TÉCNICO LISBOA

Structured Behavior Analysis on Encrypted Traffic

Understanding and Detecting Network Attacks

João Pedro Pires Carrapiço de Almeida Meira

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Pedro Miguel dos Santos Alves Madeira Adão

Eng. Nuno Miguel Lopes Marques

Examination Committee

Chairperson: Prof. Nuno João Neves Mamede

Supervisor: Prof. Pedro Miguel dos Santos Alves Madeira Adão

Member of the committee: Prof. Nuno Miguel Carvalho dos Santos

January 2021

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.

Resumo

O objetivo principal deste trabalho é estudar ataques de rede. Ao delinear o perfil dos padrões de comportamento de rede inerentes a ferramentas de *software* usadas de forma maliciosa, podemos detetar as técnicas que essas ferramentas implementam sem precisar de detetar especificamente a ferramenta com base nas suas especificidades.

Para tal, começamos por desenvolver e propôr uma ferramenta de extração de *features* de rede denominada *NetGenes*, que considera várias *features* de comunicação de rede conceituais e estatísticas baseadas exclusivamente em metadados extraídos de protocolos L1-4 (camada-OSI 1 a camada-OSI 4). A ferramenta *NetGenes*, a partir de um ficheiro de captura de rede (*PCAP*, *PCAPNG*), permite extrair *features* de três objetos de rede (*flows*, *talkers* e *hosts*) que se constroem baseados uns nos outros, agregando logicamente *features* dos objetos de rede abaixo deles, e permitindo também a criação de novas *features*.

De seguida, estudamos várias classes de ameaças, organizando-as logicamente como numa taxonomia e descrevendo as ameaças, técnicas de ataque e ferramentas que as implementam, contidas pela mesma.

Depois, criamos vários conjuntos de regras com base nos objetos de rede extraídos pelo *NetGenes* para a classe de ameaça *"Port Scan"*.

Finalmente, utilizamos os conjuntos de regras criados anteriormente ao *dataset CIC-IDS-2017*, fornecendo informações valiosas sobre as melhores formas de detetar tráfego pertencente à classe de ameaça "Port Scan" de forma transparente e direta.

Palavras-chave: Segurança de Redes, Extração de Features TCP/IP, Análise de Tráfego Cifrado, Threat Hunting em Redes.

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.

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

Table of Contents

List of Tables

List of Figures

List of acronyms and abbreviations

ARP – Address Resolution Protocol

Bwd – Backward

C2 – Command and Control (Server)

CIDR – Classless Inter-Domain Routing notation

DDoS – Distributed Denial of Service

DoS – Denial of Service

Dst – Destination

eBPF – Extended Berkeley Packet Filter

ETT – ENISA Threat Taxonomy

FTP – File Transfer Protocol

Fwd – Forward

HTTP – Hypertext Transfer Protocol

ICMP – Internet Control Message Protocol

IoC – Indicator of Compromise

IP – Internet Protocol

IPFIX – Internet Protocol Flow Information Export (NetFlow version 10)

IPv4 – Internet Protocol version 4

IPv6 – Internet Protocol version 6

IRC – Internet Relay Chat protocol

Ln – OSI Layer n

MAC – Media Access Control

MISP – Malware Information Sharing Platform

OSI – Open Systems Interconnection model

OSINT – Open-Source Intelligence

OTT – Open Threat Taxonomy

OVAL - Open Vulnerability and Assessment Language

PCAP – Packet Capture Dump File Format

PCAPNG – PCAP Next Generation Dump File Format

SCTP – Stream Control Transmission Protocol

SIEM – Security Information and Event Management

SIP – Session Initiation Protocol

SMB – Server Message Block

SQL – Structured Query Language

Src – Source

SSH – Secure Shell

SSL – Secure Sockets Layer

TAXII – Trusted Automated eXchange of Indicator Information

TCP – Transmission Control Protocol

THP – Threat Hunting Platform

TIP – Threat Intelligence Platform

TLS – Transport Layer Security

UEBA – User and Entity Behavior Analytics

UDP – User Datagram Protocol

XSS – Cross-Site Scripting

Chapter 1. Introduction

Fortunately for user privacy, there is now more encrypted network traffic than unencrypted. Right now, it is estimated that 60% of all Internet traffic and more than 80% of all Web traffic is encrypted, and the trend of encryption keeps growing [17, 45]. However, encryption poses a challenge to understanding and detecting threats because encrypted application data makes its logic unintelligible for most application analysis tools that rely on protocol parsing to detect application-specific events (e.g., WAFs, SIEM endpoint agents, etc.). For example, regex signatures are used by signature-based detection systems to detect malicious network activity. Unfortunately for these detection mechanisms, by changing a single bit of network packet data, the encrypted version of it becomes completely different from the unencrypted one and cannot be correlated to other slightly modified encrypted versions of it, fact which enables adversaries to evade and bypass such mechanisms. Even in the case that the regex-based detection mechanism is performed on the endpoint system, and therefore network traffic is decryptable, the attacker can still implement and use his own network stack and perform customized encryption (or even, simply encoding) in his malicious program to evade detection based on regex signatures, which makes regex-based detection fail to detect zero-day attacks.

On the other hand, using blacklists of contacted IPs and domains is better because, to evade detection, the attackers will have to spend money to get more IPs and domains, and additionally will have to modify their original malicious program and redistribute it. However, signature-based detection mechanisms which rely on blacklists of IPs and domains would still not detect modified malware variants because the contacted IPs and domains would not have been blacklisted yet. As such, this possibility can make intrusions unnoticed for as long as they are not added to the blacklist by security researchers, other systems or other parties which participate in adding more indicators of compromise to the blacklists. Another example is the case where the attack is specifically tailored to an organization, which can mean that indicators of compromise are unique and, therefore, not relevant anymore for detection purposes outside the organization.

Ideally, we should be able to detect network attacks (and even host attacks) based on indicators which are generic enough to be applied throughout time and independently of many circumstances, based on each attack's specific characteristics. Additionally, we should be able to separate normal from abnormal network traffic by relying on what we know about the behavioral patterns of different types of tools, threats and threat classes, and, in a standalone manner, use these to each organization's advantage and oversight against new network attacks deployed against them.

Improving Threat Detection

Problem: By using indicators of compromise (IoCs) at the network level, one can specifically target hosts (IPs and domains) and malware (regex signatures) extracted from blacklists. However, we are still left with the problem that identifying network attacks by using contacted IPs and domains is not feasible in the long run, because these are constantly changing for the same threats and threat classes. Similarly, using regex signatures has the same problem, with the aggravating that adversaries can

leverage encryption to hide their payloads from detectors, and even just use encoded payloads that get through a lot of detectors. IoCs are very useful in campaigns and incidents because they enable cybersecurity teams to efficiently detect and mitigate intrusions related to those, which allows finding victims and perpetrators of specific network intrusions. These network intrusions are caused by specific malware variants responsible for internal infection or by attacker-controlled hosts that need to be detected and blocked, but it often takes time until the relevant IoCs can be manually retrieved and input into a security feed to share with the rest of the world. Many times, these IoCs are only retrieved after the damage has already been done, when the organization should have successfully detected an anomalous event was taking place, even if the attackers' IPs, domains or regex signatures did not match any available blacklist. Similarly, threat researchers may need to study a big network capture file which deals with unknown attackers, but is mixed with all sorts of benign traffic, so it would be very useful to automate the detection of what we are looking for. In these scenarios, OSINT is very limited because you may either be patient zero, or you already were, IoCs may not be publicly available or the attacker may have specifically targeted your network(s).

Solution: To solve the previous problem, we need to successfully detect network attacks independently of the specificities of the used tools or the attacker's infrastructure. This led us to focus on higher-level definitions of the network attack than the specific software used to implement those, which we call threat classes. By using the core features of a threat class and detecting those, we can detect any software that implements that threat class, providing a trustworthy anomalous behavior detection. We propose to target both malware and attacker software by analyzing and profiling their generated network traffic. Using network behavior analysis in the study of diverse samples, independently of the traffic being encrypted or unencrypted, we propose to study and profile threats and threat classes. We also propose to perform this threat profiling by analyzing network behavioral patterns through information extrapolated from packet metadata only. We theorize that network behavioral patterns are a much stronger concept to profile software and understand how it works at the network level than indicators of compromise because it allows for a non-deprecated detection of new attack campaigns and incidents based on their real root cause. Additionally, a system built on behavioral classification can still automatically output IoCs and signature-based rules when a threat is detected.

Ultimately, this work intends to contribute to the daily activities of threat researchers who work in post-mortem analysis tasks related to network traffic, such as threat hunters, intrusion detection researchers and, also often, SOC analysts. In sum, any network security professional, or anyone who may be concerned with studying, analyzing and/or detecting malicious network activity, can make use of this thesis.

1.1. Work Objectives

Interest for Network (Security) Engineers

Network automation and the Python programming language are increasingly becoming more hand-to-hand as big networking companies like Cisco start combining both. Inclusively, the recently renewed DEVNET certification, which was put in practice in February this year (2020), now includes teaching network automation capabilities using Python (in the upcoming years, we expect that network engineers will start looking at Python's network automation capabilities much closer). It is expected that the networking community will continuously adapt and move towards an improved automation of the networking processes, for the most varied issues: performance, reliability, security, etc. Mostly, such network certifications (CCNA-, CCNP- and CCIE- levels for Cisco) often focus on the design of network architectures, deployment of vendor-specific solutions, simulating networking environments, performing advanced network traffic analysis and troubleshooting network problems [39,40]. All these activities include inherent security concerns and, particularly, the concepts learned in these type of networking courses are very closely correlated with the fields of network intrusion detection and cyber threat intelligence [35]. This constantly increasing interest of network engineers in the network security field is an additional motivation factor to perform this type of research. NetGenes will be an interesting tool for these professionals, providing them with an automated way of generating comprehensive network-object features to deeply study their networks and build monitoring systems based on it. Similar tools are reviewed and discussed in the related work.

Interest for Threat Researchers

Threat researchers mostly work with trace-files as they are concerned in studying passive data rather than real-time detection, focusing on the study of campaigns and specific incidents, further correlating these with, both, intrusion sets and TTPs. Threat researchers often look for attack patterns and indicators which can lead to correlate given incidents, campaigns, intrusion sets and TTPs to specific threat actors. Patterns and indicators are high-level concepts which are useful for threat actor profiling, while malware and attacker software are lower-level concepts which are encompassed by TTPs and are detectable in a network.

Threat hunting activities are commonly performed by using signature-based methods of detection in combination with multiple updated feeds, which leads to identifying campaigns and incidents taking place. However, in order to deeply study network threats and threat classes, and to profile threats and proactively detect them, there needs to be an analysis of network communications and the modelling of behavior patterns. As such, we propose NetGenes and associated methodologies, which can be leveraged to perform deep studies of tools, threats and threat classes based on the network traffic patterns they generate. Furthermore, these methodologies can also be used to analyze data and build classification models for threat actor network traffic to study and profile those as well. NetGenes hopes to deliver these capabilities, as well as paving the way to achieve threat-related classification models which are explainable by default.

1.2. Main Contributions

We have developed a new tool that:

- Can extract flows and talkers accurately well, comparable to *Wireshark*. We validate how accurate *NetGenes* was by comparing our results to the ones of Wireshark and *CICFlowMeter*.
- Extracts a more comprehensive flow feature-set than *CICFlowMeter*, including more statistical features, as well as information about TCP flow states.
- Extracts comprehensive flow-set based feature-sets, by means of network objects that are not considered by flow extraction tools by default.
- Allows the creation of new pre-processed datasets, similar to CICFlowMeter and Argus, which respectively generated the pre-processed *CIC-IDS-2017* and *CTU-13* datasets.

Afterwards, we defined a network threat class taxonomy, to give proper context to multiple techniques used to perform network attacks, as well as multiple tools that practically implement each threat class.

Then, we process the CIC-IDS-2017 raw dataset, composed of one PCAPNG file per weekday, we analyze the “Bot Ares” traffic present in CIC-IDS-2017 by using the flow-state features we extracted, which allow seeing patterns in the way that the traffic is created.

We then created multiple rule sets using the previously extracted *NetGenes* features to detect the “Port Scan” threat class. We applied these rule sets to the five days of traffic of this dataset, to test if the rule sets we defined are appropriate to detect the “Port Scan” flows in the CIC-IDS-2017 dataset. Our rule set successfully ignored all network objects on Monday, Tuesday and Wednesday (as it should), and detected the “Port Scan” events that the authors mention in their dataset description, presented on the CIC-IDS-2017 support website [145] and the CIC-IDS-2017 support paper [146], on Thursday and Friday. However, The TCP flow classification results are presented in chapter 5 and the direct TCP talker classification results were perfectly accurate for every day, as it can be seen in table 21 (annex).

1.3. Document Structure

This thesis is organized as follows. Chapter 2 presents related work in network traffic analysis. Chapter 3 establishes a taxonomy for network threat classes, splitting them into multiple smaller concepts. Chapter 4 explains what network objects we considered in this work, as well as how we implemented them. Chapter 5 analyzes the CIC-IDS-2017 dataset. Chapter 6 concludes this thesis.

Chapter 2. Related Work

2.1. Author's Previous Work

In a previous work [10], we developed a NIDS (dubbed *AI-NIDS*) which used all CICFlowMeter TCP flow features except for flow activeness- and idleness- related features. We exclusively considered TCP flows for detection purposes and were able to achieve very promising TCP flow classification results for three common threat classes: Denial of Service (DoS), Port Scan and L7 Brute-force Attacks. Despite this, many flows that were classified as one of the focused three classes would be wrongly classified as another category: for example, instead of being classified as a Denial of Service, a SYN Flood DoS would be put in the Port Scan category. This happened because, with the flow features that we had considered, a SYN Flood would be closer to a Port Scan that it would be to a Denial of Service. In this work, we augment the considered flow features, as well as create concepts that can encompass the flow definition and help us achieve a context for the flows.

Bots, other types of network attacks and threats (e.g., Heartbleed, data exfiltration, etc.) were not tackled properly by *AI-NIDS* because the low number of malign flows generated by these did not allow to use them in conjunction with the benign flows in a balanced manner (to obtain class balance, the number of benign flows would also need to be reduced, which would not be enough to achieve a broad enough definition of benign) for building accurate supervised ML models.

Moreover, solely using the flow classifier to output alerts would generate too many alerts for any human analyst. The alert problem happened because it proved to be unfeasible for *AI-NIDS* to directly consider any host whose flow was tagged as malign, which exclusively based the decision on the output of the flow classifier (output of the second layer of the double-layered algorithm that we proposed for the NIDS classification architecture: malign flow or benign flow). The former assumption would mean to create a very high number of irrelevant alerts for networks with a lot of traffic, even if the flow classifier itself seemingly presented low false-positive ratios like 0.12%. We quickly acknowledged that such low false positive ratios would still generate a lot of alerts because, depending on the size of the network, a lot of network flows would be continuously generated. For example, consider a network which generates 1.000.000 flows daily; this would mean that 1200 flows are falsely considered as malicious, which could possibly mean 1200 false alerts if we directly considered that the talking hosts are possibly malicious. This is clearly not the right solution for the problem because 1200 false positive results per day per threat class would be a nightmare for an analyst, and this is even considering a very low flow classification false-positive ratio (0.12%). We solved the previous problem by applying TCP flow count thresholds to downsize the number of alerts that would be output. At this moment, the multiple victims and attackers that were being output because of the flow classifier were reduced drastically due to limiting a simple feature shared between each couple of hosts. Even though this was just a quick fix to get things working, it showed us back then that it would be a very interesting idea to consider, as future work, a higher contextual level above the flows and their features, with the hope that it could improve the detection performance of malicious network traffic.

In conclusion, flow classification alone will not work properly for threat and threat class detection purposes without a proper context to operate on. This is exactly what this work worries about: create and use two higher-context concepts for network attack detection, host classification and talker classification, which we prove to be especially relevant for threat and threat class detection because the behavior patterns reflected by these concepts can capture the essence of the network attack for its threat and threat class handles, while flow classification would be more interesting for capturing specific attack phases and detect specific tools (since tools may present unique collective packet metadata features). One of the main ideas of this thesis is to be able to analyze the definitions of each label (a specific threat class, threat, tool, etc.), and try to understand which features we would need to support to be able to detect the different instances existent in each of those labels based on their exact definition. Finally, since classifying based on definitions does not conform too much with the definition of anomaly detection, it should be clarified that it is most useful when the definitions exist in the basis of a knowledge hierarchy, where the classes are well known and accepted by the domain-specific community and, thus, are more stable and less mutable. Definitions are how we establish ground truth in this work, so we can detect anomalies at the levels of threats and tools by using the definitions of their parent threat class.

2.2. Common Threat Language: Glossary and Taxonomies

A common language consists of terms and taxonomies (principles of classification) which enable the gathering, exchange, and comparison of information. Since the computer security field is relatively new and comprehends so many concepts, it is a challenge to determine a common language [46].

SANS Institute Comparison of Threat Taxonomies

The SANS Institute published a paper [50], authored by Steven Launius, in March 2018, which discusses and analyzes different threat taxonomies used by CERTs and other cybersecurity teams. The author compared four main threat taxonomies:

- Open Threat Taxonomy (OTT) [63], by James Tarala and Kelli K. Tarala
- ENISA Threat Taxonomy (ETT) [48,49], from the European Union Agency for Cybersecurity
- NIST Risk Assessment Threat Exemplary, from the National Institute of Standards and Technology
- Taxonomy of Operational Cybersecurity Risks, from the Carnegie Mellon University

Further, a survey was performed to the risk management department of a large financial company (along with 23 non-financial company respondents), and each taxonomy was evaluated according to three main metrics: completeness, complexity and clarity.

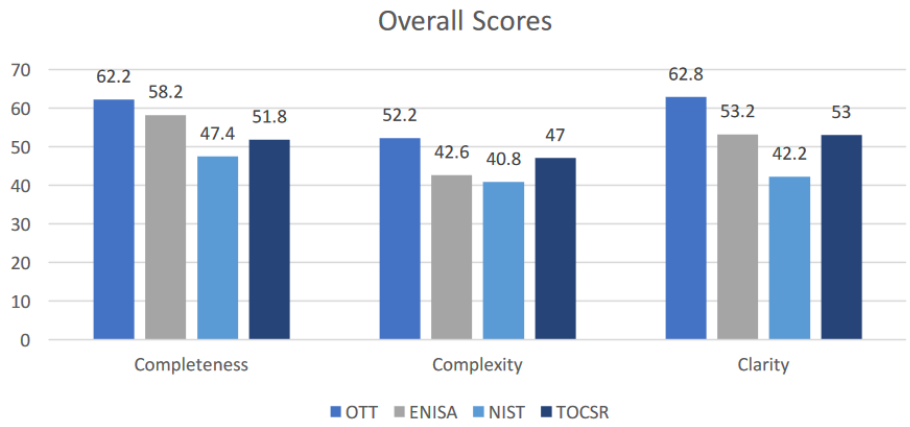

FIGURE 1. SANS SURVEY RESULTS, BY METRIC, FOR EACH THREAT TAXONOMY [50].

The author concluded that the OTT seemed to be the most preferred threat taxonomy, while remarking the fact that both the management and the non-management surveyed parties prefer the OTT to other taxonomies. Finally, the author finished by remarking that the OTT provides "a complete picture of threat actions, with clear terms, in a manner that is simple for an organization's leadership to understand". Thus, this taxonomy seems to be very appropriate for understanding threats by most parties. For that reason, we summarily provide an overview of OTT and focus on some of its encompassed concepts that are closely related to this work.

STIX 2.0

STIX 2.0 [11] is a structured language for describing cyber threat information so it can be shared, stored, and analyzed in a consistent manner. It is very popular among the cybersecurity community and it mainly focuses on high-level concepts (e.g., type of organization attacked, STIX attack pattern used) which can capture threat actors' modus operandi (expressed by their TTPs). Threat actors often act accordingly with their TTPs and perform campaigns using the same methods, allowing for correlations. Additionally, campaigns can be attributed to one or more intrusion sets if they are found to be included by them. Campaigns, incidents, intrusion sets, TTPs, attacker tools, malware and threat actor are all concepts defined by STIX 2.0 [11], also used in this work. The "attack pattern" and "indicator" concepts defined by STIX 2.0 are redefined, for the purpose of this work, to "threat actor attack pattern" and "threat actor indicator", to avoid confusion, since the first two terms have a completely different meaning for network behavior modelling and analysis. Of the previous STIX concepts, this thesis specifically focuses on fingerprinting both malware and attacker tools used by adversaries. This enables researchers to promptly associate any software (attacker tools and malware), to threat actors' TTPs and, consequently, to threat actors. However, deeply studying threat actors is not explored in this thesis but, rather, we focus on threats and threat classes, further explored in chapter 3.

Open Threat Taxonomy

The Open Threat Taxonomy [63], created by James and Kelli Tarla, was a joint collaboration between contributors from around 150 different organizations, amongst which there are multiple renowned organizations in the field of cybersecurity such as NATO (and international governments), the US Department of Defense (and other federal agencies), the US State (and municipal governments), the

Center for Internet Security (CIS), the SANS Institute and multiple Information Sharing and Analytics Centers (ISACs). People from other sectors (banking, energy, healthcare, insurance, educational institutions, etc.) involved with risk management and cybersecurity also participated in the making of this white paper.

The OTT defines four core concepts: threat agents, threat actions, threat targets and threat consequences. Threat action is “what was done” (sniffing, credential discovery, etc.), mixed with “how it was done”. The concepts and their respective definitions seem straight forward but, depending on the actual context, these concepts can be very different. For example, is “threat agent” an advanced persistent threat (APT) group, a person or is it a device? Equivalently, is “threat target” an organization, a person, a device? It all depends on the context. To try to provide such context, the OTT aggregates threat actions into four main threat categories (also called “families of threats”). Each threat category covers multiple threat actions. Summarily:

- Physical threat category: 14 threat actions
- Resource threat category: 13 threat actions
- Personnel threat category: 7 threat actions
- Technical threat category: 41 threat actions

Since the aim of this work includes possibly detecting bots in a network by analyzing bots’ behavior (among other network attacks), we want to be able to fully analyze the used techniques and try to extrapolate the associated technical intentions. By analyzing traffic metadata only, we cannot get around political or other personal motivations behind a threat. Since our aim is to profile threats by the techniques used, any other threat category other than the technical threat category is not relevant for this work. The technical threat actions we identify as being very strongly correlated to the malicious activities performed by bots and are performed by most infected devices at some point in time:

- TEC-003: Port scans (attempt to find open ports on a machine)
- TEC-008: Brute-force attacks (persistently attempt to authenticate to a service running on a machine)
- TEC-021: Denial of service attacks (attempt to overload a machine resources)
- TEC-022: Bot Infection (maintain persistence in the infected device, opportunistically connect to the command-and-control server to receive new commands and, usually, perform malicious activities on behalf of the infected device)

2.3. Automated Threat Intelligence

Threat Intelligence Gathering and Enrichment

Most automated cyber threat intelligence mechanisms today are performed by automating the treatment of available (either publicly available or available through paid cyber threat intelligence services like Cisco Talos) IoCs and applying numerous different methods to obtain useful output. IntelMQ [13] is a system which uses a message queuing protocol for collecting and processing security feeds. Blacklisted IPs (IPv4 and IPv6), domains and URLs are the most common observable in these feeds. This system comprises four types of bots: collectors, parsers, experts and outputs [25]. Collector bots regularly

(frequency defined by the user) fetch security feeds from different sources and save the unstructured data. Parser bots parse the fetched data and transform it into a structured format. Then, expert bots use the structured data to obtain more information than what was initially fetched by performing extra actions; for example, by having an IP, one can further perform IP-based lookups such as whois lookup, reverse DNS lookup, geolocation lookup, among others. Finally, output bots transform all the information to different formats (Splunk DB, MongoDB, etc.), so that it can be fed to commonly used applications. IntelMQ introduces the very interesting concept of “information enrichment”, performed by correlating information that the system obtained about an event (e.g., IPs, domains, md5/sha1/sha256 hashes) in the original feed with further information about each searchable object in various Open-Source Intelligence (OSINT) feeds.

Threat Intelligence Sharing

The Trusted Automated eXchange of Indicator Information (TAXII) standard defines a set of services and message exchanges that, when implemented, enable sharing of actionable cyber threat information across organization and product/service boundaries. TAXII [12] defines concepts, protocols, and message exchanges to exchange cyber threat information for the detection, prevention, and mitigation of cyber threats.

Furthermore, the exchange of cyber threat data between trusted partners can be used to inform and instrument network defenses. The shareable threat intelligence data is mainly comprised of indicators of compromise (IoCs), such as adversary-used IP addresses, x-mailers and malware. Such IoCs are extremely important for collaboration because they allow organizations to share and obtain relevant threat intel with one another. By combining everyone’s intel, each organization is both protecting other organizations from threats and improving their own security [26].

Implementing the former collaborative threat sharing capability allows each organization to use the automatically obtained IoCs and automatically use these IoCs to automatically generate signature-based rulesets. Commonly used signature-based rule syntaxes to identify and block network intrusions in real-time are, for example: eBPF (extended Berkeley Packet Filter), Yara, Sigma, Suricata, Snort, OVAL (Open Vulnerability and Assessment Language), OpenIOC, among many other commonly used syntaxes. The obtained rules can then be imported by the signature-based systems (such as firewalls, signature-based IDSs and SIEMs) used by each organization, thus enabling a continuous ruleset update based on up-to-date intel and, consequently, contributing to an improved real-time packet-based detection in each organization.

YETI is a practical implementation of TAXII developed by MITRE, but it was discontinued. On the other hand, MISP (Malware Information Sharing Platform) implements TAXII and much more relevant features which allow for the described threat sharing capability. The threat sharing is directed to malware and sharing IoCs related to malware, thus supporting signature-based identification of threats. Furthermore, it also supports threat classes and threat actors, and correlating threats to these using TTPs and much more. MISP falls in the Threat Intelligence Platform (TIP) category for its broadness.

Even so, MISP does not implement behavior analysis capabilities at the network traffic level, since any uploaded evidence is regarded as having been studied and its corresponding IoCs extracted

and uploaded in the system as well. THP (Threat Hunting Platform) solutions, for threat research, and SIEM (Security Information and Event Management) solutions, for active network monitoring, provide network behavior analysis capabilities by implementing built-in UEBA (User and Entity Behavior Analytics) modules and/or relevant statistics modules.

