TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

FlowBreaker

Enriched Descriptions of Network Traffic Captures

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Embedded Systems

by

Christoph Clementschitsch, Bsc.
Registration Number 11771000

to the Faculty of Electrical Engineering and Information Technology

at the TU Wien

Advisor: Senior Scientist Dr.techn. Félix Iglesias Vazquez, Msc.
Assistance: Univ.Prof. Dipl.-Ing. Dr.-Ing. Tanja Zseby

Vienna, 21st May, 2025

Erklärung zur Verfassung der Arbeit

Christoph Clementschitsch, Bsc.

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct – Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In– noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, 21. Mai 2025

Acknowledgements

I would like to express my sincere gratitude to my thesis advisor, Dr. Félix Iglesias Vazquez, for his invaluable guidance and support throughout the writing of this thesis.
My appreciation extends to Prof. Tanja Zseby and the Communication Networks Group, whose lectures and research inspired me to write my thesis in the field of IP networks and Cybersecurity.
I am deeply grateful to my parents for their support and encouragement in continuing and completing my studies in pursuit of my engineering degree.
A special thanks to my partner, Katrin Drianovska, for her continuous support during the thesis writing process.
Lastly, I wish to acknowledge my friends and colleagues at TU Wien, who have enriched my university experience, making it much more enjoyable.

Kurzfassung

IP-basierte Netzwerke werden verwendet, um Computersysteme weltweit zu verbinden, wobei enorme Datenmengen generiert werden. Die Analyse dieser Daten ist entscheidend für das Verständnis der Funktionsweise von Netzwerken und somit ein zentraler Aspekt der Cybersicherheit und des Netzwerkengineerings, welche es Forschern ermöglicht, Netzwerk-Intrusion-Detection (NID) durchzuführen, Netzwerkbeschränkungen zu verstehen und zukünftige Erweiterungen des Netzwerks zu planen. Verschiedene Werkzeuge wie z.B.: Zeek, Snort und Wireshark, wurden zu diesem Zwecke entwickelt. Diese Werkzeuge werden auch zum Labelling von IP-Traffic verwendet: entweder durch Aggregation in Flows oder durch Labelling einzelner Pakete. Daraus resultierende Labels entbehren jedoch oft einer angemessenen Begründung und liefern keine aussagekräftigen Einblicke, um tiefergehende Phänomene und Aspekte von IP-Traffic zu verstehen.
Daher ist das Hauptziel dieser Arbeit, bestehende Labels von NID-Systemen und Datensätzen zu analysieren und zu untersuchen, wie sie verbessert werden können. Zu diesem Zweck wurde FlowBreaker, ein neues Werkzeug zur Beschreibung von IP-Traffic, das auf Zeek aufbaut, entworfen und entwickelt. Die Benutzerfreundlichkeit und Performance von FlowBreaker wurden anhand des TII-SSRC-23 Benchmark-Datensatzes und eines echten Traffic-Samples aus dem MAWI-WIDE-Projekts evaluiert.
Die Ergebnisse dieser Evaluation zeigen, dass FlowBreaker in der Lage ist, die Labels des TII-SSRC-23-Datensatzes zu replizieren und dabei zusätzliche Informationen, Begründungen und Aspekte für ein tiefergehendes Verständnis der Labels zu bieten. Darüber hinaus bietet dieses Tool ein neues Framework, das benutzt werden kann um tiefergehende wissenschaftliche Arbeiten und Experimente durchzuführen.

Abstract

IP-based networks are used to connect computer systems across the globe, generating enormous volumes of data in the process. Analysing this data is vital to understanding the inner workings of networks and thus a central aspect of cybersecurity and network engineering.
This allows researchers to perform Network Intrusion Detection (NID), understand network constraints, and plan for future expansions. Different tools—such as Zeek, Snort and Wireshark—exist for this purpose. These tools are also used to label traffic: either by aggregating it into flows, or by labeling individual packets. However, such labels often lack a proper justification and fail to provide meaningful insight to deeply understand traffic phenomena and situations.
Therefore, the main objective of this thesis is to analyse existing labels of NID systems and datasets and explore how they can be improved. For this purpose, FlowBreaker—a new tool for describing traffic built upon Zeek—was designed and developed. FlowBreaker usability and performance are evaluated using the TII-SSRC-23 benchmark dataset and a real traffic sample from the the MAWI WIDE project collection.
Evaluation results show how FlowBreaker is able to replicate the labels of the TII-SSRC-23 dataset while providing enriched information, justification, and keys for the deep understanding of traffic captures. On the other hand, it provides a highly useful description of real-life data for further analysis. Furthermore, the new tool significantly improves upon user experience when compared to pre-existing solutions, thus providing a very suitable framework for scientific research and experimentation.

Contents

CHAPTER 1

Introduction

Modern computer systems are often connected to the internet. While this allows access to other networks and makes global communication possible, it also creates an attack surface for malicious actors. This raises the question of how to prevent intrusions and attacks by these actors. For this, two conditions have to be met: The communication must be observable, and indicators to discriminate benign from malicious connections must be present. This can either be achieved on the fly by analysing every packet[1] as it is received, or by capturing all packets during a given time frame and analysing them later. In this thesis we focus on the second case, also known as forensic or offline analysis. In addition to attack detection and prevention, analysis of network traffic is also necessary to detect bottlenecks and the influence of certain services on network usage. This is especially vital when building or expanding a network architecture. This thesis revolves around how traffic can be analysed, focusing on how certain parameters of network traffic can be used for threat detection.

1.1 Background - IP Traffic Analysis

The internet is built on the Open Systems Interconnection (OSI) model, which consists of seven layers, with the first layer addressing signalling across a physical medium, e.g., wires, optical fibers or 5G, and the seventh layer addressing the (end user) application, e.g., a web browser. There are several protocols involved in the OSI model, this thesis mainly addresses the 3rd and 4th layer, specifically the IP stack and the Internet Control Message Protocol (ICMP), User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) protocols that are built upon it. Higher level protocols are addressed when applicable.
With IP traffic being split into packets, a single packet does not provide much information

[1] A *packet* is the smallest unit of data exchanged between two machines in a communication network.

by itself. When grouping these packets together, however, it becomes possible to shed more light on the connection's contents, purpose and intention. These grouped packets form a flow, which can be labelled based on several characteristics, such as: duration, origin IP:Port, destination IP:Port, protocol used and service used. If these flows are then analysed in a statistical/heuristic manner based on their individual parameters, connections and their associated hosts can be further classified as benign or malicious. Significant effort has already been put in the labelling and analysis of network traffic. One of the main approaches is Flow-based Intrusion Detection. Instead of performing detection based on individual packets and their signatures (comparing them to known malicious contents), packets are aggregated into flows, and classification is performed based on higher level parameters. This has the benefit of providing more overview than individual packet analysis, while saving resources(6).
However, capturing large amounts of packets is difficult and can lead to data loss, as buffers in capturing devices are not infinite. To perform flow based analysis, first aggregation has to take place, which can overwhelm the resources of the hardware used. This makes it necessary to investigate how packets can be aggregated directly, ideally without performing computation heavy post-analysis (7).

To analyse network traffic, several tools exist, in this thesis the following ones are used: Wireshark is an open source tool for packet analysis, featuring a Graphical User Interface (GUI). It offers options to manually filter and search for packets. Snort employs a rule-based approach to filter packets and is more aimed at detecting and preventing network intrusion based on pre-defined rule sets. Suricata is an open-source network intrusion detection and prevention system, similar to Snort. Although it supports outputting traffic captures, it is more focused on monitoring entire networks. Zeek is an open source traffic analysis tool, which is also able to operate on traffic captures. It is modular, very versatile and offers detailed output. Zeek serves as the main tool used in this thesis. These analysis tools are complicated to set up and often provide complex and unsynthesised reports that are difficult for users to interpret. Our thesis focuses on how to leverage the information provided by these tools to generate high-value traffic labelling. In contrast to the common approach of labelling flows individually, the approach in this work is to describe network traffic from a host perspective (either by source or destination host).

1.2 Motivation - Why build something new?

Systems based on the Internet Protocol (IP) protocol are used in almost every field of research and daily life. This makes them a valuable target for attackers looking to extract information, steal resources, or compromise the entire system. Thus, Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) are necessary to prevent these attacks or, at least, perform damage control. However, attack detection is not the only point of interest in research in this field. With the internet serving as the de facto standard for consuming news and media in the modern age, a lot of thought has to be put into designing networks. To achieve this, researchers must be able to obtain an overview

of the traffic contained within. While achieving this overview is certainly possible with existing tools and solutions, the process is far from straightforward, often making their use frustrating. The excess of quantitative information or the deficiency of qualitative information obscures the real nature of the phenomena occurring in communications and makes it difficult to understand them. The consequence is that a high amount of pre-existing knowledge on network and OSI related protocols is required to work with the output of the aforementioned solutions. This also affects experts and researchers, and may even promote irrelevant, erratic, ineffective and inefficient practices. The reason that these problems have not been adequately addressed is, among others, that researchers working with network traffic are usually familiar and experienced with specialised tools. However, the diversification of technical and scientific work and its spread across various technological areas implies that, for example, the specialist in data analysis is often not an expert in network traffic and vice versa. The current technical situation requires lowering the barriers to entry and offering greater transparency on such technical aspects to a wider (also technical) public. Thus, initiatives that can greatly promote research in network analysis include:

1. Research on how existing solutions implement labelling and how existing measurements can be used to create more informative labels.
2. Explorations on how a simpler and more user-friendly solution for summarising network traffic can be provided.

The intention of these objectives is to help create a framework that can be used by a broader range of users while also leading to deeper and higher quality research.

1.3 Research Questions and Goals

With the scenario stated above in mind, the research questions can be set accordingly:

1. How can we label network traffic so that the information provided offers high quality descriptive knowledge, particularly in relation to network attacks?
2. How can we use existing and established tools in the Network Intrusion Detection System (NIDS) field to obtain qualitatively enriched traffic-labels?
3. To what extent do qualitatively enriched traffic-labels favour post-analysis and a deeper evaluation of classification and detection algorithms?

In order to answer these questions, three goals were set to be achieved over the course of the thesis:

1. To establish a set of minimum criteria for the summarised and qualitative description of network traffic pieces or flows. These should be suitable for the construction of NIDS dataset benchmarks, in a way that it favours the creation of meta-data, pre-analysis and post-analysis.
2. To develop a tool capable of integrating an existing NIDS (in our case, Zeek) to obtain high quality descriptive analysis summaries and enriched traffic-labels.
3. To show to what extent the availability of enriched traffic-labels enables a more discriminating and precise traffic classification from a statistical approach.

1.4 Methodology

The research questions and goals described above are addressed in this thesis according to the following methodology:

1. Exploration of the state of the art, existing methods and tools for the description and identification of traffic classes in the NIDS field.
2. Study of the type of labels provided by popular NIDS datasets benchmarks.
3. Design and development of a tool (FlowBreaker) that wraps Zeek to generate descriptive analysis summaries and enriched traffic-labels. This tool is designed under requirements of clarity, modularity, compatibility, ease of use and extensibility, and works in offline forensic conditions with captures in Packet Capture (PCAP) and Packet Capture Next Generation (PCAPNG) formats.
4. Design and development of a test environment that enables the evaluation of the discriminating capacity of traffic labelling from a statistical approach.
5. Evaluation of FlowBreaker with NIDS dataset benchmarks (TII-SSRC-23) and real-life captures from backbone networks (MAWI).

To summarise, the intent is to first explore existing ways of traffic analysis, and then improve upon them. This is achieved by selecting existing tools for labelling traffic, namely Zeek and using them as a basis for a new tool. This new tool (FlowBreaker) is then tested with existing NIDS datasets - followed by an analysis of the results.

1.5 Terminology

In this work, the following terms are used in the context of network traffic:

- **Flow** A flow is defined as a set of IP packets that have the same common properties (32). A prominent example is the five-tuple consisting of source IP, destination IP, source port, destination port and protocol.
- **Connection** In this work the term connection is used to describe flows. The term defines a custom class, which encapsulates several identifying features — most notably the five commonly used to describe flows. The term originates from Zeek's "conn.log", which is used to list connections, as described in Section 3.3.3.
- **Labelling** In this work, we consider that "labelling" implies any kind of qualitative description of a packet, flow or aggregated piece of traffic that can serve as a basis for further classification and identification. For instance, a label may describe whether a flow is benign or malicious, whether it belongs to a specific class of attack, or to an attack variant with particularities whose notification is relevant for post-analysis, monitoring or the characterization/profiling by experts.

CHAPTER 2

Related Work

2.1 Discussing Labelled Datasets in NIDS

2.1.1 An Introduction to Traffic Labelling

In data-oriented fields, the process of Labelling consists of adding relevant information to a data object, with the goal of providing a way to identify this object, or put it into context using said information (1). When it comes to Traffic Labelling, these data objects represent network traffic (e.g. packets or packets aggregated into flows). This labelling process can be broken down into a classification problem, namely differentiating normal traffic from malicious activities (2).
Labelling this data is also a process fundamentally different from (real-time) intrusion detection: Since there is no live system/network in need of protection, labelling is not time critical and misclassifications do not have such a damaging impact (i.e. runtimes are allowed to be arbitrary and an undetected threat does not cause immediate harm), although labelling is expected to be as accurate as possible to ensure its usefulness. Furthermore, the scope is different, with real-time detection classifying all traffic in a network (to reliably detect any threat), while labelling can also focus on a subset of traffic, e.g. to create a dataset for a specific type of attack (2).
There are several approaches to labelling network traffic, which can be broadly split up into two categories: *Human-Guided Labelling* and *Automatic Labelling*. Table 2.1 serves as a non exhaustive overview of these two categories and their subcategories. As the name implies, *Human-Guided Labelling* describes techniques in which the traffic is labelled under the supervision of a human. This can be further split into *Manual Labelling*, in which a researcher assigns labels to traffic traces by hand and *Human-Assisted Labelling*, in which parts of the labelling processed are performed by Machine Learning (ML).
A key aspect in *Human-Guided* is that the human does not control the network environment and thus cannot artificially generate or inject traffic (8). Since *Manual Labelling* does not scale, most efforts concentrate on either simplifying the task by providing

additional tools, or by introducing ML into the loop.
In *Human-Assisted Labelling*, we will distinguish between two main approaches: *Machine Learning* and *Active Learning*. ML by itself, can be used to label datasets but is highly sensitive to the training data used. The two main approaches consist of supervised- and unsupervised training, which simply distinguish whether the ML system is working with a dataset that has labels assigned to it or if it creates its own labels during the training process by finding and recognising patterns. The main downside of ML is the need of relevant, high quality training data, which is hard to find in the NIDS field, with details of this problem discussed in Section 2.1.2. Note that this is an oversimplification of ML, as ML by itself is outside the scope of this thesis.
"Active Learning" (AL), works by using a ML system to assign labels to unlabelled data. Whenever no clear distinction is possible, e.g. in edge cases, the system can query an "oracle", which knows the correct label (from the perspective of the ML system). In this case, the oracle is a human researcher, tasked with deciding which label to assign to the data points. These labelled data points are then reintroduced into the ML system to enhance its accuracy in labelling data. The ML system is then run again on new unlabelled data, and the cycle continues (9).
This approach has been extended into *Smart Labelling*, in which AL is combined with visual interactive labelling. This approach utilises a visual interface that presents statistical data through box plots, scatter plots, and other graphical representations. Additionally, it suggests the next potential candidate for labelling, which is most likely best suited to improve the model's performance. The key idea is to integrate the human ability of pattern-recognition into the AL approach, by allowing the researcher to directly chose relevant data points/instances (10).
The counterpart to *Human-Guided Labelling*, is *Automatic Labelling*. The key difference to the former is that the researcher now has complete control over the environment, with the possibility to inject and modify the traffic, as well as the network itself. Two of the main approaches are *Injection Timing* and *Behavioural Profiles*. As the name suggests, *Injection Timing* is based upon the idea that the traffic type is observable at any given timeslot. By leveraging this, malware traffic can be introduced at a given timeslot t_1 and terminated at 2nd timeslot t_2, with the resulting time window labelled as malicious in the resulting dataset (3).
The *Behavioural Profiles* approach consists of breaking down a network into smaller components (profiles), which are usually based on mathematical distributions — e.g. how much traffic is generated at a given time frame. These profiles are agnostic to the network they are used in and can thus be placed in different topologies. The labelling process in itself is very straightforward: Assume that a specific profile N generates normal traffic and another profile M generates malicious traffic. If N is active, the resulting traffic is labelled as "Normal", if M is active, as "Malicious" (4).
To summarise, human input is still necessary for many approaches and new approaches tend to focus on reducing the human effort necessary — either by creating automated systems or by providing new tools for researchers to use. Table 2.1 offers an overview of the main labelling approaches for Network Traffic.

Method	Sub-method	Description
Human-Guided Labelling	• Manual Labelling	Labels assigned manually by researcher
	• Human-Assisted	
	◦ Tool supported	Labels assigned by researcher, using (graphical) statistical tools
	◦ Active Learning	Labels assigned by machine learning, refined by researcher input
	◦ Machine Learning	
	▷ Supervised	Training data is labelled
	▷ Unsupervised	Training data is unlabelled
Automatic Labelling	• Injection Timing	Malicious traffic injected at specific timestamps
	• Behavioral Profiles	Network elements simplified into behavioral profiles (benign/malicious)

Table 2.1: Network Traffic Labelling Methods — A short summary of the main approaches/methods and their subcategories

2.1.2 Quality of NIDS Datasets

As discussed in Section 2.1.1, there are various methods for producing labelled datasets. However, the quality of these datasets greatly depends on the method used and is often unsatisfactory. With datasets being used as a benchmark or to train and develop new detection systems, their quality directly impacts the end result. The quality of a dataset is hard to quantify, but can be measured using several factors, including: Data Diversity, Dependency of Features, Correctly assigned Labels, Missing/Duplicate Labels and Bias. Furthermore, NIDS datasets suffer from their high specialization — a given traffic capture never contains an exhaustive list of all possible network and traffic configurations. This means that choices made during the creation of a given dataset heavily impact its representation in different scenarios. In addition, many of the most popular datasets used for NIDS are created under synthetic conditions (5). While this is not a problem in itself, it can introduce bias and also diverge from a real life scenario, thus not accurately portraying the attack (12). This is especially true when synthetic intrusion datasets are used to model production networks since they simply fail to capture the complexity and uncertainty of these larger networks. (5)
While the inapplicability to real-life systems is an inherent problem of synthetic datasets, popular datasets also suffer from other problems. One of the most notable problems in existing datasets is the poor data quality, coupled with the unclear composition — i.e. datasets contain duplicate data, some events appear too frequently or too scarcely and the samples are unrepresentative. Other issues include poor provenance and over-summarisation, which means that it is unclear how the dataset was created and features can be omitted/missing — this makes it hard to assess the origin data and its applicability (11). These issues are often prominent in existing public datasets used for NIDS (12).

The lacking quality of existing datasets raises the question of why these issues have not been addressed by the NIDS community. The answer is a combination of various difficulties inherent to the field. As already mentioned, many datasets are synthetic — the reason why little real data is available boils down to the criticality of these systems. Enterprise systems are subject to privacy laws and restrictions, making it difficult to impossible to simply capture uncensored traffic within these networks. Furthermore, threats constantly change and datasets quickly become outdated (11) (12). Combine this with the need of manual labelling and it becomes apparent that creating a representative, high quality and relevant dataset poses a great challenge.

