. 4 (2014)
8. White Paper - Cisco Public 2019 Encrypted Traffic Analytics (ETA), https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf, last accessed 2019/12/18
9. White Paper - Cisco Public 2016 StealthWatch, https://www.cisco.com/c/dam/m/en_hk/never-better/dna/pdf/stealthwatch_solution_overview_whitepaper_en.pdf, last accessed 2019/12/19
10. Radhakrishnan, S.: Detect threats in encrypted traffic without decryption, using network based security analytics (2017)
11. Security Analytics and Logging: Supercharging FirePower with Stealthwatch, https://blogs.cisco.com/security/security-analytics-and-logging-supercharging-firepower-with-stealthwatch, last accessed 2019/12/12
12. Cisco Encrypted Traffic Analytics (ETA) Promotional Video, cisco.com/go/eta, last accessed 2019/12/19
13. Cisco Encrypted Traffic Analytics: Necessity Driving Ubiquity, https://blogs.cisco.com/security/cisco-encrypted-traffic-analytics-necessity-driving-ubiquity, last accessed 2019/12/19
14. Cisco Cyber Threat Defense (CTD) design guide, https://www.cisco.com/c/dam/en/us/td/docs/security/network_security/ctd/ctd2-0/design_guides/ctd_2-0_cvd_guide_jul15.pdf, last accessed 2019/12/12
15. Cisco Advanced Malware Protection Solution Overview, https://www.cisco.com/c/en/us/solutions/collateral/enterprise-networks/advanced-malware-protection/solution-overview-c22-734228.html, last accessed 2020/01/03.
16. Cisco Threat Grid Promotional Video, https://www.cisco.com/c/en/us/products/security/threat-grid/index.html, last accessed 2020/01/03.
17. Cisco Threat Grid Demo, July 2018, https://www.youtube.com/watch?v=un2t2T_s6IY, last accessed 2020/01/03.
18. Investigating Malware with Threat Grid, https://www.youtube.com/watch?v=W7IuchQR7dA, last accessed 2020/01/03.
19. NMAP TCP Idle Scan, https://nmap.org/book/idlescan.html, last accessed 2020/11/15.
20. NMAP Service and Version Detection, https://nmap.org/book/man-version-detection.html, last accessed 2020/11/15.

21. NMAP Command-line Flags, https://nmap.org/book/port-scanning-options.html, last accessed 2020/11/15.
22. NMAP Linux Man Page, https://linux.die.net/man/1/nmap, last accessed 2020/11/15.
23. NMAP Port Scanning Techniques, https://nmap.org/book/man-port-scanning-techniques.html, last accessed 2020/11/15.
24. CIC-IDS-2017's official website, https://www.unb.ca/cic/datasets/ids-2017.html, last accessed 2020/12/28.
25. Sharafaldin, I., Lashkari, A.H., Ghorbani, A.A.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: ICISSP. pp. 108–116 (2018)
26. Almeida, F., Meira, J., Adão, P., Loura, R.: Network Intrusion Detection: Machine-Learning Techniques for TCP Flow Classification (2018, unpublished)
27. Singh Panwar, S., Raiwani, Y., Panwar, L.S.: Evaluation of network intrusion detection with features selection and machine learning algorithms on cicids-2017 dataset. In: International Conference on Advances in Engineering Science Management & Technology (ICAESMT)-2019, Uttaranchal University, Dehradun, India (2019)
28. Stiawan, D., Idris, M.Y.B., Bamhdi, A.M., Budiarto, R., et al.: Cicids-2017 dataset feature analysis with information gain for anomaly detection. IEEE Access 8, 132911–132921 (2020)