Threat Intelligence Knowledge Bases

MITRE ATT&CK is an example of a knowledge base of TTPs. Such TTPs are able to comprise threat classes, threats and threat actors, by taking into account technical and non-technical, low-level and high-level, concepts related to threat intelligence, in a structured format. At the same time, all these concepts are organized into hierarchies and intertwined to construct a heavy correlation between the different data. The main objects that MITRE ATT&CK considers are: Tactics (TA), Techniques (T), Sub-Techniques (T), Groups (G), Software (S) and Mitigations (M). The cybersecurity community which deals daily with different types of threats uses it since it is a very rich and organized source of information for anyone trying to find out more about an intrusion, a malicious campaign, or an adversary.

2.4. Network-based Feature Formats and Feature-sets

Flow Implementations and Contemplated Features

Following a growing cybersecurity trend, L5-7 data tends to be more often encrypted than not [17] and, therefore, will be unintelligible and impossible to be deeper parsed. The encryption of data at this layer is one of the main reasons why we chose to work only with L1-4 protocols and used metadata. Consequently, by not looking into L5-7 data and studying threats using their base patterns (L1-4 protocols) instead, each host's privacy is regarded in a stronger manner. Additionally, using L1-4 metadata and their inherent extractable (TCP) flow features has already proved to be very effective in modelling network behavior patterns and detecting intrusions [10]. Despite this, it is naturally expected that the most frequently used L5-7 protocols (especially the ones mostly used by malware and attacker tools), if parsed and structured into knowledgeable features by leveraging protocol-specific concepts, will be very relevant for a more fine-tuned, optimized detection. This would be particularly relevant for the implementation of a real-time system with endpoint agents capable of decrypting L5-7 traffic, but it is not the focus of this thesis.

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 [22] (a.k.a. Bro) network metadata as a superior solution in terms of knowledge depth (consequently, feature depth).

We fundamentally agree with Kevin, but he observes that researchers in the behavior analysis area often ignore deeper features in protocols above L5 while still achieving promising results [10], which implies that all the information available until L4 is enough to achieve great results in detecting specific threats. Although this is the case, the false positive ratio tends to be much higher for classifiers that use this limited data only because they cannot distinguish traffic as well as they could with decrypted data. Having L7 specific data would, no doubt, improve the false positive ratio, by diving into a deeper knowledge pool and, consequently, finding more precise fingerprints for detecting threats. However, this work will solely focus on L1-4 data because one of our main goals is to propose a sparse and comprehensive feature-set using data from these layers exclusively due to the very common usage of encryption and privacy issues.

A long-term goal that we defined for this work is that the proposed feature-set, as well as the proposed feature format based on network objects (flow, talker and host -based features), is possible of being adopted by other researchers for studying their network attacks.

L5-7 Protocol Considerations

Alas, works which consider L5-7 protocols usually take advantage of fewer features than what could be harvested and could possibly be useful threat-related indicators. In order to extract comprehensive L5-7 features, there needs to be an extensive analysis of a large number of protocols which are very complex, and data needs to be parsed in a completely different way for each protocol and structured into features that can capture the new concepts and knowledge introduced by those protocols. This task would require big development teams with network protocol know-how and a lot of time. Tools available with L5-7 protocol data parsing capabilities are, for example, Zeek [22], ntop [21] and Tranalyzer-2 [2], amongst many others.

L7 protocol features would cause the creation of new possibly useful features, but the dataset format would have to include a whole new world of concepts, from a very extensive set of protocols, for it to be relevant for a generic threat detection or threat studying system. However, for studying specific threat classes such as botnets, considering the usage of L7 protocols like IRC or HTTP and related concepts would be helpful, given that port-protocol correlation is validated.

Standalone Port-protocol Correlation

Sacramento et. al [23] assumed that any packet that had a specific source or destination port belongs to a certain protocol. We do not agree with this because there can be packets whose source port is used as an ephemeral port and is not really part of a previously started flow in a production environment. Furthermore, given that no source port was used ephemerally then the flow representation is correct, but it was still assumed that a certain protocol was being used based on the destination port of the flow. We argue that stand-alone port-protocol correlation is not the best approach. The reason that it works most of the times is because traffic is often generated using default L7 protocol configurations, which enables that frequently a correct correlation between these protocols and their default port(s) is achieved. This can and will create a weakness in the classification model that considers the used port relevant because the adversaries, once they learn about this, will be able to evade detection by configuring the considered network protocols to run in different ports, thus abusing a weak conceptual

feature. Consequently, detection algorithms which rely too much on used ports as a feature and do not perform protocol validation are more vulnerable to be evaded. To solve this problem, we propose the use of deep packet inspection (dpi) and encrypted traffic fingerprinting techniques [17, 61] in an ideal scenario, respectively for unencrypted and encrypted traffic, to promptly confirm that the protocol is being used or, in case it is not, to determine which protocol is in fact being used. Encrypted traffic fingerprinting is not 100% accurate, like any other fingerprinting technique, but it will provide much more certainty in the protocol attribution and, a plus, it is encryption-agnostic so it can be used against any network traffic.

Flow Extraction and Traffic Analysis Tools

By summarizing different tools which extract and contain flow features, we can conclude this section with important remarks:

- NetFlow leaves out a lot of flow, talker and host features, because it does not deeply focus on statistical data and sparse flow features, which is crucial for Machine-Learning models to thrive. Additionally, its features were performance-focused rather than security-focused [1], and there's also a lack of statistical features within it, which makes it less edible for machine learning. This is a problem because we want to consider both security-related features and to obtain data sparsity to increase the range of available features, consequently increasing the probability of finding more precise indicators of malicious behavior. Several tools, such as ntop [21], pmacct [20] and NfSen (which uses Nfdump [19] as its backend NetFlow feature extraction tool), utilize NetFlow (including IPFix, aka NetFlow v10) as their core flow feature format and implement further custom network-based features: more flow features specifically extended for other L7 protocols (e.g., BGP, HTTP, DNS, etc.), and more talker and host features (mostly conceptual). The process these tools perform is called protocol and feature enrichment, and it aims to achieve a broader support for a lot of different protocols and adding useful information. From this thought, the "Enhanced NetFlow" [27] concept was born, allowing the extraction of extra statistical features. It decreases performance but adds comprehensiveness, which is what one wants to achieve in a platform that is specifically intended for threat hunting, rather than real-time intrusion detection.
- Using Zeek [22] metadata as a basis enabled researchers to extract more information from it than what it was extracted with NetFlow-based extraction tools, however the features which were extracted are also limited [9] when compared to other tools like CICFlowMeter or Tranalyzer which extract statistical features in addition to conceptual ones.
- Maltrail focuses on some conceptual talker and host features and identifies threats by using both static and dynamic entries. Their static entries are fetched from various AV reports and the developers' personal research [16]. Dynamic entries are composed of blacklist feeds, i.e., lists which are continuously updated with the most recently gathered threat information (e.g., malware C2 server, sinkholes, etc.) by the blacklist owner. Flow feature-set is practically non-existent since it only considers ports. This tool means to give a good overview of the network and possible threats but does not base its threat detection in automated behavior analysis, rather in traditional indicators of compromise. It does not possess a good feature-set that can be used in the context of this thesis.

- CICFlowMeter extraction tool does not consider any talker- or host- based features, opposed to the previous tools. Still, it poses as a very good example of what a flow extraction tool should look like because it considers a lot of statistical network-object features from a trace file. The downside of it is a lack of more contextualized features, attributed to talkers and hosts.
- Haddadi et. al [6] studied, evaluated and compared five different flow extraction tools: Maji (59 flow features), YAF (46 flow features), Softflowd, Netmate and Tranalyzer-2. It was concluded that Tranalyzer-2 was the most comprehensive flow feature extractor from the five exporters. Tranalyzer-2 is a unidirectional flow extraction tool and analyzer that employs an extended version of NetFlow feature-set. Furthermore, since it is built in C, it is a lightweight and performant solution (although performance isn't the priority). Tranalyzer-2 supports 98 flow features, which are logically grouped in Time, Inter-arrival, Packets&Bytes and Flags feature-sets.

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 [6] 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 [2]. 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 [6].

We acknowledge that Tranalyzer-2 output could be worked on with a scripting language to extend some of its default features and, if necessary, a Rust plugin for Tranalyzer-2 could be developed to improve its statistical flow features and add other ones at the higher contextual levels of the talker and host. However, this was not the chosen path.

2.5. Behavior-based Network Intrusion Detection applied to Botnets

Ongun et. al.

Ongun et. al [9] 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. These datasets already contain, in the PCAP format, a clear separation between benign and malicious traffic, which is very useful because it enables any network-based extraction tool to directly work with this dataset.

Furthermore, the authors note that the amount of imbalance in cyber security is very large, which is supported by two other paper references in the same area of applying data science methods to network traffic classification, for which the authors suggest using ensemble classifiers like the Random Forest algorithm.

The authors also mention an important matter regarding choosing the train and test data: when randomly splitting the network traffic of a specific scenario at random, it will produce highly-correlated data between training and testing datasets, which will result in a disparity between the metrics calculated for determining the classifier's performances and its actual performance. Instead, by choosing to train on the data of two scenarios and testing on a third separate scenario, they are removing the possible correlations that the network traffic could have due to belonging to the same scenarios. However, we note that they are not removing the correlation out of the network threat class itself, since the same network tools are used to test, even if it is in a different timeframe using different machines. The two tools tested by the authors are two specific bots they study: "Neris" and "Rbot".

Additionally, the authors enumerate some important aspects to consider in network traffic analysis for optimizing the machine-learning model's effectiveness: feature representation (feature representations consists in logically grouping features and separately testing them. In the context of this thesis, we often refer to these as feature-sets); fine-grained labelling; algorithm choice; time-window choice.

The authors did not observe a major difference when they considered both traffic and timing features, in comparison to using only aggregated traffic features. Despite this, the authors noted that using feature representation was beneficial for the performance of their machine learning models. The authors tried three different logical feature-sets: connection features; traffic features; traffic and temporal features.

Furthermore, the authors compared their results using coarse-grained and fine-grained labelling. Their coarse-grained labelling consisted in considering every connection performed by the botnet IPs in the whole time period as malicious; on the other hand, their fine-grained labelling consisted in considering the botnet IP connections to the victim(s) when performing a DDoS attack (Rbot) in separate time windows as malicious. As expected, the fine-grained labelling technique was proven to be much better for the performance of their machine learning algorithms in detecting the presence of botnet-related network traffic.

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 [22] 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.

Gu et. al. proposed three botnet detection systems, named BotHunter [3], BotSniffer [4] and BotMiner [5]. BotHunter [3] utilizes Snort sensors and builds a customized ruleset directly integrated with Snort to specifically detect the presence of a botnet in a network. Packet payload is considered in the ruleset as well, looking for known bot generated signatures. BotHunter is publicly available and, consequently,

is one of the most used botnet detection systems by researchers to compare their own botnet detection systems and methods [6,7,15].

In the BotHunter [3] work, the authors use Snort, along with two plugins (SLADE and SCADE), in a custom way to tackle the entropy which is inherent to a bot infection. The authors focus their efforts on the victim hosts and their communications with other hosts to study and understand the infection process. Additionally, the authors consider several common actions amongst bots to try to detect a bot infection in its early stages. For instance, the following bot infection sequence was taken from a specific bot, but it is still representative for other bots:

- Event 1 – Target scanning: external-to-internal scan
- Event 2 – Vulnerability exploitation: external-to-internal exploit
- Event 3 – Bot download and execution: internal-to-external
- Event 4 – Command-and-control (C2) channel establishment: internal-to-external
- Event 5 – Outbound scanning: internal-to-external infection scanning

The presence of the above five events is tested using signature-based detection with Snort, SLADE and SCADE, which is then used to build a matrix, dubbed “network dialog correlation matrix” by the authors, showing every internal host communicating with an external entity and which events they fired considering a fixed time window for each experiment.

However, these five events should not be considered representative of all bots. For example, the infection vector might be a malicious email, which would make event 1 and event 2 irrelevant for its detection. Another example is if event 3 is the download of a dropper, rather than the bot itself, which could mean an additional downloading event would have to be considered for the case of a 2nd stage malware. Event 5 could not happen at all, or it could be a lateral movement (internal-to-internal scanning). These examples are just some possibilities of how bots could present different behaviors. At such a realization, the authors do not consider neither a strict order of events nor the existence of all the presented events in the bot infection sequence to output an alarm of a bot infection. In a prepared virtual network environment running multiple bots, BotHunter achieved a 95.1% true positive ratio.

Later, Gu et al. proposed BotSniffer [4], 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.

Khan et. al.

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.

Khan's proposed P2P botnet detection framework is based on a decision tree algorithm for feature selection which extracts the most relevant features and ignores the irrelevant features. Furthermore, the detector is based on a multi-layer approach to classify network traffic (P2P botnet traffic and non-P2P traffic) and identify botnets by applying machine learning classifiers on network features such as port filtering, DNS queries, and flow counting.

At the first layer, it filters non-P2P packets to reduce the amount of network traffic by applying port filtering using well-known ports, DNS query, and flow counting. The second layer further classifies the captured network traffic into two classes such as non-P2P and P2P. At the third layer of the model, we reduce the features which may marginally affect the classification. At the final layer, it successfully detects P2P botnets using decision tree classifier by extracting network communication features.

The proposed technique covers the limitations of single stage botnet detection, like for example the resulting class imbalance, i.e., the lack of botnet traffic in comparison to benign traffic, as Ongun et. al. [9] and our previous work [10] also mention. The accuracy of the model achieved is 98.7% and the threshold of false alarm (positive) rate is 3%. Furthermore, the authors also demonstrate that the accuracy of the proposed framework can be improved up to 99%, but at the expense of false reporting of benign files as botnets as well as false reporting of botnet as benign, so the False Positive and False Negative ratios would be affected. The authors also observed that the model's accuracy might be improved by increasing the epochs of deep learning algorithms (at the expense of more execution cost).

Finally, they performed a benchmarking of the proposed technique by testing it against diverse datasets and comparing their results with other publicly available machine learning algorithms implemented for botnet detection.

2.6. Next-generation Network Security: Cisco Solutions

Reading Cisco's article about End-to-End Visibility [37], 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 [17, 36, 41, 42, 43, 44], 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 [38] 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 [53] (Advanced Malware Protection) for endpoint threat detection.

Cisco AMP [53] 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 [54, 55, 56] 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.

Connection with This Work

In this thesis, we try to attain the same goal as Cisco CTD of detecting the presence of threats in a network, such as the presence of bots [38], but rather than focusing on the real-time detection of threats, it focuses on the process of building such detection systems and in the study of threats only at the network level (rather than studying host-level indicators as well).

Similar to Cisco Enhanced NetFlow, *NetGenes* strives to obtain the most possible extractable packet information using comprehensive conceptual and statistical flow features, and similar to the posterior flow aggregation techniques performed on top of NetFlow output, we strive to obtain the most possible flow-extractable information by implementing comprehensive conceptual and statistical talker and host features. We identify this gathering of features as being similar to Cisco Threat Grid, in the sense that we try to obtain a comprehensive network feature-set as well.

Ideally, if we were to build a full-featured Threat Hunting Platform (THP), it would use the extracted network-object features to enable an optimized study of threats in post-mortem analysis scenarios, as well as creating a way to detect them. This would be similar to Cisco StealthWatch in the sense that it performs a higher-level analysis and enables the creation of intelligence (tier-2 information, comparable to this work's host/talker information and to the detection of suspicious/malicious behavior in those contexts), from enhanced NetFlow output (tier-1 information, comparable to this work's extracted flows), all starting from network packets (raw data). However, in this work, we propose the methodology of studying and analyzing extracted network-object features to understand and detect whichever tool, threat, threat class, or inclusively any other type of traffic that we may correctly label (e.g., threat actors, types of normal traffic), that we want to study.

Note: by using *NetGenes*, which extracts the most data out of network traffic that we are able to (at the time of writing), we are optimizing the extracted data's usability. For example, if we were to develop a specific threat detection module for a real-time detection system (e.g., the NIDS developed in the context of our previous work [10]), it is very positive that the detection results are completely independent of packet L5-7 data being encrypted because the system can choose to not parse and analyze received intelligible L5-7 protocol data in real-time (since it can be a computationally heavy task to do so) and go straight to the detection of L1-4 events (e.g., detecting the core scenarios of a port scan, as we do in subsection 5.3 by creating simple but effective rule sets that target it) or, if implemented, flow fingerprinting may also be employed at the L1-4 level to detect L5-7 events (e.g., common HTTP GET requests, SSH initial login request, etc.). Similarly, for threat hunting, ignoring the encryption state of traffic does not limit us to finding threats in unencrypted network traffic only. Thus, on top of *NetGenes* network-object output data, similarly to NetFlow, it is possible to build whole solutions on top, such as the ones described above and much more: that is the purpose of *NetGenes*.

Concluding and contextualizing this subsection to our work, we observe that software solutions such as the ones briefly described above contain very similar solutions to what most literature tries to achieve. Another interesting observation is that, most of the times, the literature exclusively focuses on other literature and open-source tools, but the truth is that domain-specific companies, such as Cisco

in the networking (and network security) field, are usually ahead. As such, we propose that researchers in this field look at these companies and try to take away key notes and ideas from their publicly available documentation and solutions. Doing a good research on state-of-the-art solutions will likely be an eye-opener and improve researchers' smaller solutions which, very often, aim to achieve common goals. This way, the literature might better accompany research in domain-specific areas such as this one and strive to, inclusively, improve specific concepts and techniques used by these solutions to achieve the same objectives in a possibly better way. We firmly believe that this is what we performed within the context of this thesis but, truth be told, there are a lot of different tools out there and a more extensive study of these tools is required.

2.7. Detected Issues

We overlook closed-source and commercial software solutions which cannot be studied more in-depth for lack of verifiability. Given the studied related work, we identify the following problems:

1. Lack of L1-4 features: NetFlow-based feature extraction tools, Zeek, Maji, YAF, Softflowd, Netmate, pmacct. CICFlowMeter (TCP) and Tranalyzer-2 include a more comprehensive set of L1-4 flow features, but these flow features can still be extended even further.

2. Lack of talker- and host- based features: CICFlowMeter, Tranalyzer-2. Tranalyzer-2 has features on these higher abstraction levels, but it still lacks some of them (many which are extendable from flow features) which we want to consider, even though it also considers some features that we haven't implemented at the time. These higher-level features provide more context than packet- and flow-based features; despite this, they are very lightly considered in most datasets, traffic analysis tools and the literature in general [14].

3. Problem 1 and problem 2 both result in a different research-related problem: a lack of standardized format for building processed network-based datasets. This is a problem that we identify in the research community around the intrusion detection field, mainly due to so many different processed datasets being made available [14]. This lack of format for network-based features leads to each researcher using raw trace-files or NetFlow logs made available and using diversified methods and tools to extract knowledgeable features from these. This is a problem because researchers are, one, spending time in finding useful features extractable from those standard formats and two, creating privately processed datasets. For the previously mentioned reasons, researchers are forced to compare their work results with other works which use completely different network feature formats and feature-sets.

4. Problem 3, consequently, results in at least two other problems:

- Research related to Machine Learning algorithms applied to the network intrusion detection problem will be, in the most part, inherently inconclusive, because the dataset features worked by each classifier have different formats. Classifier benchmarking is often performed in processed datasets using custom network-based feature formats, so very often the work becomes incomparable with other researchers' work. At the same time, used network traffic quality is difficult to be evaluated and compared because instead of there being a focus only on the traffic, the focus

of the research papers constantly shifts between the used feature-sets (such as those generated by tools and datasets), the labelling methodology (e.g., fine-grained, coarse-grained, manual) and the traffic used to generate datasets.

- Extended signature-based botnet detection techniques, for example, often focus on packet-based features and use flow-based features [3] to aid detection. Additionally, by analyzing different anomaly- and behavior- based botnet detection techniques, we observed that there exists a focus on flow-based features and a few conceptual talker- and host- based features; furthermore, some works also consider the contacted ports and directly assume that a L7 protocol is being used [23] which might introduce unintended bias. To consider the used L7 protocol, we recommend extending the port-protocol correlation with the use of traffic validation techniques like L5-7 protocol fingerprinting in case the traffic is encrypted, or a direct validation of the protocol in case traffic is unencrypted. This thesis does not contribute with such techniques and would also assume port-protocol correlation for a more optimized detection if needed, because this work would not be doable for all relevant L5-7 protocols in the due time. However, it is important that weaknesses like this are duly documented and that solutions are eventually put in-place, since adversaries can leverage those weaknesses to evade detection. The reason we propose L5-7 protocol fingerprinting to solve this problem is because it is an encryption-agnostic solution. This and other improvements are mentioned as future work.

Chapter 3. Network Threat Class Taxonomy

The current chapter is divided in multiple subsections enumerating threat classes, and the taxonomy that we define in this chapter is mostly based on our experience on network attacks, as well as knowledge acquired from multiple references [67]-[137] marked as “network attack research reference”.

Threat Class	Label	OSI Layer	Threat
3.1. Host Discovery	3.1.1	L2	ARP Host Discovery
	3.1.2	L3	IP Protocol Host Discovery
	3.1.2	L3+	ICMP Host Discovery
	3.1.4	L4	UDP Host Discovery
	3.1.5	L4	TCP Host Discovery
3.2. Port Scan	3.2.1	L4	UDP Port Scan
	3.2.2	L4	TCP Port Scan
	3.2.3	L4	SCTP Port Scan
3.3. L3 Service Discovery	3.3.1	L3	IP Protocol Scan over IPv4 (-sO)
	3.3.2	L3	IP Protocol Scan over IPv6
3.4. L7 Brute Force Attack	3.4.1	L7	FTP Brute Force Attack
	3.4.2	L7	SSH Brute Force Attack
	3.4.3	L7	Telnet Brute Force Attack
	3.4.4	L7	SMTP Brute Force Attack
	3.4.5	L7	POP3 Brute Force Attack
	3.4.6	L7	RDP Brute Force Attack
	3.4.7	L7	HTTP-application Brute Force Attack
	3.4.8	L7	HTTPS-application Brute Force Attack
3.5. L3+ Resource Exhaustion Denial of Service Attack	3.5.1	L3+	ICMP Denial of Service Attack
3.6. L4 Resource Exhaustion Denial of Service Attack	3.6.1	L4	UDP Denial of Service Attack
	3.6.2	L4	TCP Denial of Service Attack
	3.6.3	L4	SCTP Denial of Service Attack
3.7. HTTP Resource Exhaustion Denial of Service Attack	3.7.1	L7	HTTP Low and Slow Attack
	3.7.2	L7	HTTP Flood
3.8. Logical Denial of Service Attack	3.8.1	L1-7	Network Protocol Exploitation
	3.8.2	L7	Application Layer Logical Exploitation

TABLE 1. THREATS AND LABELS BY THREAT CLASS.

The main objective of this chapter is establishing a technical taxonomy on common network threat classes, establishing a strong ground truth concept for each. The following concepts, used in this chapter, relate as follows:

- A Threat Class is implemented by one or more Threats.
- A Threat Class encompasses one or more Generic Attack Techniques.
- A Threat Class has an Intent.
- Intent describes the main objectives behind a Threat Class.
- A Threat encompasses one or more Specific Attack Techniques.
- A Generic Attack Technique is implemented by one or more Programs.
- A Specific Attack Technique is implemented by one or more Programs.
- Program Applicability describes criteria for acceptable Programs.

Table 1 shows each threat considered by each threat class, as well as their labels, which we use to refer to them in each of the following subsections. In the context of this chapter only, labels in-between square brackets are used to refer to their associated threat classes and threats.

3.1. L2-L4 Threat Class: Host Discovery

Intent: Probe multiple selected hosts to find active ones.
Generic Attack Technique(s):

- CIDR Selection - the attacker probes multiple hosts contained by a network range written in the Classless Inter-Domain Routing (CIDR) notation. [3.1]
- Host Range Selection - the attacker probes multiple hosts by specifying a range of IP addresses. [3.1]

Specific Attack Technique(s):

- ARP Ping Scan (-sn -PR) - within a Local Area Network (LAN), the attacker sends an ARP request to a destination MAC address, which can either be a single MAC address, a multicast MAC address or the broadcast MAC address (the most common). If any device is listening on those channels, it will respond (given a normal system configuration) to the request with a valid MAC address associated with the IP, according to its ARP table, given that default dynamic ARP table entries are enabled. This scan is very powerful to find hidden devices in a network, since ARP requests will very likely be responded to by whoever owns the requested information and is actively listening on those channels. In case of a response, we have confirmation that the host is active, unless the respondent host had ARP table entries that should already have expired or, in a more unusual case, if the respondent host was ARP spoofed. [3.1.1]
- IP Protocol Ping (-sn -PO) - for each host, the attacker sends multiple raw IP packets containing the IP protocol number in the IP header. For example, the attacker can send six raw IP packets, each containing a different protocol: ICMP (protocol 1), IGMP (protocol 2), IP-in-IP (protocol 4), TCP (protocol 6), UDP (protocol 17) and SCTP (protocol 132). This method looks for either responses using the same protocol (host supports protocol) or ICMP protocol unreachable messages (host doesn't support protocol), both indicating that the target host is alive. [3.1.2]

- ICMP Echo Request Scan (-sn -PE) - for each host, the attacker sends an ICMP type 8 packet. If the host responds with an ICMP type 0 packet, it is up, else the host may be down or the packet was filtered. [3.1.3]
- ICMP Timestamp Request Scan (-sn -PP) - for each host, the attacker sends an ICMP type 13 packet. If the host responds with an ICMP type 14 packet, it is up, else the host may be down or the packet was filtered. [3.1.3]
- ICMP Information Request (does not exist in Nmap) - for each host, the attacker sends an ICMP type 15 packet. If the host responds with an ICMP type 16 packet, it is up, else the host may be down, this service is not implemented on the end device or the packet was filtered. [3.1.3]
- ICMP Address Mask Request Scan (-sn -PM) - for each host, the attacker sends an ICMP type 17 packets. If the host responds with an ICMP type 18 packet, it is up, else the host may be down, this service is not implemented on the end device or the packet was filtered. [3.1.3]
- UDP Ping Scan (-sn -PU) - for each host, the attacker sends a UDP request to one given port. If the host responds, the host is up, else the host may be down or the packet was filtered. [3.1.4]
- TCP SYN Ping Scan (-sn -PS) - for each host, the attacker sends a TCP request with the SYN flag activated to one test port. If the host responds with SYN-ACK or RST, the host is up, else the host may be down or the packet was filtered. [3.1.5]
- TCP ACK Ping Scan (-sn -PA) - for each host, the attacker sends a TCP packet with the ACK flag activated to one test port. If the host responds with SYN-ACK or RST, the host is up, else the host may be down or the packet was filtered. [3.1.5]

Program Applicability: Programs that can communicate over a network can eventually be used for host discovery, given that the protocols used to communicate are supported by the targeted machine. However, we will only consider a host discovery program as such if it complies with at least one of the following conditions:

- It supports sending and interpreting ARP probes for multiple hosts
- It supports sending and interpreting IP protocol probes (raw IP packets specifying the probed IP protocol number on the IP header) for multiple hosts
- It supports sending and interpreting TCP, UDP and ICMP probes for multiple hosts
- Optionally, these programs can also support other protocols such as SCTP. Also, the existence of any L5-7 protocol is irrelevant for this category.