2.1.3 Future Directions in NIDS

With these problems regarding quality and methodology in mind, we can now turn to the future prospects in the NIDS field. It is evident that the general quality of datasets is low and this has a variety of consequences on research performed with/on them.
As already mentioned in Section 2.1.1, the human element is crucial for obtaining high quality labels. Since this involves a high amount of labour, the general trend is to reduce effort and introduce automation. This is either done by leveraging Machine Learning, which itself depends on quality datasets, or by providing new tools to aid researchers in portraying traffic (9). Other approaches try to bypass the inherent difficulties associated with using pre-existing traffic captures by providing the user with the possibility to create personalised datasets by utilising existing traffic captures. One example of this provides a framework for generating NIDS datasets, that works by taking real traces, sanitising them (thus removing identifiable information) and mixing them with real malware traffic. This traffic is then fed into another tool which takes the generated output and provides a feature list and labels to be used in an ML approach (13). Another approach also relies on the user to provide background traffic and subsequently merge malicious attack traffic into said background traffic. The way traffic is injected can be customised by the user through the use of an Application Programming Interface (API), instead of just blindly injecting it, i.e. taking a step further than simple *Injection Timing* (14). The intention here is to solve the problem of outdated, irrelevant and out of scope datasets, so the user can create their own NIDS dataset tailored to a specific use case.
With these tools at least partially relying on existing traffic captures, the analysis, labelling and description process of these captures will remain relevant in the foreseeable future. This also leads into the next topic, which discusses how traffic is captured and analysed, as well as NIDS and Network Intrusion Prevention System (NIPS) and how they operate to detect and prevent threats.

2.2 Working with Network Traffic

Network traffic can either be live monitored or analysed by extracting it from traffic captures. Traffic analysis/monitoring is necessary for a number of reasons: Either to detect anomalies, e.g. attacks taking place, or to simply monitor network activity; for example, to impose limitations on certain hosts. Furthermore, analysing traffic is vital for understanding network constraints and planning future expansions of the network. Common tools for capturing traffic are: tcpdump, Wireshark, tshark or SmartSniff. Whereas well-known tools with traffic monitoring and logging capabilities are: Zeek, Suricata, Snort, ntop/ngrep. In cloud and distributed traffic capture settings we find approaches like AWS VPC Traffic Mirroring, Google Cloud Packet Mirroring or Microsoft Azure Network Watcher. But cloud environments go far beyond the scope of this thesis. This section is intended to give an overview of the different methods used for capturing and analysing traffic, either by live surveillance or post-processing. Furthermore, different tools for traffic analysis are discussed.

2.2.1 Capturing Network Traffic

In order to work with network traffic, it must first be captured using appropriate software. This software, e.g. Wireshark, tcpdump, then monitors a network interface, e.g. the Local Area Network (LAN) port of the PC it is running on, and logs every IP packet it sees on this interface. There are two disadvantages however. Running locally, it is only possible to log traffic routed from or to the local host. Additionally, most modern protocols use encryption, meaning that the payload is end-to-end encrypted using Secure Sockets Layer (SSL)[1]. Without the encryption key, it is not possible to read the actual data being transmitted. The first problem can be solved by monitoring traffic at a central spot, e.g. a router. Furthermore, more sophisticated software such as Suricata, Snort or Zeek can be used for network wide logging. The second problem cannot be solved (at least not without breaching a lot of security), but it is often also not necessary to be able to decrypt the actual payloads. The reason is that IP packet headers (also headers of protocols in the transport layer, e.g. TCP, UDP, ICMP) usually contain enough information to understand the purpose of the traffic. This is why we concentrate on interpreting such unencrypted data together with aggregated statistics. The most popular way to store traffic captures at packet-level is through PCAP and PCAPNG format files (Section 2.2.2), which is a natural format for most traffic analysis tools, particularly if we refer to forensic or offline analysis (Section 2.2)

2.2.2 PCAP & PCAPNG files

PCAP “Packet Capture” files make use of the pcap API, which itself consists of the libpcap project (28), that provides a framework for low-level network monitoring. This framework is then used by higher-level applications, such as Wireshark, to capture network traffic. After doing so, these applications output a pcap file, containing the raw

[1]SSL is a protocol for encrypting traffic and ensuring data integrity between hosts.

packet data, as well as additional logging information. The original pcap format was later extended into PCAPNG, with NG standing for "Next Generation", providing additional fields such as user comments, network card information etc.(29). For all intents and purposes, PCAP and PCAPNG are used interchangeably in this thesis. With PCAPs being binary data and not human-readable, they first need to be processed before working on them (30).

2.2.3 Network Intrusion Detection

There are several different fields of traffic analysis: IDS, as the name implies, serves to monitor a network and raise an alert when it detects an intrusion. Apart from logging and passively monitoring network traffic, it does not interact with the network. An IPS goes a step further and actively filters traffic on the network. The goal here is to perform the same goal as an IDS but act on a raised alarm directly to prevent damage. Finally, there is post analysis, which works on pcap files or flow records. Especially when using online NIDS, it is common not to work with pcaps due to their large size but rather directly capture traffic as flows using flow records. Tools that capture flows instead of all packets are: sFlow, Netflow and IPFIX. They operate by either sampling individual packets and using them as representation for the entire flow, thus reducing bandwith (sFlow)[2], or by aggregating packets that share the same key details (source, destination, ports, protocol, etc.) and saving them as a flow after the connection has been terminated (Netflow/IPFIX)[3]. To reduce complexity, this thesis focuses exclusively on traffic captured in pcaps.

2.3 Tools for Capturing and Describing Network Traffic

There are several tools for capturing and analysing network traffic. This section addresses a selection of the available solutions and introduces how one of them (Zeek) is used as a starting point for developing a new tool.

2.3.1 Wireshark

Wireshark (34) is one of the most popular, if not the most popular, open-source network packet analyser. It comes with a GUI and is cross-platform compatible. It can work on existing pcaps or live capture traffic directly and export it to a pcap. It is feature-rich and provides filtering options as well as visual representations of those filters. Wireshark offers the ability to quickly figure out what is going on in a network or in a traffic capture, but does not provide intrusion detection or any alarms in that regard. It is rather suited for short manual searches and less for automatisation. Furthermore, Wireshark, upon opening a pcap file, reads its entire contents into memory, meaning that for large pcaps it becomes unresponsive and prone to freezing on normal machines.

[2] *sFlow* was introduced in RFC 3176 (31).

[3] *Netflow* was introduced by Cisco, later superseded by IPFIX in RFC 5101 (32) and included in the IETF standard track.

2.3.2 Snort

Snort (35) is an open-source network intrusion detection system. It works by using a series of pre-defined rules describing malicious network traffic and matches packets in the network to those rules. This makes it suitable as an IDS, or as an IPS, with Snort being deployed inline to stop detected packets. While Snort is open source, it offers a "Snort Subscriber Ruleset" provided to customers of Cisco, in addition to the "Community Ruleset" which is maintained by the Snort community - with the former being more sophisticated. The intrusion detection mode of Snort is highly configurable and adaptable. In addition to this main feature, Snort also offers a sniffer mode which simply reads packets off the network and displays them to the console and a logger mode, which writes captured packets to disk. Like Wireshark, Snort is able to work with pcaps.

2.3.3 Suricata

Suricata (39) is an open-source network intrusion detection and prevention system, similar to Snort. It is mainly targeted at monitoring an entire network, but also supports outputting pcap files, either on condition or capturing all traffic. Its detection system works on inspection by signature, rulesets, Indicator of Compromise (IoC) matching (e.g. domains, fingerprints, hashes) and Lua[4] scripts. It is thus also highly customisable but requires a significant amount of effort to be configured properly.

2.3.4 Zeek

Zeek (40), formerly known as Bro, is a passive open-source traffic analyser. It can be operated from the command line and provides its own Turing-complete scripting language, offering a framework for working with network traffic. It is highly customisable and able to directly work with pcaps. By writing custom scripts, it is also possible to directly control how the tool processes and monitors traffic. When analysing a pcap, Zeek generates detailed reports containing every connection it processed. These logs can then be easily ported to other tools or further analysed manually. In addition to its scripting language, Zeek is also modular, meaning that future support for new protocols can be patched in and the analysis of existing protocols can be fine-tuned and modified. Finally, Zeek is also very resource-efficient and able to run on basic hardware.

2.4 Zeek as a Basis for Informative Labelling

Pcap files are encoded in binary and thus are not easy to parse into other formats. Since they usually capture all packets present on a network interface, it is necessary to preprocess them, before being able to parse individual packet information. However, this is only one step before analysis can take place. The packets have to be aggregated into flows first, as individual packet inspection is very time- and resource consuming.

[4] *Lua* (33) is an an embeddable scripting language

Therefore, an existing program needs to be chosen for performing the task of preprocessing pcaps and outputting the data in a structured manner. Afterwards, the now structured data can be analysed and summarised into a final result.

2.4.1 Why Zeek

Out of the options above, the one that proved to be the most viable was Zeek. The reason for this is that Zeek already implements a very detailed logging system and just needs to be configured on what to log. This way it is possible to process a pcap through Zeek by just providing a startup script listing all the modules that Zeek has to execute. After finishing its analysis, Zeek then outputs a well structured "conn.log", listing all connections made and further .log files for each submodule, e.g. "dns.log". This improves modularity, making it possible to easily change direction in terms of scope, without having to rebuild everything from scratch. When looking at the other options, Wireshark does not implement such an output system, while Snort and Suricata are more focused on providing logs on potential threats that they have already identified.

2.4.2 The Benefits of Zeek

One upside of Zeek is its ease of use compared to Snort and Suricata. Being a Command Line Interface (CLI) utility, apart from a startup script, it does not take much to provide an output to a given pcap. However, the biggest advantage is its comprehensive and easy to follow documentation (41). It describes how Zeek's scripting language works, how the output files are structured, what each entry means and how to interpret it. Furthermore, Zeek is fully open source, without any business license, it instead opts for an approach of complete accessibility. This means that it will remain usable in the future, without being impacted by potential business decisions.

2.4.3 The Drawbacks of Zeek

Zeek is built for Unix — while this makes sense for its use case, it is very hard to have it run on other operating systems. This means that, in order to process pcap files, it is best to do so on a Linux machine and then either proceed on this machine or port the results to the operating system on which one wishes to perform further analysis. Another drawback is that the high configurability and scripting may overwhelm users, and no GUI is provided. These issues can be somewhat mitigated by building a wrapper around Zeek with Docker and providing a simple-to-use frontend for users.

2.5 Existing Ways of Traffic Classification

To understand the current state of traffic labels and how they can be interpreted, it is necessary to highlight the different output formats of the solutions presented above. This section concentrates on providing a short overview, with more detail added in the chapters Methodology and Evaluation.

2.5.1 Zeek

As mentioned above, Zeek gives a very detailed account of network traffic by grouping packets into flows and then displaying these flows in a formatted manner. If Zeek is configured to detect anomalies, they are saved in the "notice.log". Listing 2.1 shows a sample entry, which includes a timestamp, a reason why this notification was raised, followed by the source IP causing the anomaly and the actions taken (in this case a notice was written). Moreover a time frame is defined, preventing this notice from being raised again within that frame. Additionally, an email address for notification of the event can be specified.

```
{"ts":"2025-01-01T05:00:00.475398Z",
"note":"Traceroute::Detected",
"msg":"64.88.110.57 seems to be running
       traceroute using icmp",
"src":"64.88.110.57",
"actions":["Notice::ACTION_LOG"],
"email_dest":[],
"suppress_for":3600.0}
```

Listing 2.1: Zeek Example Output

The details of how Zeek's system is used, with a more in depth explanation, can be found in Section 3.2 In general, Zeek is very flexible in its output but the flows it generates still need to be post-processed to provide an overview.

2.5.2 Snort

Similar to Zeek, Snort also provides an alert system. The difference is that Zeek is more targeted at analysis, while Snort is equipped to intercept packets in real time. This is apparent in Listing 2.2(60), in which Snort detected an arbitrary code execution[5] attempt.

```
{ "timestamp" : "07/16-09:23:39.153899",
"pkt_num" : 5,
"proto" : "TCP",
"pkt_gen" : "stream_tcp",
"pkt_len" : 97,
"dir" : "C2S",
"src_ap" : "192.168.1.2:50284",
"dst_ap" : "192.168.2.3:80",
"rule" : "1:1000000:0",
"action" : "would_drop",
"msg" : "SERVER-WEBAPP Apache Log4j arbitrary
```

[5]This example corresponds to an attack form where a hacker executes his own malicious code/program on a target.

```
                code execution attempt",
"class" : "Attempted User Privilege Gain" }
```

Listing 2.2: Snort Example Output

When compared to Zeek, this alert includes more details, such as the protocol used, packet length, etc. This would appear in Zeek's "conn.log" and can then be cross referenced if needed. This example gives some insight into how Snort operates. While it can be used for traffic classification, it is rather aimed at preventing real time threats as can be seen in the attack above.

2.5.3 Suricata

Suricata provides IDS and IPS services and is comparable to Snort. Listing 2.3 demonstrates an example alert which was raised based on packet signature (61).

```
[1:2100498:7]
GPL ATTACK_RESPONSE id check returned root [**]
[Classification: Potentially Bad Traffic]
[Priority: 2]
{TCP}
217.160.0.187:80 -> 10.0.0.23:41618
```

Listing 2.3: Example Alert raised by Suricata

Again, it is apparent that a connection was made, with the source and destination IP:Port tuples provided. The protocol used is TCP and a reason for the classification is also given. Like Snort, Suricata can be configured to perform actions based on rules, for example dropping the packet, sending a notification, or raising an alert. The problem with this type of classification is again the focus on individual packets and real time analysis. This makes Suricata well suited for threat protection but less suited for analysis of traffic captures.

2.5.4 Wireshark

Unlike the other utilities, Wireshark is used for inspecting and analysing packets directly. In Figure 2.1 the Source IP 192.168.0.118 establishes a connection to google.at, which resolves to 142.251.39.57. The protocol used is Quick UDP Internet Connections (QUIC), which is establishing itself as a replacement for TCP in some instances and in this case, is simply used to load the Google front page. Some information about the purpose and content of the packet is displayed by Wireshark but, with QUIC being encrypted, only "Protected Payload" is displayed. This makes Wireshark a good choice for understanding individual packets and connections, but not suited for larger scale flow analysis and labelling.

No.	Time	Source	Destination	Protocol	Lengt	Info
71	1.552951	142.251.39.67	192.168.0.118	QUIC	1292	Initial, SCID=e6ac0a1073b6b902, PKN: 5, CRYPTO, PADDING
72	1.554161	142.251.39.67	192.168.0.118	QUIC	347	Protected Payload (KP0)
73	1.554161	142.251.39.67	192.168.0.118	QUIC	982	Protected Payload (KP0)
74	1.554161	142.251.39.67	192.168.0.118	QUIC	67	Protected Payload (KP0)
75	1.554450	192.168.0.118	142.251.39.67	QUIC	1292	Handshake, DCID=e6ac0a1073b6b902
76	1.554517	192.168.0.118	142.251.39.67	QUIC	73	Protected Payload (KP0), DCID=e6ac0a1073b6b902

Figure 2.1: Example of Wireshark's User Interface

2.5.5 TII-SSRC-23 Dataset

Used for testing purposes, the TII-SSRC-23 dataset contains several pcaps with sample traffic of attacks or benign traffic and a csv file which contains detailed information on the flows within the pcap files. The dataset provides a total of 85 labels in the csv file, out of which the labels in Listing 2.4 have been selected for demonstration purposes:

```
{ "Flow ID" : "192.168.1.42-192.168.1.220-43808-53-17",
"Src IP" : "192.168.1.42",
"Src Port" : 43808.0,
"Dst IP" : "192.168.1.220",
"Dst Port" : 53,
"Protocol" : 17.0,
"Timestamp" : "07/02/2023 04:04:00 PM",
"Flow Duration" : 5005274.0,
"Label" : "Malicious",
"Traffic Type" : "Mirai",
"Traffic Subtype" : "Mirai DDoS ACK"}
```

Listing 2.4: Labels selected from TII-SSRC-23 Dataset for demonstration purposes

The flow above is taken from the "Mirai_DDoS_ACK" pcap, which contains a dedicated denial of service attack using ACK statements in the TCP protocol. The fields provided are similar to those of Zeek and Snort. The source and destination IP:Port tuples are provided, along with a timestamp, flow duration and flow id (to uniquely label this flow). What is more interesting is the classification provided by the dataset: the label states "Malicious", indicating an attack. However the traffic type is simply set to "Mirai", referring to the Mirai botnet[6] and the traffic subtype reads "'Mirai DDoS ACK". These fields leave room for improvement. For instance, the traffic type is redundant when looking at the traffic subtype. Furthermore, no specifics about the attack are provided, meaning that all flows contained within the dataset would have to be processed again to gain more insight into the number of requests or which IPs took part in the attack.
In conclusion, the labels provided by existing tools and methods are either too detailed, or not detailed enough, requiring post-processing. In the next chapter, the state of the art is analysed in more detail, discussing how the existing forms of traffic analysis can be improved and extended upon.

[6]The Mirai botnet consists of a network of infected Internet of Things (IoT) devices used to perform DDoS attacks (45).

CHAPTER 3

Methodology

This chapter describes the working methodology of this thesis. We begin by contextualizing the traffic analysis and the Network Intrusion Detection Systems explored. We then expose and discuss the development of a tool (FlowBreaker) built on Zeek for performing Network Intrusion Detection and traffic analysis. Later on, the developed test setup for evaluating this tool is presented based on the datasets used for the experiments, i.e. the TII-SSRC-23 NIDS dataset benchmark and its default labelling system and a capture of the real-world capture of MAWI WIDE project.

3.1 Explored NIDS

We narrowed down the selection to the following three tools:

- Snort
- Wireshark
- Zeek

3.1.1 Snort & Rule based Detection

Snort is an intrusion detection system able to operate on both pcap files and live traffic. Its main feature is its rule based detection system, which labels packets based on criteria defined in a ruleset. To clarify how this is done in practice, here is an example rule from the Snort Wiki (36):

```
alert tcp $EXTERNAL_NET 80 -> $HOME_NET any
```

This rule tells Snort to send an alert if a TCP connection from the "EXTERNAL_NET" on source port 80 to any port in the "HOME_NET" is established. Snort can not only send an alert but also perform other actions, such as blocking the connection, dropping the packet, or sending a response to the source, based on the rule provided. This means that Snort allows for a very fine grade approach that can be tailored to any network and its threat model. There are two pre-existing rulesets: one provided by the Snort community and one provided for a subscription fee to customers of Cisco. While Snort is an excellent choice for monitoring and protecting a live network, it is rather unfit for labelling/processing pcap files, as this would require creating a very extensive and detailed set of rules. Furthermore, statistical analysis would require additional overhead based on the results provided by Snort after applying the aforementioned ruleset.

3.1.2 Wireshark

Unlike Snort, Wireshark is not directly used for NID, but rather as an open source packet inspection tool. It focuses on describing packets and their contents and, while being able to aggregate packets into flows, flow labelling is not its main focus. Wireshark offers an user friendly GUI and optionally a command line utility called TShark for automation purposes (37). This makes Wireshark a suitable candidate for in depth manual analysis, using its assortment of functions to filter and search for packets. It is rather unfit however for the task of filtering aggregated flows/connections on a larger scale, as this would require us to construct a wrapper that first aggregates all packets into suitable flows, before any labelling can take place.

3.1.3 Zeek

While Wireshark focuses on the contents of packets, Zeek directly abstracts this layer and instead aggregates all packets into packet flows, which directly describe a connection from one IP and Port tuple to another. Using these tuples, Zeek generates an unique identifier for each flow. Additionally, Zeek is able to extract descriptors, such as the number of bytes transmitted/received that can then be used to evaluate the purpose of a connection. Furthermore, Zeek provides its own scripting language and is built with modularity in mind (38). We can therefore configure it to provide all details in need of assessment and then label flows accordingly. Paired with its rich documentation, open source approach and the ability to work directly on pcap files, its feature set makes it the most suitable solution for the task at hand.

3.2 Using Zeek as a Basis for Informative Labelling

With Zeek offering modularity and extensive, detailed log files after processing a pcap file, it is well suited as a foundation for labelling IP traffic. This section will address how Zeek works, how to use and configure its ouput and finally how a custom frontend for processing pcaps with Zeek through a web browser was built.

3.2.1 Working with Zeek

Zeek's main target operating system is Linux. To make future updates and possible migrations of the system easier and independent of the host machine, it was decided to use Docker to run an instance of Zeek. The developers of Zeek provide a docker image for this use case, making the setup process very simple — with the process being discussed in detail in the section 3.2.2 (42). When running Zeek, we can either have it live capture traffic on an interface, e.g. the host machines ethernet port, or provide a pcap file to Zeek for analysis. Since the focus of this thesis is not live traffic labelling, we will proceed with the latter. For a simple analysis of a pcap file, using default parameters, the following command is sufficient: (43)

```
zeek -r mypackets.trace local
```

This command runs Zeek with the mypackets.trace file as an input pcap file. The word "local" instructs Zeek to use the "local.zeek" script, which is included in the standard configuration and can be found in the official Github repository (44). After running Zeek in this manner, it puts out several log files—the most notable being the "conn.log", which lists all connections that have been established and further details, e.g. timestamp, protocol, service, and bytes transferred. Additionally, Zeek creates a separate log file for each enabled submodule, e.g. "http.log" and "dns.log", each of which contain specific information on connections using the Hyper Text Transfer Protocol (46) and the Domain Name System (47) respectively.

3.2.2 Building a Frontend for Zeek

After performing initial test runs with Zeek, the next step was to create a testing environment to test Zeek on existing datasets. When running Zeek as a Docker image, it is necessary to manually manage the pcap files, then run Zeek with a custom script, and then copy back the output. This gets very tedious when analysing multiple files. For this reason, a more user friendly solution using docker-compose was conceptualised. With the docker-compose.yml file provided in the Appendix A.3, the following Docker containers are created:

- Zeek
- Debian
- PHP server
- nginx webserver

All images are pulled directly from Dockerhub. The nginx webserver then hosts a simple PHP: Hypertext Preprocessor (PHP) webpage which provides the possibility to upload

and download files. To actually run PHP, which is a scripting language, a PHP server is necessary. To move and delete files, a Debian Linux container is used. Finally the Zeek container perfoms the pcap analysis. The simplified workflow is as follows: First a pcap file is uploaded by the user through the php webpage. This file is then located in the "uploads" directory. The Debian container runs a shell script, monitoring changes in this directory. If a new file is detected, the Debian container executes another script which runs a command in the Zeek container using the file as input. After Zeek has finished its analysis, a "console.log" is created, containing the command line return value of Zeek. The Debian container detects this newly created file and compresses the output log files into a zip file. This zip file is then placed in the "zipped-logs" directory, from which it can be downloaded using the webpage. The zip file contains the name of the analysed pcap and the timestamp of the analysis. Finally, the temporary log files are deleted. To save on storage space, another shell script runs in the background, deleting the oldest file once the uploads directory reaches 8 gigabytes in size.

Figure 3.1 gives a brief, simplified overview of how a user uploads a pcap file, which is then analysed by Zeek, with the results zipped by the Debian container and provided back to the user. The nginx and PHP Server container have been omitted in this flowchart. Figure 3.2 illustrates this system in more detail, with the cylinders representing the folders "uploads" and "zipped-logs" and the rounded rectangles representing Docker containers. The arrows indicate the actions taken by the containers. The "Management Script" is responsible for monitoring the upload directory, running the "Zeek Script" when a new pcap file is uploaded, zipping the .log files produced by Zeek, and saving them to the "zipped-logs" directory.

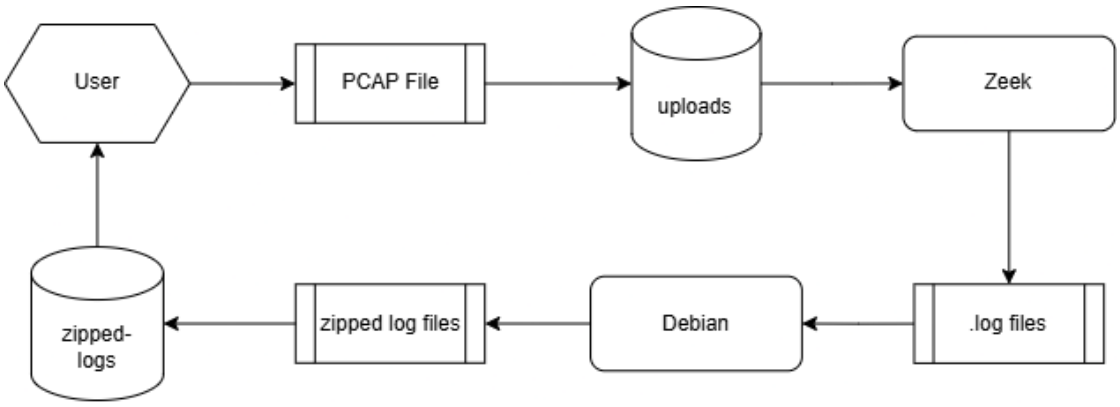

Figure 3.1: Flowchart showing how a pcap is processed using Zeek, simplified for demonstration purposes, omitting the nginx and php components.

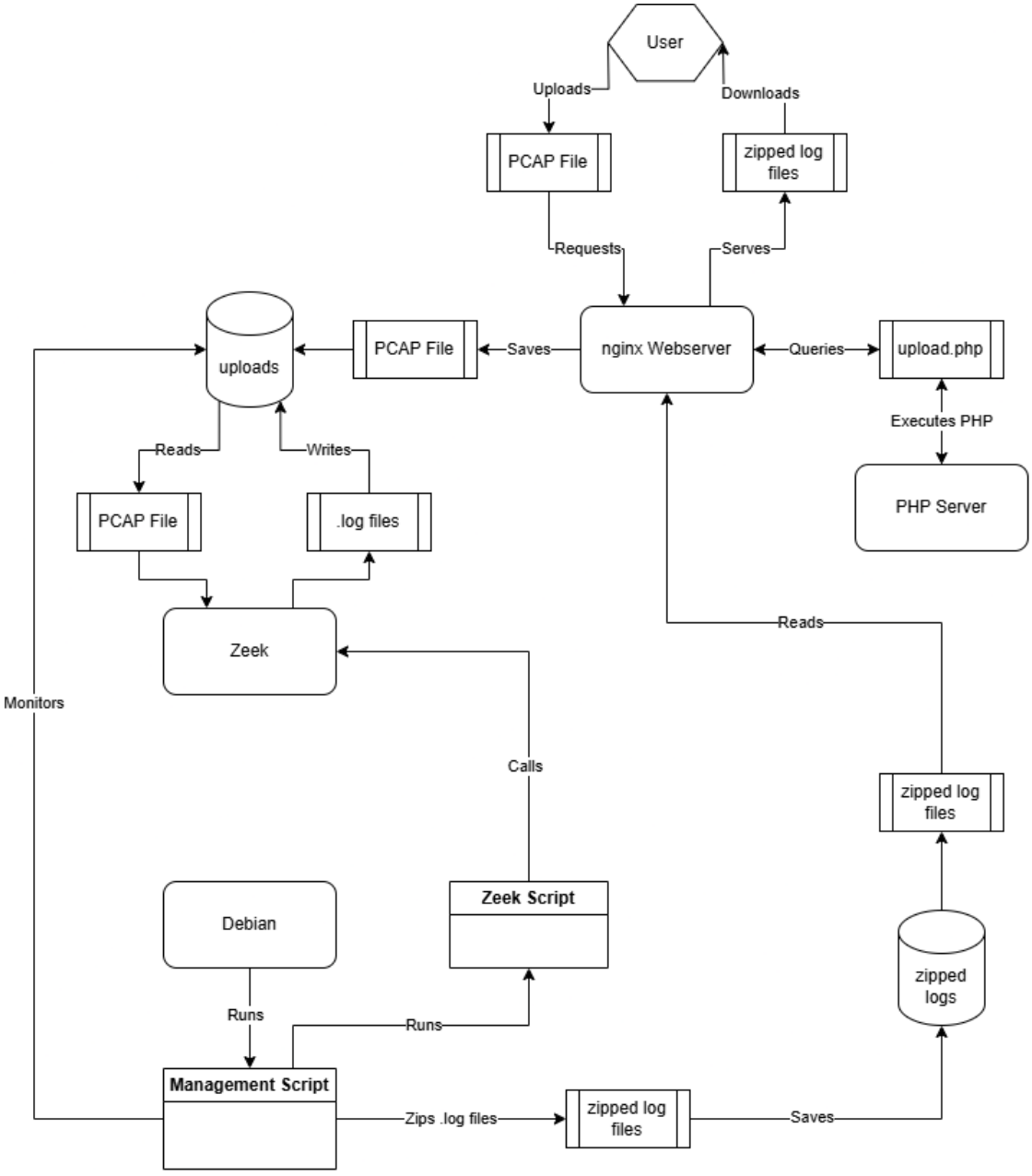

Figure 3.2: Flowchart illustrating the Frontend for Zeek, showcasing how Docker containers communicate and operate.

3.3 FlowBreaker: A Smart Wrapper for Zeek

This section revolves around the planning and development of FlowBreaker: a tool that wraps Zeek's output and processes it to detect the attacks mentioned in Chapter 2. It

also provides additional insight into the traffic and the hosts associated with it. First the scope is set, then FlowBreaker's structure is explained, and finally we give details on how traffic is classified.

3.3.1 Setting the Scope

As already mentioned in this thesis, it is necessary to limit the goals and expectations, as the field of analysing IP traffic is massive. Therefore, the following design goals were set early on in the planning phase of FlowBreaker:

1. CLI only: A GUI is not necessary for operation and it requires significantly more development-time to implement. Only using a CLI simplifies development in exchange for slightly reduced usability. This way all user input can be provided as text using a CLI. Furthermore, CLIs are easier to integrate with Shell scripts, which facilitates automated processing of multiple input files.
2. Outsourcing of pcap analysis: Aggregating packets into flows by processing a pcap is difficult to implement, as one would need to implement a way of formatting the binary data into readable data and then iterate over said data. With Zeek, this is not necessary because it already provides aggregated packet flows in its output data. These flows can then be parsed into custom classes, which are then used internally.
3. Providing a framework rather than a fully featured solution: This way the end result can be easily modified in the future. Furthermore, features can be developed independently of each other, which simplifies future additions.
4. Concentrating on detecting certain types of attacks first: Detecting all feasible attacks would be out of scope, as we want to provide a proof of concept, that can be validated first. By concentrating on a smaller subset of attacks, we can trade quantity for quality. Furthermore, the resulting detection methods could be reused and adapted to other forms of attacks in the future, thus reducing development time.

With this in mind, the expectations can be set accordingly: The resulting tool is supposed to take the already processed pcap file, then perform its own analysis based on the measurements contained within the pcap, and finally output a summary of the file's contents with additional labels with a high informative value in place. Therefore, the end result should be cross-platform-compatible and provide an easy way for user configuration.

3.3.2 Divide & Conquer

In order to be able to work on features individually, FlowBreaker had to be structured in a modular way, involving breaking functionality up into independent subclasses. This

ensures code readability and maintainability. Additionally, since the data provided is not pre-sorted, the used language has to be capable of efficiently handling lists that contain several 100,000 entries. This is why C# was used. Unlike interpreted languages such as Python, C# builds on an object-oriented approach and strong typing, providing significantly faster execution due to its compiled nature. Additionally, C# offers Language Integrated Query (LINQ), which enables queries similar to Structured Query Language (SQL) on any custom classes containing data. These features, combined with the asynchronous support and the excellent IDE (Integrated Development Environment) provided by Visual Studio, make C# a solid foundation for this thesis. With data queries sorted, the next step is processing the raw input data into a structured format that can be handled by other modules.

3.3.3 Wrapping Zeek's Output

As mentioned above, Zeek outputs several log files after processing a pcap file. The most important one is the "conn.log", which lists all connections Zeek is able to recognise.

An example line of this log file can be found in Listing 3.1.

With this in mind, it is already possible to extract several key details, such as the origin and destination, bytes transmitted, protocols used and duration of the connection. Furthermore we can cross reference this entry to other log files, such as the "http.log", using the "uid" parameter. To handle this kind of connection, the "Connection class", which acts as a simple wrapper and provides access to the attributes below, is introduced. Furthermore, standard methods such as "ToString()", [1] were introduced.
With the existing data formatted, the next step was to aggregate these connections into larger flows and associate them with a given host IP address—hence the "Connection-Group" class is introduced. This class stores a list of all communications associated with a given IP and provides additional fields for statistic description, such as the number of unique source ports used, average number of bytes transmitted per connection, number of connections, etc. The goal of this class is to provide a starting point for later classifying IPs as malicious or benign. All attributes and methods implemented are discussed in the next sections, along with an overview of the program flow and the other classes involved.

Please note that FlowBreaker's "connections" refer to TCP, UDP and ICMP communications, with the term being used interchangeably, rather than just for TCP connections.

```
"ts": "1970-01-01T00:42:10.881299Z",
"uid": "CF2GticcWn8tTg866",
"id.orig_h": "192.168.1.90",
"id.orig_p": 35900,
"id.resp_h": "192.168.1.70",
"id.resp_p": 8080,
"proto": "tcp",
```

[1] *ToString() is a method which returns a String summarising an object's properties*

```
"service": "http",
"duration": 25.995668000000023,
"orig_bytes": 137,
"resp_bytes": 4487402,
"conn_state": "SF",
"local_orig": true,
"local_resp": true,
"missed_bytes": 0,
"history": "ShADadtttfF",
"orig_pkts": 2565,
"orig_ip_bytes": 147597,
"resp_pkts": 3410,
"resp_ip_bytes": 4690493,
"ip_proto": 6
```


Listing 3.1: Example of a Zeek conn.log entry

The flowchart in Figure 3.3 provides a rough overview of the tool's inner workings. As explained above, the log files provided by Zeek are first read in, then formatted into a list of individual connections. Optional log files, e.g. the "dns.log", are parsed as well and provided if required for analysis later on. Since some of the information contained in them, such as the "id.orig_h" (which denominates the origin IP address), are redundant and already contained in the "conn.log", they are only queried if needed and are not structured into individual connection groups. To actually sort connections effectively they are first broken up by the following protocols:

- **Transmission Control Protocol (TCP)**
  This protocol is used for reliable data transfers, meaning that each packet is confirmed by the receiver. It is used for purposes such as transmitting files and authentication (48).
- **User Datagram Protocol (UDP)**
  This protocol is used for unreliable data transfers, this is useful when speed matters more than data integrity; for example, in streaming video (49).
- **Internet Control Message Protocol (ICMP)**
  Unlike TCP and UDP, this protocol is directly embedded in the IP stack and is used for basic controls and messages, e.g. the "destination unreachable" notification/message (50).

The reason for this is that TCP, UDP, and ICMP are the most frequently used IP protocols, and attacks are prone to use them separately (i.e. most attacks commonly focus on one of these protocols). For example a DNS amplification attack, an attack abusing Domain Name System (DNS) to overwhelm a victim (20), is carried out using UDP, as DNS is a service almost exclusively using this protocol (47). Another example

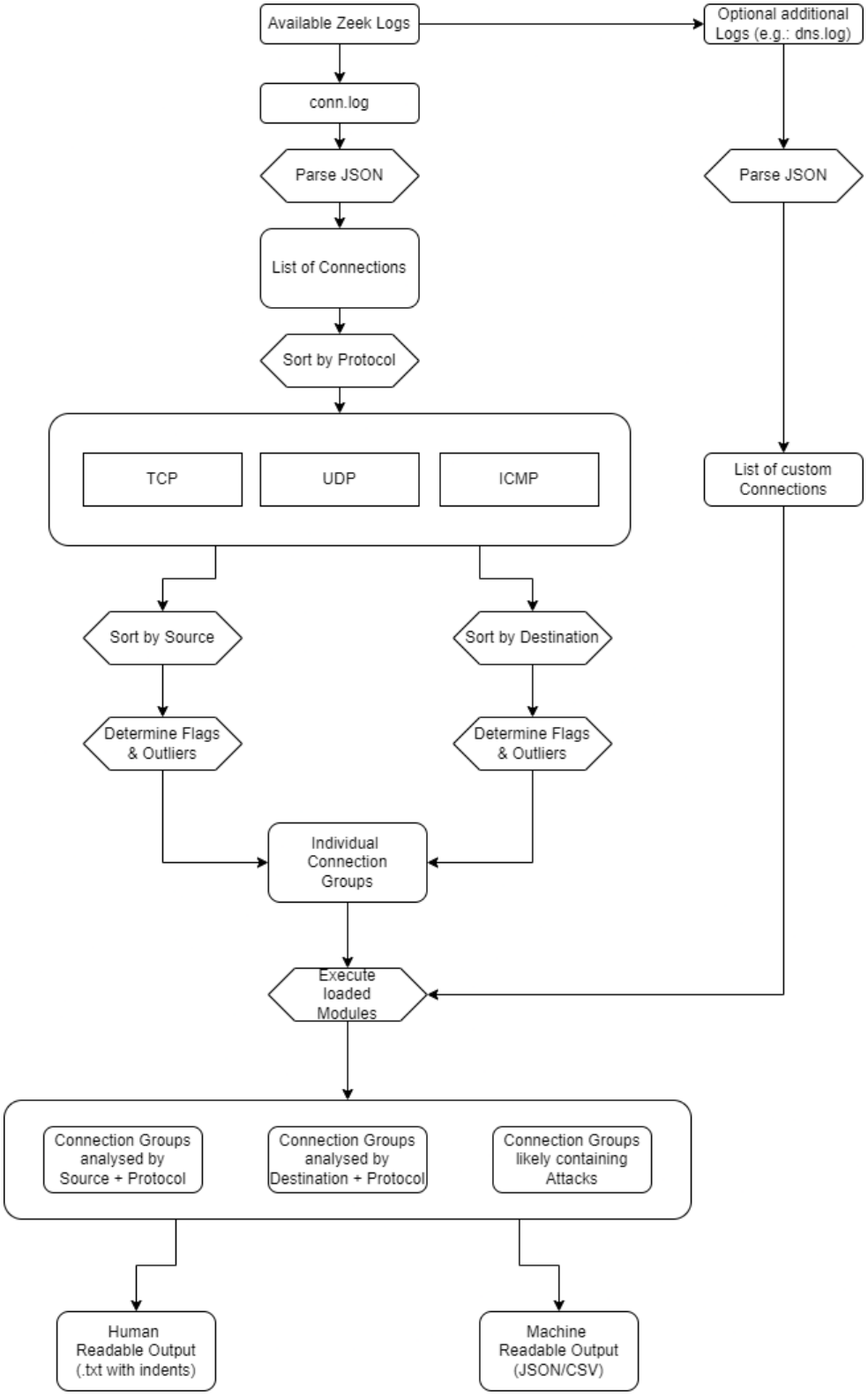

Figure 3.3: This Flowchart outlines how FlowBreaker processes Zeek input files, performs attack detection and writes the results to disk.

would be brute force for guessing passwords, which aims at protocols such as Hypertext Transfer Protocol Secure (HTTPS), or Secure Shell (SSH), that are built on TCP (19). After splitting the connections up by TCP, UDP and ICMP, the three resulting lists are once sorted by source IPs and once by destination IPs (six lists in total). This is done because some attacks, for example DDoS, use many source IPs on a single destination IP, while others, e.g. scans for connected hosts, originate from a single source.
By using two separate connection groups, it is easier to evaluate what is happening in a given traffic capture. During the process of aggregating the connections into connection groups, the additional statistical parameters ("Flags & Outliers"), which are discussed in Section 3.3.6, are also calculated and set. Finally the processed connection groups are parsed into the individual submodules, which run detection methods on the groups. The results are then saved to the output directory in the txt, csv and json formats. This also applies to the six processed connection groups, containing all TCP, UDP and ICMP connections.
To allow for flexibility in FlowBreaker's behaviour, each module along with its methods is configurable by using a configuration file. This file called "config.toml" is formatted in the toml (52) format, which allows for comments and better readability[2].