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

- NetDiscover (ARP)
- UnicornScan - 3.1.1, 3.1.2, 3.1.3 (TCP, UDP, ICMP)
- Nmap - 3.1.1, 3.1.2, 3.1.3 (ARP, raw IP, ICMP, UDP, TCP, SCTP)
- Ncat - 3.1.1, 3.1.2, 3.1.3, 3.1.4, 3.1.5 (UDP, TCP, SCTP)
- Hping3 - 3.1.1, 3.1.2, 3.1.3, 3.1.4 (raw IP, ICMP, UDP, TCP)
- AngryIPScanner - 3.1.1, 3.1.2 (ICMP, UDP, TCP)
- Masscan - 3.1.1, 3.1.2 (ICMP, UDP, TCP)
- ZMap - 3.1.1, 3.1.2 (ICMP, UDP, TCP)

3.2. L4 Threat Class: Port Scan

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. [3.2]
- 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. [3.2]

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. [3.2.1]
- 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. [3.2.2]
- 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. [3.2.2]
- 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. [3.2.2]
- 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. [3.2.2]
- 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. [3.2.2]
- 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. [3.2.2]

- 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. [3.2.2]
- 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. [3.2.2]
- 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. [3.2.2]
- 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. [3.2.3]
- 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. [3.2.3]

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 - 3.2.1, 3.2.2, 3.2.3 (TCP, UDP, ICMP)
- Nmap - 3.2.1, 3.2.2, 3.2.3 (ARP, raw IP, ICMP, UDP, TCP, SCTP)
- Ncat - 3.2.1, 3.2.2, 3.2.3, 3.2.4, 3.2.5 (UDP, TCP, SCTP)
- Hping3 - 3.2.1, 3.2.2, 3.2.3, 3.2.4 (raw IP, ICMP, UDP, TCP)
- AngryIPScanner - 3.2.1, 3.2.2 (ICMP, UDP, TCP)
- Masscan - 3.2.1, 3.2.2 (ICMP, UDP, TCP)
- ZMap - 3.2.1, 3.2.2 (ICMP, UDP, TCP)

3.3. L3 Threat Class: L3 Service Discovery

Intent: Find out information about a target host using raw L3 requests.

Generic Attack Technique(s): Unspecified.

Specific Attack Technique(s):

- IP Protocol Scan over IPv4 (-sO) - the objective of this scan is determining what IP protocols, running over IPv4, are available in the target host. The attacker sends an IPv4 packet to the target host, with the "Protocol" field filled in the IPv4 header for each targeted IP protocol number. For example, the attacker can send six IP packets asking for six IP protocols support: ICMP (protocol 1), IGMP (protocol 2), IP-in-IP (protocol 4), TCP (protocol 6), UDP (protocol 17) and SCTP (protocol 132). If the attacker receives a response from the target host using the probed protocol or an ICMP "Destination Unreachable - Port Unreachable" error (ICMP Type

3, code 3), the protocol is supported (open). If an ICMP "Destination Unreachable - Protocol Unreachable" error (ICMP Type 3, code 2) is received, the protocol is marked as unsupported (closed). Other ICMP "Destination Unreachable" errors (ICMP Type 3; codes 0, 1, 9, 10, or 13) cause the protocol to be marked filtered. If no response is received after retransmissions, the protocol is marked as possibly supported (open or filtered). [3.3.1]

- IP Protocol Scan over IPv6 - the objective of this scan is determining what IP protocols, running over IPv6, are available in the target host. The attacker sends an IPv6 packet to the target host, with the "Next Header" field filled in the IP header for each targeted IP protocol number. Similarly to the IPv4 protocol scan, if the attacker receives a response from the target host using the probed protocol then the protocol is supported. However, the interpretation of the responses will differ since ICMPv4 and ICMPv6 responses differ. [3.3.1]

Program Applicability: Any program that allows sending and interpreting multiple L3 service-related probes.

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

- Nmap - 3.3.1, 3.3.2, 3.3.3 (ARP, raw IP, ICMP, UDP, TCP, SCTP)
- Hping3 - 3.3.1, 3.3.2, 3.3.3, 3.3.4 (raw IP, ICMP, UDP, TCP)

3.4. L7 Threat Class: L7 Brute Force Attack

Intent: Test multiple credential combinations in a continuous manner to find out correct ones.

Generic Attack Technique(s):

- Traditional Brute Force Attack - test multiple passwords per few accounts. [3.4]
- Reverse Brute Force Attack (a.k.a. Password Spraying Attack) - test few passwords per multiple accounts. [3.4]

Specific Attack Technique(s): Unspecified.

Program Applicability: Any program that supports brute-forcing credentials associated with a L7 protocol.

Programs - <name> (<L1-4 protocols supported>) (<L5-7 protocols supported>):

- Ncat (UDP, TCP, SCTP) (None in particular)
- Patator (TCP) (FTP, SSH, Telnet, SMTP, HTTP/HTTPS, RDP, AJP, POP, IMAP, LDAP, SMB, SNMP)
- ncrack (TCP) (SSH, RDP, FTP, Telnet, HTTP/HTTPS, HTTP/HTTPS WordPress websites, POP3/POP3S, IMAP, CVS)
- CrackMapExec (SMB)

3.5. L3+ Threat Class: L3+ Resource Exhaustion Denial of Service Attack

Intent: Overwhelm a target system with multiple malicious L3-level control queries, with the goal of exhausting that system's network and/or computational resources.

Generic Attack Technique(s):

- Distributed Denial of Service (DDoS) Attack - use multiple systems to attack a target system. [3.5]
- Reflection and Amplification Attack - the attacker uses systems which are running specific network protocols that respond to small requests with large responses. This fact provides an attacker the possibility of sending multiple spoofed requests with the target's IP address (as source address) and redirect those systems' responses to the target system, resulting in a Distributed Denial of Service (DDoS) attack. [3.5]

Specific Attack Technique(s):

- ICMP Ping (Type 8) Flood - the attacker sends multiple ICMP "Echo" (ICMP Type 8) request packets to the target system. [3.5.1]
- ICMP Destination Unreachable (Type 3) Flood - the attacker sends multiple ICMP "Destination Unreachable" (ICMP Type 3) packets to the target system. Although this ICMP packet type is a response, since the ICMP protocol is not stateful, the packet will still be processed. [3.5.1]
- ICMP Time Exceeded (Type 11) Flood - the attacker sends multiple ICMP "Time Exceeded" (ICMP Type 11) packets to the target system. Although this ICMP packet type is a response, since the ICMP protocol is not stateful, the packet will still be processed. [3.5.1]
- Smurf Attack (specific "Reflection and Amplification Attack") - the attacker broadcasts spoofed ICMP "Echo Request" packets on a network, so that systems which are listening on the IP broadcast address send ICMP "Echo Reply" response packets to the target system. [3.5.1]
- Fraggle Attack (specific "Reflection and Amplification Attack") - the attacker sends spoofed UDP requests to multiple systems at ports 7 (Echo Protocol) and 19 (CHARGEN Generator Protocol), so that those systems send ICMP "Destination Unreachable - Port Unreachable" (ICMP Type 3, code 3) response packets to the target system. [3.5.1]

Program Applicability: Every program that is able to perform multiple malicious requests against a L3-level control service (subset of L3+ services) to cause network and computational resource exhaustion on the targeted system, ultimately resulting in a lack of availability to legitimate L3-level control queries.
Programs - <name> (<L1-4 protocols supported>): Hping3 (raw IP, ICMP, UDP, TCP)

3.6. L4 Threat Class: L4 Resource Exhaustion Denial of Service Attack

Intent: Overwhelm a target system with multiple malicious L4-level requests directed towards a network service using a given L4 protocol on a given port, with the goal of exhausting the target system's network and/or computational resources.
Note: Since any received packet needs to be processed by the network stack of the targeted system, a denial of service attack may still occur against closed ports, which is why we often use the term "system" rather than "server", which designates a system running a network service usable by clients. We use the term "server" whenever the attack is only applicable against one.
Generic Attack Technique(s):

- High-Rate Attack - quickly and continuously launch multiple requests against a target system. [3.6]
- Distributed Denial of Service (DDoS) Attack - the attacker uses multiple systems to attack a target system. [3.6]
- Reflection and Amplification Attack - the attacker uses servers which are running specific network protocols that respond to small requests with large responses. This fact provides an attacker the possibility of sending multiple spoofed requests with the target's IP address (as source address) and redirect those servers' responses to the target system, resulting in a Distributed Denial of Service (DDoS) attack. [3.6]
- Low-Rate Attack - the attacker launches multiple L4-level requests against a target server and, for each established connection, slowly sends data back to the server to keep it holding to the connection as long as possible. [3.6]

Specific Attack Technique(s):

- UDP Reflection and Amplification Attack - the attacker sends multiple spoofed UDP packets to appear as if these packets originated from the target's network IP address, to multiple systems running UDP services. This results in those multiple systems reflecting large UDP response packets to the target's network, resulting in a Distributed Denial of Service (DDoS) attack. DNS and NTP are examples of UDP services that are very usually used to perform this kind of attack, but many others can be used as well. More recently, in February 2018, the Memcached service was used for this kind of attack with an unprecedented amplification factor. [3.6.1]
- TCP SYN-ACK Reflection and Amplification Attack - the attacker sends multiple spoofed TCP packets to appear as if these packets originated from the target's network IP address, to multiple systems running TCP services. This results in those multiple systems reflecting TCP SYN-ACK response packets to the target's network, resulting in a Distributed Denial of Service (DDoS) attack. [3.6.2]
- TCP SYN Flood Attack - the attacker sends multiple SYN packets to a target server, resulting in multiple SYN-ACK responses, only to never send any ACK back to the target server. This results in the target server maintaining multiple sockets occupied for the initiated half-open connections, resulting in a denial of service for legitimate clients who want to connect to those ports. [3.6.2]
- TCP "Tsunami" Flood Attack - similar to the TCP SYN Flood attack, however sent packets contain garbage data to cause the server additional stress when processing each request. [3.6.2]
- TCP Custom Flag Floods - the attacker sends multiple TCP packets with custom sets of TCP flags. Some already used attacks based on custom flag combinations are: URG-PSH-SYN Flood, URG-PSH-RST Flood, "All TCP Flags" Flood (Xmas Flood), ACK-SYN Flood, PSH-RST-FIN Flood, URG-ACK-FIN Flood, among others. [3.6.2]
- TCP Connection Flood - the attacker sends multiple SYN packets to a target server, resulting in multiple SYN-ACK responses, to which the attacker will respond with ACK packets ideally in the longest time possible before the server times out from the connection attempt. This results

in multiple longest-time connection initiations (3-way handshakes) between the attacker and the server to exhaust server's resources for the longest time possible, which the attacker may complement with additional measures to keep the connection active for the longest time as well.

- TCP Connection Flood Stress (TCP Sockstress Attack 1) - similar to the TCP connection flood. [3.6.2]
- TCP Zero Window Connection Stress (TCP Sockstress Attack 2) - the attacker initiates a TCP connection with the target server. The attacker sends zero-sized window TCP packets, begun to be specified in the last ACK packet of the 3-way handshake, expressing a false unavailability to receive any packets with a data size greater than 0 bytes. In response to the former, the server stores in memory all the data it has yet to send. The attacker will then continuously request the expected X-byte sized chunks at a specified rate and in specified intervals, which will optimally be the slowest rate and intervals at which the target server keeps the connection active, for the longest time possible and avoiding any timeout event. Since the server will have to hold on to the stored data, it will incur in excessive memory consumption. [3.6.2]
- TCP Small Window Stress Attack (TCP Sockstress Attack 3) - the attacker initiates a TCP connection with the target server. The attacker sends small-sized window TCP packets, begun to be specified in the last ACK packet of the 3-way handshake, expressing a false unavailability to receive packets with a data size greater than X bytes (the Sockstress's framework defines 4 bytes as the default window size). In response to the former, the server splits up the data it has yet to send into multiple X-byte chunks and stores it in memory. The attacker will then continuously request the expected X-byte sized chunks at a specified rate and in specified intervals, which will optimally be the slowest rate and intervals at which the target server keeps the connection active, for the longest time possible and avoiding any timeout event. Since the server will have to hold on to the data that it is very slowly being sent, it will incur in excessive memory consumption. [3.6.2]
- TCP Segment Hole Stress (TCP Sockstress Attack 4) - the attacker initiates a TCP connection with the target server. The attacker sends 4 bytes to the beginning of the TCP window, then sends 4 bytes to the end of the TCP window, and then sets the windows size to zero. The network stack vulnerable servers may respond to the former attack by allocating multiple pages of kernel memory per connection made, incurring in excessive memory consumption. Note: this attack is yet unclear in its execution, it would need to be further analyzed. [3.6.2]
- TCP Req Fin Pause Stress (TCP Sockstress Attack 5) - the attacker initiates a TCP connection with the target server. The attacker sends a L7 application payload (e.g. HTTP GET) inside a TCP PSH packet. The attacker then sends a FIN packet with a zero size window, to which vulnerable servers will not respond with a FIN-ACK packet to close the connection, but rather will maintain the connection open on their side and indefinitely keep the socket occupied on the FIN_WAIT_1 state (which means that the socket knows the remote computer has closed the connection, but it is still waiting for the local application that was using the socket to acknowledge the end of the connection and finally allow releasing the socket). [3.6.2]

- TCP Activate Reno Pressure Stress (TCP Sockstress Attack 6) - the attacker initiates a TCP connection with the target HTTP server, sends a L7 application payload (e.g. HTTP GET) inside a TCP PSH packet and sends three duplicate ACK packets. Note: it would be interesting to find more reliable information about this attack, however I could not find any more information on it. [3.6.2]
- SCTP INIT Flood - the attacker sends multiple SCTP INIT packets to a target system that supports SCTP. [3.6.3]
- SCTP Address Camping - the attacker connects to an SCTP server and "camps upon" or "holds up" a valid peer's IP address, preventing the legitimate peer from communicating with the server. This technique targets the SCTP's multi-homing feature and directly affects the peers' ability to establish a connection with the server. [3.6.3]
- SCTP Reflection and Amplification Attack (dubbed SCTP Bombing Attack) 1, 2, 3, 4 and 5 – The five attacks are specified in RFC 5062 [3.6.3]

Program Applicability: Every program that can perform multiple malicious requests against a L4 service or any application running over a L4 service to cause network and computational resource exhaustion on the targeted server, ultimately resulting in a lack of availability to legitimate clients.

Programs - <name> (<L1-4 protocols supported>) (<L5-7 protocols supported>):

- Ncat (UDP, TCP, SCTP) (None in particular)
- Hping3 (raw IP, ICMP, UDP, TCP) (None in particular)
- DoS Goldeneye (TCP) (HTTP/HTTPS)
- DoS Hulk (TCP) (HTTP/HTTPS)
- DoS Slowloris (TCP) (HTTP/HTTPS)

3.7. L7 Threat Class: HTTP Resource Exhaustion Denial of Service Attack

Intent: Overwhelm a target HTTP server with multiple malicious HTTP requests in order to exhaust its network and/or computational resources.

Generic Attack Technique(s):

- High-Rate Attack - quickly and continuously launch multiple requests against a target HTTP server. [3.7]
- Distributed Denial of Service (DDoS) Attack - the attacker uses multiple systems to attack a target HTTP server. [3.7]
- Reflection and Amplification Attack - the attacker uses servers which are running specific network protocols that respond to small requests with large responses. This fact provides an attacker the possibility of sending multiple spoofed requests with the target's IP address (as source address) and redirect those servers' responses to the target server, resulting in a Distributed Denial of Service (DDoS) attack. [3.7]

- Low-Rate Attack - the attacker launches multiple HTTP requests against a target server and, for each established connection, slowly sends data back to the server to keep it holding to the connection as long as possible. [3.7]

Specific Attack Technique(s):

- Slowloris - the attacker performs multiple HTTP persistent connections with the target server and slowly sends partial HTTP headers to it, which will keep it waiting for the receival of the rest of the headers. Timeouts are avoided by periodically sending "Keep alive" (not to confuse with the HTTP header value "keep-alive" used on the "connection" field) packets, i.e., "PSH-ACK" TCP packets transporting partial headers on the data field. [3.7.1]
- R.U.D.Y (R-U-DEAD-YET) - the attacker generates multiple POST requests to fill out form fields and tells the server how many bytes it should expect using the "Content-Length" HTTP header field. Then, the attacker sends small-sized TCP packets with the expected data at very slow rates, which results in the server holding on to the TCP socket to receive the rest of the data for a long time. [3.7.1]
- HTTP GET Flood - the attacker sends multiple HTTP GET requests to the target server. [3.7.2]
- HTTP POST Flood - the attacker sends multiple HTTP POST requests to the target server. [3.7.2]

Program Applicability: Every program that is able to perform multiple malicious requests specifically against an HTTP application to cause network and computational resource exhaustion on the targeted server, ultimately resulting in a lack of availability to legitimate clients.

Programs - <name> (<L1-4 protocols supported>) (<L5-7 protocols supported>):

- DoS Goldeneye (TCP) (HTTP/HTTPS)
- DoS Hulk (TCP) (HTTP/HTTPS)
- DoS Slowloris (TCP) (HTTP/HTTPS)

3.8. L1-7 Threat Class: Logical Denial of Service Attack

Intent: Exploit a network service or application, vulnerable to a logic flaw, running on the target system. Exploitation of those is performed through specific actions that highly depend on very specific vulnerabilities.

Generic Attack Technique(s): Unspecified.

Specific Attack Technique(s):

- Specially Crafted Packets - this technique involves the exploitation of a logical flaw in a network service that is actively running on the target system by sending a set of packets that, far from the expected format, are able to create a malfunction in the network service itself. This technique may make use of an unforeseen vulnerability in a designed network protocol or, if not on the protocol itself, a vulnerability in its code implementation. "Teardrop" (L3), "Ping of Death" (L3+), "Land" (L4 - TCP) and "SCTP Association Redirection" (L4 - SCTP) are examples of this technique. [3.8.1]

- Application Layer Logical Exploitation - this technique leverages logical mistakes in a specific application to cause its unavailability. As an example, if a server requires default guest user credentials to provide data to any user, all users must locally own those credentials (even if "under the hood") to authenticate to the server. As such, users might also be able to issue a password change request to the server to change those credentials if this "guest user" account is not treated with caution server-side. If no control is put in place for this situation, a single user could be able to deny every other user from authenticating to and receiving data from the servers, since every users' locally saved guest user credentials would not be valid anymore. [3.8.2]

Program Applicability / Programs: Custom exploits.

Chapter 4. Network Objects

We consider four different network objects: packets, flows, talkers, and hosts. Below, we describe the relations between these four network objects, before studying the datasets using these:

- Legend: PS – Protocol Stack; SH – Source Host; SP – Source Port; DH – Destination Host; DP – Destination Port; H – Host; FSC – Flow Separation Counter.
- Packet: Single unit. Defined by SH, SP, DH, DP and PS.
- Flow: gathers SH-SP<->DH-DP forward and backward packets:
    - 5-tuple Flow: Includes the protocol stack in the flow definition but does not use the highest-layer protocol to logically separate it. A 5-tuple flow is orderly defined by SH, SP, DH, DP, PS.
    - 6-tuple Flow: Includes the protocol stack and uses the highest-layer protocol to logically separate the flow, thus requiring a sixth parameter (which we name “inner flow counter” or “flow separation counter”). In this work, an example of this is the TCP flow, which requires separation using the TCP flags and the TCP sequence (SEQ) and acknowledgement (ACK) numbers. A 6-tuple flow is orderly defined by SH, SP, DH, DP, PS and FSC.
- Talker: gathers SH<->DH forward and backward 6-tuple flows. A talker is orderly defined by SH, DH and PS.
- Host: gathers H forward and backward talkers. A host is orderly defined by H and PS.

The Flow, Talker and Host objects we use are bidirectional and can also be referred to as Bi-Flow, Bi-Talker and Bi-Host. We developed a network object extraction tool dubbed “NetGenes”, which generates these network objects from multiple packets (captured on a PCAP or PCAPNG file). Inspired by CICFlowMeter, it generates a high number of Flow features, both conceptual and statistical, as well as Talker (flow set) and Host (talker set) data points.

4.1. NetGenes: network-object feature extraction 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. Host-based flow-set features can also be created but were not implemented. We consider “eth-eth-ipv4” as the protocol stack for talkers and hosts, and we uniquely identify them using their IPv4.

Network Object	Aggregated Network-object Features
Flow	Packet-set based features
Talker	Flow-set based features
Host	Talker-set (and flow-set) based features

TABLE 2. NETGENES NETWORK OBJECTS AND FEATURE SOURCE.

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’ summarized architecture is presented in figure 2 below. The currently implemented network-object features are presented in the following annex tables: table 15 presents the packet features; table 16 presents the flow features; table 17 presents the talker features; table 18 presents the host features (not considering flow-set based features).

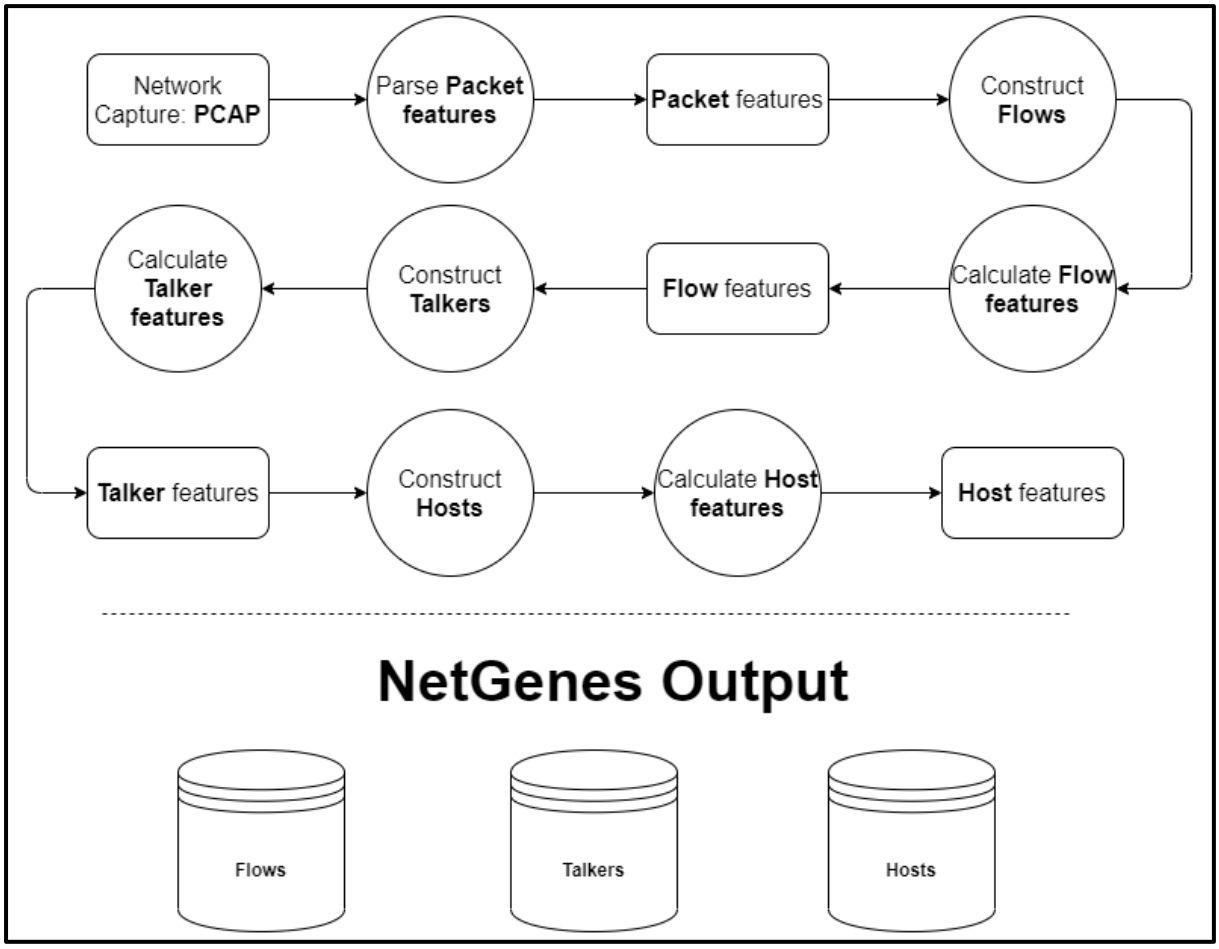

FIGURE 2. NETGENES SUMMARIZED ARCHITECTURE.

4.2. 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 also because custom parameters can be given to these tools (and other methods as well) to alter the generated network traffic enough to be able to evade packet-based and signature-based detection, as well as evade flow feature analysis methods that do not solely focus on the core features of the threat class. Additionally, since neither CICFlowMeter nor other pure “flow” extraction tools extract flow-set based features, most researchers usually only use flow-based features to feed ML models and study threat classes, which is a big loss of perspective on the information.

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 multiple statistical flow-based features and attempt to model a whole threat class around those features. This

may achieve great results because ML models are capable of creating very complex rule sets within themselves, but the main problem is that these models will most likely rely in features that do not truly define the threat class because the train data is not broad enough to help the algorithm ignore non-core features. If the train data is not broad enough to create a generic model, it will result in network attacks falling undetected to the generated algorithm when they completely drop said non-core features. It is a common problem that a ML model will overfit around multiple non-core features based on the train dataset and achieve 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 commonalities within the used train and test datasets that allows detecting those instances.

The previous problem is the reason why, for ML-based research for network traffic analysis specifically applied to network attacks, if we want to find relevant commonalities that lead us to better understand a threat class, it is important to use broad train and test datasets with a preference for multiple scenarios within the same threat class using a single label between all scenarios. It is also very difficult to truly understand a ML classifier (unless good explainable AI methods are employed) and we cannot easily outline its limitations, as there is a lack of transparency in the way classifications are made. To avoid this issue, we created rule sets to directly detect the threat class and its core scenarios.