The values in this file specify thresholds and markers used in the detection methods for the classification of individual hosts. One example would be how many SSH connections need to be initiated by a source IP for the source IP to be flagged as malicious, which is specified using the field "MinConnections" in the section [SSHBruteForce]. An example for a list of values would be the "CommonPorts" list, which is specified in the section [CommonPortsAttack] and denominates each port that FlowBreaker watches. The purpose of each configuration value is explained in Section 3.3.4 alongside the detection methods they are used in.

[2]An example for this file can be found in the Appendix A.1.

3.3.4 FlowBreaker's Modules

This section gives an overview of the implemented analysis modules, including a short description of the attacks they can detect, and Table 3.1 providing an overview of their submethods and relevant configuration parameters. In the current version of FlowBreaker there are four modules implemented:

Module	Submodule	Sorts By	Config Values
Basic Analysis	N/A	Src. & Dest. IP	Flag Setting Behaviour
Scanning	DetectPortScans	Source IP	Connection_Threshold, Unique_Port_Threshold
	DetectHostDiscoveryScans	Source IP	Unique_IP_Threshold
	DetectProtocolSpecificScans	Source IP	SYN_Scan_Threshold
	DetectVersionScans	Source IP	Connection_Threshold, Min_Port_Number, Max_Bytes_Transferred, Common_Ports
	DetectServiceEnumeration	Source IP	Connection_Threshold, Min_Port_Number, Min_Bytes_Transferred, Common_Ports
BruteForce	DetectCommonPortAttacks	Destination IP	MinConnectionsPerPort
	DetectPasswordSpraying	Source IP	PasswordSprayingThreshold
	DetectSSHBruteForce	Source IP	MinConnections
	DetectSSLBruteForce	Source IP	MinConnections
	DetectHTTPBruteForce	Source IP	MinConnections
DDoS	DetectSYNFlood	Destination IP	SYNThreshold
	DetectUDPFlood	Destination IP	UDPThreshold
	DetectICMPFlood	Destination IP	ICMPThreshold
	DetectDNSAmplification	Source IP	DNSThreshold, MaxDomainRepetitions
	DetectNTPAmplification	Source IP	NTPThreshold
	DetectSSDPAmplification	Source IP	SSDPThreshold
	DetectConnectionExhaustion	Destination IP	ConnectionThreshold, MaxBytes, MinDuration
	DetectSlowlorisAttack	Destination IP	HalfOpenThreshold, MinDuration

Table 3.1: FlowBreaker Module Overview: Listing Submodules, relevant configuration options and how output is sorted

1. **Basic Analysis**
   This module is responsible for creating six outputs: First the connections are split into three groups: TCP, UDP and ICMP. These are then broken up into Connection Groups sorted by destination IP, and Connection Groups sorted by source IP, resulting in a total of six groups. In addition, the custom flag fields are set based on the config.toml and the average values are calculated.

2. **Scanning**
   This module detects attacks that search for vulnerabilities on a target machine by using the following methods:

   a) **DetectPortScans**
   This method detects Port Scans, where an attacker tries to find open ports on a target machine. Two thresholds are configurable to achieve this: Connection_Threshold, which sets the minimum number of connections a host needs to establish and Unique_Port_Threshold which sets the minimum number of unique ports that need to be addressed by the attacker. The output is sorted by source IP.

   b) **DetectHostDiscoveryScans**.
   This method detects Host Discovery Scans based on the number of unique destination IPs targeted by a host. If this number exceeds Unique_IP_Threshold, the source IP is flagged and written into the output file. The output is sorted by source IP.

   c) **DetectProtocolSpecificScans**.
   This method detects SYN Scans, which consist of the attacker sending TCP Syn Packets to a victim, in order to detect open ports/protocols. Zeek provides the "S state" in the "history field", indicating that only a SYN packet without the ACK bit set was seen coming from the source IP. FlowBreaker flags this as an attack if the minimum number of these connections set in SYN_Scan_Threshold is exceeded. The output is sorted by source IP.

   d) **DetectVersionScans**.
   This method detects Version Scans, which try to detect the specific protocol version being used on a given port. This is done by searching for repeated connections to commonly used ports, with minimal data transfer. The following parameters can be defined: Connection_Threshold (minimum number of connections per port), Min_Port_Number (minimum number of unique ports targeted), Max_Bytes_Transferred (the maximum number of bytes), Common_Ports (a list determining which ports are monitored). The output is sorted by source IP.

   e) **DetectServiceEnumeration**
   This method works identical to the method above, with the only difference being that, instead of a maximum number of bytes being defined, a minimum

number of bytes is defined with Min_Bytes_Transferred. The output is sorted by source IP.

3. **BruteForce**
   This module implements very basic methods that trigger based on the number of connections made, with a high number indicating a potential Brute Force attempt.

   a) **DetectCommonPortAttacks**
   This method triggers if the number of established connections from a single IP to a port in the CommonPorts list exceeds the MinConnectionsPerPort variable. The results are sorted by destination IP.

   b) **DetectPasswordSpraying**
   This method listens for authentication attempts on the ports listed in CommonPorts. It triggers if the number of login attempts (ssh, ssl or http) exceeds PasswordSprayingThreshold. The output is sorted by source IP.

   c) **DetectSSHBruteForce**
   This method triggers if the amount of SSH connections exceeds MinConnections. The output is sorted by source IP.

   d) **DetectSSLBruteForce**
   This method triggers if the amount of SSL/TLS connections with failed handshakes exceeds MinConnections. The output is sorted by source IP.

   e) **DetectHTTPBruteForce**
   This method triggers if the amount of Hypertext Transfer Protocol (HTTP) requests that contain "password=" in their "uri" field exceeds MinConnections. The Unique Resource Identifier (URI) field(59) serves as the authentication to a server, meaning that a high number of these requests indicates password guessing. The output is sorted by source IP.

4. **DDoS**
   This module implements methods that search for attacks that attempt to exhaust the resources of a target by flooding it with requests.

   a) **DetectSYNFlood**
   This methods triggers if the number of connections with their conn_state set to "S0" or "REJ" (a field used by Zeek, describing TCP connections that contain only a single SYN packet or have been rejected) exceeds SYNThreshold. The output is sorted by destination IP.

   b) **DetectUDPFlood**
   This method triggers if the number of UDP connections exceeds the UDPThreshold. The output is sorted by destination IP.

   c) **DetectICMPFlood**
   This method triggers if the number of ICMP connections exceeds the ICMPThreshold. The output is sorted by destination IP.

d) **DetectDNSAmplification**
This method works by detecting redundant DNS queries, by checking the following conditions: The number of DNS requests that query the same domain more than MaxDomainRepetitions, have to exceed DNSThreshold. If MaxDomainRepetitions is set to 0, only the total number of DNS requests has to exceed DNSThreshold. The output is sorted by source IP. (Because from Zeek's interpretation, the victim is the source of the request)

e) **DetectNTPAmplification**
This method triggers if the amount of connections with 123 being the origin port exceeds NTPThreshold. The output is sorted by source IP. (Because from Zeek's interpretation, the victim is the source of the request)

f) **DetectSSDPAmplification**
This method triggers if the amount of connections with 1900 being the origin port exceeds SSDPThreshold. The output is sorted by source IP. (Because from Zeek's interpretation, the victim is the source of the request)

g) **DetectConnectionExhaustion**
This method works by checking for a high number of connections with minimal data transfer that stay open for a long duration. The minimum number of connections is defined in ConnectionThreshold, the maximum number of bytes in MaxBytes and the minimum duration in seconds in MinDuration. The output is sorted by destination IP.

h) **DetectSlowlorisAttack**
This method triggers if the number of half-open connections, therefore connections in the "S1" or "SF" state (a field used by Zeek, describing TCP connections that have not been terminated), exceeds HalfOpenThreshold, with MinDuration again defining the minimum time a connection needs to stay open to be counted. The output is sorted by destination IP.

3.3.5 FlowBreaker's Workflow

FlowBreaker works by breaking up its input into smaller, sorted lists, that describe network flows. Understanding this process is key to understand the output of the program. As can be seen in Figure 3.3, FlowBreaker starts by reading in the Zeek Logs, to then further analyse them. Internally, FlowBreaker uses the custom class "ConnectionGroup", which stores a single IP as its key, a list of connections relating to this IP, as well as the values, fields and flags calculated based on this IPs activity. This is carried out in the following order:

1. **Discrimination by Protocol**
   The input is broken up by the underlying protocol used, namely: TCP, UDP and ICMP.

2. **Discrimination by IP**
   The three resulting groups are now sorted twice into two dictionaries: First, "SortBySource", which contains all IPs that have at least initiated a single connection; and, second, "SortByDestination", which contains all IPs that have at least received a single connection.

3. **Processing "SortBySource" and "SortByDestination" Dictionaries**
   From Steps 1 and 2 we obtained a total of six dictionaries, which are all processed individually. FlowBreaker calculates the average values across all IPs and their associated connections for all six dictionaries and uses them as a reference when determining custom activity flags, which can be used to identify IPs with high or low activities.

4. **Running additional Modules**
   If configured, FlowBreaker runs each module added to the configuration file on the lists generated above. These outputs (if they actually exist) are then saved to separate files.

3.3.6 Interpreting FlowBreaker's Output

After executing a submodule, FlowBreaker saves its output into three different output file types: .txt, .csv, .json, with the first serving as a human readable summary and the latter two as means for automated processing. In these output files, FlowBreaker offers "Text Fields", "Activity Flags", "Flagging Scores" and "Average Values". Text Fields provide a general description of the analysed IP addresses and the communications associated with them (Note that "Connections" refers to TCP as well as UDP and ICMP based communications). "Activity Flags" are binary values, serving to indicate high activity of the IP question compared to the other addresses in a traffic capture. Flagging Scores are float values, which are used as a basis for setting the "Activity Flags". Their value can be understood as a multiple of the average value in a given traffic capture and is calculated based on the equations in Equation . Finally, "Average Values" is an assortment of float

values describing the average of e.g. communications or ports per IP address. The details of how these calculations are performed can be found in Equations 3.2 to 3.8.

Text Fields

In its output, FlowBreaker provides several text fields for each host contained in the input files. These fields are meant to offer human-readable information, describing the activity of the host in question:

1. **IP**
   The IP associated with the connections, which is either the source or the destination, but never both in the same file.
2. **Total Connections**
   The total number of communications associated with this IP.
3. **Protocol**
   The transport protocol used, either TCP, UDP or ICMP
4. **Service**
   If the Connection Group only contains a single type of network protocol, this field is set to this protocol, e.g. "http". If not, it is set as "UNDEF". The protocol identification is done based on Zeek's output. Note that Connection Group is an internal class name of FlowBreaker, which is created for every unique IP address.
5. **Classification**
   This text field is only set if the Connection Group was identified as malicious, otherwise it is set to "UNDEF".
6. **Connection Summary**
   This field either refers to source or destination IPs. It contains the following lists and their related figures/aggregations:
   a) **Source/Destination IPs**
      A list of all unique IPs and the number of connections they are associated with, sorted in descending order by the number of connections.
   b) **Unique Destination Ports**
      A list of all unique destination ports used and the number of connections they are associated with, sorted in ascending order based on port number.
   c) **Unique Source Ports**
      A list of all unique source ports used and the number of connections they are associated with, sorted in ascending order based on port number.
7. **Services**
   This field contains a list with the number of services logged by Zeek together with

their number of occurrences. Note that "ssl"(55) simply refers to encrypted traffic and is used by Zeek to express that a Transport Layer Security (TLS) encryption was recognised (56) (SSL was superseded by TLS and both acronyms are often used interchangeably).

Activity Flags

Based on the configuration file, FlowBreaker sets eight flags according to the IPs activity and Equation 3.1. This equation defines the threshold of what is considered a "high" amount by FlowBreaker. The flags can be understood as follows:

1. *High Outgoing Port Activity*: If set, this flag indicates that the IP was associated with a "high" entropy of destination ports, meaning that the IP targeted many different ports in its communications.

2. *High Incoming Port Activity*: If set, this flag indicates that the IP was associated with a "high" entropy of source ports, meaning that the IP used many different source ports in its communications.

3. *High Outgoing IP Activity*: If set, this flag indicates that the IP was associated with a "high" entropy of destination IPs, meaning that the IP targeted many different IP addresses in its communications.

4. *High Incoming IP Activity*: If set, this flag indicates that the IP was associated with a "high" entropy of source IPs, meaning that the IP was addressed by many different IP addresses.

5. *High Number of Outgoing Connections*: If set, this flag indicates that the IP was associated with a "high" amount of incoming connections, meaning that the IP initiated many connections/communications (either to the same or different destinations).

6. *High Number of Incoming Connections*: If set, this flag indicates that the IP was associated with a "high" amount of outgoing connections, meaning that the IP was targeted in many connections/communications (either to the same or different sources).

7. *Listener*: This flag is set if the IP has more incoming than outgoing communications, meaning it "listens" more than it "speaks" (this flag is mutually exclusive with the Speaker flag).

8. *Speaker*: This flag is set if the IP has more outgoing than incoming communications, meaning it "speaks" more than it "listens" (this flag is mutually exclusive with the Listener flag).

All flags are consistent across output files for the same IP.

$$\text{Flag}_i = \begin{cases} \text{True}, & \text{if } v_i \geq \bar{v} \times f \\ \text{False}, & \text{otherwise} \end{cases} \tag{3.1}$$

Where v_i is the value for the specific IP, $\bar{v}$ is the average value across all IPs and their connections, and f is the multiplication factor provided in the configuration file. All v_i values are logged in console during runtime but are not contained in the output files. To save them, it is necessary to either copy the console log or pipe the output to a file.

Flagging Scores

This section simply contains the results of equation 3.1 for each individual flag. The idea here is to provide more detail than just a binary value. If the flag cannot be set, the value is initialised as "-1" — for example, in an output file that only contains source IP addresses, the "Incoming IP activity" would be "-1", because this file only addresses outgoing connections, not incoming ones.

Average Values

To clarify the various variables used internally by FlowBreaker, Table 3.2 provides an overview of key variables alongside short explanations. The Average Values are calculated as follows: With `src_ips` referring to the number of unique source IPs, `dest_ips` referring to the number of unique destination IPs, `src_ports` to the number of unique source ports and `dest_ports` to the number of unique destination ports.

Variable	Description
`src_ips`	Number of unique source IPs
`dest_ips`	Number of unique destination IPs
`src_ports`	Number of unique source ports
`dest_ports`	Number of unique destination ports
`connections`	Total number of connections (communications) for a given protocol/IP
`orig_bytes`	Origin bytes sent (excluding IP header) - calculated by Zeek based on packet numbers, can be inaccurate
`resp_bytes`	Response bytes sent (excluding IP header) - calculated by Zeek based on packet numbers, can be inaccurate
`orig_ip_bytes`	Origin IP bytes sent (including headers) - based on bytes actually seen by Zeek, accurate
`resp_ip_bytes`	Response IP bytes sent (including headers) - based on bytes actually seen by Zeek, accurate

Table 3.2: Overview of Variables used internally by FlowBreaker

$$\text{Connections per Source IP} = \begin{cases} \text{connections}, & \text{for sortBySource} \\ \frac{\text{connections}}{\text{src\_ips}}, & \text{for sortByDestination} \end{cases} \tag{3.2}$$

$$\text{Connections per Destination IP} = \begin{cases} \frac{\text{connections}}{\text{dest\_ips}}, & \text{for sortBySource} \\ \text{connections}, & \text{for sortByDestination} \end{cases} \tag{3.3}$$

$$\text{Connections per Source Port} = \frac{\text{connections}}{\text{src\_ports}} \tag{3.4}$$

$$\text{Connections per Destination Port} = \frac{\text{connections}}{\text{dest\_ports}} \tag{3.5}$$

$$\text{Connections per Unique IP} = \begin{cases} \text{Connections per Destination IP}, & \text{for sortBySource} \\ \text{Connections per Source IP}, & \text{for sortByDestination} \end{cases} \tag{3.6}$$

$$\text{Average Bytes transferred per Connection} = \frac{\sum_{i=1}^{n}(\text{orig\_bytes}_i + \text{resp\_bytes}_i)}{\text{connections}} \tag{3.7}$$

$$\text{Average IP Bytes transferred per Connection} = \frac{\sum_{i=1}^{n}(\text{orig\_ip\_bytes}_i + \text{resp\_ip\_bytes}_i)}{\text{connections}} \tag{3.8}$$

Where in 3.7 and 3.8:

- n is the number of connections in the group.
- orig_bytes_i and resp_bytes_i are the original and response bytes for connection i.
- orig_ip_bytes_i and resp_ip_bytes_i are the original and response IP bytes for connection i.
- connections is the total number of connections in the Connection Group - a class used by FlowBreaker to describe a given host and its associated connections.

The values “orig_ip_bytes” and “resp_ip_bytes” represent the number of bytes that have been sent by the source IP address and by the destination IP address, including headers. These numbers are accurate and correspond to the actual packets transmitted, as seen by Zeek. The values “orig_bytes” and “resp_bytes” however are calculated by

Zeek based on the sequence numbers of the packets, without including the IP header. This means that they are prone to be inaccurate in attack scenarios, where large numbers of packets are sent. To illustrate the fields and calculations explained in this section, the example in Listing 3.2 shows FlowBreaker's output for a single IP address with a single DNS connection.

```
IP: 192.168.0.118
Total Connections: 1
Protocol: UDP
Service: dns
Classification: UNDEF
Connection Summary:
    Unique Destination IPs: 1
            Destination IPs:
                    1.1.1.1: 1 connections
    Unique Destination Ports: 1
            Destination Ports:
                    53: 1 connections
    Unique Source Ports: 1
            Source Ports:
                    32431: 1 connections
Services:
Activity Flags:
    High Outgoing Port Activity: False
    High Incoming Port Activity: False
    High Outgoing IP Activity: False
    High Incoming IP Activity: False
    High Number of Outgoing Connections: False
    High Number of Incoming Connections: False
    Listener: False
    Speaker: True
Flagging Scores (Current Value/Average Value):
    Outgoing Port Activity: 1
    Incoming Port Activity: -1
    Outgoing IP Activity: 1
    Incoming IP Activity: -1
    Outgoing Connections: 1
    Incoming Connections: -1
Average Values:
    Connections per Destination IP: 1
    Connections per Source IP: 1
    Connections per Destination Port: 1
    Connections per Source Port: 1
    Connections per Unique IP: 1
    Bytes transferred per Connection: 0
    IP Bytes transferred per Connection: 80
```

Listing 3.2: Example .txt Output of FlowBreaker for a single DNS connection

3.4 Analysing Labels provided by popular NIDS Datasets

With the testbench created, we can now move on to running Zeek on existing datasets. In this section the labels provided by the TII-SSRC-23 dataset are addressed and possible improvements upon these labels are discussed.
Furthermore, a single traffic capture file, provided by the Measurement and Analysis on the WIDE Internet (MAWI) Working Group[3], is selected, analysed and then processed with FlowBreaker. The results of this analysis are summarised in the Evaluation chapter. While the traffic captures of MAWI do not necessarily contain any intrusion attempts, they can be used to showcase and assess FlowBreaker's capability to summarise traffic data.

3.4.1 Overview of the TII-SSRC-23 dataset

The TII-SSRC-23 dataset (15) provides 32 pcap files containing malicious and non malicious traffic and a csv file labelling the traffic of each file as either benign or malicious, and, if applicable, the type of attack taking place. The csv file was split by traffic subtype, with the resulting labels compiled in Table 3.4. It is apparent that this labelling is rather basic. All traffic flows contained in the pcap file share the same label, with no additional explanation given. For example, the "information-gathering" label does not provide any further insight into what kind of information is actually being gathered. This could range from a port scan (scanning for open ports), to a host discovery scan (scanning for all IPs in a network), to service enumeration (identifying which version of a service is being used on a host), or simply an attacker looking for vulnerable webpages.

3.4.2 Selecting subtypes from the TII-SSRC-23 Labels

After assessing the attack types contained within the dataset, the attacks in Table 3.3 were selected for further assessment. (D)DoS refers to "(Dedicated) Denial of Service", and it sets out to overwhelm a victim by sheer volume of requests. This can take place by abusing protocols like DNS or using multiple hosts (20). "Bruteforce" refers to an attacker searching for passwords or other confidential data with no prior information, hence blindly guessing or using "brute force" (19). Finally "information gathering", as discussed above, can refer to various attacks that generally aim at extracting information on possible targets.

The reason why these types and subtypes of attacks were selected, is that they are very common and should be easy to filter from benign traffic. To label these different types, we will identify key markers used by attacks, such as the port numbers used, the amount of activity associated with a given host (e.g. how many TCP connections have

[3]The MAWI (Measurement and Analysis on the WIDE Internet) Working Group performs network traffic measurement, analysis, evaluation, and verification of the WIDE (Widely Integrated Distributed Environment) project. This project captures packet traces from backbone networks, typically from a transpacific link between Japan and other regions.

been initiated) and the protocols involved, e.g. DNS for the pcap “mirai_ddos_dns”. The attack types listed in Table 3.3 should depict very distinct patterns in these markers. For example, DNS is served on port 53 using UDP, while SSH is typically served on port 22 using TCP (19). This means, that the pcap “bruteforce_ssh”, should contain an unusually high number of TCP connections to port 22, while “miraid_ddos_dns” should contain a high number of communications associated with port 53, possibly from different source IPs (20). We will therefore filter the input data accordingly and test if FlowBreaker’s detection methods, mentioned in Section 3.3.4, can detect the attack patterns mentioned in Table 3.3. Furthermore, we will assess if FlowBreaker is able to provide further descriptions to the pcaps listed in Table 3.3. The next step will be to process all other pcaps listed in Table 3.4, assessing FlowBreaker’s performance when processing attack patterns it was not directly designed to detect.

pcap	Label	Traffic Type	Traffic Subtype
syn_tcp_dos	Malicious	DoS	DoS_SYN
udp_dos	Malicious	DoS	DoS_UDP
mirai_ddos_dns	Malicious	Mirai-Botnet	Mirai_DDoS_DNS
mirai_ddos_syn	Malicious	Mirai-Botnet	Mirai_DDoS_SYN
mirai_ddos_udp_udpplain	Malicious	Mirai-Botnet	Mirai_DDoS_UDP
mirai_scan_bruteforce	Malicious	Mirai-Botnet	Mirai_Scan_Bruteforce
information_gathering	Malicious	Information_Gathering	Information_Gathering
bruteforce_ftp	Malicious	Bruteforce	Bruteforce_FTP
bruteforce_http	Malicious	Bruteforce	Bruteforce_HTTP
bruteforce_ssh	Malicious	Bruteforce	Bruteforce_SSH
bruteforce_telnet	Malicious	Bruteforce	Bruteforce_Telnet

Table 3.3: Selected Attack Types from the TII-SSRC-23 Dataset: Label denominates classification, Traffic Type the broader category, Traffic Subtype provides more detailed information

pcap file	Label	Traffic Type	Traffic Subtype
ack_tcp_dos	Malicious	DoS	DoS_ACK
cwr_tcp_dos	Malicious	DoS	DoS_CWR
ecn_tcp_dos	Malicious	DoS	DoS_ECN
fin_tcp_dos	Malicious	DoS	DoS_FIN
http_dos	Malicious	DoS	DoS_HTTP
icmp_dos	Malicious	DoS	DoS_ICMP
mac_dos	Malicious	DoS	DoS_MAC
psh_tcp_dos	Malicious	DoS	DoS_PSH
rst_tcp_dos	Malicious	DoS	DoS_RST
syn_tcp_dos	Malicious	DoS	DoS_SYN
udp_dos	Malicious	DoS	DoS_UDP
urg_tcp_dos	Malicious	DoS	DoS_URG
audio	Benign	Audio	Audio
background	Benign	Background	Background
http	Benign	Video	Video_HTTP
rtp	Benign	Video	Video_RTP
udp	Benign	Video	Video_UDP
text	Benign	Text	Text
mirai_ddos_ack	Malicious	Mirai-Botnet	Mirai_DDoS_ACK
mirai_ddos_dns	Malicious	Mirai-Botnet	Mirai_DDoS_DNS
mirai_ddos_greeth	Malicious	Mirai-Botnet	Mirai_DDoS_GREETH
mirai_ddos_greip	Malicious	Mirai-Botnet	Mirai_DDoS_GREIP
mirai_ddos_http	Malicious	Mirai-Botnet	Mirai_DDoS_HTTP
mirai_ddos_syn	Malicious	Mirai-Botnet	Mirai_DDoS_SYN
mirai_ddos_udp_udpplain	Malicious	Mirai-Botnet	Mirai_DDoS_UDP
mirai_scan_bruteforce	Malicious	Mirai-Botnet	Mirai_Scan_Bruteforce
information_gathering	Malicious	Information-Gathering	Information_Gathering
bruteforce_dns	Malicious	Bruteforce	Bruteforce_DNS
bruteforce_ftp	Malicious	Bruteforce	Bruteforce_FTP
bruteforce_http	Malicious	Bruteforce	Bruteforce_HTTP
bruteforce_ssh	Malicious	Bruteforce	Bruteforce_SSH
bruteforce_telnet	Malicious	Bruteforce	Bruteforce_Telnet

Table 3.4: TII-SSRC-23 Classification of network traffic in pcaps: Label denominates classification, Traffic Type the broader category and Traffic Subtype further specifies the contents of the pcap

3.4.3 Overview of the MAWI Working Group Traffic Archive

The MAWI Working Group, is a research group that, among other things, collects traffic traces at the Widely Integrated Distributed Environment (WIDE) backbone, which is one of the major backbones connecting the Japanese internet(62). These traffic captures are available in the MAWI Working Group Traffic Archive, which provides daily captures, that cover a 15 minute time span. The captures are provided as zipped pcap files, with a single one of them reaching upwards of seven Gigabytes in size. Since these samples contain "natural" traffic and are not pre-analysed, there are no labels provided. This means, that unlike with the TII-SSRC-23 dataset, it will not be possible to verify results of an analysis independently.

3.4.4 Using the MAWI dataset for Evaluation

As mentioned above, a single traffic capture is very large. For this reason, a single pcap file from 01.01.2025, capturing the time frame from 14:00 to 14:15 was used for evaluation purposes (63). MAWI provides a breakdown of the protocols contained in the pcap file, which is visible in Table 3.5. It is apparent that roughly 1/5 of the traffic consists entirely of http and https — this can be used to test whether some of this traffic will be labelled anomalous due to the high volume. The same can be said about UDP and DNS, which account for 17.48% and 3.05%. What supersedes this is ICMP, which accounts for 1/4 of the traffic, which is unusual, as ICMP (at least in smaller networks) normally accounts for less traffic. A possible explanation for this could be ICMP probing, which describes the process of scanning the network using ICMP to detect which hosts respond and track outages, mostly caused by the USC ANT project (16).
Similar to the the TII-SSRC-23 dataset, we use this pcap file in our test experiments by using the same process and steps described in Section 3.2.2. Note that this network capture was not previously labelled; however, analysing it with FlowBreaker for evaluation purposes has a meaningful value since this is real-world data (i.e. not specifically designed to contain different attack collections). In Section 4.2 we contrast results with the information provided in Table 3.5 to check whether attack patterns can be recognised.

Protocol	Packets (%)	Bytes (%)	Bytes/Pkt
Total	136925174 (100.00%)	136355160176 (100.00%)	995.84
IP4	117883811 (86.09%)	111906629655 (82.07%)	949.30
TCP	54038536 (39.47%)	88454071193 (64.87%)	1636.87
http	7828571 (5.72%)	16190032422 (11.87%)	2068.07
https	20204061 (14.76%)	38530889698 (28.26%)	1907.09
smtp	344497 (0.25%)	172257246 (0.13%)	500.03
ftp	22806 (0.02%)	1526087 (0.00%)	66.92
ssh	1717789 (1.25%)	2918533088 (2.14%)	1699.01
dns	128856 (0.09%)	14688217 (0.01%)	113.99
bgp	33728 (0.02%)	3558809 (0.00%)	105.51
other	23755489 (17.35%)	30622421286 (22.46%)	1289.07
UDP	23937668 (17.48%)	15143624827 (11.11%)	632.63
dns	4170755 (3.05%)	745019703 (0.55%)	178.63
https	7935225 (5.80%)	7229154457 (5.30%)	911.02
other	11829767 (8.64%)	7169283557 (5.26%)	606.04
ICMP	34085962 (24.89%)	2154229712 (1.58%)	63.20
gre	183676 (0.13%)	53480495 (0.04%)	291.17
ipsec	5636496 (4.12%)	6101040602 (4.47%)	1082.42
ip6	75 (0.00%)	8020 (0.00%)	106.93
other	1398 (0.00%)	174806 (0.00%)	125.04
frag	25072 (0.02%)	17766520 (0.01%)	708.62
IP6	18683211 (13.64%)	24426690737 (17.91%)	1307.41
TCP6	10409565 (7.60%)	19206176865 (14.09%)	1845.05
http	573986 (0.42%)	1015165130 (0.74%)	1768.62
https	6252212 (4.57%)	12757842508 (9.36%)	2040.53
smtp	25854 (0.02%)	55290950 (0.04%)	2138.58
ftp	5834 (0.00%)	727043 (0.00%)	124.62
ssh	89575 (0.07%)	138668069 (0.10%)	1548.07
dns	49059 (0.04%)	9978130 (0.01%)	203.39
bgp	9184 (0.01%)	2278185 (0.00%)	248.06
other	3403861 (2.49%)	5226226850 (3.83%)	1535.38
UDP6	6836812 (4.99%)	4858585074 (3.56%)	710.65
dns	3176525 (2.32%)	902937014 (0.66%)	284.25
https	3457144 (2.52%)	3858158217 (2.83%)	1116.00
other	203143 (0.15%)	97489843 (0.07%)	479.91
ICMP6	554825 (0.41%)	46524625 (0.03%)	83.85
gre	881995 (0.64%)	315391243 (0.23%)	357.59
other6	14 (0.00%)	12930 (0.00%)	923.57

Table 3.5: Protocols contained in the MAWI capture of 01.01.2025, split by percentage

CHAPTER 4

Evaluation

This chapter revolves around testing FlowBreaker's performance on existing traffic capture datasets. This process is split into describing the test setup and comparing potential pre-existing labels to the labels assigned by FlowBreaker.

4.1 Evaluating FlowBreaker on the TII-SSRC-23 Dataset

In order to process the pcap files provided by the dataset with FlowBreaker, they first needed to be pre-processed with Zeek. This was done by using the docker compose stack, discussed in Section 3.2.2. This stack hosts a webserver and a PHP frontend for uploading pcap files, which are then automatically processed by Zeek. After preparing the relevant pcap files, the output was processed with FlowBreaker, using the default configuration toml file, which can be found in the Appendix A.1.
FlowBreaker's modules detect most attacks through statistical means, e.g. if the amount of "suspicious" connections crosses a threshold defined in the configuration file, the source IP in question is labelled as malicious. In order to depict the additional labels provided by FlowBreaker accurately, the configuration used for the experiments employs rather low thresholds. The thresholds were not configured higher because the analysis of the benign traffic in the dataset showed no false positives, so this fine-tuning was deemed unnecessary.
The results of the processing are split into three subsections, with Section 4.1.1 addressing attacks FlowBreaker was built to recognise, Section 4.1.2 addressing attacks not yet implemented in FlowBreaker and the Section 4.1.3 giving detailed information on how FlowBreaker recognises attacks in the "information-gathering" pcap file.

When comparing FlowBreaker's labels/descriptions to the ones provided by the TII-SSRC-23 dataset, we face the challenge that FlowBreaker assigns labels to individual IP addresses and flows/communications associated with them, while the TII-SSRC-23 dataset labels the entire content of a given pcap with a single classification, including

every IP address contained within. Since this makes it hard to compare the results directly, an aggregation was performed. If FlowBreaker detects an attack pattern in a given pcap, it creates a file, listing all IP addresses and flows that were deemed suspicious, e.g. if FlowBreaker detects a HTTP Brute Force attack, it will include all HTTP flows that took part in it in said file.
However, these labels are not mutually exclusive, meaning that a flow can appear in multiple output files, classified under different labels. To actually gain comparable data, all output files of FlowBreaker (if any) for a given pcap were grouped together, listing the amount of flows associated with the attack type. This is apparent in Table 3.3, 4.2 and 4.4. The pcaps provided by the TII-SSRC-23 dataset almost exclusively contain IP addresses, which either fill the attacking role or are under attack. This means that the IP addresses by themselves hold little descriptive power and were thus omitted. Instead, we took the route of counting the total amount of flows in all pcaps provided by the dataset with FlowBreaker, with the results shown in Table 4.3. Using these results, we can assess how many, out of the total flows present, were recognised as malicious by FlowBreaker. Furthermore, we will discuss if FlowBreaker was able to provide more relevant insight into its labels/descriptions.

4.1.1 Labelling Attacks implemented in FlowBreaker

As explained in Section 3.4.1, the TII-SSRC-23 dataset contains 32 pcap files, out of which 10 pcaps are presented in Table 3.3 and 4.2 and have been selected for direct evaluation using FlowBreaker. The Flow Count column next to the attack classification is the total number of connections/communications[1] that FlowBreaker associated with this attack. These values are based on the connections provided by Zeek in the “conn.log”. For comparison, the total number of connections/flows in each pcap, split by protocol, is listed in Table 3.3.

First we will start by taking a look at the attacks in the TII-SSRC-23 dataset, for which FlowBreaker offers a dedicated detection method. Looking at Table 3.3, it is apparent that, when used to search for attack patterns it was designed to detect, FlowBreaker was able to recognise at least one attack pattern in each file. Starting with the first entry in the dataset, labelled as “bruteforce_ftp”, we can see that FlowBreaker labelled this as a “Common Port Attacks” with 3470 connections.

Now, what does this label mean? When looking for simple Brute Force attacks, it is sufficient to monitor certain ports and protocols and set a limit for established connections(19).
File Transfer Protocol (FTP) (53) is usually served on port 20 and 21. As the name implies, it is used for transferring files, which often involves authentication. An attacker would thus attempt multiple user and password combinations to see if any of them are

[1]The term connections in the scope of FlowBreaker refers to TCP, UDP and ICMP communications/flows.

valid, resulting in hundreds of attempts. A legitimate usage would only result in a few connections, as connections are reused to save on overhead.

Since FlowBreaker saw 3470 flows originating from a small amount of IPs to a single destination port, it was labelled as excessive and marked as “Common Port Attacks”.

pcap/sub-label	TII-label	Attack Classifications	Flow Count
bruteforce_ftp	bruteforce	Common Port Attacks	3470
bruteforce_http	bruteforce	HTTP Brute Force Attack	331 [1]
		Slowloris	210
bruteforce_ssh	bruteforce	Common Port Attacks	1970
		SSH Brute Force	1970
bruteforce_telnet	bruteforce	Common Port Attacks	2333
information_gathering	Info.-gathering	See below	N/A
mirai_ddos_dns	mirai-botnet	UDP Flood	134016
		DNS Amplification	2 [2]
mirai_ddos_udp_udpplain	mirai-botnet	UDP Flood	1784
mirai_scan_bruteforce	mirai-botnet	Common Port Attacks	4766
		Slowloris	1270
syn_tcp_dos	dos	SYNFlood	1810966
udp_dos	dos	UDP Flood	348492

[1] HTTP Brute Force Attack: Additionally, 16358 suspicious HTTP requests
[2] DNS Amplification: Additionally, 27880 suspicious DNS requests.