4.3. NetGenes-based rule set guide

Creating a rule set based on NetGenes features should follow some guidelines:

- Avoid using time-based flow features - time-based flow features can be inaccurate. For example, if a network interface is being flooded with packets, it uses its processing buffer to store packets that it cannot process in the current time. This causes that a packet that arrives at a certain moment *t* will only be processed by the software in the moment *t*+*n* and, unfortunately, *t*+*n* is the one that is stored as the packet timestamp, rather than the real packet arrival time *t*.
- Avoid using forward non-core flow features – forward flow features are controllable by the adversary. When we want to detect flows using these, it is best that these features are necessary (core features) for the malicious activity we want to detect. This guideline is commonly broken with ML models, as it will be more inclined to perform a complex form of flow fingerprinting based on its training data, which also breaks the following guideline.
- Avoid using too many flow features – avoid using too many features, which usually leads to being too specific about what flows to detect. If this is needed, we should at least be aware that we are being too specific about it, as it is likely that it represents flow fingerprinting, which is more specific than flow profiling and should not be used to detect threat class flows but, instead, more specific flows (e.g., flows created by specific software). Then, we should improve the rule set in the future to generalize it enough for the threat class, when we confirm that the flows filtered by this rule will, with a high probability, belong to the target threat class.

4.4. Network traffic analysis: packets, flows and flow sets

Some common examples of packet- and flow- based traffic analysis are:

- **Firewall Stateless rules –** packet-based filtering rules, these rules use packet metadata to allow and route or to block those packets.
- **Firewall Stateful rules –** flow-based filtering rules, these rules use packet metadata to maintain track of active connections, to allow or to block those connections.
- **Deep Packet Inspection (DPI) –** packet-based detection technique, it uses the information present in a packet to detect it based on its transport protocol's data. This technique allows parsing and creating detection rules for protocols above OSI-layer 5. In the case of decrypted traffic, it allows for signature-based detection based on L5-7 protocol specifics, as well as detection based on regex filters.
- **Flow Fingerprinting –** flow-based detection technique, it uses information extracted from all the packets of a certain flow to identify the software that generated it. It works because many tools are consistent in the way that they generate unique identifiable network flows. It works on encrypted traffic because it exclusively uses packet metadata extractable below OSI-layer 4 to precisely identify these flows. It allows guessing flow (and packet) contents and context, even if encrypted, based on multiple L1-L4 features only.

Flow fingerprinting can be differentiated from the flow-based detection techniques we employ in this work for its stronger specificity in analyzing a significant number of features which serve as a "fingerprint" identification of a given flow. Thus, if the considered flow features are not core to the threat class, an attack's signature may become outdated in time; additionally, an adversary has the possibility of tampering with those non-core flow features to avoid detection, while still maintaining attack's effectiveness.

In this work, we employ flow-based detection methods focused on core flow features, aided by talker- and host- based detection methods that allow logically grouping flows by core flow-set features. We call this flow-set based detection.

In this matter, perspective is a key element. Packet-based detection is comparable to trying to solve a labyrinth in the view of the player. Flow-based detection is like having a limited perspective of the labyrinth from above. Flow-set based detection is comparable to viewing the whole labyrinth, providing the context that is needed to improve the abstraction level and consequently facilitate problem solving.

As an example, for port scan detection, a flow-based detection method does not retrieve or otherwise consider data points that are core to its detection, like the unique destination port count based on source host - destination host pairs (talker-based detection) or destination hosts (host-based detection), which flow-set based detection provides by default.

4.5. NetGenes: Limitations and Considerations

Regarding talker features, we defined the direction of forward and backward flows using the talker's direction as reference, which by convention is set as the direction of the first flow of the talker. On the other hand, regarding host features, we defined the direction of forward and backward talkers using each host as its own reference. This means that the number of forward talkers in a host represents the number of talkers whose first flow was initiated from/by the current host, while the number of backward talkers represents the number of talkers whose first flow was initiated to the current host. Although this convention is needed to properly define the talker and the host object, we should have also kept flow-derived features in host features as they would allow us to answer simple host queries that cannot otherwise be answered by talker-derived features. For example, implementing flow aggregation by host enables us to answer the question "How many services did host A have accessed by other hosts?". Answering the previous question is like answering the question "How many services did host B attempt to access on host A?", already answered by "bitalker_fwd_biflow_n_unique_dst_ports" and "bitalker_bwd_biflow_n_unique_dst_ports" talker features, but it is relative to detecting a single host's probed ports, which is only doable using the "bihost_bwd_biflow_n_unique_dst_ports" host feature. Why does this not work with talker's unique destination ports data? Because, if there are multiple talkers for one host, we will not be able to determine the unique destination ports of each talker that overlap with one another; we need to use the flow object to directly fetch this data.

NetGenes TCP flows may disregard several packets because their flows were not initiated within the packet capture. This is not a limitation, but rather a choice of disregarding incomplete flows from the generated traffic data.

The Talker and Host network objects, both considering flow-set and talker-set features, are not considered in most works, mainly due to the prevalence of ready-to-use data using only flow extraction tools. By performing flow aggregation, *NetGenes* facilitates traffic analysis.

Chapter 5. CIC-IDS-2017 analysis

Based on the threat class definitions, we can define rule sets based on the extracted features to detect network attacks while leaving out benign traffic.

Through-out this chapter, we will always sum 3 hours to the times defined by the Canadian Institute for Cybersecurity (CIC) in their CSV files and official website [145]. The times we consider are the ones that the PCAP files we used to extract the data show, which are in relation to the Lisbon time zone (WET).

The CSV files we always refer to are the ones in the “GeneratedLabelledFlows.zip” file (md5: “5ca3f8f69e3514950681615824149973”, last seen 2020/12/10), since the “MachineLearningCSV.zip” (md5: “4f83860afbf29cac8163854095bf6cf7”, last seen 2020/12/10) file just contains the same CSV files with 79 columns instead of the original 85 columns, and we need the 6 removed columns (“Flow ID”, “Source IP”, “Source Port”, “Destination IP”, “Protocol” and “Timestamp”) to correctly map CICFlowMeter-generated flows to NetGenes-generated flows. There are 83 features in these 85 columns, as “Flow ID” and “Label” are not considered features.

We found there are CIC-IDS-2017 flows that are mislabeled in files that have mixed benign and malicious traffic, i.e., all files except Monday’s file. The following information was used to check the CIC-IDS-2017 CSV dataset files for errors in labels, to crosscheck with the information supplied in the official CIC-IDS-2017 website [145], the official CIC-IDS-2017 paper [146] and their dataset:

- TCP regular expression for CIC-IDS-2017 flow CSV files (IP protocol number “6”): “^[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+\-[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+\-.*\-6\,.*<CIC-IDS-2017 Label>”
- UDP regular expression for CIC-IDS-2017 flow CSV files (IP protocol number “17”): “^[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+\-[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+\-.*\-17\,.*<CIC-IDS-2017 Label>”
- NetGenes-generated bi-flows, bi-talkers and bi-hosts, and threat-class rule sets.

Furthermore, the way we study the CIC-IDS-2017 dataset is per weekday. We use our different rule sets to understand what is going on each day and interpret the network traffic. We also point out some mislabeled network traffic that we have detected along the way, which we hope the Canadian Institute for Cybersecurity notices and eventually correct for future researchers.

5.1. Metrics

In this subsection, we define metrics for a binary classification contemplating only class and non-class results. A class may be represented any labelable object, such as a threat class, a threat, or a tool. A result may be represented by any network object, such as a host, a talker, or a flow.

The classification results are grouped in:

- **True Positive** results (abbreviated **TP**) are results correctly classified as class results.
- **True Negative** results (abbreviated **TN**) are results correctly classified as non-class results.
- **False Positive** results (abbreviated **FP**) are results incorrectly classified as class results.
- **False Negative** results (abbreviated **FN**) are results incorrectly classified as non-class results.

The classification results are made comparable by calculating the following metrics:

- **Sensitivity** or **True Positive Rate** (abbreviated **TPR**) is the rate of results correctly classified as class results among class results.

$$TPR = \frac{TP}{TP + FN} \quad (1)$$

- **Specificity** or **True Negative Rate** (abbreviated **TNR**) is the rate of results correctly classified as non-class results among non-class results.

$$TNR = \frac{TN}{TN + FP} \quad (2)$$

- **Fallout** or **False Positive Rate** (abbreviated **FPR**) is the rate of results incorrectly classified as class results among non-class results.

$$FPR = \frac{FP}{TN + FP} \quad (3)$$

- **Miss Rate** or **False Negative Rate** (abbreviated **FNR**) is the rate of results incorrectly classified as non-class results among class results.

$$FNR = \frac{FN}{TP + FN} \quad (4)$$

We also calculate and use the following metrics:

- **Overall Accuracy** is the rate of correctly classified results among all results. It can be interpreted as the probability that a classification is correct.

$$Overall\ Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (5)$$

- **Precision** is the rate of correctly classified class results among results classified as class results. It can be interpreted as the probability that a class classification is correct. The higher the precision is, the more probable it is that a class-flagged instance is correctly classified, which means that a class classification will yield a high relevance. This is the most important metric to us because we only want to flag a network attack when we are certain that it occurred.

$$Precision = \frac{TP}{TP + FP} \quad (6)$$

- **F1-Score** is the harmonic mean of Sensitivity and Precision. It weighs false positive and false negative results altogether.

$$F1\ Score = \frac{2 \times TP}{2 \times TP + FP + FN} \quad (7)$$

- **Matthews Correlation Coefficient**, abbreviated **MCC**, is used as a measure of the quality of binary (two-class) classifications. The classifications we perform are binary classifications because, even though we distinguish between multiple classes, our rule sets only consider class and non-class instances. The MCC is a very important metric to us because it takes into account the four result groupings (TP, TN, FP and FN) and weighs their sizes to provide a balanced metric. It shows that we can output a high number of suspicious instances (less FN, more TP), without compromising the TN and FP. Unlike the other considered metrics, the MCC varies between -1 (-100%) and 1 (100%). The higher the MCC is, the best we can provide correct (high TP over TP+FP results; high TN over TN+FN results) and complete (high TP over TP+FN results; high TN over TN+FP results) class and non-class classifications.

$$MCC = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}} \quad (8)$$

We are interested in maintaining a balance between all metrics and getting great results for those metrics. However, in many cases, it is not possible to do both, so we must focus on this thesis's objective: "Helping network threat analysts studying and detecting network attacks". We want to make sure that we can present the analyst with malicious instances only, ignoring others. As such, we want to maximize **precision** as much as possible, by maximizing **TP** and minimizing **FP**.

5.1.1. Metrics applied to rule sets

In practice, what does it mean to have a high-precision rule set and why is it so relevant? It means that what we care about the most is the true positives (TP) and false positives (FP). This happens because TP and FP results are the ones that an analyst will actually see when a rule set is applied.

A rule set with a high precision will likely retrieve most output class flows, but may not retrieve a high rate of class flows in relation to all available class flows (i.e., high FNR). It provides enough information to pursue an investigation on more class flows (in other words, a high rate of true maliciously classified flows in relation to all truly malicious flows) by successfully identifying the talker and the hosts involved, as well as the identified set of malicious flows. Precision is our most important metric.

On the other hand, a rule set with a high sensitivity (TPR) will likely retrieve a high rate of class flows in relation to all available class flows, but may result in a higher fallout (FPR). As such, we instead use the F1-Score (harmonic mean of the precision and the sensitivity) as our second most important metric. The F1-Score helps us evaluate rule sets in their ability to be correct (precision) at the same time that they are complete (sensitivity).

Finally, we use the MCC as our third most important metric, to measure the rule set's ability of correctly and completely classify class and non-class instances.

5.2. Network Object Statistics

From this point on, to avoid repetition, we implicitly refer to TCP network objects when referring to network objects, unless we explicitly refer to the UDP protocol.

Using three different tools, *NetGenes*, *Wireshark* and *CICFlowMeter*, we generate statistics about this dataset in table 3. The following acronyms are used for this table: BT – Bi-Talker, UT – Uni-Talker, T – Talker, C – IPv4 Conversation (Wireshark) w/ "eth && tcp" filter, F – Flow, P – Packet.

By analyzing table 3, we can see that the way that we implemented the network objects we implemented in NetGenes is practically paired with Wireshark. Like Wireshark, we create a TCP flow considering the usual flag combinations that matter in the flow initiation and termination phases, i.e., SYN, ACK, RST and FIN packets. We also consider the TCP sequence and acknowledgement numbers to not only validate subsequent packets but, also, to find incorrectly ordered packets in the current 5-tuple flow (excluding previously separated 6-tuple flows within it) and ignore the timestamp order. Considering the SEQ/ACK numbers significantly improved our TCP flow parsing and extraction. One key difference between our tool and Wireshark is that we chose to only consider the packets that belong to a valid detected flow, i.e., we only consider a flow and its packets if the flow has been initiated within

the current capture, so we will explicitly discard all flows and their packets if no initiation has been seen. This choice of ours might be able to explain the subtle difference between our network objects and Wireshark's network objects, but since we want to be consistent in multiple data points, we choose to discard those incomplete flows.

Day	TCP/IPv4 (Eth-Eth-IPv4-TCP protocol stack)									
	NetGenes				Wireshark			CICFlowMeter		
	UT	BT	F	P	T / C	F	P	T	F	P
Monday	21562	21557	132218	10656505	21563	131765	10718485	N/A	305423	10665332
Tuesday	19187	19183	109164	10658296	19188	108195	10710320	N/A	245781	10670088
Wednesday	19666	19663	273858	12701208	19669	278209	12943381	N/A	489450	12896520
Thursday	18394	18387	167903	8466187	18394	159810	8538185	N/A	274624	5689865
Friday	17559	17554	347994	9140249	17563	345060	9192354	N/A	514276	9155169

TABLE 3. NETGENES, WIRESHARK AND CICFLOWMETER: PER-DAY TCP NETWORK-OBJECT STATISTICS.

As for CICFlowMeter, our tool was initially inspired by it to make the most out of packet metadata. We grew our flow features and, further, we created two more network objects above the flow with their own features. In terms of network object parsing and extraction, CICFlowMeter does not implement the "Talker" network object and, by analyzing table 3 results, we can see that CICFlowMeter was not consistent with NetGenes and Wireshark results. Hence, since Wireshark is a tool that is in the industry for years and is commonly accepted by network and cybersecurity specialists, we consider this a good sign that NetGenes is on the right track of achieving a good flow definition.

Day	Label	TCP/IPv4 (Eth-Eth-IPv4-TCP protocol stack) Network Objects		
		(Bi-)Talkers*	Flows	Packets
Monday	None	7 uni- and bi-talkers	10	10
	BENIGN	21561 uni-talkers and 21556 bi-talkers	132208	10656495
Tuesday	FTP-Patator	172.16.0.1-192.168.10.50-TCP	2505	65004
	SSH-Patator	172.16.0.1-192.168.10.50-TCP	2935	161332
	None	13 uni- and bi- talkers	31	31
	BENIGN	19180 uni-talkers and 19177 bi-talkers	103693	10431929
Wednesday	DoS slowloris	172.16.0.1-192.168.10.50-TCP	2089	41181
	DoS Slowhttptest	172.16.0.1-192.168.10.50-TCP	1245	17524
	DoS Hulk	172.16.0.1-192.168.10.50-TCP	154659	2028799
	DoS GoldenEye	172.16.0.1-192.168.10.50-TCP	7458	105160
	None	14 uni- and bi- talkers	22	38
	BENIGN	19666 uni-talkers and 19663 bi-talkers	108384	10459210
Thursday	Web Attack – Brute Force	172.16.0.1-192.168.10.50-TCP	143	22371
	Web Attack – XSS	172.16.0.1-192.168.10.50-TCP	23	5416
	Web Attack – Sql Injection	172.16.0.1-192.168.10.50-TCP	9	94
	Infiltration	192.168.10.8-205.174.165.73-TCP	20	59754
	None	24 uni- and bi-talkers	95	192
	BENIGN	18387 uni-talkers and 18383 bi-talkers	167613	8378360
Friday	PortScan	172.16.0.1-192.168.10.50-TCP	158980	320401
	DDoS	172.16.0.1-192.168.10.50-TCP	68212	891556
	Bot	192.168.10.12-52.6.13.28-TCP (label only) 192.168.10.14-205.174.165.73-TCP 192.168.10.15-205.174.165.73-TCP 192.168.10.5-205.174.165.73-TCP 192.168.10.8-205.174.165.73-TCP 192.168.10.9-205.174.165.73-TCP 192.168.10.17-52.7.235.158-TCP (label only)	2208	12853
	None	5 uni- and bi- talkers	173	213
	BENIGN	17556 uni-talkers and 17552 bi-talkers	118421	7915226

TABLE 4. NETGENES: PER-DAY PER-LABEL TCP NETWORK-OBJECT STATISTICS.

*The malicious bi-talkers are specified in this column, considering the dataset authors' labels

*The "label only" tag means that the malicious talker was not provided in the CIC-IDS-2017's official website, but was present in the CIC-IDS-2017 authors' labels

CIC-IDS-2017 original dataset presents 158930 port scan flows, 158923 TCP flows (protocol field = 6), 1 UDP flow (protocol field = 17) and 6 unspecified-protocol flows (protocol field = 0). Table 4 shows the TCP network-object statistics based on the mapping between CICFlowMeter and NetGenes flows before correcting any label:

- Thursday's traffic was corrected to 71809 port scan flows and 95804 benign flows
- Friday's traffic was corrected to 2206 bot flows and 118423 benign flows

As for UDP, Friday's flows were also corrected, from 1 labeled port scan flow to 195 port scan flows that should have been labeled as port scan. We talk about all these flows in more detail further.

5.2.1. Benign Traffic Overview

The issue with port-protocol correlation is not detecting an adversary that is making use of a certain protocol's commonly used port to communicate using an unexpected protocol; if the traffic was decryptable, there would be no issue in this since we can validate the L5-7 traffic according to the used L5-7 protocol specification. However, since we assume encryption, we would not be able to validate this specification and would have to resort to flow fingerprinting to detect common L5-7 events in each supported protocol to provide us with some level of comfort about the fact that it is that protocol which is running. Even so, because we have considered Monday's benign traffic as ground truth and we are interested in showing what CIC-IDS-2017's benign traffic is constituted of, and not anything more, we assume port-protocol correlation in this subsection.

For providing a small overview of CIC-IDS-2017's traffic, we strictly overviewed Monday's traffic, which according to the dataset authors is comprised only of benign traffic and is labeled as such in the dataset. Monday's TCP benign traffic overview is presented in table 19 (annex), while Monday's UDP benign traffic is presented in table 20 (annex).

As can be seen in table 20, the most common UDP traffic was DNS traffic (UDP/53), as well as protocols such as Kerberos (UDP/88), NetBIOS (UDP/137), LDAP (UDP/389), NTP (UDP/123) and SSDP (UDP/1900). As for UDP/443 traffic, which produced the highest number of flows after DNS traffic, these communications were performed to external servers whose public IPs belong to Google, according to the WHOIS records at the time of writing.

On the other hand, as can be seen in table 21, the most common TCP traffic was HTTPS (TCP/443) and HTTP (TCP/80) traffic, as well as protocols such as SSH (TCP/22), LDAP (TCP/389 and TCP/3268), FTP (TCP/21), SMTP over SLL (TCP/465), Kerberos (TCP/88), SMB (TCP/445) and NetBIOS (TCP/139).

5.2.2. Bot ARES Traffic

We have successfully converted the CTU-13 original dataset to a NetGenes-based dataset, as we did with CIC-IDS-2017. However, time restraints did not allow analyzing the CTU-13 dataset and properly defining the “Bot” threat class. Since we were also initially targeting this threat class, we present a simple run-down of the CIC-IDS-2017’s Bot flows based on flow connection states, which logically divides the traffic of the 5 bi-talkers marked as “Bot” by CIC-IDS-2017’s official website (5 out of 7) in two time-ranges: 13:04-14:02 and 14:03-15:59. The following are the Bot uni-talkers found:

- **192.168.10.12-52.6.13.28-TCP Uni-Talker (UT-1) –** 12:34:14-12:35:14, 1 flow, three-way-handshake initiation & full-duplex connection established & graceful termination. Flow labeled as “Bot”, but it is not mentioned in the CIC-IDS-2017’s official website.
- **192.168.10.5-205.174.165.73-TCP Uni-Talker (UT-2) –** 13:29:01-14:01:44, 109 flows, three-way-handshake initiation & full-duplex connection established & (graceful termination | null termination); 14:03:24-15:59:35, 210 flows, two-way-handshake initiation & rejected connection & abort termination (forward “syn” + backward “rst-ack”).
- **192.168.10.8-205.174.165.73-TCP Uni-Talker (UT-3) –** 13:36:11-14:02:03, 117 flows, three-way-handshake initiation & full-duplex connection established & (graceful termination | null termination); 14:03:17-15:59:53, 420 flows, two-way-handshake initiation & rejected connection & abort termination (forward “syn” + backward “rst-ack”)
- **192.168.10.9-205.174.165.73-TCP Uni-Talker (UT-4) –** 13:04:14-14:01:44, 151 flows, three-way-handshake initiation & full-duplex connection established & (graceful termination | null termination); 14:03:24-15:59:36, 210 flows, two-way-handshake initiation & rejected connection & abort termination (forward “syn” + backward “rst-ack”)
- **192.168.10.14-205.174.165.73-TCP Uni-Talker (UT-5) –** 13:24:29-14:01:43, 139 flows, three-way-handshake initiation & full-duplex connection established & graceful termination; 14:03:23-15:59:34, 210 flows, two-way-handshake initiation & rejected connection & abort termination (forward “syn” + backward “rst-ack”)
- **192.168.10.15-205.174.165.73-TCP Uni-Talker (UT-6) –** 13:06:55-14:01:59, 220 flows, three-way-handshake initiation & full-duplex connection established & (graceful termination | null termination); 14:03:14-15:59:51, 420 flows, two-way-handshake initiation & rejected connection & abort termination (forward “syn” + backward “rst-ack”)
- **192.168.10.17-52.7.235.158-TCP Uni-Talker (UT-7) –** 14:20:40-14:21:41, 1 flow, three-way-handshake initiation & full-duplex connection established & graceful termination. Flow labeled as “Bot”, but it is not mentioned in the CIC-IDS-2017’s official website.
- All “Bot” flows were initiated to destination port **TCP/8080**, where the C2 server ran.

Timeline	Uni-Talker	Full-duplex Connection Flows	Rejected Connection Flows
12:34:14-12:35:14	UT-1	1	0
13:04:14-14:01:44	**UT-4**	**151**	**0**
13:06:55-14:01:59	**UT-6**	**220**	**0**
13:24:29-14:01:43	**UT-5**	**139**	**0**
13:29:01-14:01:44	**UT-2**	**109**	**0**
13:36:11-14:02:03	**UT-3**	**117**	**0**
14:03:14-15:59:51	UT-6	0	420
14:03:17-15:59:53	UT-3	0	420
14:03:23-15:59:34	UT-5	0	210
14:03:24-15:59:35	UT-2	0	210
14:03:24-15:59:36	UT-4	0	210
14:20:40-14:21:41	UT-7	1	0

TABLE 5. BOT ARES: UNI-TALKER TIMELINE ANALYSIS BASED ON FLOW STATES.

Table 5 shows the Uni-Talker timeline of Bot ARES based on the full-duplex connection and rejected connection states:

- The first (UT-1) and last uni-talker (UT-7) traffic, marked red, is labeled as belonging to Bot ARES traffic, but is never mentioned by the authors neither on CIC-IDS-2017 website nor on their support paper.
- In the first time-range (13:04-14:02), marked bold, the C2 server is accepting connections from its bot victims and providing the fetched commands. The bot was fetching commands from the C2 server in 10-second intervals (occasionally using 100-second intervals instead).
- In the second time-range (14:03-15:59), the C2 server was shut down and the system in which it was run now has closed the port; when the bot victims contact the system's closed port, it responds with rst2-ack2 packets, rejecting all incoming connections. This traffic is correctly labeled in the dataset, but not mentioned on the website or the paper. The presented data indicates that the bot software continued to run on the victim hosts, consistently attempting to connect to the C2 server at port TCP/8080 even after 14:02. When a bot got their first rejected connection, they started consistently attempting to reach the C2 server in 100-second intervals instead, which kept consistently rejecting the connections. Curiously, in this time-range, each internal host repeatedly initiated a 210-multiple number of rejected-connection flows to the C2 server (a single outside host), initiating and terminating flows at approximately the same timestamps between each other.

Bot ARES Discussion

CIC-IDS-2017's "Bot" traffic is very limited in comparison to CTU-13, but it allows understanding that there are noticeable patterns that may be exploited even though most of these patterns are based on the "Bot ARES" tool specificities and used configurations, rather than the "Bot" threat class itself.

It could be interesting to consider detecting fixed communication timings (considering flow inter-initiation and inter-termination timings), since these timings would give away the automated nature of the network traffic, but timings could eventually be randomized enough to make the bot traffic undistinguishable from human-generated traffic. Assuming that fixed timings are maintained and we just detect the traffic by timing, other benign automated tools would be detected that way as well. If we chose to detect bot traffic this way, all the rest of the benign automated network traffic would have to eventually be whitelisted for this type of analysis to be useful for an analyst. As such, this type of analysis based on timing would be perfect for a network that is continuously monitored by an analyst, who can whitelist the benign automated traffic of the network over time, in order to be able to distinguish and detect the malicious automated traffic as well based on this communication timing indicator (in this case, the network traffic generated by a bot). A more core indicator, however, would be the fact that a bot will try to consistently connect to its C2 server, independently of it being up or down (bot always needs to re-check if it is up again). This latter might be the most relevant takeaway from CIC-IDS-2017 Bot ARES traffic, as every other parameter results from specific configuration.

5.3. Port Scan

We start by remembering that a Port Scan's intent is to "probe multiple ports of a given host, for a given L4 protocol", leading us to create the TR-1 rule and filter talkers by their unique destination ports count using a fixed threshold. We use the previous rule to create the FR-TR-Default rule and filter flows for those talkers. Then, we create other flow rules to logically narrow down relevant flows within the talkers, given flow initiations and terminations, connection states, and more, that are common in port scans.