Table 4.1: Attacks implemented in FlowBreaker in the TII-SSRC-23 Dataset

Attack Classification	Flow Count
Port Scans	567671
Protocol Specific Scans	72055
Service Enumeration	4433
SYN Flood	269572
UDP Flood	1404
Common Port Attacks	1362431

Table 4.2: FlowBreaker’s Classifications of the Flows contained within the “information_gathering” pcap of the TII-SSRC-23 Dataset

These “Common Port Attacks”, (as explained in Section 3.3.4) are classified by checking a list of commonly used ports for certain protocols, such as FTP, to see if any

of these ports were targeted in excess, i.e. a high number of flows/connections have been established. It is then up to the researcher to decide if the classification was justified. The same label is assigned to the pcaps "bruteforce_ssh" and "bruteforce_telnet". Both protocols use authentication and provide control access and are thus valuable targets. SSH stands for Secure Shell (54) and allows users to log into a remote machine, with Telnet (57) achieving a similar goal. Since Zeek provides a specific log file for SSH, there is also an additional subclassification by FlowBreaker named "SSH Brute Force" based on this file. As already explained for the attack targeted at FTP, these labels are assigned based on exceeded thresholds.
Moving onto the "bruteforce_http" file, FlowBreaker was once again able to recognise two attacks, but here things differ from the "Common Port Attacks". The "HTTP Brute Force Attack" has 331 connections, which at first glance, seems little compared to the former Brute Force attacks. However, in this case, a connection does not represent an attempt. The total requests (and therefore attempts) are listed as a footnote (16358). These requests are taken from Zeek's http.log file and contain "password=" in their "uri" field and are thus labelled as suspicious by FlowBreaker. This field is used for HTTP authentication (58). In this case, this means that a client uses it to authenticate itself to a server to gain access to resources. Again, such a high number of requests is very likely malicious, since a user would not access that many resources in a short time frame.
The final classification "Slowloris", with 210 associated connections, is a false label. Slowloris is an attack that works by stressing a server by keeping many connections open over a long time period, therefore preventing other clients from connecting (27). FlowBreaker recognised this by searching for connections that have a small data transfer (indicating that no useful data is transmitted) but a long duration. In this instance, connections are reused for HTTP requests containing no data but the authentication headers, which is why FlowBreaker assigned this label. However, it is up to the researcher to recognise this misclassification and potentially alter the configuration.
The next pcap file analysed, is "mirai_ddos_dns", which contains a DNS amplification attack by the mirai-botnet (45). This type of attack exploits the DNS protocol by sending spoofed requests with the target's IP address as the apparent source (20). This causes the target to be flooded with unwanted DNS responses. FlowBreaker successfully recognised this attack type by listing 27880 suspicious DNS requests. In this case, the amount of connections is misleading, as Zeek only saw two connections that were actually initiated by the source IP, even though it was flooded with DNS responses by other hosts. The second classification set by FlowBreaker is "UDP Flood", which just indicates that the amount of UDP connections made by a single host crossed a threshold. This classification is triggered because DNS uses UDP, which is why many UDP connections were seen by FlowBreaker. An UDP Flood also has the goal of overwhelming a victim with a high volume of UDP requests/connections (21). This is possible because UDP is stateless and can be used with very little effort to spam a target with unwanted data. The target then has to deny all these connections, which can be resource intensive. FlowBreaker successfully recognised this attack type in two other pcaps, namely "udp_dos" and "mirai_ddos_udp_udpplain", with "DOS" referring to "Denial of Service" and "DDoS"

to “Dedicated Denial of Service”. Both serve the same goal of overwhelming a victim with useless requests but the latter involves multiple sources of attack, such as the mirai botnet. Furthermore, they can be understood as a subtype of Distributed Denial of Service (DDoS) (23). The “mirai_scan_bruteforce” pcap is similar to the other Brute Force attacks discussed above. The only difference is that this time it was carried out by a botnet. Once again, FlowBreaker recognised these attacks correctly, with Slowloris being a false label. Before moving onto the “information_gathering” pcap, the last pcap on the list is “syn_tcp_dos”. FlowBreaker correctly identified the traffic in this file as “SYNFlood”. A SynFlood works by sending a lot of SYN packets to a victim, which initiates the TCP negotiation process (22). The victim then has to acknowledge these requests and store the connection state. The attacker never replies to the acknowledgements (SYN-ACK) sent by the victim, causing stress on the victim’s side, as it has to retain a lot of open, unfinished connections, while still answering to new ones.
Table 4.2 provides an overview of all the attack types FlowBreaker recognised within the “information_gathering” pcap. While the TII-SSRC-23 dataset only provides the label and sub-label “information_gathering”, FlowBreaker was able to provide a more detailed overview of the contents. Port Scans, Common Port Attacks, SYN Floods and UDP Floods function in the same manner as above. The new attack types not yet explained, are “Protocol Specific Scans” and “Service Enumeration”. These attacks are similar to a Port Scan (24), but go a step further: The former simply sends SYN packets to many different ports and waits for replies to see which ports are open. FlowBreaker specifically searches for connections containing only a SYN packet to label this type of attack. A Service Enumeration attack addresses the same port and protocol multiple times to check which versions of the service running on that port are available. This is useful information as certain versions are vulnerable to different types of attacks. The SYN Flood and UDP Flood are false labels in this instance. As explained above, many SYN packets are sent to gather information but in a SYN-Flood, the primary goal is overwhelming the victim. FlowBreaker still recognised this as a SYN Flood, as the behaviour is the same. The UDP Flood label is also very likely assigned due to FlowBreaker recognising UDP connections that are used to gather information as a denial of service attack. This could be remedied by setting a higher limit in the configuration file.

To summarise, FlowBreaker can recognise the attacks it was designed to recognise, while also offering a description on why it classified the attack as such. When it comes to comparing the assigned classifications to the ones provided by the TII-SSRC-23 dataset, FlowBreaker offers comparable labels or, in case of false labels, a plausible reasoning why these labels were assigned, which should make them easy for a researcher to spot and correct. Compared to the TII-SSRC-23 dataset, FlowBreaker labels on a per-IP basis, instead of classifying all flows inside a pcap with a given attack label. This provides a more fine-grained approach when searching for attack patterns in unfiltered traffic but makes it hard to compare the labels/classifications directly. We can thus conclude, that the limits based approach used in FlowBreaker’s detection methods, as described in Section 3.3.4, is indeed capable of filtering attack patterns. Furthermore, although omitted

in this section for reasons of simplicity, FlowBreaker's output files put its classifications into context, i.e. a researcher can use the Average Values and Flags described in Section 3.3.6 to further assess whether a given host's activity is anomalous/malicious or not.

pcap/sub-label	TII-label	TCP	UDP	ICMP
ack_tcp_dos	dos	944,204	0	0
cwr_tcp_dos	dos	859,356	0	0
ecn_tcp_dos	dos	862,067	0	0
fin_tcp_dos	dos	1,276,452	0	0
http_dos	dos	41,399	0	0
icmp_dos	dos	0	0	7
mac_dos	dos	1,919	0	0
psh_tcp_dos	dos	899,906	0	0
rst_tcp_dos	dos	1,663,731	0	0
udp_dos	dos	0	348,623	0
urg_tcp_dos	dos	889,464	0	0
bruteforce_dns	bruteforce	0	24,061	0
bruteforce_ftp	bruteforce	3,470	0	0
bruteforce_http	bruteforce	331	0	0
bruteforce_ssh	bruteforce	1,970	0	0
bruteforce_telnet	bruteforce	2,333	0	0
audio	benign	62	17	7
background	benign	3	11	7
video_http	benign	191	8	8
video_rtp	benign	3	248	8
video_udp	benign	3	117	8
text	benign	80	39	5
information_gathering	recon	1,363,105	1,404	44
mirai_ddos_ack	mirai-botnet	9,529	37	4
mirai_ddos_dns	mirai-botnet	10	134,038	1
mirai_ddos_greeth	mirai-botnet	10	6,844	5
mirai_ddos_greip	mirai-botnet	10	4,008	5
mirai_ddos_http	mirai-botnet	5,896	94	1
mirai_ddos_syn	mirai-botnet	30,527	83	5
mirai_ddos_udp_udpplain	mirai-botnet	12	1,862	5
mirai_scan_bruteforce	mirai-botnet	4,791	373	8

(The values provided are the sum of all connections/flows of all unique hosts contained in each pcap)

Table 4.3: Total amount of Connections/Flows in every pcap of the TII-SSRC-23 Dataset as analysed and calculated by FlowBreaker

4.1.2 Labelling Attacks not implemented in FlowBreaker

This section details how FlowBreaker recognised attacks for which no specific detection mechanism had been implemented. For 14 of the 21 pcaps listed in Table 4.4, FlowBreaker provides at least one classification, alongside the number of flows contained in the pcap that were classified as such. For simplicity reasons, the output files were again aggregated, omitting the individual IP addresses, as those are not relevant in the scope of the TII-SSRC-23 dataset — instead, the total flows responsible for the classification are listed. FlowBreaker does not perform any deeper analysis of traffic contents, i.e. it will not output any "benign" labels if the input was not classified as an attack. The reason why no classification is given for "icmp_dos" and "mac_dos", is that the attacks are not accurately captured by Zeek, as ICMP traffic was omitted in the output and Medium Access Control (Address) (MAC) traffic is out of scope, resulting in FlowBreaker being unable to give more insight. This is also evident in Table 4.3, with FlowBreaker only listing seven ICMP flows for the former and zero flows for the latter.

All attacks present in Table 4.4 have been explained already, with the only exception being "Connection Exhaustion", a label defined by FlowBreaker 3.3.4. This attack has the goal of depleting the resources of a victim. FlowBreaker checks for this type of attack by searching for connections that have a high duration, but have minimal data transfer. If the amount of these connections exceeds a threshold, the label is assigned. Unlike the submethod for detecting a Slowloris attack, which only includes unterminated TCP connections, this submethod includes all TCP connections meeting above mentioned criteria.
Since (Dedicated) Denial of Service attacks mostly work by generating useless and long lasting traffic, it makes sense that FlowBreaker assigned this label to the pcaps labelled as "dos" by the dataset. Interestingly, FlowBreaker assigned the "UDP Flood" label to the "bruteforce_dns" pcap. DNS Brute Force in the scope of the dataset means that a server is checking specific subdomains of a domain, such as "database.domain.com", to find potential targets. This generates a high amount of DNS requests. Since DNS utilises mostly UDP, FlowBreaker assigns this label based on the high volume of UDP packets seen.
Without addressing each pcap file listed in Table 4.4 individually, it is apparent that FlowBreaker is able to provide insight into the traffic by offering classifications due to the attacks contained within them sharing symptoms with more specialised attacks: Even though the label "Service Enumeration" was not initially meant to detect DoS attacks, it is useful for the recognition of the repeated targeting of ports prevalent in these types of attacks. The same is true for FlowBreaker recognising hosts that send a lot of unwanted SYN packets, labelling such cases as "SynFlood" or with enough port variety "Port Scans" as evident in the "mirai_ddos_syn" pcap.
Another case of unintended use is the "Slowloris" label. Originally meant for identifying attacks that aim at blocking resources, it is also useful for detecting "unproductive" connections that do not carry any data, but remain open for a long time. Since many attacks share this pattern, this indicates that this detection label could be expanded into

a more general indicator when searching for malicious traffic.
To summarise, even when used on traffic data containing attack patterns it was not designed to recognise, FlowBreaker is still able to correctly label this traffic as malicious. Furthermore, the labels assigned by FlowBreaker are justified based on patterns/measurements present in the dataset, meaning that they are verifiable by the researcher using FlowBreaker.

Looking at Table 4.4, it is also evident that FlowBreaker did not output any false labels for the pcaps classified as "benign" by the dataset, resulting in zero labelled flows and no classification given.

pcap/sub-label	TII-label	Attack Classifications	Flow Count
ack_tcp_dos	dos	Connection Exhaustion	5833
		Service Enumeration	233
bruteforce_dns	bruteforce	UDP Flood	24061
cwr_tcp_dos	dos	Connection Exhaustion	6492
		Service Enumeration	215
ecn_tcp_dos	dos	Connection Exhaustion	6161
		Service Enumeration	221
fin_tcp_dos	dos	Service Enumeration	301
http_dos	dos	Slowloris	39327
		SYNFlood	2065
icmp_dos	dos	-	-
mac_dos	dos	-	-
mirai_ddos_ack	mirai-botnet	Port Scans	3727
mirai_ddos_greeth	mirai-botnet	UDP Flood	4390
mirai_ddos_greip	mirai-botnet	UDP Flood	1883
mirai_ddos_http	mirai-botnet	Slowloris	742
mirai_ddos_syn	mirai-botnet	Port Scans	4617
		SynFlood	8696
psh_tcp_dos	dos	Connection Exhaustion	6957
		Service Enumeration	216
rst_tcp_dos	dos	Common Port Attacks	1324
		Port Scan	1311
urg_tcp_dos	dos	Connection Exhaustion	6129
		Service Enumeration	208
audio	benign	-	-
background	benign	-	-
video_http	benign	-	-
video_rtp	benign	-	-
video_udp	benign	-	-
text	benign	-	-

Table 4.4: Results of analysing attacks and benign traffic in the TII-SSRC-23 Dataset not implemented in FlowBreaker: Listing Classification and associated Flow Count

4.1.3 Detecting Malicious Traffic in the "Information Gathering" pcap

Compared to the labels provided in the original dataset, FlowBreaker's output is much more detailed and provides additional reasoning to the user, including the calculations and thresholds used for determining the type of attack. Additionally, statistical labels are provided, which are absent in the dataset and would have to first be calculated manually.

We will first discuss the additional labels, to then reason how they can be leveraged to improve attack detection. In detail, FlowBreaker provides the following additional calculations, as discussed in Section 3.3.5:

- **Outgoing IP/Port activity** — Indicates if the host uses a high number of different IPs/Ports for outgoing connections/flows, in comparison to the other hosts in the pcap file.
- **Incoming IP/Port activity** — Indicates if the host uses a high number of different IPs/Ports for incoming connections/flows, in comparison to the other hosts in the pcap file.
- **Unique IPs and their flow/connection count** — How many flows are associated with each Source/Destination IP.
- **Unique Ports and their flow/connection count** — How many flows are associated with each Source/Destination Port.
- **Average Bytes transferred per Connection** — How much data was transferred per connection/flow on average for a given host.

Furthermore, the following labels are provided:

- **Classification** - The subtype of the attack in question.
- **Reason** - Why this traffic was flagged as malicious.
- **Services** - The high level protocols (e.g. HTTP, SSH) used.

Using the calculations above, FlowBreaker gives a more detailed overview of the traffic within a pcap. This is apparent when looking at the"Information_Gathering" pcap of the original dataset: With the labels provided being:

1. Label: Malicious
2. Traffic Type: Information Gathering
3. Traffic Subtype: Information Gathering

FlowBreaker is able to subdivide these labels further, as seen in Table 4.2. This section focuses on this individual pcap to explain how FlowBreaker uses different parameters to arrive at a potential classification. Using these parameters, we will discuss how malicious traffic can be distinguished from benign traffic. The key parameters used for the classifications in Table 4.2 are:

- **Number of S0 States** — S0 is a label assigned by Zeek, which means that only a SYN packet was sent during a TCP connection.
- **Number of REJ States** — REJ is a label assigned by Zeek, which means that the SYN packet was rejected at the destination.
- **Unique Port Numbers Used** — The average number of unique ports gives insight on how many different services a client accesses on a target machine. If this value exceeds a certain threshold, e.g. ≥ 100, it can be assumed that these connections are malicious and an attack such as a Port Scan is taking place.
- **Connections per Port** — The average number of connections per port is often inversely correlated to the number of unique ports: A legitimate client often accesses a single resource on a target machine, e.g. a video stream through a web browser. This means that the number of unique ports would be one, as only one port is necessary for HTTP, additionally the number of connections per port would be high, as the same port is addressed multiple times when streaming video.
- **Services Used** — The services used by a host can shed light on its intentions: A high variety could hint at a Service Enumeration or other types of scanning attacks.
- **Bytes Transferred** — The average number of bytes transferred is of interest, because it serves as a measurement of the "productiveness" of the connections/flows associated with a host. When transmitting files or streaming video, this value would be very high, but in a SynFlood or Port Scan it would be very low.

With these indicators it is now possible to compare the contents of the benign pcap "video_http" to those of the malicious pcap "information_gathering". In the "video_http" pcap, a single client streams video from a single source using http. In the "information_gathering" pcap, a single client leverages different attack patterns to gain information on a target client/network. After analysing both pcaps with FlowBreaker, the results were compiled in Figures 4.1, 4.2, 4.3 and 4.4. The results can be understood as follows: Since only one relevant IP address is contained in each pcap, the output of FlowBreaker has been aggregated to compare the parameters individually in each figure. This means that Figure 4.1 compares the amount of *Total Connections* (i.e. flows) associated with the relevant IP in both pcaps, Figure 4.2 the amount of *Unique Destination Ports* targeted by the relevant IP, Figure 4.3 the *Average Bytes* transferred per flow/connection between the host and the target, and finally, Figure 4.4 compares *S0 and REJ States* between the malicious and the benign client. When looking at benign

traffic in general, both the source and destination try to conserve resources, leading to the following behaviour: TCP connections are established properly and with purpose, thus the amount of SYN only connections, or S0/REJ States, is ideally zero. Moreover, the client will only target specific destination ports, commonly in the single digit area for a single purpose. Furthermore, since connection establishment and teardown takes time, the amount of individual connections is lower, with a high amount of transferred bytes. In malicious traffic, we can expect the opposite to be true: A Port Scan will try to look for open ports, thus targeting a high amount of destination ports, while transferring little to no data (24). Similarly, we can expect a high number of initiated connections, if an attacker is scanning for other hosts in the network — as none of these connections is wanted nor expected at the destination, many of them will be rejected (25).

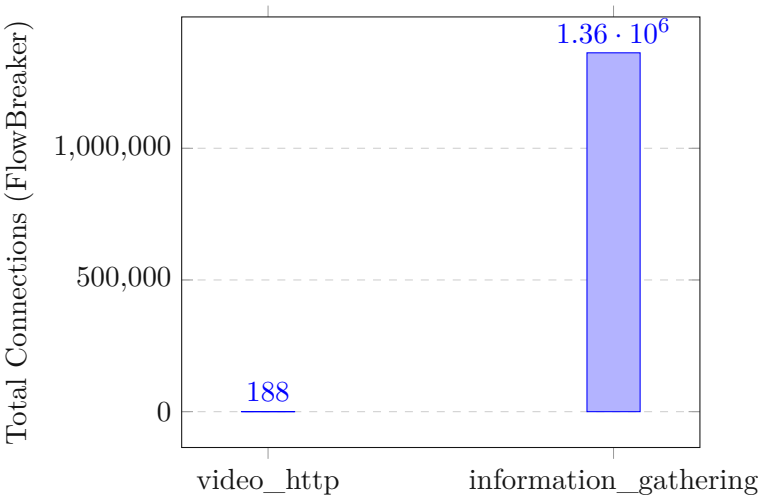

Figure 4.1: Comparison of Total Connections counted by FlowBreaker in the pcaps containing HTTP Video (benign) and Information-Gathering (malicious)

In Figure 4.1, it is apparent that in the "video_http" pcap, the benign client establishes only 188 connections, while the malicious client in the "information_gathering" pcap is several magnitudes above that number. Both pcaps contain only one source IP, thus we can use this measure as a direct comparison. This hints at suspicious activity, as normally connections/flows are not established blindly, but rather with purpose. The high number of connections/flows associated with a single IP in "informationg_gathering" hints at a network mapping taking place. In such an attack, the client targets many different IP addresses, to see which of them respond, thus creating a map of the network **??**. With FlowBreaker providing the total amount of connections/flows for every analysis it performs, it is possible to quickly recognise suspicious IP addresses in a given pcap (though in this case only one IP is present). To achieve the same result by using the labels of the TII-SSRC-23 dataset, it would be necessary to count every flow and associate it to an IP. We will look at the *Number of Unique Destination Ports* next to see how many

ports were targeted.

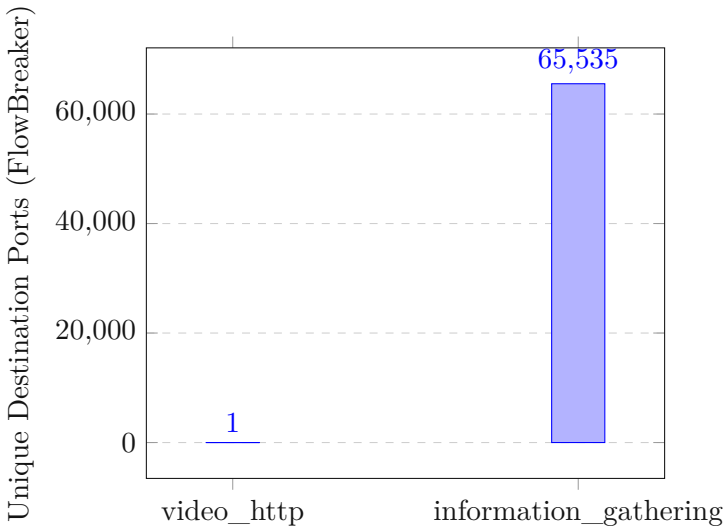

Figure 4.2: Comparison of Unique Destination Ports counted by FlowBreaker in the pcaps containing HTTP Video (benign) and Information-Gathering (malicious)

Looking at Figure 4.2, we see that the client in the "video_http" pcap only targets a single destination port. This is to be expected, as the client only accesses a single service using HTTP on a single destination. Meanwhile for gathering information, the client accesses as many ports as possible with the intent of discovering open ports — reaching the maximum amount of addressable unique ports 65.535. FlowBreaker provides the value of *Unique Destination Ports* for every IP address in a given pcap, thus highlighting malicious activity — in this specific case, we can directly compare the single IPs present in both pcaps. This leads us to the conclusion, that this value can help in discovering suspicious activity, which is also the reason why the *Unique Destination Ports* parameter is used by FlowBreaker to identify possible Port Scans taking place. This parameter is also unique to FlowBreaker and not present in the TII-SSRC-23 dataset. Knowing that the source IP in the "information_gathering" pcap targets a high number of destination ports and has a very high number of connections/flows associated with it, we will now take a look at the amount of data transferred.

FlowBreaker calculates the *Average IP Bytes per Connection*, which can be understood as the amount of data transferred on average per connection/flow, including the IP Packet Headers (i.e. not just application data is counted). We can leverage this information to gauge how "productive" the connections/flows associated with a given IP are — if the value is very low, little data is transferred, causing overhead. As this calculation is performed by FlowBreaker, it would again be necessary to iterate over every flow in a pcap to achieve the same result when using pre-existing labels. This becomes apparent in Figure 4.3, which compares the averages of both pcaps. For HTTP Video, the amount

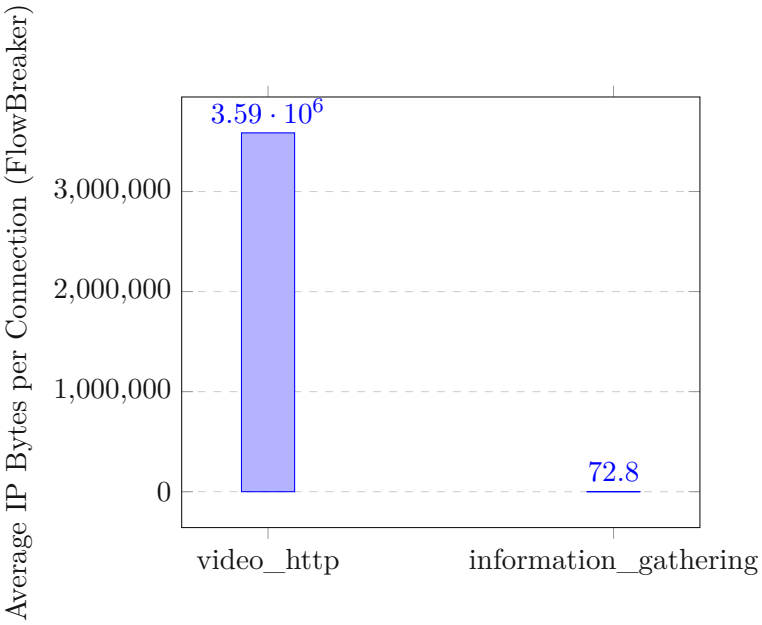

Figure 4.3: Comparison of Average Bytes per Connection (including Headers) calculated by FlowBreaker in the pcaps containing HTTP Video (benign) and Information-Gathering (malicious)

of bytes transferred is very high, because large amounts of data are streamed over a single connection. Contrary to that, the average in the Information Gathering pcap is very low. This is due to the Port Scans taking place: Here no data is transmitted, as the attacker only listens for the SYN-ACK response, indicating an open port. Using this information provided by FlowBreaker, we can further conclude that the client in the "information_gathering" pcap uses a lot of resources, but transfers little data, which is undesirable in most applications.

Figure 4.4 compares the S0 and REJ States in both pcaps. In the legitimate traffic, no S0 state was seen, as the used port is open and thus there should always be a SYN-ACK response. A Port Scan addresses many unavailable ports by design, thus resulting in the 83,028 S0 States of the Information Gathering pcap. The high number of REJ States is due to the same reason, with the difference that the connection attempt is actively rejected at the destination (e.g. because no service is running on that port, or the firewall is closed). The 46 REJ States that occurred in the HTTP Video stream can be explained by connection issues or timeouts in the TCP protocol and are to be expected in low quantities.
Using the information provided by FlowBreaker about the S0 and REJ states, it is possible to conclude that almost all connections/flows initiated by the malicious client in the "information_gathering" pcap are unwanted at the destination. S0 and REJ states are assessed by Zeek, with FlowBreaker counting the occurences and attributing them

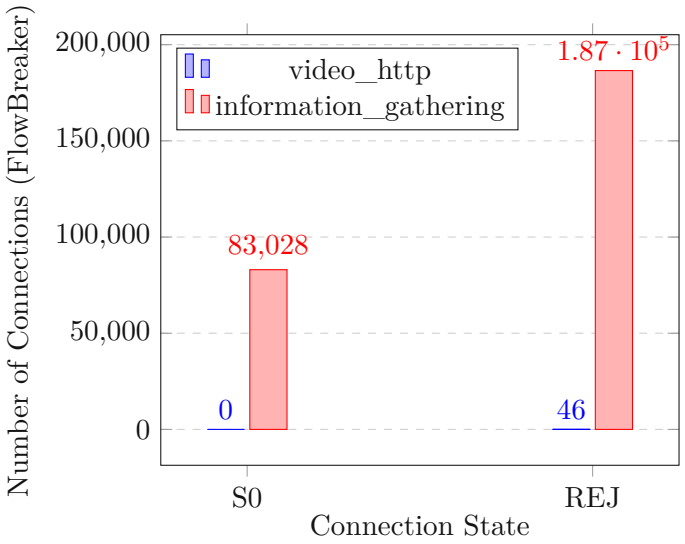

Figure 4.4: Comparison of S0 and REJ Connections counted by FlowBreaker in the pcaps containing HTTP Video (benign) and Information-Gathering (malicious)

to individual IPs. To achieve a similar descriptive power by using these values through other tools would prove difficult, as it involves filtering for SYN packets and tracking the state of TCP connections through an entire pcap.
To summarise, the TII-SSRC-23 dataset does not reflect the information provided in the diagrams in this section in its labels. In order to extract it, each flow in the dataset would need to be iterated, counting the ports used, bytes transmitted and requests made. Also, since the connection state identification is done by Zeek, it is not directly available from the dataset. Therefore, to achieve the same descriptive power as FlowBreaker, the labels provided by the dataset would need to be processed again and cross-referenced with information gathered by Zeek after processing the pcap files. This leads us to the conclusion that FlowBreaker offers a high amount of convenience to the user when assessing the contents of a given pcap, as the provided parameters prove to be relevant in the benchmark environment. Furthermore, FlowBreaker offers insight into its justification for classifications, making it easier for researchers to interpret the results when compared to the labels provided by the TII-SSRC-23 dataset.

4.2 Evaluating FlowBreaker on a MAWI Archive Traffic Capture

Given that it is uncertain whether the pcap provided by the MAWI Archive contains any attacks, it is interesting to see if FlowBreaker labels any of the traffic as malicious. This section first discusses the output generated by FlowBreaker, to then evaluate the findings and provide reasoning on their validity.

4.2.1 General Statistics provided by FlowBreaker

With Section 4.1 focusing on labelling attacks, one key feature of FlowBreaker still remains to be highlighted. FlowBreaker generates very detailed, human readable logs, which can be used as reference when analysing a traffic capture. Table 4.5 lists the statistics FlowBreaker provides during runtime when analysing the provided pcap from the MAWI archive. This information is useful to assess the behaviour of a given IP by comparing its individual values to the average values of the pcap.

Metric	TCP	UDP
Total Source IPs	22,659	10,851
Total Destination IPs	152,770	5,931
Total Connections	258,114	104,170
Avg. unique destination IPs	10.900	2.009
Avg. unique source IPs	1.617	3.675
Avg. unique destination ports	1.595	1.000
Avg. unique source ports	7.251	9.469
Avg. connections per destination IP	11.393	9.600
Avg. connections per source IP	1.690	17.564

Table 4.5: TCP and UDP Connection Statistics, provided by FlowBreaker

To assess the potential of FlowBreaker, let us analyse the output it generates when processing a randomly selected IP (here, 36.182.230.11). These are shown in Listing 4.1. Note that the IP in question connected to 12291 unique destination IPs. FlowBreaker lists such destination IPs together with their respective connection counts; however, we omit this information to avoid taking up an unnecessarily large amount of space for our example. Additionally, the Source and Destination Ports are also listed, with the number of connections associated to them.
In this case, it is obvious that the IP connected to port 6379 every time. If this were an attack, this could be a command and control server, but it can also be done for legitimate reasons, e.g. a centralised update. For further confirmation, a researcher could then look at the connection states of that IP in more detail, for example whether or not each destination replied to the connection attempt.
Another useful indicator is the average amount of IP bytes transferred per connection, which is calculated for each individual IP by FlowBreaker. In this instance it is 60, indicating that the connections are all identical (because it is not a fraction — this can happen when all connections share the same value). Furthermore, it shows that very little information has been transferred; this could indicate a synchronisation protocol or an attacker scanning if the port is open.

Finally, port 6379 serves the Redis service (64), which is an in-memory database[2], making it a potential target for attacks. Unfortunately, without external labels, it is hard to verify whether the use is actually nefarious or legitimate.

In Table 4.5, it can be seen that FlowBreaker provided information about TCP and UDP connections/flows, but none about ICMP flows, even though almost 25% of the traffic capture consists of them. This is due to Zeek's pre-processing. After consulting Zeek's logs, it became apparent that Zeek recognised several hosts running a trace-route[3] using ICMP and thus omitted these packets from its final output. This behaviour would have to be changed in the script used to run Zeek (mentioned in Section 3.2.2) when processing a pcap file through the frontend, which could be achieved by simply telling Zeek to include ICMP based communications directly in its output instead of overriding them with its internal analysis. This is thus not an inherent limitation to FlowBreaker and could be solved in future versions of the frontend. However, since the problem arose at the end of the evaluation phase, it was not addressed in order to keep the testing environment consistent.

Looking at the other statistics provided in Table 4.5, we cannot verify them directly using the information provided in Table 3.5, since FlowBreaker works with flows and the reference data is based on individual packets.
As evident in this section, the output in Listing 4.1 is useful for assessing individual IPs. The fields not yet addressed are: Activity Flags and Flagging Scores. The Activity Flags can be seen as a binary indicator whether or not an IP has a high amount of activity, compared to the rest of the capture file. This is explained in more detail in Section 3.3.6, but their main purpose is to quickly filter if an IP is an outlier or not.
As mentioned in Section 3.3.6, the Flagging Scores simply serve as the basis for these flags, indicating the ratio of the current IP's value to the average value (note that "-1" indicates that the value could not be calculated but can be found in the other output file provided by FlowBreaker, which sorts by source IP). In this instance, the Outgoing Port Activity is 1138,715 times higher than the average, indicating that this IP is very clearly an outlier that should be investigated further.

[2]in-memory database: A database stored in the Rapid Access Memory (RAM) for faster access times.
[3]Traceroute is a command used for mapping a network's topology (51).

```
IP: 36.182.230.11
Total Connections: 12291
Protocol: TCP
Service: UNDEF
Classification: UNDEF
Connection Summary:
    Unique Destination IPs: 12291
            Destination IPs:
                    150.66.15.241: 1 connections
                    150.66.15.247: 1 connections
                    150.66.15.225: 1 connections
                    ...
    Unique Destination Ports: 1
            Destination Ports:
                    6379: 12291 connections
    Unique Source Ports: 8257
            Source Ports:
                    32770: 1 connections
                    32774: 1 connections
                    32776: 1 connections
                    ...
Services:
Activity Flags:
    High Outgoing Port Activity: True
    High Incoming Port Activity: False
    High Outgoing IP Activity: True
    High Incoming IP Activity: False
    High Number of Outgoing Connections: True
    High Number of Incoming Connections: False
    Listener: False
    Speaker: True
Flagging Scores (Current Value/Average Value):
    Outgoing Port Activity: 1138,7146
    Incoming Port Activity: -1
    Outgoing IP Activity: 1127,5695
    Incoming IP Activity: -1
    Outgoing Connections: 1078,862
    Incoming Connections: -1
Average Values:
    Connections per Destination IP: 1
    Connections per Source IP: 12291
    Connections per Destination Port: 12291
    Connections per Source Port: 1,4885552
    Connections per Unique IP: 1
    Bytes transferred per Connection: 0
    IP Bytes transferred per Connection: 60
```

Listing 4.1: FlowBreaker's Output for IP 36.182.230.11 in the MAWI traffic capture

FlowBreaker also outputs data using the csv format, which is useful for processing data further. The top 10 entries of FlowBreaker's csv formatted output can be seen in the following Tables: Table 4.6 lists the 10 most active source IPs using the UDP protocol, ranked by the amount of connections/flows associated with them. Similarly, Table 4.7 lists the 10 most active destination IPs using the UDP protocol, also ranked by the amount of connections/flows associated with them. The same has been done for TCP, listing the most active sources in Table 4.8 and the most targeted destinations in Table 4.9.
These tables can be used as reference by a researcher for plotting data and to quickly look up which IP addresses are the most active (which usually indicates relevance). For example, it can be seen that the most active IP using UDP in Table 4.7 serves a total of 17,738 connections, probably indicating a DNS server. When investigating the top IP in Table 4.8, it seems that each destination IP is only targeted once. This, combined with the fact that the same port is targeted every time, suggests that this IP is relevant. Looking at Listing 4.1, we can see that IP 36.182.230.11 targets Port 6379 exclusively. We can thus conclude that this IP is likely carrying out malicious activity, likely scanning multiple hosts for Port 6379. Unlike the top IP in Table 4.8, the top IPs in Table 4.6 and 4.9 do not immediately reveal useful information that could be used as an indicator for malicious activity. However, these tables are useful for demonstrating how the top five IPs account for far more connections than the remaining entries.

After highlighting FlowBreaker's capabilites of providing an overview, the next section addresses attack patterns found by FlowBreaker in the MAWI traffic capture.

IP	Total Connections	Unique Dest IPs	Unique Source IPs	Unique Dest Ports	Unique Source Ports
203.178.182.222	3,009	889	1	1	2,934
2001:4230:f7cc:e60c: 202:4a00:e79a:a54a	2,671	615	1	1	2,538
203.178.180.55	2,151	330	1	1	2,079
131.113.133.45	1,009	182	1	1	979
131.113.133.43	965	149	1	1	941
2001:4d70:f2ed:beb4: 702:7c0b:cc7f:fc64	904	178	1	1	878
2001:4d70:f097:9ed: 7809:77ce:3ffe:643	742	79	1	1	737
2001:4d70:f097:9ed: 7809:77ce:3ffe:647	702	101	1	1	695
61.166.65.100	630	1	1	1	630
2404:93cf:7f8:e201: fdee:c17f:98df:8465	569	2	1	1	569

(Top 10 Source UDP IPs sorted by total connections)

Table 4.6: Top 10 Source UDP IPs, based on FlowBreaker's Analysis

IP	Total Connections	Unique Dest IPs	Unique Source IPs	Unique Dest Ports	Unique Source Ports
2001:4230:81d:fde7: ff0f:8019:f3fe:3934	17,738	1	3,124	1	14,712
192.12.106.202	15,529	1	2,781	1	13,626
202.243.164.209	9,323	1	1,016	1	8,527
192.151.56.21	6,101	1	109	1	4,621
192.151.56.54	6,089	1	99	1	4,593
2620:7be:71ec:1fc: 9f2:820:307e:106	5,934	1	90	1	4,482
2620:7be:71ec:1fc: 9f2:820:307e:141	5,744	1	74	1	4,305
202.243.162.173	3,081	1	470	1	2,997
2001:4d51:ffc4:3ef: f105:7bef:fc7e:3a3b	1,429	1	202	1	1,392
88.166.193.213	889	1	4	1	876

(Top 10 Destination UDP IPs sorted by total connections)

Table 4.7: Top 10 Destination UDP IPs, based on FlowBreaker's Analysis

IP	Total Connections	Unique Dest IPs	Unique Source IPs	Unique Dest Ports	Unique Source Ports
36.182.230.11	12,291	12,291	1	1	8,257
141.157.180.187	3,835	3,829	1	2	84
141.157.180.105	3,753	3,753	1	1	1
185.90.3.177	3,605	3,569	1	1	3,389
193.21.207.59	3,426	3,334	1	3	3,191
193.84.89.48	3,415	3,325	1	1	3,232
185.141.3.226	3,372	3,372	1	1	3,165
193.21.207.47	2,988	2,953	1	3	2,842
172.40.178.50	2,348	2,348	1	1	2,257
193.167.179.5	2,327	2,315	1	1	2,232

(Top 10 Source TCP IPs sorted by total connections)

Table 4.8: Top 10 Source TCP IPs, based on FlowBreaker's Analysis

IP	Total Connections	Unique Dest IPs	Unique Source IPs	Unique Dest Ports	Unique Source Ports
131.112.115.241	677	1	136	5	358
203.178.182.97	497	1	228	4	476
163.93.12.153	479	1	76	2	471
203.178.184.83	422	1	117	4	415
150.66.56.65	404	1	145	7	391
185.141.3.245	358	1	358	355	1
2001:4d70:f2ed:beac: fff9:7bd4:3ff:61ed	209	1	52	66	174
202.116.68.67	190	1	51	3	143
23.109.46.227	172	1	76	1	76
2001:4230:81d:fde7: ff0f:8019:f3fe:3934	156	1	148	2	156

(Top 10 Destination TCP IPs sorted by total connections)

Table 4.9: Top 10 Destination TCP IPs, based on FlowBreaker's Analysis

4.2.2 Addressing Attack Patterns found by FlowBreaker

As explained in Section 3.3.4 when scanning for Brute Force attacks, FlowBreaker has a conservative tendency to label some traffic with specific characteristics as UDP Flood, which may be wrong in some cases. Since this label is assigned based on the number of UDP flows, this can be mitigated by simply raising the limits.

Looking at Table 4.10, it is reasonable to think that not all entries listed are experiencing an UDP Flood, but they are most likely higher level DNS servers[4].
This shows that FlowBreaker's configuration needs to be adapted to each use case and also highlights the differences between traffic in smaller networks and traffic at a backbone.

IP	Total Connections	UDP Connections	DNS Queries	Unique Dest IPs	Unique Source IPs	Unique Source Ports
2001:4230:81d:fde7: ff0f:8019:f3fe:3934	17,738	17,738	23,989	1	3,124	14,712
192.12.106.202	15,529	15,529	16,972	1	2,781	13,626
202.243.164.209	9,323	9,323	9,512	1	1,016	8,527
192.151.56.21	6,101	6,101	8,991	1	109	4,621
192.151.56.54	6,089	6,089	9,040	1	99	4,593
2620:7be:71ec:1fc: 9f2:820:307e:106	5,934	5,934	8,710	1	90	4,482
2620:7be:71ec:1fc: 9f2:820:307e:141	5,744	5,744	8,646	1	74	4,305
202.243.162.173	3,081	3,081	3,072	1	470	2,997
2001:4d51:ffc4:3ef: f105:7bef:fc7e:3a3b	1,429	1,429	1,428	1	202	1,392

(All IPs involved in UDP Flood attack are shown)

Table 4.10: UDP Flood Attack Statistics, based on FlowBreaker's Analysis

[4]The DNS system is hierarchical, with the top handling a higher volume of requests

Table 4.11 addresses entries which were labelled by FlowBreaker as Password Spraying. Password Spraying is the practice of trying (different) passwords on many different ports/hosts (26). FlowBreaker detects this by watching a port list in the configuration used, this includes Ports: 21 (FTP), 22(SSH), 23(Telnet), 3389(RDP[5]) and 5900 (VNC[6]). If more than 50 connections are made to one of these ports, the label is assigned to the IP in question. Even for a backbone, it is unusual to have a single IP connect to that many different destinations using the protocols mentioned above.
It is thus very likely that Table 4.11 indeed lists IPs scanning for potential victims, especially because the number of unique destination ports is 1.