5.3.1. Used nmap parameters

The CIC-IDS-2017 authors mention that they have used the following nmap flags in the Friday's flows: "-sS, sT, -sF, -sX, -sN, -sP, -sV, -sU, -sO, -sA, -sW, -sR, -sL, -sI, -b". In the "Port Scan" threat class context, all these nmap flags used were previously mentioned/described, except for:

- List Scan (-sL) – this option simply prints a list of hosts in the specified network range, so no traffic is generated.
- TCP Window Scan (-sW) – this option does not generate new traffic, it just changes the interpretation of the scanned ports. Using this analysis method, the attacker may analyze the response packet to check whether a port is open: a positive window size indicates an open port and a zero-size window indicates a closed port; if the response packet time to live (ttl) is lower than the rest of the received RST packets the port is likely to be open. This applies to most systems, but there are other systems that may return the inverse.

- IP protocol scan (-sO) – traffic generated by this option belongs to the “L3 Service Discovery” threat class, since it consists of raw IPv4 packets (no L4 traffic) with no capabilities of probing a port (which is a L4 concept).
- Ping scan (-sn): this option tells nmap to not do a Port Scan after host discovery. The “-sP” flag is just an alias for the “-sn” flag, which we use in the “Host Discovery” threat class. By default, it sends an ICMP echo request, TCP SYN to port 443, TCP ACK to port 80, and an ICMP timestamp request.
- Service/Version Detection (-sR) – the “-sR” flag is just an alias for the previously described “-sV” flag (Service/Version Detection) since March 2011 (before, it was used for the “RPC Scan”, which is now implicitly included in this option).

5.3.2. Defining 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.

The factor that makes the features in HR-1 and TR-1 core features is the fact that they are directly extrapolated from the very definition of port scan, which very basically consists in communicating with various ports to unveil their status. Similarly, all the flow rules define the core scenarios of a port scan: filtered port scans, closed port scans and open port scans.

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.

5.3.3. Defining rule sets

For studying the “Port Scan” threat class, we use the following flow rule sets:

- **RS1 –** “TR-1 n=100, FR-TR-Default” – Flows whose bi-talkers have more than 100 unique destination ports. This rule is used in all further rule sets.
- **RS2 –** “TR-1 n=100, FR-TR-Default & FR-1” – Flows that feature an unanswered connection request as flow initiation. It captures probes against filtered ports, which result in dropped connections.
- **RS3 –** “TR-1 n=100, FR-TR-Default & FR-2” – Flows that feature a two-way-handshake as flow initiation. It captures probes against closed ports, which result in rejected connections, as well as probes against open ports, which result in half-duplex connections in most cases that do not require a full-duplex connection (e.g., Connect scan, Version scan, custom adversarial scan).
- **RS4 –** “TR-1 n=100, FR-TR-Default & FR-3” – Flows that are initiated using a three-way handshake and result in an established full-duplex connection that is later aborted by the source host (who initiated it), without the destination host ever sending another packet other than the

syn2-ack2 packet. It captures “Connect Scan” and similar probes against open ports that are not very well known, which causes just a TCP full-duplex connection and nothing more except abort-terminating the flow.

- **RS5 –** “TR-1 n=100, FR-TR-Default & FR-2.1” – Flows that feature a rejected connection. It captures probes against closed ports.
- **RS6 –** “TR-1 n=100, FR-TR-Default & FR-2.2” – Flows that feature a half-duplex connection. It captures probes against open ports (except previously mentioned cases).
- **RS7 –** “TR-1 n=100, FR-TR-Default & FR-2.2.1” – Flows that feature a half-duplex connection and, at the same time, are aborted by the same host that initiated the flow. It captures probes against open ports (except previously mentioned cases).
- **RS8 –** “TR-1 n=100, FR-TR-Default & (FR-1 | FR-2)” – Flows that feature an unanswered connection request, a rejected connection, or a half-duplex connection. It captures probes against closed ports, filtered ports and open ports (except previously mentioned cases that require a full-duplex connection).
- **RS9 –** “TR-1 n=100, FR-TR-Default & (FR-1 | FR-3)” – Flows that feature an unanswered connection request or a full-duplex connection. It captures probes against filtered ports and open ports.
- **RS10 –** “TR-1 n=100, FR-TR-Default & (FR-2 | FR-3)” – Flows that feature a rejected connection, a half-duplex connection, or a full-duplex connection. It captures probes against closed ports and open ports.
- **RS11 –** “TR-1 n=100, FR-TR-Default & (FR-1 | FR-2 | FR-3)” – Flows that feature an unanswered connection request, a rejected connection, a half-duplex connection or a specific full-duplex connection. It captures probes against closed ports, filtered ports and open ports (including a specific full-duplex connection case).

PS Rule Set	PS Flow Rules	Port State		
		Closed Port	Filtered Port	Open Port
RS1	None	N/A	N/A	N/A
RS2	FR-1	NO	YES	NO
RS3	FR-2	YES	NO	YES
RS4	FR-3	NO	NO	YES
RS5	FR-2.1	YES	NO	NO
RS6	FR-2.2	NO	NO	YES
RS7	FR-2.2.1	NO	NO	YES
RS8	FR-1 | FR-2	YES	YES	YES
RS9	FR-1 | FR-3	NO	YES	YES
RS10	FR-2 | FR-3	YES	NO	YES
RS11	FR-1 | FR-2 | FR-3	YES	YES	YES

TABLE 6. PORT SCAN RULE SET SUMMARY.

5.3.4. File investigation

Using the previously defined rule sets, we investigated each day's file:

- Monday:
  - UDP "Benign" traffic: we determined that a benign NetBIOS name service had been responding over udp/137, from time "12:01:14" to time "19:58:53", from source host "192.168.10.12" to destination host "192.168.10.25", from source port "137" to 161 different destination ports in the range "49173-49295", as well as a more consistent flow of data to destination port "137". See the "NetBIOS note" below the bullet points for further details.
  - No malicious activity was detected.
- Tuesday:
  - UDP "Benign" traffic: we determined that a benign NetBIOS name service had been responding over udp/137, from time "12:02:18" to time "19:59:19", from source host "192.168.10.50" to destination host "192.168.10.25", from source port "137" to 188 different destination ports in the range "49184-49605", as well as a more consistent flow of data to destination port "137". See the "NetBIOS note" below the bullet points for further details.
  - No malicious activity was detected.
- Wednesday:
  - UDP "Benign" traffic: we determined that a benign NetBIOS name service had been responding over udp/137, from time "11:46:29" to time "20:01:29", from source host "192.168.10.50" to destination host "192.168.10.25", from source port "137" to 167 different destination ports in the range "49184-49353", as well as a more consistent flow of data to destination port "137". See the "NetBIOS note" below the bullet points for further details.
  - No malicious activity was detected.
- Thursday:
  - UDP "Benign" traffic: we determined that a benign NetBIOS name service had been responding over udp/137, from time "12:23:16" to time "20:01:55", from source host "192.168.10.19" to destination host "192.168.10.25", from source port "137" to 128 different destination ports in the range "49194-49401", as well as a more consistent flow of data to destination port "137". See the "NetBIOS note" below the bullet points for further details.
  - TCP "Port Scan" mislabeled traffic: we determined that a TCP Port Scan occurred, from time "17:00:32" to "17:00:46", from host "172.16.0.1" to host "192.168.10.51", from source port "50122" and "50133" to 997 different destination ports in the range "1-65389". This occurrence was never mentioned by the authors. Regarding the rest of the traffic in the "172.16.0.1-196.168.10.51-TCP" talker, 5 flows marked "Benign" were created to the HTTP port (tcp/80) from "16:14:21" to "16:14:22", which are in fact "Benign" flows. Hence, in this 1 talker, we caught 998 mislabeled NetGenes-generated flows which should have been marked as "Port Scan" instead of "Benign". By using the TR-1 rule, we have successfully detected an unmentioned TCP port scan. We manually corrected these 998 flows as "Port Scan" afterwards.
  - TCP "Port Scan" mislabeled traffic: in the official CIC-IDS-2017 website, the authors state that the third "Infiltration" attack, dubbed "Infiltration – Dropbox Download", divided in two steps,

occurs in Thursday's "18:04-18:45" time range. The first step was correctly marked as "Infiltration", but the second step, which was supposed to be a "Port Scan" using "nmap", was marked as "Benign". We determined that there were 11 TCP port scans launched against 11 hosts, from time "18:05:14" to time "18:44:35", from host "192.168.10.8" to 11 different internal hosts ("192.168.10.5", "192.168.10.9", "192.168.10.12", "192.168.10.14", "192.168.10.15", "192.168.10.16", "192.168.10.17", "192.168.10.19", "192.168.10.25", "192.168.10.50", "192.168.10.51"), from 451 different source ports in the range "33264-65243" to 1038 different destination ports in the range "1-65389". The 11 bi-talkers encompass 73150 flows, of which 2265 flows are not in the "18:05:14-18:44:35" time range, of which 74 flows that target the destination port 5060 (Session Initiation Protocol, SIP) are seemingly "Benign". Hence, in these 73150 flows, there were several mislabeled NetGenes-generated flows which should have been marked as "Port Scan" instead of "Benign", which according to our previously mentioned manual labelling might be 70811 (73150 – 2265 – 74) "Port Scan" flows. In this manual labelling, on the filtered bi-talkers, we stumbled upon 813 flows with the source port range 1266-3215, which we were not sure about, so we still classified these as "Port Scan" as our base criteria only directed us to bi-talkers with the source host "192.168.10.8", within the timings presented in CIC-IDS-2017's official website. Since we are dealing with a very significant number of flows, it is quite difficult to manually label them in the most accurate way. We would either need the authors' labels corrected or more specific details on this traffic to be able to be perfectly accurate. The TR-1 rule helped us detecting the network pivoting step (11 TCP port scans) of an on-going infiltration, along with the TCP port scan we described in the previous bullet-point. As a final step, we manually labeled the incorrect 70811 flows as "Port Scan" afterwards.

- Friday:
  - UDP "Port Scan" mislabeled traffic: In the official CIC-IDS-2017 website, the authors state that the flag "sU" was used in Friday's "18:11-18:12" time range, even though only 1 UDP flow that targeted the destination port tcp/123 was marked as "Port Scan" (CIC flow id "172.16.0.1-192.168.10.50-38260-123-17") in this file. We determined that a UDP port scan ("nmap" with "sU" flag) did occur, from time "18:11:11" to time "18:12:32", from host "172.16.0.1" to host "192.168.10.50", from source port ranges "38260-38268" and "38271-38276" (15 source ports) to 82 different destination ports in the range "42-65024". Regarding the rest of the traffic in the "172.16.0.1-192.168.10.50-UDP" talker, 39 flows marked "None" targeted udp/21 and udp/22 from "17:17:10" to "17:19:09", while the remaining 34 flows marked "None" targeted udp/40125 from "18:13:14" to "18:21:28". Hence, in this 1 talker, there were 195 mislabeled NetGenes-generated flows which should have been marked as "Port Scan" rather than "None" (meaning a lack of existence in the original dataset) to join the 1 flow correctly classified as a "Port Scan". The TR-1 rule, once again, proved useful to finding labeling errors in the dataset. Weirdly, the 1 UDP "Port Scan" flow was the only maliciously tagged traffic in the whole CIC-IDS-2017 dataset.
  - UDP "Benign" traffic: we determined that a benign NetBIOS name service had been responding over udp/137, from time "12:01:23" to time "20:01:22", from source host "192.168.10.50" to

destination host “192.168.10.25”, from source port “137” to 161 different destination ports in the range “49184-49401”, as well as a more consistent flow of data to destination port “137”. See the “NetBIOS note” below the bullet points for further details.

- TCP “Port Scan” traffic: we determined that there was 1 TCP port scan launched, from time “16:05:34” to time “18:23:53”, from host “172.16.0.1” to host “192.168.10.50”, from 14140 different source ports in the range “32768-64915” to 998 different destination ports in the range “1-65389”. Finally, since this “Port Scan” is the only one that is correctly labeled by CIC, we tested our results with the original CIC-IDS-2017 labeled flows.

“Benign” NetBIOS traffic: Even though there was more NetBIOS traffic, the above-mentioned NetBIOS flows were the ones that generated, by far, the most traffic to different multiple destination ports within the same bi-talkers. For an average of 8 hours per day, a new UDP flow was created within an exact 3-minute time span. NetBIOS traffic can be distinguished from a typical UDP port scan due to its UDP communication coming from NetBIOS reserved port “137” to dynamic destination ports over a significantly high time span, while maintaining a communication flow between source and destination ports udp/137. Since port scans may also be spanned over greater periods of times and be made to simulate the above port conditions, the rule is that any non-dynamic destination port different than udp/137 that is scanned from source port udp/137 of the same host should be flagged. This represents automated benign behavior that should either be treated in a higher layer than OSI layer 4 or be treated as an exception, given a real-world scenario where we may know where NetBIOS instances are running. This traffic can be seen marked yellow in table 22.

Mislabeled “Port Scan” traffic: None of the previously mentioned mislabeled “Port Scan” traffic was detected or referenced by any related work that studied the CIC-IDS-2017 dataset.

5.3.5. Applying the rule sets

Before we apply our rule sets, we note that the threshold could be put lower in a real-world scenario, where we would know which exception cases needed to be handled. It is highly unusual that a source host accesses a lot of network services on a destination host (bi-talker level). Even a single destination host running more than 5 network services is not usual (bi-host level). In CIC-IDS-2017 files, we have traffic classified as “Benign” that is a product of CIC’s ML-based B-Profile tool, which is meant to generate “Benign” traffic. Unfortunately, by not considering source and destination hosts, although this tool seems to create a highly accurate definition of a “Benign” bi-flow, it does not create the most accurate “Benign” definition of bi-talkers and bi-hosts. In this file, among the 17554 bi-talkers, 4 bi-talkers have [5-10] ports, 7 bi-talkers have [11-50] ports and 4 bi-talkers have [51-100] ports, with all their bi-flows marked as “Benign”. We still want to be able to detect these types of cases, so we believe that a fair threshold would be a maximum of 5 ports. With such a low threshold, a positive hit would not directly be considered as a “Port Scan” at the talker level, but rather a suspicion of a “Port Scan” that needs to be validated by an analyst and can lead to finding automated benign exceptions. Furthermore, the flow rules we implemented may also be applied to help the analyst distinguish benign bi-talkers from port scan bi-talkers. As for the B-Profile tool’s case, the fact that it constantly communicates with the same host for multiple network services should be handled as an exception put by an analyst who knows (or, *a posteriori*, gets to know) such a tool is performing queries from the same IP to multiple

services on a single host, as well as knowing the hosts which are running more than 5 network services and that, consequently, are prone to receive benign traffic in more than 5 ports.

As a first step, we filter bi-talkers for the number of unique destination ports (TR-1 rule) using a threshold of 100. This means that talkers with bitalker_fwd_biflow_n_unique_dst_ports>100 or bitalker_bwd_biflow_n_unique_dst_ports>100 will match. Among the 17554 Friday talkers, there is only 1 talker that matches this rule: “172.16.0.1-192.168.10.50-TCP”, active from “14:47:47” to “19:16:12”, with bitalker_fwd_biflow_n_unique_dst_ports=999. On Friday, this 1 talker is the only one that is marked as “Port Scan”, according to CIC-IDS-2017 labels and their official website. At this point, we have already determined that the host “172.16.0.1” was likely responsible for launching a Port Scan against the host “192.168.10.50”, filtering out 17553 (99.994%) of Friday’s talkers.

If we use a threshold of 5 instead in the first step, we match 17 talkers. Here, in the worst-case scenario, the analyst does not know anything at all about the network and does not consider that there are multiple automated “Benign” network scripts running and accessing multiple network services on specific hosts. In this scenario, 17537 talkers (99.903%) are still filtered out.

As a second step, we use the “bitalker_id” talker data column as filter for the flows (FR-TR-Default rule), by filtering the “unitalker_id” flow data column for the bi-talker “172.16.0.1-192.168.10.50-TCP” or, practically, filtering for both uni-talkers “172.16.0.1-192.168.10.50-TCP” and “192.168.10.50-172.16.0.1-TCP” (in this case, the only uni-talker that exists is the first one, since this bi-talker doesn’t encompass any backward flow). Out of the 347994 flows, the bi-talker filter leads to 255794 filtered flows, which is an unusually high number of flows for a single bi-talker.

Third, the “biflow_eth_ipv4_tcp_initiation_two_way_handshake” flow feature (FR-2 rule) is used to filter flows with a two-way handshake initiation (FR-2 rule), filtering flows whose connection is rejected by the server (port is closed, hence the connection is rejected with a RST packet), as well as flows that establish a half-duplex connection (port is open, server accepts the connection but client closes it before finalizing the three-way handshake). Additionally, in Port Scan scenarios, this half-duplex connection is usually aborted by the source host. The filter returns 158485 flows, of which 39 flows are marked “None”, 32 flows are marked “Benign”, and 158414 flows are marked “Port Scan”.

With the previous filters (“TR-1 n=100, FR-TR-Default & FR-2” rules), we have successfully detected 158414 (99.644%) flows out of the 158980 “Port Scan” flows. The reason why we are not detecting the remaining 566 (0.357%) “Port Scan” flows is that the scans that require a three-way-handshake initiation and a full-duplex connection will not be caught if the previous flow filter is applied, such as, for example, the “Connect Scan” and the “Service/Version Detection Scan”, which account for 565 “Port Scan” flows, together with only 1 dropped “Port Scan” flow.

Furthermore, if we further filter the 158485 flows, we can see that 158000 were rejected connections, while only 485 were established half-duplex connections:

- Regarding half-duplex connections, the half-duplex connection filter returned 484 flows (due to a bug regarding corrupted packet timestamps messing with correct packet order, we suspect). This sole missing flow shares the same TCP flag features of the captured half-duplex connection flows, so we suspect it is a half-duplex connection like the others. The 1 sole flow we mentioned is marked “Benign”, while the initially filtered 484 flows are marked “Port Scan”.

- Regarding rejected connections, 39 flows are marked “None”, 31 are marked “Benign” and 157930 are marked “Port Scan”.

Above, we detailed the “TR-1 with n=100” rule, joined with the “FR-TR-Default” and “FR-2” rule variations, for detecting Friday’s correctly labeled “Port Scan” flows. We summarize the results we obtained for all rule sets which we automatically applied to the dataset, in the subsections below. We analyze Thursday traffic as well, using Thursday’s mislabeled “Port Scan” flows, which we manually corrected. After manually having manually labeled Thursday’s mislabeled port scan flows based on the file investigation we previously carried out, we can analyze our flow rule sets in the Thursday’s file as well.

5.3.5.1. “Port Scan” Rule sets applied to Thursday

	#Flows			
	TP	**TN**	**FP**	**FN**
RS1	71809	93748	2346	0
RS2	30174	94060	2034	41635
RS3	40987	96014	80	30822
RS4	0	96094	0	71809
RS5	40593	96041	53	31216
RS6	393	96067	27	71416
RS7	1	96094	0	71808
RS8	71161	93980	2114	648
RS9	30174	94060	2034	41635
RS10	40987	96014	80	30822
RS11	71161	93980	2114	648

TABLE 7. THURSDAY: “TR-1 N=100” FLOW RULE SET RESULTS.

	Flow Classification Metrics (%)							
	TPR	**TNR**	**FPR**	**FNR**	**Overall Accuracy**	**Precision**	**F1-Score**	**MCC**
RS1	100.000	97.559	2.441	0.000	98.603	96.836	98.393	97.197
RS2	42.020	97.883	2.117	57.980	73.992	93.685	58.017	50.140
RS3	57.078	99.917	0.083	42.922	81.595	99.805	72.623	65.600
RS4	N/A	100.000	N/A	100.000	57.232	N/A	N/A	N/A
RS5	56.529	99.945	0.055	43.471	81.337	99.870	72.194	65.228
RS6	0.547	99.972	0.028	99.453	57.450	93.571	1.088	5.142
RS7	0.001	100.000	0.000	99.999	57.232	100.000	0.003	0.282
RS8	99.098	97.800	2.200	0.902	98.355	97.115	98.096	96.664
RS9	42.020	97.883	2.117	57.980	73.992	93.685	58.017	50.140
RS10	57.078	99.917	0.083	42.922	81.595	99.805	72.623	65.600
RS11	99.098	97.800	2.200	0.902	98.355	97.115	98.096	96.664

TABLE 8. THURSDAY: “TR-1 N=100” FLOW RULE SET METRICS.

We only compare “TR-1 n=100” to avoid comparing both talker rules, which present very similar results. More information on the “TR-1 n=100” filtered bi-talkers can be found in table 21 (annex).

Considering only the “TR-1 n=100” talker rule, we compare every flow rule set using three metrics. The results using “TR-1 n=100, FR-TR-Default”, which considers all flows of the filtered bi-talkers, achieves a high precision (96.836%) and the highest MCC (97.197%) in this case because most flows in the filtered bi-talkers are “Port Scan” flows. Amongst the “Port Scan” flow rule sets:

- The RS7 rule set (“FR-2.2.1”) achieved a perfect precision. In the context of Thursday’s flows, the only flow that had a half-duplex connection whose source host aborted it afterwards (“FR-2.2.1”), was a Port Scan flow.
- The RS11 (“FR-1 | FR-2 | FR-3”) and RS8 (“FR-1 | FR-2”) rule sets generically performed the best for Thursday’s flows with a shared MCC of 96.664%, with a precision of 97.115% and F1-Score of 98.096%, and with a sensitivity of 99.098%. This means that Thursday’s Port Scan flows are mostly comprised of dropped (“FR-1”) and rejected (“FR-2”) packets.

These mislabeled Thursday port scans were never found or evaluated by the considered related work. At the time of writing, we did not find any other paper that referenced this traffic as having been mislabeled, even though the authors talk about this traffic in their paper and in the dataset’s website.

5.3.5.2. “Port Scan” Rule sets applied to Friday

	#Flows			
	TP	**TN**	**FP**	**FN**
RS1	158980	92200	96814	0
RS2	1	188089	925	158979
RS3	158414	188943	71	566
RS4	474	189013	1	158506
RS5	157930	188944	70	1050
RS6	484	189014	0	158496
RS7	484	189014	0	158496
RS8	158415	188018	996	565
RS9	475	188088	926	158505
RS10	158888	188942	72	92
RS11	158889	188017	997	91

TABLE 9. FRIDAY: “TR-1 N=100” FLOW RULE SET RESULTS.

	Flow Classification Metrics (%)							
	TPR	**TNR**	**FPR**	**FNR**	**Overall Accuracy**	**Precision**	**F1-Score**	**MCC**
RS1	100.000	48.779	51.221	0.000	72.179	62.152	76.659	55.061
RS2	0.001	99.511	0.489	99.999	54.050	0.108	0.001	-4.726
RS3	99.644	99.962	0.038	0.356	99.817	99.955	99.799	99.631
RS4	0.298	99.999	0.001	99.702	54.451	99.789	0.595	4.016
RS5	99.340	99.963	0.037	0.660	99.678	99.956	99.647	99.353
RS6	0.304	100.000	0.000	99.696	54.454	100.000	0.607	4.069
RS7	0.304	100.000	0.000	99.696	54.454	100.000	0.607	4.069
RS8	99.645	99.473	0.527	0.355	99.551	99.375	99.510	99.097
RS9	0.299	99.510	0.490	99.701	54.186	33.904	0.592	-1.504
RS10	99.942	99.962	0.038	0.058	99.953	99.955	99.948	99.905
RS11	99.943	99.473	0.527	0.057	99.687	99.376	99.659	99.372

TABLE 10. FRIDAY: “TR-1 N=100” FLOW RULE SET METRICS.

We only compare “TR-1 n=100” to avoid comparing both talker rules, which present very similar results. More information on the “TR-1 n=100” filtered bi-talkers can be found in table 21 (annex).

Considering only the “TR-1 n=100” talker rule, we compare every flow rule set using three metrics. The results using “TR-1 n=100, FR-TR-Default”, which considers all flows of the filtered bi-

talkers, is not enough to achieve a high precision in this case because there are a lot of other flows in the same bi-talker that are not “Port Scan” flows, specifically flows belonging to the DDoS attack generated with the LOIC tool (the tool’s acronym is mistaken twice, one in the CIC-IDS-2017’s official website and one in their paper).

Additionally, note that the RS10 rule set (“FR-2 | FR-3”) generically performed the best for Friday’s flows. This means that most Friday flows are either rejected connections or full-duplex connections that were established and immediately terminated. We can also see that “FR-2” is the dominant flow rule, as its individual rule set (RS3) correctly classified many more flows than the “FR-3” rule set (RS4). Despite that fact, in the real world, flows are not always rejected when a Port Scan occurs. In this case, the authors turned off the firewall rules which would have normally caused more dropped packets.

Friday’s flows are the only “Port Scan” flows that were correctly labeled and are also the only ones that are evaluated by other works. Since only Friday flows are considered for training and testing data, there is a chance that they might miss other types of Port Scan flows. For example, they might miss flows that are dropped since only 1 dropped Port Scan flow was presented in Friday’s flows.

5.3.5.3. Common (Thursday + Friday) rule set evaluation

	#Flows			
	TP	**TN**	**FP**	**FN**
RS1	230789	185948	99160	0
RS2	30175	282149	2959	200614
RS3	199401	284957	151	31388
RS4	474	285107	1	230315
RS5	198523	284985	123	32266
RS6	877	285111	27	229912
RS7	485	285108	0	230304
RS8	229576	281998	3110	1213
RS9	30649	282148	2960	200140
RS10	199875	284956	152	30914
RS11	230050	281997	3111	739

TABLE 11. THURSDAY & FRIDAY: “TR-1 N=100” FLOW RULE SET RESULTS.

	Flow Classification Metrics (%)							
	TPR	**TNR**	**FPR**	**FNR**	**Overall Accuracy**	**Precision**	**F1-Score**	**MCC**
RS1	100.000	65.220	34.780	0.000	80.779	69.947	82.316	67.542
RS2	13.075	98.962	1.038	86.925	60.540	91.070	22.867	24.413
RS3	86.400	99.947	0.053	13.600	93.887	99.924	92.671	88.156
RS4	0.205	100.000	0.000	99.795	55.356	99.789	0.410	3.361
RS5	86.019	99.957	0.043	13.981	93.722	99.938	92.458	87.851
RS6	0.380	99.991	0.009	99.620	55.432	97.013	0.757	4.405
RS7	0.210	100.000	0.000	99.790	55.359	100.000	0.419	3.410
RS8	99.474	98.909	1.091	0.526	99.162	98.663	99.067	98.309
RS9	13.280	98.962	1.038	86.720	60.632	91.193	23.184	24.665
RS10	86.605	99.947	0.053	13.395	93.978	99.924	92.789	88.326
RS11	99.680	98.909	1.091	0.320	99.254	98.666	99.170	98.496

TABLE 12. THURSDAY & FRIDAY: “TR-1 N=100” FLOW RULE SET METRICS.

According to CIC-IDS-2017's labels, only Friday featured a port scan, however, according to CIC-IDS-2017's official website and our rule set, both days featured a port scan. Nevertheless, the above metrics join the only two days where a port scan was detected by our rule sets (the other three days were correctly discarded by the rule sets). We recall the simplified interpretations of the above listed metrics:

- High Precision: in the studied context, most flows that are filtered by the rule set are Port Scan flows.
- High F1-Score/MCC: in the studied context, most flows that are filtered by the rule set are Port Scan flows and there are not much more Port Scan flows left to detect.

We summarize each "Port Scan" flow rule set (considering "FR-TR-Default" applied with "TR-1 n=100"):

- The RS1 rule set (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.
- The RS2 rule set ("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.
- The RS3 rule set ("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.
- The RS4 rule set ("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.
- The RS5 rule set ("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.
- The RS6 rule set ("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.
- The RS7 rule set ("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.
- The RS8 rule set ("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.
- The RS9 rule set ("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.

- The RS10 rule set (“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.
- The RS11 rule set (“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: Thursday, TR-1 n=5, FR-TR-Default & (FR-1 | FR-2 | FR-3)					
Results	**Flows**	**Metric**	**Value (%)**	**Metric**	**Value (%)**
TP	71161	**Sensitivity / TPR**	99.098	**Overall Accuracy**	98.354
TN	93979	**Specificity / TNR**	97.799	**Precision**	97.114
FP	2115	**Fallout / FPR**	2.201	**F1-Score**	98.096
FN	648	**Miss Rate / FNR**	0.902	**MCC**	96.662
Port Scan: Friday, TR-1 n=5, FR-TR-Default & (FR-2 | FR-3)					
Results	**Flows**	**Metric**	**Value (%)**	**Metric**	**Value (%)**
TP	158888	**Sensitivity / TPR**	99.942	**Overall Accuracy**	99.952
TN	188940	**Specificity / TNR**	99.961	**Precision**	99.953
FP	74	**Fallout / FPR**	0.039	**F1-Score**	99.948
FN	92	**Miss Rate / FNR**	0.058	**MCC**	99.904

TABLE 13. THURSDAY & FRIDAY: “TR-1 N=5” FLOW RULE SET METRICS FOR THE MOST GENERICALLY PERFORMANT “PORT SCAN” FLOW RULE SET ON EACH DAY.

Table 13 shows the “TR-1 n=5” talker rule being test tested with the most generically performant flow rule sets for each day: the “FR-TR-Default & (FR-2 | FR-3)” rule set for Friday and the “FR-TR-Default & (FR-1 | FR-2 | FR-3)” rule set for Thursday. The “TR-1 n=5” talker rule can be overviewed using table 21 (annex) results to see how different thresholds affect the number of filtered talkers and filtered flows, while a more in-depth analysis is only possible through directly testing with the dataset.

The results between bi-talker filters “TR-1 n=100” and “TR-1 n=5” were not very different, but we recall that it is important that an analyst should be aware of any “Benign” automated behavior in a network that they may need to analyze or monitor, in order to create relevant exceptions and obtain the best results out of the automated analysis. Given these exceptions, a threshold of 5 would still be able to match only the relevant bi-talkers. In conclusion, in a real-world scenario where an analyst is able to whitelist benign automated traffic, we could lower the threshold down and incur in no loss at all.

5.3.6. Rule set discussion

If we use an hourly time window for defining bi-talkers (and bi-hosts, if we use HR-1) rather than the daily time window used in this work, we can achieve better results because most traffic that correlates to each other is often very close in time, including in this dataset. For example, Friday’s “Port Scan” flows span over 41 minutes and, since the first few seconds, it is already detectable by the unique destination port count rule (TR-1 rule).

In a real-world setup, a smaller time window can be used to greatly improve the classification overall metrics and precision, but it is very important that the smaller window does not become a “blindfold”: if not combined with lengthier time windows, such as the daily time window or an even

lengthier one, a smaller window may miss network attacks spanned over greater time ranges. CIC-IDS-2017 does not include any port scan that may be considered slow, but even if it did, as long as it scanned more than TR-1's threshold ports per day, we would be able to detect it. If we want to be completely safe about detecting all port scan instances, a lengthier time window may be implemented, as well as working towards minimizing the TR-1's threshold in each time window by correctly white-listing all the "Benign" exceptions detected overtime.

The metrics that we calculated tell us how our rule sets correlate with the classification problem at hand, given the studied datasets. The best part about using rule sets is that we know exactly what we are looking for and detecting, so we have complete control over what we want to detect. The hardest part about rule sets is that we need to effectively find the threat class's core features and explicitly implement them; else, the reliable way to create performant classification systems would still be ML classifiers, even though they would be more prone to be bypassed by an adversary capable of customizing their flows. Examples of adversarial evasion to our rule sets are considered in the next subsection (5.3.7).

Training ML models with network attack data that is inherently limited leads to limiting the ML models to that same inherently limited data. A ML model will train and test itself with data that is not broad enough to successfully measure its performance, by not considering unforeseen edge cases. This limitation issue does not come from a misconfiguration of the ML classification system, but from the train data which is not enough to emulate the data that an adversary would be able to try out to confuse the classifier. In this scenario, unless the ML model only attributes feature importance to threat class core features (which is very unlikely given that many statistical features are strongly correlated to these core features and NetGenes includes plenty statistical features), train data may not be enough to help the classifier correctly unmix these features. As such, in this same scenario, the only way to make the ML model focus on the core features would be doing manual feature selection, so we would still have to go through our threat class definitions to find their core features and explicitly implement core features from scratch. This results in not needing to deduce unimplemented core features anymore, as well as creating detection systems that can detect network attacks without having random vulnerabilities (against knowledgeable attackers, or simply against tools that do not respect the same non-core rules that the classifier incorrectly employs). However, the truth is that ML-based approaches can lead to great results detecting the class instances in the test datasets, while at the same time not yielding a true relevance of the used features and algorithms to really detect the network attack by its root causes, mainly due lack of data and a lot of non-core features. This means the measured classification results can be high for the considered data but will fail to flag different traffic that represents the same fundamental threat class while maintaining the same attack effectiveness. If the previous issue was not enough, there is also a need to make sure that class imbalance is not an issue by resampling (for example, reducing class samples to the least common denominator, as it was done in the previous work). The lack of data broadness issue and subsequent limitations will be there as long as there are endless traffic possibilities that do not affect attack effectiveness using non-core features.

In summary, in a non-adversarial scenario, the train/test data will likely not be broad enough to detect every variation of a network attack, and, inclusively, may lead to a misclassification of similar

non-class traffic based on the non-core features it uses. In an adversarial scenario, this means that the utilized weaker features can eventually be tampered with by an adversary to make network attacks invisible to the classification system while still maintaining their effectiveness.

A practical example of the previous issue is a ML model that trains with TCP flag counts from multiple datasets and will classify every flow that has certain combinations of TCP flag count ranges as malicious. While these combinations may be true for the contemplated training and test cases, an adversary can fool this classifier by, for example, sending more packets with the accounted TCP flags activated, or messing around with other random features if the training data (and, consequently, the classifier as well) had a strong bias towards these. With flow initiation, flow connection flow end states being correctly implemented in the data extraction phase, this is no longer an issue. Another practical example is the TCP SYN Flood Attack, which considering only flow features can be incorrectly detected as a Port Scan for its packet-set (flow) similarities, as we saw during the test phase of our previous work. However, by simply analyzing flow-set (talker/host) based data, which not many related works did, we can see that the considered flows consistently contact a specific port, which may or may not have an active service in this case.

The solution to solve the previous issues is undertaking manual feature selection before employing any type of Machine Learning, as the real issue can be narrowed down to the automated feature selection that does not know any better other than the data that it is presented with and will choose features solely based on this data, so the issue is not the Machine Learning usage *per se*, but the assumption that the utilized data is complete enough to make the ML algorithm select the correct core features only among all the considered features.

Moreover, without the correct extracted data features, as well as broad train and testing data, there could be many unforeseen blind spots to a classifier. Defining broad testing data is difficult for network attacks due to so much customization (switching tools, customizing parameters or customizing the sent network traffic in any given way) that is possible and, on the other hand, it is assumed that an adversary will try every possible way to bypass the detection mechanisms. Since we do not want to be blindly playing mouse and cat with our detection mechanisms, we avoid using classifiers which we do not have our complete control over.

Also, it is of utmost importance that we think from scratch about possible adversarial moves against our detection mechanisms, independently of the detection mechanisms being rule sets or more complex ML algorithms. A critical difference between the last two is that rule sets are easier to understand and debug, and they also allow us to have a more fine-grained control over what exactly we want to detect. Although ML can help us detect useful features which hold a significant logical value for the classification problem, it will not do so unless we have implemented this feature at least partially: for example, if we had not implemented flow states, a well-trained performant ML model could indicate us that the syn and ack counts are relevant features, which may be seen as a hint that flow states are relevant. If we had not implemented the TCP flow flag counts, there would be no way that we could deduce that flow states were useful for detection by means of Machine Learning. However, in our case, the TCP flow state features were implemented because we thought that flag counts were not enough to tell what was happening for the TCP connection itself, and we wanted to be able to filter flows as if

we were working with blocking packets and connections in iptables. At the same time, by better studying how a port scan worked, we were able to logically match the important port scan scenarios to their associated core features.

At the flow level, flow initiation types, connection states and termination types needed to be extracted because using TCP flag counts only is not enough to properly query the data for these types of features. However, we must note that even though the classifier may not have knowledge about the order in which the flags were used, as well as all their possible combinations between one another, it is very possible that a ML model can find a correlation between TCP flag counts and each type of connection state, which ultimately leads to correctly classifying a “Port Scan” flow correctly in most cases where an adversary does not properly customize their traffic. With these features explicitly implemented, a ML model could now use these features instead but, as we already mentioned, the used training/testing data would have to be broad enough to show us a clear difference in terms of tangible results.

Finally, one important thing that needs to be pointed out is that the false-positive ratio in flow classifications is not as important anymore when using flow-set based rules. This is because the number of alerts regarding malicious talkers and hosts is going to become very limited now, whereas if we only apply flow classification, we may have a low false-positive ratio and still capture non-malicious talkers and hosts, creating the possibility of false alerts. Flows need to be analyzed in sets, not only individually.

5.3.7. Adversarial evasion

“Attackers could span the port scan over lengthy periods of time.” – solved by increasing the bi-talker time window. These lengthier bi-talkers should complement smaller time window bi-talkers, rather than replace them, as we have discussed before.

“Attackers could start performing full-duplex connections for every open port.” – This situation messes with the FR-2.2 and FR-2.2.1 rules, which only detect half-duplex connections. We will still detect every closed port (FR-2.1 - rejected connection) and filtered port (FR-1 – dropped connection) attempts, which make up most of the flows. The FR-2.2 and FR-2.2.1 rules are very precise rules, but they depend on the source host to be fulfilled because only the source host has the power to decide if he wants to close the connection in this state, or if he wants to acknowledge the connection, thus creating a full-duplex connection, and only then terminating it.

“The attacker could use multiple of their owned IPs to perform the port scan” – This situation requires using host’s backward flows’ unique destination port counts, while maintaining the same flow rule set. By answering the question “How many services did host A have accessed by other hosts?”, the “bihost_bwd_biflow_n_unique_dst_ports” host feature would be the most relevant feature to detect a distributed port scan. Since this type of port scan requires focusing on a single host’s destination ports, independently of the source host, it can only be reliably detected using host-based rules. Based on the CIC-IDS-2017 authors’ labels, the official website and our dataset analysis, a distributed port scan attack does not seem to have happened.

“The attacker could perform a port scan to a limited number of ports only, enough to evade detection.” – this case would not be detected, but the effectiveness of the Port Scan is very limited. The adversary may only scan *n-1* ports before the port scan is detected. To detect this, TR-1’s threshold would need

to be lowered. The lower the threshold, the lower the effectiveness of the Port Scan will be without it being flagged. However, it is expected that more false-positive results arise, especially false-positive talker results. For example, if false-positive flow results increase by 100, this is not a big deal given the flows' order of greatness (FPR will slightly increase here), but if the talker rule set TR-1 is a weak rule set, false-positive talker results will also increase by a similar value, which means the FPR will greatly increase and, consequently, that would mean the analyst would have a lot of hosts to validate.

"The attacker customizes multiple port scan packets in every possible way they can (e.g., add garbage data, use multiple flag combinations, slow-down the packet pace, all of the above, etc.)" – we attribute zero relevance to features other than the core features considered in the various port scan rule sets, and our classification will either remain the same in every rule set that is not slightly controllable by the adversary, or simply not output anything in the rule sets that are. Among the rule sets we defined, the high-precision flow rules "FR-3", "FR-2.2.1" and "FR-2.2" can be fooled by a knowledgeable adversary: in order to bypass FR-3, the adversary needs to receive at least 1 packet after the full-duplex connection is established, which is achievable if he sends more packets (or by performing a graceful termination, which would require at least a second packet from the destination host); similarly, to bypass FR-2.2 and FR-2.2.1, the adversary may choose to perform full-duplex connections whenever a port is open, rather than the usual half-duplex connection, and send more packets each time to bypass the FR-3 rule as well again. Finally, bypassing a TR-1 with a low threshold, as well as the FR-1, FR-2 and FR-2.1 rules, is much more difficult for a scan that intends to find open ports on the network, unless the adversary can successfully guess the ports that are open (which obviously defeats the purpose of the port scan) all the times. This is why core features work very well. Also, in real-world scenarios, TR-1's threshold can be safely lowered because most hosts will not usually access more than 1 or 2 different ports of a destination host in one single day (let alone in 1 hour, or any other smaller time window that we might want to use); in the cases where they go up the threshold, the flow rule sets can effectively eliminate false-positive results.

5.5. Chapter Conclusions

Works	Detected Port Scans in CIC-IDS-2017	Best Overall Accuracy for CIC-IDS-2017 Friday's "Port Scan" Flow Classification
Previous work (2018) [10]	1	99.73%
Singh et. al, ICAESMT-2019 (2019) [148]	1	99.9815%
Stiawan et. al, IEEE Access 8, 132911–132921 (2020) [149]	1	99.7%
Current work (2020)	13	99.953%

TABLE 14. WORK COMPARISON IN CIC-IDS-2017 PORT SCAN DETECTION.

As we have already defined earlier in the previous chapter, core features are features that can successfully describe the core scenarios of a threat class, with either low possibilities of evasion or severely affecting that class's effectiveness if not detected. We believe we have achieved this for the Port Scan threat class, as you can see by analyzing each rule's effectiveness and the fact that the rules that can be evaded by the attacker represent the detection of a low percentage of flows – open port rules, which can always be made to look benign by truly accessing and using the service as a normal user would just to evade those rules at the L1-L4 level if the adversary intends to do so. The other rules, however, are more resilient, as they do not depend on L5-7 behavior.

Table 14 shows a comparison between our current work and three other works that have applied their detection mechanisms to the CIC-IDS-2017 dataset. 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 [145,146], 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 [148], 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 the already explained 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, as we have already shown in the beginning of this chapter: 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, as we have already detailed in subsection 5.2). 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.

Chapter 6. Conclusion

6.1. Main contributions and takeaways

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 (IPv4 and IPv6 “Conversations”). Summarily, Wireshark (and tshark for that matter) has the various network-object definitions well implemented, however it does not present us a set of features as rich as CICFlowMeter (as well as tranalyzer-2 and other tools that we’ve talked about) does, as it provides us a more human-friendly and efficient real-time visualization that does not consider the same number of features that we can consider to better study traffic. Our solution was to develop our own tool that tries to combine the best of Wireshark, which has a good definition of the concepts of flow, talker and host, to the best of CICFlowMeter, which gives us a vast set of data ready to analyze.

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.

With this work, we proved that ML-based classification is not a necessary requirement to achieve good results in network traffic analysis; in fact, we argue that it may lead to weaker classification systems in what it comes to being generic, not because of the algorithms, but because of the inherent lack of broadness in train and test data and leading to use non-core features. The features we implemented for flows, talkers, and hosts, provide a simple and effective way of querying and quickly analyzing traffic data. Machine Learning algorithms can always still be used for flow fingerprinting using

all the features that NetGenes provides (the only Boolean values are even provided one-hot encoded for this reason), but avoid providing higher relative importance to features other than the threat class's previously thought-out and explicitly implemented core features. Moreover, we have not found any related work that tackles the correct classification of hosts against the false-positive flow results, except for us in the context of the previous work, where we recognized that we were correctly identifying most test dataset flows using ML models, as many other related works were doing (at the time, for other datasets), but a high number of hosts would still be incorrectly tagged as malicious as a result of the false-positive flow classifications. The "Talker" and "Host" object introduced in this work provide the needed features to avoid false-positive talker and host classifications. This matter is very important because a human analyst should be given a low number of alerts regarding hosts, talkers or flows, to further investigate only alerts that are truly relevant; if we often output incorrect alerts, a correct alert will likely be discarded with the rest (like the story "The Boy Who Cried Wolf"). The highest the precision of a rule set (given that it is correctly built with the specifics of the threat class in mind) is, the higher relevance an output alert has. Furthermore, by using manually created rule sets that directly query the data, we were able to study how each rule set can individually contribute to achieve great flow classification results.

We do not need to worry about getting broad train datasets to prepare ML models for multiple tools. By focusing on the core features of a threat class, we do not require network traffic from multiple tools to model their common threat or threat class. Based on our previous work results, ML would be only able to help if we could gather a broad-enough dataset for every threat class we have to study; however, the problem is that software tools can be customized in endless ways, so the network traffic that is generated is different each time but still maintains the effectiveness of the malicious tool. This means that, to detect anomalous instances, it is not sufficient to extract multiple statistical features and delegate the classification job to the Machine Learning algorithm. It is our job to ensure the Machine Learning algorithms are being explicitly fed with broad data but, also, with core features that enable the "correct" detection of the threat class. Thus, rather than focusing on the algorithms we should use, we focused more on extracting important traffic features to best detect the threat classes we ought to detect. By successfully detecting the threat class itself, we are creating generic rules to detect threats and software tools that implement those threat classes, including new software tools that implement it which may not even have been created by someone. The practical application of a new tool or threat results in a completely new network attack, but we can still detect it because it is bound to maintain the threat class's core features. This shows our work's intent to fit in the anomaly detection category the best possible way that we can.

Additionally, our rule sets are not restrained by class-imbalanced datasets, nor is it restrained by the relative size of different, often mutually exclusive, types of instances (e.g., the difference between the number of dropped, rejected and accepted connections in the "Port Scan" threat class). The first often results in a classification bias towards higher-sized classes, while the latter results in a modelling bias of the class towards higher-sized types of instances within the same class. The first is a widely researched issue in related works about ML-based network traffic analysis, while the latter is a more hidden issue.

Another important aspect of the proposed method is that we can more easily understand what went wrong with our classification system, when it misses TP results in favor of FN ones, by directly applying the individual rule set responsible for the results and analyzing the flows that are left out by it. Additionally, an analyst could also think about new rule sets that would be better for their case and easily modify the classification system to their liking, by considering new custom rules, tuning rule parameters and/or deactivating default rules. On the other hand, a classification system based on pre-trained ML algorithms provides less of this type of flexibility. Additionally, a classification system based on our proposed flow rule sets provides more transparency than most ML-based works. Rather than specifying statistical features importance, we specify how we use them in high-precision rule sets.

By filtering out irrelevant flow sets, talker-based and host-based rule sets not only improve the detection results at the flow level, but more important, substantially reduce suspicions on benign talkers and hosts. The previous matter is very important because when a single flow is deemed as malicious, it could be incorrectly assumed that its talker and the two hosts are malicious, when that flow cold have just been a false positive. This is the reason why false-positive results in anomaly-based intrusion detection systems is an issue, and we propose that talker features and host features are used in order to avoid misclassifications of talkers and hosts solely based on flow rule sets.

6.2. Future Work

As future work, we propose to associate other L1-4 threat classes to specific L1-4 network object features. If the previous network object features are not implemented, implement those features, create the rule sets and experiment with datasets. If no new ideas come up on how to best detect those threat classes, even after extensively studying them, we propose to try to use ML classifiers to study the targeted threat class by studying how the ML model classifies it with great accuracies. If the ML model has not overfit to the training dataset, the feature importance values can provide hints about the core features (e.g., SYN/ACK flag counts can hint about the importance of the flow initiation, among others). Finally, we enumerate other future work that is performable based on this work:

- Further develop *NetGenes*: first, continuously include more L1-4 protocols and features; second, parse and extract relevant conceptual and statistical features from L5-7 protocols as well to enable working on the network behavior analysis capabilities of endpoint agents such as a HIDPS (Host-based Intrusion Detection and Prevention System) for a real-time use-case, or a malware sandbox analysis tools for a threat hunting use-case. Examples of this extraction can be seen in Cisco StealthWatch solution, which investigates TLS, DNS and HTTP in more depth to create conceptual features which are useful for threat identification. In theory, this will allow starting to tackle the detection of L5-7 threat classes by their core features rather than using ML-based classification to flow-fingerprint these threat classes.
- For rule sets concerned with data from specific ports and the L5-7 protocols used: implement L5-7 protocol fingerprinting to validate direct protocol-port correlation, using L5-7 parsers for decrypted traffic and using statistical analysis- or ML- based L1-4 detection for encrypted traffic, similar to TCP/IP stack fingerprinting.

- Implement a THP (Threat Hunting Platform): an integrating part of a larger Threat Intelligence Platform (TIP), it intends to automate and facilitate most of the work performed by threat researchers when analyzing threat-related network traffic (in the form of a trace-file), mostly by simplifying common data science and artificial intelligence use cases specifically applied to threat analysis. It should integrate *NetGenes* data output into a database and should also allow applying semi-automated labelling (e.g., labelling by host, by talker or by flow) in a user-friendly way. Finally, by using datasets with a common feature set, researchers would be able to compare their training-set network traffic and labelling methods more easily, rather than focusing on getting comprehensive and logically organized feature sets, which is a very big challenge by itself. Furthermore, this platform should also allow researchers to assess what features of the supported ones are relevant for their specific threat-related use-cases. Finally, this would also enable researchers to deeply study network attacks without needing to program or script anything, but rather simply use the platform.
- Generate explicit behavior-based classification rules to obtain white-box classification systems using advanced explainable AI methods [58, 59]. It could accelerate the discovery of core features by making the classification process as transparent as possible, while maintaining the great classification results ML offers for the default cases considered in the train dataset. The resulting rule sets would still need to be assessed against adversarial evasion.
- Implement a low-level TIP: real-time system with an involved community capable of generating shareable behavior-based and signature-based rulesets and integrating a THP capability.
- Implement a real-time NIDPS capable of using *NetGenes*'s extracted network-object features as a basis for building behavioral classification rules capable of operating at the flow-, talker- and host- levels. Furthermore, when it is available a real-time TIP with an involved community and support for shareable IoCs and signature and behavioral classification rules, add extra capability to the platform by integrating the NIDPS to enhance each constituency's detection capabilities, with automatically updatable classifiers based on the collaborative threat intelligence gathering performed in each network. Such a capability aims to automatically protect each constituency network from threats detected in collaborative constituencies, thus unifying security among all entities. More in-depth, Packet-based filtering will be performed by the usual signature-based methods, using external blacklists and threat intelligence feeds, but its rules will be continually updated by the community because IoCs (IPs, domains and malware signatures) will be shared between constituencies when a real detection occurs, given a confirmation. Behavioral classification rules, at the flow, talker and host levels, obtained through explainable AI methods and signature-based packet-level rules will be automatically generated locally and shared with the TIP community, preventing the infection of collaborative networks.
- Design a fully functional centralized SIEM architecture and implement it. It must be capable of integrating with the previous systems and being interoperable with other commonly used tools and standards.

Bibliography

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. Tranalyzer Website, http://tranalyzer.com, last accessed 2019/12/02
3. Gu, G., Porras, P.A., Yegneswaran, V., Fong, M.W., Lee, W.: Bothunter: Detecting malware infection through ids-driven dialog correlation. In: USENIX Security Symposium. vol. 7, pp. 1–16 (2007)
4. Gu, G., Zhang, J., Lee, W.: Botsniffer: Detecting botnet command and control channels in network traffic (2008)
5. Gu, G., Perdisci, R., Zhang, J., Lee, W.: Botminer: Clustering analysis of network traffic for protocol- and structure- independent botnet detection (2008)
6. 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)
7. Haddadi, F., Le Cong, D., Porter, L., Zincir-Heywood, A.N.: On the effectiveness of different botnet detection approaches. In: International Conference on Information Security Practice and Experience. pp. 121–135. Springer (2015)
8. Haddadi, F., Zincir-Heywood, A.N.: Botnet behaviour analysis: How would a data analytics-based system with minimum a priori information perform? International Journal of Network Management **27**(4), e1977 (2017).
9. 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)
10. Almeida, F., Meira, J., Adão, P., Loura, R.: Network Intrusion Detection: Machine-Learning Techniques for TCP Flow Classification (2018, unpublished)
11. White Paper - STIX Project, https://stixproject.github.io/getting-started/whitepaper/, last accessed 03/12/2019
12. White Paper - TAXII Project, https://taxiiproject.github.io/getting-started/whitepaper/, last accessed 04/12/2019
13. IntelMQ GitHub, https://github.com/certtools/intelmq, last accessed 04/12/2019
14. Ring, M., Wunderlich, S., Scheuring, D., Landes, D., Hotho, A.: A survey of network-based intrusion detection data sets. Computers & Security (2019)
15. Zhao, D., Traore, I., Sayed, B., Lu, W., Saad, S., Ghorbani, A., Garant, D.: Botnet detection based on traffic behavior analysis and flow intervals. Computers & Security **39**, 2–16 (2013)
16. Maltrail GitHub, https://github.com/stamparm/maltrail, last accessed 2019/12/05
17. 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
18. Khan, R.U., Zhang, X., Kumar, R., Sharif, A., Golilarz, N.A., Alazab, M.: An adaptive multi-layer botnet detection technique using machine learning classifiers. Applied Sciences **9**(11), 2375 (2019)
19. Nfdump GitHub, https://github.com/phaag/nfdump, last accessed 2019/12/06
20. Pmacct GitHub, https://github.com/pmacct/pmacct, last accessed 2019/12/07
21. Ntopng GitHub, https://github.com/ntop/ntopng, last accessed 2019/12/07
22. Zeek GitHub, https://github.com/zeek/zeek, last accessed 2019/12/08
23. Sacramento, L., Medeiros, I., Bota, J., Correia, M.: Flowhacker: Detecting unknown network attacks in big traffic data using network flows. In: 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). pp. 567–572. IEEE (2018)
24. Sacramento, L., Medeiros, I., Bota, J., Correia, M.: Detecting botnets and unknown network attacks in big traffic data. Botnets: Architectures, Countermeasures, and Challenges p. 237 (2019)
25. IntelMQ Bots Documentation, https://intelmq.readthedocs.io/en/latest/Bots/, last accessed 2019/12/09
26. The Githubification of InfoSec, https://medium.com/@johnlatwc/the-githubification-of-infosec-afbdbfaad1d1, last accessed 2019/12/10
27. Rohmad, M.S., Azmat, F., Manaf, M., Manan, J.A.: Enhanced netflow version 9 (e-netflow v9) for network mediation: Structure, experiment and analysis. In: 2008 International Symposium on Information Technology. vol. 3, pp. 1–6. IEEE (2008)
28. CSE-CIC-IDS2018 dataset, https://www.unb.ca/cic/datasets/ids-2018.html, last accessed 2019/12/11

29. ISCX-Bot-2014 dataset, https://www.unb.ca/cic/datasets/botnet.html, last accessed 2019/12/11
30. CTU-13 dataset, https://www.stratosphereips.org/datasets-ctu13, last accessed 2019/12/11
31. ISOT Botnet dataset, https://www.uvic.ca/engineering/ece/isot/datasets/, last accessed 2019/12/11
32. RFC 7676 - IPv6 Support for Generic Routing Encapsulation (GRE), https://tools.ietf.org/html/rfc7676, last accessed 2019/12/11
33. An overview of correlation measures between categorical and continuous variables, https://medium.com/@outside2SDs/an-overview-of-correlation-measures-between-categorical-and-continuous-variables-4c7f85610365, last accessed 2019/12/12
34. Feature Correlation and Feature Importance Bias with Random Forests, http://rnowling.github.io/machine/learning/2015/08/11/random-forest-correlation-bias.html, last accessed 2019/12/12
35. Cisco Security Certifications Overview, https://www.cisco.com/c/dam/en_us/training-events/certifications/shared/docs/sec_oView_dSheet.pdf, last accessed 2019/12/12
36. 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
37. Cisco VMDC Cloud Security 1.0 Design Guide, chap. 4 (2014)
38. 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
39. New Cisco Certification Updates 2019, https://blog.ine.com/new-cisco-certification-updates-2019, last accessed 2019/12/13
40. Cert News: New Cisco Certifications Coming in 2020, https://www.cbtnuggets.com/blog/career/career-progression/cert-news-new-cisco-ccna-coming-in-2020, last accessed 2019/12/13
41. Cisco Encrypted Traffic Analytics (ETA) Promotional Video, cisco.com/go/eta, last accessed 2019/12/19
42. 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
43. Radhakrishnan, S.: Detect threats in encrypted traffic without decryption, using network based security analytics (2017)
44. Cisco Encrypted Traffic Analytics: Necessity Driving Ubiquity, https://blogs.cisco.com/security/cisco-encrypted-traffic-analytics-necessity-driving-ubiquity, last accessed 2019/12/19
45. Orans, L., Hils, A., D'Hoinne, J., Ahlm, E.: Gartner Predicts 2017: Network and Gateway Security (2016)
46. Howard, J.D., Longstaff, T.A.: A common language for computer security incidents. Tech. rep., Sandia National Labs., Albuquerque, NM (US); Sandia National Labs (1998)
47. White Paper - Evaluation of Comprehensive Taxonomies for Information Technology Threats, https://www.sans.org/reading-room/whitepapers/threatintelligence/paper/38360, last accessed 2019/12/20
48. ENISA Threat Taxonomy, https://www.enisa.europa.eu/topics/threat-risk-management/threats-and-trends/enisa-threat-landscape/threat-taxonomy, last accessed 2019/12/20
49. Sfakianakis, A., Douligeris, C., Marinos, L., Lourenço, M., Raghimi, O.: Enisa threat landscape report 2018: 15 top cyberthreats and trends. DOI **10**, 622757(2019)
50. Steven Launius, Evaluation of Comprehensive Taxonomies for Information Technology Threats, SANS Institute (2018), https://www.sans.org/reading-room/whitepapers/threatintelligence/evaluation-comprehensive-taxonomies-information-technology-threats-38360, last accessed 2019/12/22.
51. NIST TTP definition, https://csrc.nist.gov/glossary/term/Tactics-Techniques-and-Procedures, last accessed 2020/01/03.
52. STIX TTP definition, https://stixproject.github.io/data-model/1.2/ttp/TTPType/, last accessed 2020/01/03.
53. 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.
54. Cisco Threat Grid Promotional Video, https://www.cisco.com/c/en/us/products/security/threat-grid/index.html, last accessed 2020/01/03.
55. Cisco Threat Grid Demo, July 2018, https://www.youtube.com/watch?v=un2t2T_s6IY, last accessed 2020/01/03.

56. Investigating Malware with Threat Grid, https://www.youtube.com/watch?v=W7IuchQR7dA, last accessed 2020/01/03.
57. Feature Importance: Explainable Artificial Intelligence, https://medium.com/time-to-work/feature-importance-c837e0e27155, last accessed 2020/01/04.
58. Interpretable AI or How I Learned to Stop Worrying and Trust AI: Techniques to build Robust, Unbiased AI Applications, https://towardsdatascience.com/the-how-of-explainable-ai-post-modelling-explainability-8b4cbc7adf5f, last accessed 2020/01/04.
59. The How of Explainable AI: Post-modelling Explainability, https://towardsdatascience.com/the-how-of-explainable-ai-post-modelling-explainability-8b4cbc7adf5f, last accessed 2020/01/04.
60. Hindy, H., Brosset, D., Bayne, E., Seeam, A., Tachtatzis, C., Atkinson, R., Bellekens, X.: A taxonomy and survey of intrusion detection system design techniques, network threats and datasets. arXiv preprint arXiv:1806.03517 (2018)
61. Leroux, S., Bohez, S., Maenhaut, P.J., Meheus, N., Simoens, P., Dhoedt, B.: Fingerprinting encrypted network traffic types using machine learning. In: NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium. pp. 1–5. IEEE (2018)
62. Garcia, S., Grill, M., Stiborek, J., Zunino, A.: An empirical comparison of botnet detection methods. computers & security **45**, 100–123 (2014)
63. White Paper – Open Threat Taxonomy (Version 1.1), https://www.auditscripts.com/resources/open_threat_taxonomy_v1.1a.pdf, last accessed 31/03/2020
64. https://securitytrails.com/blog/top-scanned-ports, last accessed 2020/08/03.
65. TCP Flags summarized, https://www.keycdn.com/support/tcp-flags, last accessed 2020/08/02.
66. IPv4Security.com - TCP Start timeout / TCP Session timeout / TCP End timeout on a CheckPoint firewall, http://www.ipv4security.com/packet_flow/tcp_timeout.txt, last accessed 2020/08/04.
67. Network attack research reference, https://www.yeahhub.com/15-most-useful-host-scanning-commands-kalilinux/, last accessed 15/11, last accessed 2020/11/15.
68. Network attack research reference, https://cheatsheetseries.owasp.org/cheatsheets/Denial_of_Service_Cheat_Sheet.html, last accessed 15/11.
69. Network attack research reference, http://www.cs.columbia.edu/~dgen/papers/conferences/conference-07.pdf, last accessed 2020/11/15.
70. Network attack research reference, https://betangel.kayako.com/article/107-how-do-i-secure-my-dns-resolver-against-amplification-attacks, last accessed 2020/11/15.
71. Network attack research reference, https://nmap.org/man/pt_PT/man-host-discovery.html, last accessed 2020/11/15.
72. Network attack research reference, https://nmap.org/book/man-port-scanning-techniques.html, last accessed 2020/11/15.
73. Network attack research reference, https://medium.com/@iphelix/port-scanning-techniques-7661839d182e, last accessed 2020/11/15.
74. Network attack research reference, https://nmap.org/book/idlescan.html, last accessed 2020/11/15.
75. Network attack research reference, https://nmap.org/misc/split-handshake.pdf, last accessed 2020/11/15.
76. Network attack research reference, https://en.wikipedia.org/wiki/SYN_flood, last accessed 2020/11/15.
77. Network attack research reference, https://www.sciencedirect.com/topics/computer-science/denial-of-service, last accessed 2020/11/15.
78. Network attack research reference, https://cheatsheetseries.owasp.org/cheatsheets/Denial_of_Service_Cheat_Sheet.html, last accessed 2020/11/15.
79. Network attack research reference, https://link.springer.com/referenceworkentry/10.1007%2F978-1-4419-5906-5_262, last accessed 2020/11/15.
80. Network attack research reference, https://nmap.org/book/man-version-detection.html, last accessed 2020/11/15.
81. Network attack research reference, https://nmap.org/book/osdetect-ipv6-methods.html, last accessed 2020/11/15.
82. Network attack research reference, https://nmap.org/book/port-scanning-options.html, last accessed 2020/11/15.
83. Network attack research reference, https://linux.die.net/man/1/nmap, last accessed 2020/11/15.

84. Network attack research reference, https://www.cloudflare.com/learning/ddos/ddos-attack-tools/slowloris/, last accessed 2020/11/16.
85. Network attack research reference, https://www.cloudflare.com/learning/ddos/ddos-low-and-slow-attack/, last accessed 2020/11/16.
86. Network attack research reference, https://www.cloudflare.com/learning/ddos/http-flood-ddos-attack/, last accessed 2020/11/16.
87. Network attack research reference, https://blog.qualys.com/securitylabs/2011/11/02/how-to-protect-against-slow-http-attacks, last accessed 2020/11/16.
88. Network attack research reference, https://www.imperva.com/learn/application-security/slowloris/, last accessed 2020/11/16.
89. Network attack research reference, https://help.fortinet.com/fos50hlp/54/Content/FortiOS/fortigate-firewall-52/Concepts/SCTP.htm, last accessed 2020/11/16.
90. Network attack research reference, https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/, last accessed 2020/11/16.
91. Network attack research reference, https://kb.mazebolt.com/knowledgebase/slowloris-attack/, last accessed 2020/11/16.
92. Network attack research reference, https://www.imperva.com/learn/application-security/rudy-r-u-dead-yet/, last accessed 2020/11/16.
93. Network attack research reference, https://www.imperva.com/learn/application-security/http-flood/, last accessed 2020/11/16.
94. Network attack research reference, https://www.techrepublic.com/blog/data-center/sockstress-a-new-and-effective-dos-attack/, last accessed 2020/11/16.
95. Network attack research reference, https://en.wikipedia.org/wiki/Sockstress, last accessed 2020/11/16.
96. Network attack research reference, https://security.radware.com/ddos-knowledge-center/ddospedia/sockstress/, last accessed 2020/11/17.
97. Network attack research reference, https://security.radware.com/ddos-knowledge-center/ddospedia/tcp-flood/, last accessed 2020/11/17.
98. Network attack research reference, https://security.radware.com/ddos-knowledge-center/ddospedia/http-flood/, last accessed 2020/11/17.
99. Network attack research reference, https://tools.ietf.org/html/rfc5062, last accessed 2020/11/17.
100. Network attack research reference, https://www.stationx.net/nmap-cheat-sheet/, last accessed 2020/11/17.
101. Network attack research reference, https://en.wikipedia.org/wiki/Denial-of-service_attack, last accessed 2020/11/18.
102. Network attack research reference, https://www.f5.com/services/resources/glossary/icmp-flood-ping-flood-smurf-attack, last accessed 2020/11/18.
103. Network attack research reference, https://www.imperva.com/learn/application-security/smurf-attack-ddos/, last accessed 2020/11/18.
104. Network attack research reference, https://security.radware.com/ddos-knowledge-center/ddospedia/teardrop-attack/, last accessed 2020/11/18.
105. Network attack research reference, https://security.radware.com/ddos-knowledge-center/ddospedia/land-attack/, last accessed 2020/11/18.
106. Network attack research reference, https://en.wikipedia.org/wiki/Smurf_attack, last accessed 2020/11/18.
107. Network attack research reference, https://www.cloudflare.com/learning/ddos/dns-amplification-ddos-attack/, last accessed 2020/11/18.
108. Network attack research reference, https://www.cloudflare.com/learning/ddos/ntp-amplification-ddos-attack/, last accessed 2020/11/18.
109. Network attack research reference, https://www.darkreading.com/attacks-breaches/new-ddos-attacks-leverage-tcp-amplification-/d/d-id/1336339, last accessed 2020/11/19.
110. Network attack research reference, https://www.trendmicro.com/vinfo/hk-en/security/news/cyber-attacks/ddos-attacks-that-employ-tcp-amplification-cause-network-congestion-secondary-outages, last accessed 2020/11/19.
111. Network attack research reference, https://blog.verisign.com/security/dns-based-threats-dns-reflection-amplification-attacks/, last accessed 2020/11/19.
112. Network attack research reference, https://www.us-cert.gov/ncas/alerts/TA14-017A, last accessed 2020/11/19.

113. Network attack research reference, https://isc.sans.edu/forums/diary/How+did+it+all+start+Early+Memcached+DDoS+Attack+Precursors+and+Ransom+Notes/23437/, last accessed 2020/11/19.
114. Network attack research reference, https://www.cloudflare.com/learning/ddos/ping-icmp-flood-ddos-attack/, last accessed 2020/11/19.
115. Network attack research reference, https://link.springer.com/content/pdf/10.1007%2F978-0-387-34827-8_6.pdf, last accessed 2020/11/19.
116. Network attack research reference, https://www.akamai.com/uk/en/resources/our-thinking/threat-advisories/connection-less-lightweight-directory-access-protocol-reflection-ddos-threat-advisory.jsp, last accessed 2020/11/19.
117. Network attack research reference, https://beyondsecurity.com/scan-pentest-network-cisco-ssh-malformed-packet-dos-vulnerability.html?cn-reloaded=1, last accessed 2020/11/19.
118. Network attack research reference, https://security.radware.com/ddos-knowledge-center/ddospedia/low-rate-attack/, last accessed 2020/11/19.
119. Network attack research reference, https://br.netscout.com/what-is-ddos/slow-post-attacks, last accessed 2020/11/19.
120. Network attack research reference, https://samsclass.info/123/proj10/sockstress.htm, last accessed 2020/11/19.
121. Network attack research reference, https://activereach.net/resources/ddos-knowledge-centre/ddos-dictionary/, last accessed 2020/11/19.
122. Network attack research reference, https://securiteam.com/securitynews/6b0031f0ka/, last accessed 2020/11/19.
123. Network attack research reference, https://ddos-guard.net/en/terminology/attack_type/naptha-attack, last accessed 2020/11/19.
124. Network attack research reference, https://blog.radware.com/security/2016/06/2016-republican-national-convention-rnc-democratic-national-convention-dnc-will-be-cyber-attacked/, last accessed 2020/11/19.
125. Network attack research reference, https://kb.mazebolt.com/knowledgebase_category/layer-3/, last accessed 2020/11/19.
126. Network attack research reference, https://kb.mazebolt.com/knowledgebase_category/layer-4/, last accessed 2020/11/19.
127. Network attack research reference, https://kb.mazebolt.com/knowledgebase_category/layer-7/, last accessed 2020/11/19.
128. Network attack research reference, https://kb.mazebolt.com/knowledgebase/ip-fragmented-flood/, last accessed 2020/11/19.
129. Network attack research reference, https://kb.mazebolt.com/knowledgebase/icmp-ping-flood/, last accessed 2020/11/19.
130. Network attack research reference, https://kb.mazebolt.com/knowledgebase/icmp-time-exceeded-flood/, last accessed 2020/11/19.
131. Network attack research reference, https://kb.mazebolt.com/knowledgebase/icmp-destination-unreachable-flood/, last accessed 2020/11/19.
132. Network attack research reference, https://www.gont.com.ar/papers/tn-03-09-security-assessment-TCP.pdf, last accessed 2020/11/19.
133. Network attack research reference, https://www.grc.com/sn/notes-164.htm, last accessed 2020/11/19.
134. Network attack research reference, https://ddos-guard.net/en/terminology/attack_type, last accessed 2020/11/19.
135. Network attack research reference, https://www.varonis.com/blog/smb-port/, last accessed 2020/11/19.
136. Network attack research reference, https://securitytrails.com/blog/top-scanned-ports, last accessed 2020/11/19.
137. Network attack research reference, https://www.extrahop.com/company/blog/2016/how-to-recognize-malicious-network-scanning-port-scanning/, last accessed 2020/11/19.
138. Service overview and network port requirements for Windows, https://docs.microsoft.com/en-us/troubleshoot/windows-server/networking/service-overview-and-network-port-requirements, last accessed 2020/11/23.
139. IANA Service Name and Transport Protocol Port Number Registry, https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml, last accessed 2020/11/23.

140. SpeedGuide.Net Port “XYZ” Information, https://www.speedguide.net/port.php?port=XYZ, last accessed 2020/11/23.
141. NMAP - Subverting Intrusion Detection Systems, https://nmap.org/book/subvert-ids.html, last accessed 2020/11/24.
142. Panigrahi, R., Borah, S.: A detailed analysis of cicids2017 dataset for designing intrusion detection systems. International Journal of Engineering & Technology **7** (3.24), 479–482 (2018).
143. Zhang, Y., Chen, X., Jin, L., Wang, X., Guo, D.: Network intrusion detection: Based on deep hierarchical network and original flow data. IEEE Access 7, 37004–37016 (2019).
144. Confusion Matrix Wikipedia – Metric Calculations, https://en.wikipedia.org/wiki/Confusion_matrix, last accessed 2020/12/15.
145. CIC-IDS-2017’s official website, https://www.unb.ca/cic/datasets/ids-2017.html, last accessed 2020/12/28.
146. 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)
147. Gustavsson, V.: Machine learning for a network-based intrusion detection system: An application using zeek and the cicids2017 dataset (2019)
148. 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)
149. 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)
150. Ullah, I., Mahmoud, Q.H.: A two-level flow-based anomalous activity detection system for iot networks. Electronics **9**(3), 530 (2020)

Annex

The annex is organized as follows

Table 15. NetGenes Packet features.

Table 16. NetGenes Flow features.

Table 17. NetGenes Talker features.

Table 18. NetGenes Host features (without flow-set based features).

Table 19. Monday: TCP Benign Traffic Overview.

Table 20. Monday: UDP Benign Traffic Overview.

Table 21. TCP Bi-Talker Forward and Backward “Unique Destination Port Count” analysis.

Table 22. UDP Bi-Talker Forward and Backward “Unique Destination Port Count” analysis.

Table 21 and Table 22 Legend:

- Abbreviations: V – Values; T – Talkers; F – Flows
- Colors: Color Absence – Not “Port Scan”; Yellow – “Port Scan” Exception; Red – “Port Scan”

Protocol	Additional Information	Packet Features			
Generic	timestamp				
IPv4	src_ip	ipv4_header_len		ipv4_data_len	
	dst_ip	ipv4_df_flag		ipv4_mf_flag	
L4	src_port	l4_header_len		l4_data_len	
	dst_port				
TCP		tcp_seq_numb		tcp_ack_numb	
		tcp_fin_flag	tcp_syn_flag	tcp_rst_flag	tcp_psh_flag
		tcp_ack_flag	tcp_urg_flag	tcp_ece_flag	tcp_cwr_flag

TABLE 15. NETGENES PACKET FEATURES.

Protocol	Ad. Info.		Flow Features	
Generic	biflow_any_first_packet_time	biflow_any_last_packet_time	biflow_any_duration	biflow_any_n_packets
			biflow_fwd_n_packets	biflow_bwd_n_packets
			biflow_any_packets_per_sec	biflow_fwd_packets_per_sec
			biflow_bwd_packets_per_sec	biflow_any_bytes_per_sec
			biflow_fwd_bytes_per_sec	biflow_bwd_bytes_per_sec
			biflow_any_packet_iat_total	biflow_any_packet_iat_mean
			biflow_any_packet_iat_std	biflow_any_packet_iat_var
			biflow_any_packet_iat_max	biflow_any_packet_iat_min
			biflow_fwd_packet_iat_total	biflow_fwd_packet_iat_mean
			biflow_fwd_packet_iat_std	biflow_fwd_packet_iat_var
			biflow_fwd_packet_iat_max	biflow_fwd_packet_iat_min
			biflow_bwd_packet_iat_total	biflow_bwd_packet_iat_mean
			biflow_bwd_packet_iat_std	biflow_bwd_packet_iat_var
			biflow_bwd_packet_iat_max	biflow_bwd_packet_iat_min
IPv4	bihost_fwd_id	bihost_bwd_id	biflow_any_eth_ip_n_active_df_flags	biflow_fwd_eth_ip_n_active_df_flags
			biflow_bwd_eth_ip_n_active_df_flags	biflow_any_eth_ip_n_active_mf_flags
			biflow_fwd_eth_ip_n_active_mf_flags	biflow_bwd_eth_ip_n_active_mf_flags
			biflow_any_eth_ipv4_header_len_total	biflow_any_eth_ipv4_header_len_mean
			biflow_any_eth_ipv4_header_len_std	biflow_any_eth_ipv4_header_len_var
			biflow_any_eth_ipv4_header_len_max	biflow_any_eth_ipv4_header_len_min
			biflow_fwd_eth_ipv4_header_len_total	biflow_fwd_eth_ipv4_header_len_mean
			biflow_fwd_eth_ipv4_header_len_std	biflow_fwd_eth_ipv4_header_len_var
			biflow_fwd_eth_ipv4_header_len_max	biflow_fwd_eth_ipv4_header_len_min
			biflow_bwd_eth_ipv4_header_len_total	biflow_bwd_eth_ipv4_header_len_mean
			biflow_bwd_eth_ipv4_header_len_std	biflow_bwd_eth_ipv4_header_len_var
			biflow_bwd_eth_ipv4_header_len_max	biflow_bwd_eth_ipv4_header_len_min
			biflow_any_eth_ipv4_data_len_total	biflow_any_eth_ipv4_data_len_mean
			biflow_any_eth_ipv4_data_len_std	biflow_any_eth_ipv4_data_len_var

			biflow_any_eth_ipv4_data_len_max	biflow_any_eth_ipv4_data_len_min
			biflow_fwd_eth_ipv4_data_len_total	biflow_fwd_eth_ipv4_data_len_mean
			biflow_fwd_eth_ipv4_data_len_std	biflow_fwd_eth_ipv4_data_len_var
			biflow_fwd_eth_ipv4_data_len_max	biflow_fwd_eth_ipv4_data_len_min
			biflow_bwd_eth_ipv4_data_len_total	biflow_bwd_eth_ipv4_data_len_mean
			biflow_bwd_eth_ipv4_data_len_std	biflow_bwd_eth_ipv4_data_len_var
			biflow_bwd_eth_ipv4_data_len_max	biflow_bwd_eth_ipv4_data_len_min
L4	biflow_src_port	biflow_dst_port	biflow_any_eth_ipv4_l4_n_data_packets	biflow_fwd_eth_ipv4_l4_n_data_packets
			biflow_bwd_eth_ipv4_l4_n_data_packets	biflow_any_eth_ipv4_l4_data_packets_per_sec
			biflow_fwd_eth_ipv4_l4_data_packets_per_sec	biflow_bwd_eth_ipv4_l4_data_packets_per_sec
			biflow_any_eth_ipv4_l4_header_len_total	biflow_any_eth_ipv4_l4_header_len_mean
			biflow_any_eth_ipv4_l4_header_len_std	biflow_any_eth_ipv4_l4_header_len_var
			biflow_any_eth_ipv4_l4_header_len_max	biflow_any_eth_ipv4_l4_header_len_min
			biflow_fwd_eth_ipv4_l4_header_len_total	biflow_fwd_eth_ipv4_l4_header_len_mean
			biflow_fwd_eth_ipv4_l4_header_len_std	biflow_fwd_eth_ipv4_l4_header_len_var
			biflow_fwd_eth_ipv4_l4_header_len_max	biflow_fwd_eth_ipv4_l4_header_len_min
			biflow_bwd_eth_ipv4_l4_header_len_total	biflow_bwd_eth_ipv4_l4_header_len_mean
			biflow_bwd_eth_ipv4_l4_header_len_std	biflow_bwd_eth_ipv4_l4_header_len_var
			biflow_bwd_eth_ipv4_l4_header_len_max	biflow_bwd_eth_ipv4_l4_header_len_min
			biflow_any_eth_ipv4_l4_data_len_total	biflow_any_eth_ipv4_l4_data_len_mean
			biflow_any_eth_ipv4_l4_data_len_std	biflow_any_eth_ipv4_l4_data_len_var
			biflow_any_eth_ipv4_l4_data_len_max	biflow_any_eth_ipv4_l4_data_len_min
			biflow_fwd_eth_ipv4_l4_data_len_total	biflow_fwd_eth_ipv4_l4_data_len_mean
			biflow_fwd_eth_ipv4_l4_data_len_std	biflow_fwd_eth_ipv4_l4_data_len_var
			biflow_fwd_eth_ipv4_l4_data_len_max	biflow_fwd_eth_ipv4_l4_data_len_min
			biflow_bwd_eth_ipv4_l4_data_len_total	biflow_bwd_eth_ipv4_l4_data_len_mean
			biflow_bwd_eth_ipv4_l4_data_len_std	biflow_bwd_eth_ipv4_l4_data_len_var
			biflow_bwd_eth_ipv4_l4_data_len_max	biflow_bwd_eth_ipv4_l4_data_len_min
TCP	flow_separation_counter		biflow_eth_ipv4_tcp_initiation_requested_connection	biflow_eth_ipv4_tcp_initiation_two_way_handshake
			biflow_eth_ipv4_tcp_initiation_three_way_handshake	biflow_eth_ipv4_tcp_connection_redropped
			biflow_eth_ipv4_tcp_connection_rejected	biflow_eth_ipv4_tcp_connection_established_half_duplex
			biflow_eth_ipv4_tcp_connection_established_full_duplex	biflow_eth_ipv4_tcp_termination_abort
			biflow_eth_ipv4_tcp_termination_null	biflow_eth_ipv4_tcp_termination_graceful
			biflow_any_eth_ipv4_tcp_n_active_fin_flags	biflow_any_eth_ipv4_tcp_n_active_syn_flags
			biflow_any_eth_ipv4_tcp_n_active_rst_flags	biflow_any_eth_ipv4_tcp_n_active_psh_flags
			biflow_any_eth_ipv4_tcp_n_active_ack_flags	biflow_any_eth_ipv4_tcp_n_active_urg_flags
			biflow_any_eth_ipv4_tcp_n_active_ece_flags	biflow_any_eth_ipv4_tcp_n_active_cwr_flags
			biflow_fwd_eth_ipv4_tcp_n_active_fin_flags	biflow_fwd_eth_ipv4_tcp_n_active_syn_flags
			biflow_fwd_eth_ipv4_tcp_n_active_rst_flags	biflow_fwd_eth_ipv4_tcp_n_active_psh_flags
			biflow_fwd_eth_ipv4_tcp_n_active_ack_flags	biflow_fwd_eth_ipv4_tcp_n_active_urg_flags
			biflow_fwd_eth_ipv4_tcp_n_active_ece_flags	biflow_fwd_eth_ipv4_tcp_n_active_cwr_flags
			biflow_bwd_eth_ipv4_tcp_n_active_fin_flags	biflow_bwd_eth_ipv4_tcp_n_active_syn_flags
			biflow_bwd_eth_ipv4_tcp_n_active_rst_flags	biflow_bwd_eth_ipv4_tcp_n_active_psh_flags
			biflow_bwd_eth_ipv4_tcp_n_active_ack_flags	biflow_bwd_eth_ipv4_tcp_n_active_urg_flags
			biflow_bwd_eth_ipv4_tcp_n_active_ece_flags	biflow_bwd_eth_ipv4_tcp_n_active_cwr_flags

TABLE 16. NETGENES FLOW FEATURES.

Protocol	Ad. Info.		Talker Features	
Generic	bitalker_any_first_biflow_initiation_time	bitalker_any_last_biflow_termination_time	bitalker_any_duration	bitalker_any_n_biflows
			bitalker_fwd_n_biflows	bitalker_bwd_n_biflows
			bitalker_any_biflows_per_sec	bitalker_fwd_biflows_per_sec
			bitalker_bwd_biflows_per_sec	bitalker_any_biflow_bytes_per_sec
			bitalker_fwd_biflow_bytes_per_sec	bitalker_bwd_biflow_bytes_per_sec
			bitalker_any_biflow_n_packets_total	bitalker_any_biflow_n_packets_mean
			bitalker_any_biflow_n_packets_std	bitalker_any_biflow_n_packets_var
			bitalker_any_biflow_n_packets_max	bitalker_any_biflow_n_packets_min
			bitalker_fwd_biflow_n_packets_total	bitalker_fwd_biflow_n_packets_mean
			bitalker_fwd_biflow_n_packets_std	bitalker_fwd_biflow_n_packets_var
			bitalker_fwd_biflow_n_packets_max	bitalker_fwd_biflow_n_packets_min
			bitalker_bwd_biflow_n_packets_total	bitalker_bwd_biflow_n_packets_mean
			bitalker_bwd_biflow_n_packets_std	bitalker_bwd_biflow_n_packets_var
			bitalker_bwd_biflow_n_packets_max	bitalker_bwd_biflow_n_packets_min
			bitalker_any_biflow_duration_total	bitalker_any_biflow_duration_mean
			bitalker_any_biflow_duration_std	bitalker_any_biflow_duration_var
			bitalker_any_biflow_duration_max	bitalker_any_biflow_duration_min
			bitalker_fwd_biflow_duration_total	bitalker_fwd_biflow_duration_mean
			bitalker_fwd_biflow_duration_std	bitalker_fwd_biflow_duration_var
			bitalker_fwd_biflow_duration_max	bitalker_fwd_biflow_duration_min
			bitalker_bwd_biflow_duration_total	bitalker_bwd_biflow_duration_mean
			bitalker_bwd_biflow_duration_std	bitalker_bwd_biflow_duration_var
			bitalker_bwd_biflow_duration_max	bitalker_bwd_biflow_duration_min
			bitalker_any_biflow_iit_total	bitalker_any_biflow_iit_mean
			bitalker_any_biflow_iit_std	bitalker_any_biflow_iit_var
			bitalker_any_biflow_iit_max	bitalker_any_biflow_iit_min
			bitalker_fwd_biflow_iit_total	bitalker_fwd_biflow_iit_mean
			bitalker_fwd_biflow_iit_std	bitalker_fwd_biflow_iit_var
			bitalker_fwd_biflow_iit_max	bitalker_fwd_biflow_iit_min
			bitalker_bwd_biflow_iit_total	bitalker_bwd_biflow_iit_mean
			bitalker_bwd_biflow_iit_std	bitalker_bwd_biflow_iit_var
			bitalker_bwd_biflow_iit_max	bitalker_bwd_biflow_iit_min
			bitalker_any_biflow_itt_total	bitalker_any_biflow_itt_mean
			bitalker_any_biflow_itt_std	bitalker_any_biflow_itt_var
			bitalker_any_biflow_itt_max	bitalker_any_biflow_itt_min
			bitalker_fwd_biflow_itt_total	bitalker_fwd_biflow_itt_mean
			bitalker_fwd_biflow_itt_std	bitalker_fwd_biflow_itt_var
			bitalker_fwd_biflow_itt_max	bitalker_fwd_biflow_itt_min
			bitalker_bwd_biflow_itt_total	bitalker_bwd_biflow_itt_mean
			bitalker_bwd_biflow_itt_std	bitalker_bwd_biflow_itt_var
			bitalker_bwd_biflow_itt_max	bitalker_bwd_biflow_itt_min

IPv4	bihost_fwd_id	bihost_bwd_id	bitalker_any_biflow_eth_ipv4_data_lens_total	bitalker_any_biflow_eth_ipv4_data_lens_mean
			bitalker_any_biflow_eth_ipv4_data_lens_std	bitalker_any_biflow_eth_ipv4_data_lens_var
			bitalker_any_biflow_eth_ipv4_data_lens_max	bitalker_any_biflow_eth_ipv4_data_lens_min
			bitalker_fwd_biflow_eth_ipv4_data_lens_total	bitalker_fwd_biflow_eth_ipv4_data_lens_mean
			bitalker_fwd_biflow_eth_ipv4_data_lens_std	bitalker_fwd_biflow_eth_ipv4_data_lens_var
			bitalker_fwd_biflow_eth_ipv4_data_lens_max	bitalker_fwd_biflow_eth_ipv4_data_lens_min
			bitalker_bwd_biflow_eth_ipv4_data_lens_total	bitalker_bwd_biflow_eth_ipv4_data_lens_mean
			bitalker_bwd_biflow_eth_ipv4_data_lens_std	bitalker_bwd_biflow_eth_ipv4_data_lens_var
			bitalker_bwd_biflow_eth_ipv4_data_lens_max	bitalker_bwd_biflow_eth_ipv4_data_lens_min
L4			bitalker_any_biflow_n_unique_dst_ports	bitalker_fwd_biflow_n_unique_dst_ports
			bitalker_bwd_biflow_n_unique_dst_ports	bitalker_any_biflow_n_unique_src_ports
			bitalker_fwd_biflow_n_unique_src_ports	bitalker_bwd_biflow_n_unique_src_ports
			bitalker_any_eth_ipv4_l4_biflow_n_data_packets_total	bitalker_any_eth_ipv4_l4_biflow_n_data_packets_mean
			bitalker_any_eth_ipv4_l4_biflow_n_data_packets_std	bitalker_any_eth_ipv4_l4_biflow_n_data_packets_var
			bitalker_any_eth_ipv4_l4_biflow_n_data_packets_max	bitalker_any_eth_ipv4_l4_biflow_n_data_packets_min
			bitalker_fwd_eth_ipv4_l4_biflow_n_data_packets_total	bitalker_fwd_eth_ipv4_l4_biflow_n_data_packets_mean
			bitalker_fwd_eth_ipv4_l4_biflow_n_data_packets_std	bitalker_fwd_eth_ipv4_l4_biflow_n_data_packets_var
			bitalker_fwd_eth_ipv4_l4_biflow_n_data_packets_max	bitalker_fwd_eth_ipv4_l4_biflow_n_data_packets_min
			bitalker_bwd_eth_ipv4_l4_biflow_n_data_packets_total	bitalker_bwd_eth_ipv4_l4_biflow_n_data_packets_mean
			bitalker_bwd_eth_ipv4_l4_biflow_n_data_packets_std	bitalker_bwd_eth_ipv4_l4_biflow_n_data_packets_var
			bitalker_bwd_eth_ipv4_l4_biflow_n_data_packets_max	bitalker_bwd_eth_ipv4_l4_biflow_n_data_packets_min
TCP			bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_total	bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_mean
			bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_std	bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_var
			bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_max	bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_min
			bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_total	bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_mean
			bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_std	bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_var
			bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_max	bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_min
			bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_total	bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_mean
			bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_std	bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_var
			bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_max	bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_min
			bitalker_eth_ipv4_tcp_biflow_connections_redropped_total	bitalker_eth_ipv4_tcp_biflow_connections_redropped_mean
			bitalker_eth_ipv4_tcp_biflow_connections_redropped_std	bitalker_eth_ipv4_tcp_biflow_connections_redropped_var
			bitalker_eth_ipv4_tcp_biflow_connections_redropped_max	bitalker_eth_ipv4_tcp_biflow_connections_redropped_min
			bitalker_eth_ipv4_tcp_biflow_connections_rejected_total	bitalker_eth_ipv4_tcp_biflow_connections_rejected_mean
			bitalker_eth_ipv4_tcp_biflow_connections_rejected_std	bitalker_eth_ipv4_tcp_biflow_connections_rejected_var
			bitalker_eth_ipv4_tcp_biflow_connections_rejected_max	bitalker_eth_ipv4_tcp_biflow_connections_rejected_min
			bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_total	bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_mean
			bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_std	bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_var
			bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_max	bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_min
			bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_total	bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_mean
			bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_std	bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_var
			bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_max	bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_min
			bitalker_eth_ipv4_tcp_biflow_abort_terminations_total	bitalker_eth_ipv4_tcp_biflow_abort_terminations_mean
			bitalker_eth_ipv4_tcp_biflow_abort_terminations_std	bitalker_eth_ipv4_tcp_biflow_abort_terminations_var
			bitalker_eth_ipv4_tcp_biflow_abort_terminations_max	bitalker_eth_ipv4_tcp_biflow_abort_terminations_min
			bitalker_eth_ipv4_tcp_biflow_null_terminations_total	bitalker_eth_ipv4_tcp_biflow_null_terminations_mean
			bitalker_eth_ipv4_tcp_biflow_null_terminations_std	bitalker_eth_ipv4_tcp_biflow_null_terminations_var
			bitalker_eth_ipv4_tcp_biflow_null_terminations_max	bitalker_eth_ipv4_tcp_biflow_null_terminations_min
			bitalker_eth_ipv4_tcp_biflow_graceful_terminations_total	bitalker_eth_ipv4_tcp_biflow_graceful_terminations_mean
			bitalker_eth_ipv4_tcp_biflow_graceful_terminations_std	bitalker_eth_ipv4_tcp_biflow_graceful_terminations_var
			bitalker_eth_ipv4_tcp_biflow_graceful_terminations_max	bitalker_eth_ipv4_tcp_biflow_graceful_terminations_min

TABLE 17. NETGENES TALKER FEATURES.

Protocol	Ad. Info.		Host Features	
Generic	bihost_any_first_bitalker_initiation_time	bihost_any_last_bitalker_termination_time	bihost_any_duration	bihost_any_n_bitalkers
			bihost_fwd_n_bitalkers	bihost_bwd_n_bitalkers
			bihost_any_bitalkers_per_sec	bihost_fwd_bitalkers_per_sec
			bihost_bwd_bitalkers_per_sec	bihost_any_bitalker_bytes_per_sec
			bihost_fwd_bitalker_bytes_per_sec	bihost_bwd_bitalker_bytes_per_sec
			bihost_any_bitalker_n_biflows_total	bihost_any_bitalker_n_biflows_mean
			bihost_any_bitalker_n_biflows_std	bihost_any_bitalker_n_biflows_var
			bihost_any_bitalker_n_biflows_max	bihost_any_bitalker_n_biflows_min
			bihost_fwd_bitalker_n_biflows_total	bihost_fwd_bitalker_n_biflows_mean
			bihost_fwd_bitalker_n_biflows_std	bihost_fwd_bitalker_n_biflows_var
			bihost_fwd_bitalker_n_biflows_max	bihost_fwd_bitalker_n_biflows_min
			bihost_bwd_bitalker_n_biflows_total	bihost_bwd_bitalker_n_biflows_mean
			bihost_bwd_bitalker_n_biflows_std	bihost_bwd_bitalker_n_biflows_var
			bihost_bwd_bitalker_n_biflows_max	bihost_bwd_bitalker_n_biflows_min
IPv4	bihost_id		bihost_any_bitalker_any_biflow_eth_ipv4_data_lens_total	bihost_any_bitalker_any_biflow_eth_ipv4_data_lens_mean
			bihost_any_bitalker_any_biflow_eth_ipv4_data_lens_std	bihost_any_bitalker_any_biflow_eth_ipv4_data_lens_var
			bihost_any_bitalker_any_biflow_eth_ipv4_data_lens_max	bihost_any_bitalker_any_biflow_eth_ipv4_data_lens_min
			bihost_fwd_bitalker_any_biflow_eth_ipv4_data_lens_total	bihost_fwd_bitalker_any_biflow_eth_ipv4_data_lens_mean
			bihost_fwd_bitalker_any_biflow_eth_ipv4_data_lens_std	bihost_fwd_bitalker_any_biflow_eth_ipv4_data_lens_var
			bihost_fwd_bitalker_any_biflow_eth_ipv4_data_lens_max	bihost_fwd_bitalker_any_biflow_eth_ipv4_data_lens_min
			bihost_bwd_bitalker_any_biflow_eth_ipv4_data_lens_total	bihost_bwd_bitalker_any_biflow_eth_ipv4_data_lens_mean
			bihost_bwd_bitalker_any_biflow_eth_ipv4_data_lens_std	bihost_bwd_bitalker_any_biflow_eth_ipv4_data_lens_var
			bihost_bwd_bitalker_any_biflow_eth_ipv4_data_lens_max	bihost_bwd_bitalker_any_biflow_eth_ipv4_data_lens_min
L4			bihost_any_bitalker_any_biflow_n_unique_dst_ports_total	bihost_any_bitalker_any_biflow_n_unique_dst_ports_mean
			bihost_any_bitalker_any_biflow_n_unique_dst_ports_std	bihost_any_bitalker_any_biflow_n_unique_dst_ports_var
			bihost_any_bitalker_any_biflow_n_unique_dst_ports_max	bihost_any_bitalker_any_biflow_n_unique_dst_ports_min
			bihost_fwd_bitalker_any_biflow_n_unique_dst_ports_total	bihost_fwd_bitalker_any_biflow_n_unique_dst_ports_mean
			bihost_fwd_bitalker_any_biflow_n_unique_dst_ports_std	bihost_fwd_bitalker_any_biflow_n_unique_dst_ports_var
			bihost_fwd_bitalker_any_biflow_n_unique_dst_ports_max	bihost_fwd_bitalker_any_biflow_n_unique_dst_ports_min
			bihost_bwd_bitalker_any_biflow_n_unique_dst_ports_total	bihost_bwd_bitalker_any_biflow_n_unique_dst_ports_mean
			bihost_bwd_bitalker_any_biflow_n_unique_dst_ports_std	bihost_bwd_bitalker_any_biflow_n_unique_dst_ports_var
			bihost_bwd_bitalker_any_biflow_n_unique_dst_ports_max	bihost_bwd_bitalker_any_biflow_n_unique_dst_ports_min
			bihost_any_bitalker_any_biflow_n_unique_src_ports_total	bihost_any_bitalker_any_biflow_n_unique_src_ports_mean
			bihost_any_bitalker_any_biflow_n_unique_src_ports_std	bihost_any_bitalker_any_biflow_n_unique_src_ports_var
			bihost_any_bitalker_any_biflow_n_unique_src_ports_max	bihost_any_bitalker_any_biflow_n_unique_src_ports_min
			bihost_fwd_bitalker_any_biflow_n_unique_src_ports_total	bihost_fwd_bitalker_any_biflow_n_unique_src_ports_mean
			bihost_fwd_bitalker_any_biflow_n_unique_src_ports_std	bihost_fwd_bitalker_any_biflow_n_unique_src_ports_var
			bihost_fwd_bitalker_any_biflow_n_unique_src_ports_max	bihost_fwd_bitalker_any_biflow_n_unique_src_ports_min
			bihost_bwd_bitalker_any_biflow_n_unique_src_ports_total	bihost_bwd_bitalker_any_biflow_n_unique_src_ports_mean
			bihost_bwd_bitalker_any_biflow_n_unique_src_ports_std	bihost_bwd_bitalker_any_biflow_n_unique_src_ports_var
			bihost_bwd_bitalker_any_biflow_n_unique_src_ports_max	bihost_bwd_bitalker_any_biflow_n_unique_src_ports_min
TCP			bihost_any_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_total	bihost_any_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_mean
			bihost_any_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_std	bihost_any_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_var
			bihost_any_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_max	bihost_any_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_min
			bihost_fwd_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_total	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_mean
			bihost_fwd_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_std	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_var
			bihost_fwd_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_max	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_min
			bihost_bwd_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_total	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_mean
			bihost_bwd_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_std	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_var
			bihost_bwd_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_max	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_requested_dropped_connection_initiations_min
			bihost_any_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_total	bihost_any_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_mean
			bihost_any_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_std	bihost_any_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_var

		bihost_any_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_max	bihost_any_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_min
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_total	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_mean
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_std	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_var
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_max	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_min
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_total	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_mean
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_std	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_var
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_max	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_two_way_handshake_initiations_min
		bihost_any_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_total	bihost_any_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_mean
		bihost_any_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_std	bihost_any_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_var
		bihost_any_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_max	bihost_any_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_min
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_total	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_mean
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_std	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_var
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_max	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_min
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_total	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_mean
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_std	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_var
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_max	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_three_way_handshake_initiations_min
		bihost_any_bitalker_eth_ipv4_tcp_biflow_connections_redropped_total	bihost_any_bitalker_eth_ipv4_tcp_biflow_connections_redropped_mean
		bihost_any_bitalker_eth_ipv4_tcp_biflow_connections_redropped_std	bihost_any_bitalker_eth_ipv4_tcp_biflow_connections_redropped_var
		bihost_any_bitalker_eth_ipv4_tcp_biflow_connections_redropped_max	bihost_any_bitalker_eth_ipv4_tcp_biflow_connections_redropped_min
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_connections_redropped_total	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_connections_redropped_mean
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_connections_redropped_std	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_connections_redropped_var
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_connections_redropped_max	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_connections_redropped_min
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_connections_redropped_total	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_connections_redropped_mean
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_connections_redropped_std	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_connections_redropped_var
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_connections_redropped_max	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_connections_redropped_min
		bihost_any_bitalker_eth_ipv4_tcp_biflow_connections_rejected_total	bihost_any_bitalker_eth_ipv4_tcp_biflow_connections_rejected_mean
		bihost_any_bitalker_eth_ipv4_tcp_biflow_connections_rejected_std	bihost_any_bitalker_eth_ipv4_tcp_biflow_connections_rejected_var
		bihost_any_bitalker_eth_ipv4_tcp_biflow_connections_rejected_max	bihost_any_bitalker_eth_ipv4_tcp_biflow_connections_rejected_min
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_connections_rejected_total	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_connections_rejected_mean
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_connections_rejected_std	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_connections_rejected_var
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_connections_rejected_max	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_connections_rejected_min
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_connections_rejected_total	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_connections_rejected_mean
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_connections_rejected_std	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_connections_rejected_var
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_connections_rejected_max	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_connections_rejected_min
		bihost_any_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_total	bihost_any_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_mean
		bihost_any_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_std	bihost_any_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_var
		bihost_any_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_max	bihost_any_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_min
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_total	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_mean
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_std	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_var
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_max	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_min
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_total	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_mean
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_std	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_var
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_max	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_half_duplex_connections_established_min
		bihost_any_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_total	bihost_any_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_mean
		bihost_any_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_std	bihost_any_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_var
		bihost_any_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_max	bihost_any_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_min
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_total	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_mean
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_std	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_var
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_max	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_min
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_total	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_mean
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_std	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_var
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_max	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_full_duplex_connections_established_min
		bihost_any_bitalker_eth_ipv4_tcp_biflow_abort_terminations_total	bihost_any_bitalker_eth_ipv4_tcp_biflow_abort_terminations_mean
		bihost_any_bitalker_eth_ipv4_tcp_biflow_abort_terminations_std	bihost_any_bitalker_eth_ipv4_tcp_biflow_abort_terminations_var
		bihost_any_bitalker_eth_ipv4_tcp_biflow_abort_terminations_max	bihost_any_bitalker_eth_ipv4_tcp_biflow_abort_terminations_min
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_abort_terminations_total	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_abort_terminations_mean
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_abort_terminations_std	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_abort_terminations_var
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_abort_terminations_max	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_abort_terminations_min
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_abort_terminations_total	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_abort_terminations_mean
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_abort_terminations_std	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_abort_terminations_var
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_abort_terminations_max	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_abort_terminations_min
		bihost_any_bitalker_eth_ipv4_tcp_biflow_null_terminations_total	bihost_any_bitalker_eth_ipv4_tcp_biflow_null_terminations_mean
		bihost_any_bitalker_eth_ipv4_tcp_biflow_null_terminations_std	bihost_any_bitalker_eth_ipv4_tcp_biflow_null_terminations_var
		bihost_any_bitalker_eth_ipv4_tcp_biflow_null_terminations_max	bihost_any_bitalker_eth_ipv4_tcp_biflow_null_terminations_min
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_null_terminations_total	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_null_terminations_mean
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_null_terminations_std	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_null_terminations_var
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_null_terminations_max	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_null_terminations_min
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_null_terminations_total	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_null_terminations_mean
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_null_terminations_std	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_null_terminations_var
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_null_terminations_max	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_null_terminations_min

		bihost_any_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_total	bihost_any_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_mean
		bihost_any_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_std	bihost_any_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_var
		bihost_any_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_max	bihost_any_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_min
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_total	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_mean
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_std	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_var
		bihost_fwd_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_max	bihost_fwd_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_min
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_total	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_mean
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_std	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_var
		bihost_bwd_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_max	bihost_bwd_bitalker_eth_ipv4_tcp_biflow_graceful_terminations_min

TABLE 18. NETGENES HOST FEATURES (WITHOUT FLOW-SET BASED FEATURES).

#Flows	Destination Port	Port to L5-7 Protocol
94203	**443**	**HTTPS**
34113	**80**	**HTTP**
1054	22	SSH
537	8313-65490	Multiple applications
476	389	LDAP
458	21	FTP
367	465	SMTP over SSL
327	88	Kerberos
218	3268	LDAP Global Catalog
181	445	SMB
108	139	NetBIOS Session Service
71	8080	HTTP alternative port
70	135	RPC
29	4502-7905	Multiple applications
4	8081	HTTP alternative port
1	53	DNS
1	843	Unidentified

TABLE 19. MONDAY: TCP BENIGN TRAFFIC OVERVIEW.

#Flows	Destination Port	Port to L5-7 Protocol
114707	**53**	**DNS**
844	443	Application Traffic (Google)
582	88	Kerberos
388	137	NetBIOS Name Service
246	389	LDAP
148	123	NTP
126	19302-63056	Multiple applications
111	1900	SSDP
66	5355	LLMNR
40	1124	HP VMM Control
40	3289	Citrix
35	3478	VoIP STUN
9	138	NetBIOS Datagram Service
8	5353	MDNS
5	7725	Nitrogen Service
5	8612	Canon BJNP Port 2
3	8610	Canon MFNP Service
1	42	WINS

TABLE 20. MONDAY: UDP BENIGN TRAFFIC OVERVIEW.

TCP Protocol										
“bitalker_fwd_biflow_n_unique_dst_ports” – Talker Feature										
V	Monday		Tuesday		Wednesday		Thursday		Friday	
	T	F	T	F	T	F	T	F	T	F
[1,5]	21542	129769	19167	106421	19647	271091	18360	91294	17538	89530
[6,10]	4	474	5	656	4	423	1	114	4	521
[11, 50]	10	1778	10	1889	6	1032	7	1116	7	1267
[51,100]	1	197	1	198	6	1312	4	844	4	882
[101, 500]	0	0	0	0	0	0	0	0	0	0
[501, +∞ [	0	0	0	0	0	0	12	74155	1	255794
bitalker_bwd_biflow_n_unique_dst_ports – Talker Feature										
[0,5]	21556	132086	19182	108975	19662	273753	18386	167590	17553	347868
[6,10]	1	132	0	0	1	105	1	313	1	126
[11, 50]	0	0	1	189	0	0	0	0	0	0
[51,100]	0	0	0	0	0	0	0	0	0	0
[101, 500]	0	0	0	0	0	0	0	0	0	0
[501, +∞ [	0	0	0	0	0	0	0	0	0	0

TABLE 21. TCP BI-TALKER “UNIQUE DESTINATION PORT COUNT” ANALYSIS.

UDP Protocol										
“bitalker_fwd_biflow_n_unique_dst_ports” – Talker Feature										
V	Monday		Tuesday		Wednesday		Thursday		Friday	
	T	F	T	F	T	F	T	F	T	F
[1,5]	636	117244	593	103246	151	96174	205	98625	191	102363
[6,10]	0	0	0	0	1	12683	10	236	0	0
[11, 50]	1	18	0	0	0	0	0	0	0	0
[51,100]	0	0	0	0	0	0	0	0	1	268
[101, 500]	1	105	1	188	1	167	1	129	1	162
[501, +∞ [	0	0	0	0	0	0	0	0	0	0
“bitalker_bwd_biflow_n_unique_dst_ports” – Talker Feature										
[0,5]	638	117367	594	103434	152	98263	216	98990	193	102793
[6,10]	0	0	0	0	1	10761	0	0	0	0
[11, 50]	0	0	0	0	0	0	0	0	0	0
[51,100]	0	0	0	0	0	0	0	0	0	0
[101, 500]	0	0	0	0	0	0	0	0	0	0
[501, +∞ [	0	0	0	0	0	0	0	0	0	0

TABLE 22. UDP BI-TALKER “UNIQUE DESTINATION PORT COUNT” ANALYSIS.