IP	Total Connections	Unique Dest IPs	Unique Source IPs	Unique Dest Ports	Unique Source Ports
172.40.178.50	2,348	2,348	1	1	2,257
87.135.246.132	1,911	1,911	1	1	1,843
141.157.180.137	1,840	1,825	1	1	1,785
87.135.240.77	319	318	1	1	317
198.28.88.199	216	216	1	1	1
205.48.29.63	210	210	1	1	1
87.119.147.4	193	193	1	1	1
205.48.29.221	184	184	1	1	1
205.48.29.145	182	182	1	1	1
211.152.206.223	182	127	1	1	1

(Top 10 IPs out of 50 results)

Table 4.11: Password Spraying Attack Statistics, based on FlowBreaker's Analysis

Listing 4.2 contains an excerpt of FlowBreaker's output file for Password Spraying. It can be seen that all connections were initiated to port 5900, with little actual data being transferred (just 40.324 Bytes on average). This, paired with the high Flagging Scores of 311.2606, 215.4042, 206.09944, expressing a large amount of connections compared to other IPs in the pcap, confirms the suspicion indicated by the information in Table 4.11, that this host is indeed Password Spraying (or at least looking for machines with an open 5900 port).

[5]Remote Desktop Protocol (RDP) is a protocol used by Windows for remote access
[6]Virtual Network Computing (VNC) is also a protocol for remote access

```
IP: 172.40.178.50
Total Connections: 2348
Protocol: TCP
Service: UNDEF
Classification: Password Spraying
Reason:
Attempts to many unique destinations: 2348
SSH attempts: 0
SSL attempts: 0
HTTP attempts: 0
Connection Summary:
    Unique Destination IPs: 2348
            Destination IPs:
                    203.178.205.253: 1 connections
                    203.178.172.34: 1 connections
                    203.178.254.47: 1 connections
            ...
    Unique Destination Ports: 1
            Destination Ports:
                    5900: 2348 connections
    Unique Source Ports: 2257
            Source Ports:
                    32770: 1 connections
                    32771: 1 connections
                    32780: 1 connections
            ...
    Services:
    Activity Flags:
            High Outgoing Port Activity: True
            High Incoming Port Activity: False
            High Outgoing IP Activity: True
            High Incoming IP Activity: False
            High Number of Outgoing Connections: True
            High Number of Incoming Connections: False
            Listener: False
            Speaker: True
    Flagging Scores (Current Value/Average Value):
            Outgoing Port Activity: 311,2606
            Incoming Port Activity: -1
            Outgoing IP Activity: 215,4042
            Incoming IP Activity: -1
            Outgoing Connections: 206,09944
            Incoming Connections: -1
    Average Values:
            Connections per Destination IP: 1
            Connections per Source IP: 2348
            Connections per Destination Port: 2348
            Connections per Source Port: 1,040319
            Connections per Unique IP: 1
            Bytes transferred per Connection: 0
            IP Bytes transferred per Connection: 40,32368
```

Listing 4.2: FlowBreaker's Output for Password Spraying in the MAWI traffic capture

Table 4.12 lists all Host Discovery Scans identified by FlowBreaker. Due to the high traffic volume to single servers, this label is not as indicative as it would be in other scenarios (e.g. when inspecting the TII-SSRC-23 dataset). Comparing this information with Table 4.13, we can see that the top four IPs of both tables are the same.
This is relevant because Table 4.13 addresses Protocol-Specific Scans, which work by detecting SYN-only connections. This indicates that a connection is unwanted by the destination host. Given that all connections of these 4 top IPs are SYN-only, we can conclude that these hosts are malicious and/or performing some kind of active information gathering.
However, we cannot confirm if these hosts are indeed scanning for hosts or for a single open port (except for address 172.40.178.50, which was confirmed targeting port 5900 in Listing 4.2). In any case, exclusively having SYN-only connections, all IPs listed in Table 4.13 are highly suspicious of carrying out undesired network traffic activities.

IP	Total Connections	Unique Dest IPs	Unique DNS Queries	Unique Source IPs	Unique Source Ports	Avg Connections per Dest IP
36.182.230.11	12,291	12,291	0	1	8,257	1.0000
141.157.180.187	3,835	3,829	0	1	84	1.0016
141.157.180.105	3,753	3,753	0	1	1	1.0000
185.90.3.177	3,605	3,569	0	1	3,389	1.0101
193.21.207.59	3,426	3,334	0	1	3,191	1.0276
193.84.89.48	3,415	3,325	0	1	3,232	1.0271
185.141.3.226	3,372	3,372	0	1	3,165	1.0000
193.21.207.47	2,988	2,953	0	1	2,842	1.0119
172.40.178.50	2,348	2,348	0	1	2,257	1.0000

(Top 10 IPs out of 32 results)

Table 4.12: Host Discovery Scan Attack Statistics, based on FlowBreaker's Analysis

IP	Total Connections	SYN-only Connections	Failed SSL Handshakes	Unique Dest IPs	Unique Dest Ports	Unique Source Ports
36.182.230.11	12,291	12,291	0	12,291	1	8,257
141.157.180.105	3,733	3,733	0	3,733	1	1
141.157.180.187	3,695	3,695	0	3,695	1	1
185.90.3.177	3,519	3,519	0	3,519	1	3,314
185.141.3.226	3,325	3,325	0	3,325	1	3,130
193.84.89.48	3,267	3,267	0	3,267	1	3,106
193.21.207.59	3,264	3,264	0	3,264	1	3,069
193.21.207.47	2,930	2,930	0	2,930	1	2,795
172.40.178.50	2,329	2,329	0	2,329	1	2,240
193.167.179.5	2,294	2,294	0	2,294	1	2,203

(Top 10 IPs out of 32 results)

Table 4.13: Protocol-Specific Scan Attack Statistics, based on FlowBreaker's Analysis

The last attack type that FlowBreaker detected is Port Scans. In Table 4.14, several IPs are listed with all of their connections labelled as SYN-only (S0) or rejected connections (REJ).
While some of these connections are not unusual, the entries listed in this table seem to be looking for open ports. Compared to the results in Tables 4.12 and 4.13, the amount of unique destination ports is higher this time, indicating that the list of scanned ports is longer.

IP	Total Connections	Unique Dest IPs	Unique Dest Ports	Unique Source Ports	S0	REJ
83.192.141.160	1,823	1,810	52	1	1,813	10
154.170.107.192	1,758	1,753	865	2	1,746	11
23.123.132.156	1,742	1,729	29	1,697	1,730	12
83.192.141.42	1,290	1,283	456	1	1,287	3
92.7.86.251	1,063	1,056	173	1	1,040	22
79.131.203.211	877	875	66	2	871	5
115.24.50.56	610	606	269	608	596	14
91.7.33.92	578	577	201	1	573	5
5.132.204.32	545	543	191	1	538	7
91.7.33.198	502	500	186	1	501	1

(Top 10 IPs out of 447 results)

Table 4.14: Port Scan Attack Statistics, based on FlowBreaker's Analysis

After reviewing the attacks listed by FlowBreaker, the provided summaries and classifications seem to be consistent. In spite of the fact that it is not possible to directly confirm the labels without being able to verify them at the source, or by contrasting with other tools for Network Intrusion Detection, results are promising. This underlines

FlowBreaker's utility as a solution for analysing traffic and its ability to automatically detect attacks if given a configuration file adjusted to the specific use case and traffic analysed. With the evaluation concluded, the next chapter addresses open issues and future improvements.

CHAPTER 5

Open Issues and Future Directions

After concluding the evaluation phase, the results look very promising.

By utilising the MAWI Traffic Archive, as well as the TII-SSRC-23 dataset, we were able to demonstrate, how a limit-based approach, i.e. filtering communications in IP based traffic by certain parameters and sort them using pre-defined thresholds, is able to effectively describe traffic captures. In doing so, we developed a tool, FlowBreaker, which offers significant insight into traffic captures with little operational overhead. However, some open issues and room for improvement also arose. This section summarises the potential improvements and sets a course for future directions in terms of using a limit-based approach as well as our practical implementation of this approach.

5.1 Critiquing the Approach of Host Description

In our work we use pre-defined limits for filtering traffic to generate descriptions of a given traffic capture, similar to the ruleset-based approach employed by NIDS and NIPS such as Snort. In doing so, we created a tool that could be used in Human-Guided Labelling, as described in Section 2.1.1. What distinguishes our approach from other works, is that, rather than labelling 5 Tuple flows directly as typically done in NIDS datasets (15) (2), is that we assign labels to entire hosts/IPs and provide a description as to why this label has been assigned, based on certain parameters. Given that a host can display multiple behaviours and attack patterns, a single flow can have multiple attack labels assigned.

5.1.1 Using Flow Descriptions for ML

ML relies on accurately labelled datasets, which is one of the greatest issues when using NIDS datasets, as they are often out of date, irrelevant or simply of poor quality (12). In

ML, data points are typically attributed to a single label, i.e. a flow can not be labelled as a part of two different attack patterns simultaneously. This makes our approach of allowing for flows to have multiple labels, as we aim at describing host behaviour, rather than flow behaviour, unfit for the direct use with ML. Thus, in order to use this approach for ML, a researcher would have to take FlowBreaker's suggestions/descriptions and decide which labels to assign by hand. If combined with other tools mentioned in Section 2.1.1, our approach could provide additional descriptive power, but by itself it is limited.

5.1.2 Reliance on Humans

With our approach offering host descriptions in various output files, rather than describing every flow individually, a human is almost always necessary to make sense of the descriptions. This is especially true as filtering using limit-based filters is limited in terms of context-awareness. This means that, as observed in Section 4.1.3, different attack patterns can match to the same traffic and it is up to a human to make a distinction. This makes our approach useful when assessing traffic captures by hand, but less fit for automatisation.

5.2 Suggested Enhancements for FlowBreaker

When designing FlowBreaker, one of the top priorities was modularity and ease of use. While both have been achieved, it became clear during the evaluation phase, that further improvements can be made:
The modularity of FlowBreaker, providing a separate output file for each attack, directly contradicts the ease of use, by scattering output across several locations. Furthermore, there is some overlap between attack patterns, leading to multiple labels for the same IP. While this can be the case, it would benefit the user experience if these labels could be summarised into a single file. Since FlowBreaker was built on top of the output of Zeek, it is necessary to host a separate instance of Zeek for pre-processing files. This somewhat complicates the use of FlowBreaker for the end user. Finally, FlowBreaker offers detection methods for many generic attacks, but these only cover a small (nonetheless very important) subset of the actual attacks taking place in real life scenarios.

5.2.1 Detection of Attacks

While FlowBreaker covers some popular attacks, the list of them is certainly not exhaustive. Most of FlowBreaker's modules revolve around threshold based detections, meaning that if a number of connections, bytes, states or ports is exceeded, the IP in question is labelled as malicious. Furthermore, since most attacks follow similar patterns, it is not always clear which attack is actually taking place. For now the solution for this was to label an attack with all labels triggered by it and leave the distinction up to the user. Another issue is that currently the timing information provided by Zeek is not being issued for threat detection. This could also be a valuable asset in detecting attacks based on packet timestamps, e.g. to recognise a covert channel being used for data exfiltration.

Another issue that became evident is Zeek's handling of ICMP based attacks. With ICMP integrated more closely into the IP stack, it is handled a bit differently than TCP and UDP. This results in Zeek sometimes directly detecting hosts running the trace route command and labelling these patterns as such. This resulted in FlowBreaker being unable to properly label ICMP based attacks. In order to improve on this behaviour, the script used to run Zeek would need to be altered, followed by further testing with FlowBreaker.

5.2.2 Output

With FlowBreaker providing its information in three different output formats, there is already a lot of flexibility available. However, the output formatting could be further improved, making it more readable and providing a better overview. At the time of development, flexibility was a higher priority. Thus FlowBreaker provides multiple output files, with each corresponding to a detected type of attack or being a summary of the connections of a given host. In hindsight, this flexibility also leads to a loss of overview and redundant information in the output files. Furthermore, attack patterns could be grouped together, offering multiple classifications in the same file, instead of just one.

Moreover, as mentioned in Section 5.1, our approach of labelling hosts/IPs, rather than individual flows, makes the results difficult to compare to existing NIDS datasets, which are typically labelled on a per-flow basis. This issue could be addressed by creating a new module, that generates e.g. a .csv styled flow list, containing all descriptions/labels using the traditional 5 Tuple-based approach. Doing so, would make our approach easier to automate and integrate with other tools.

In order to improve upon these aspects, further testing and research with FlowBreaker is needed. The findings could then be summarised and added to the existing functionality.

5.2.3 User Interface and Experience

While one of FlowBreaker's main goals is ease of use and it certainly provides a very simple CLI interface, the user still has to pre-process pcaps externally using Zeek and then run FlowBreaker on the output. This is an additional barrier as Zeek is cumbersome to set up under non-Linux operating systems. While somewhat mitigated by the custom frontend provided in this thesis, future versions of FlowBreaker could bypass this need and process pcaps directly. Another issue that comes with CLI interfaces, is that they have to be configured and interfaced with using only text. When trying to automate processes, this is a significant advantage, given that shell scripts can be run using different configuration files with little setup time. However, when manually processing traffic captures, the process can become tedious. A solution for this could be a helper GUI, which acts as an interface for inputting parameters and then runs FlowBreaker's CLI version with said parameters.

5.3 Future Directions

With suggested enhancements for FlowBreaker now laid out, this section revolves around a few proposals on how to tackle them, as well as further ideas for research based on FlowBreaker and the research conducted here, or using FlowBreaker as the core of future research in the field of network security.

5.3.1 Additional Modules

As mentioned above, there is more than one dimension for improvement worth exploring with FlowBreaker. Ranking the afore discussed issues by urgency, one of the top priorities would be extending the capabilities of FlowBreaker in detecting more variants of attacks. This can easily be achieved by writing new submodules. With the existing modules already covering most attacks detectable by numeric patterns, e.g. number of ports used, bytes transferred, etc., a module concentrating on covert channels or hidden data exfiltration, command and control servers or other nefarious server structures would make a great addition to FlowBreaker's repertoire. Furthermore, existing modules could be used for cross referencing, bundling the information into a single detection method.

5.3.2 Research on Labelling Methods

After addressing the critiques regarding per-flow-based labelling and implementing the changes proposed in this section, FlowBreaker could be used to directly analyse existing NIDS benchmark datasets. Thus, using FlowBreaker to carry out a comprehensive comparison on the datasets published in the last 10 years would prove to be a major contribution to the science community. Especially the low quality of NIDS datasets (12), and its impact on ML-based tools could then be further assessed. Furthermore, the descriptions provided by FlowBreaker could offer a new perspective on existing data, potentially identifying patterns that have previously been missed.

5.3.3 Field Research based on FlowBreaker

To further the research made in this thesis, there are several fields of interest that could be explored. One such field would be to run FlowBreaker on more samples provided by the MAWI Working Group. Doing so would provide more data, which could then be cross-referenced. This means that the attack patterns recognised by FlowBreaker could be validated using statistical means, e.g. how often a certain attack pattern emerged, from which IPs, etc..

In general, FlowBreaker's approach of not indexing flows and associated sessions, but rather host activity, offers a new perspective in exploring traffic.
One interesting aspect would be to analyse larger networks, characterise host behaviour/activity and use FlowBreaker to highlight potential weak spots or constraints. This could then be used to map network topologies or develop potential expansions of said networks. In itself, the idea of describing network activity in relation to NIDS from a higher level

perspective is not entirely new. Previous works include tracking sessions/applications through the use of vectors for identification to detect attacks despite TLS encryption (18), as well as characterising IP hosts through IP headers, rather than decrypting the payloads (17). Doing so has proven as an effective measure for characterising networks and communications. This means that simply capturing traffic in different networks and analysing it with FlowBreaker opens up new ways of research and possibilities for describing network phenomena.
To summarise, pre-existing ways of labelling could be combined with the descriptions provided by FlowBreaker to offer new ways of classifying, interpreting and indexing traffic in relation to existing threat potentials.

CHAPTER 6

Conclusion

6.1 Assessing the Research Goals

In order to properly compare the findings of this thesis to the initial expectations, it is necessary to first briefly restate the research goals. Objectives were:

1. Establish a set of minimum criteria to provide a summarised and qualitative description of network flows.
2. Develop a tool which integrates an existing NIDS to obtain the above mentioned summaries and descriptions (describing traffic flows).
3. Show to what extent these enriched labels offer a more discriminating and precise traffic classification.

The first goal was accomplished indirectly when designing FlowBreaker. In its output, FlowBreaker lists several measurements: ports used, number of connections made, data transferred, etc. Additionally, several custom indicators have been created: The Activity Flags which serve to indicate high activity as well as the Flagging Scores which are used to set the aforementioned Flags. Furthermore, FlowBreaker calculates the averages values of the amount of ports used, unique IP addresses seen and transferred bytes per connection. This, combined with the modules targeted at detecting specific attacks, serves as a summarised and qualitative description of individual network flows.
The second goal was reached by building FlowBreaker, which works on top of Zeek, thus integrating an existing NIDS for more detailed labels and analysis. The final goal was achieved in the Evaluation chapter of this thesis. First, by using the TII-SSRC-23 dataset to directly compare the labels assigned by FlowBreaker to those provided by dataset creators. Second, by using a traffic capture from the MAWI Working Group to directly test FlowBreaker on real traffic.

We can therefore conclude that our approach of filtering and describing traffic by defining limits on certain parameters delivers comparable and promising results when benchmarked using a NIDS dataset, but also when used on real traffic captures. Next, we will address how this approach was used in FlowBreaker to help answering the research questions.

6.2 Answering the Research Questions

The following research questions are discussed and answered in this section:

1. How can we label network traffic so that the information provided offers high quality descriptive knowledge, particularly in relation to network attacks?
2. How can we use existing and established tools in the NIDS field to obtain qualitatively enriched traffic-labels?
3. To what extent do qualitatively enriched traffic-labels favor post-analysis and a deeper evaluation of classification and detection algorithms?

The approach towards labelling and describing network traffic in this thesis was to look at network traffic from host perspective, rather than addressing flows individually. This diverges from the more common approach in NIDS datasets of using 5 tuple flows. We created detection methods for some of the most common attack patterns in the NIDS field, such as (D)DoS, Brute Force and Portscans, which assign labels to individual hosts. These labels contain descriptions on why the label was assigned, including measurements and reference values. Furthermore, statistical values and descriptions are provided for each analysed traffic capture, even if no attack patterns were identified in this capture. Using the TII-SSRC-23 dataset, the detection methods were benchmarked and evaluated. In the course of this evaluation, our approach proved to provide labels offering a higher quality descriptive knowledge when compared to the reference labels.
While reviewing pre-existing approaches towards labelling traffic, tools such as Snort, Suricata, Wireshark and Zeek were assessed in the task of obtaining qualitatively enriched traffic-labels.

After this assessment, Zeek was selected to serve as a foundation to provide these labels. The reason for this is its high modularity and configurability, paired with its detailed output. The measurements provided by Zeek were then used to create custom flows, which perform cross-comparisons among these features. These cross-comparisons served to improve attack detection and thus traffic labelling.

For assessing the final research question, we have to take a look at the Evaluation Chapter, specifically Section 4.1. The labels provided by FlowBreaker are consistent with the labels provided by the TII-SSRC-23 dataset, but offer more insight into the reason of the classification. This more descriptive approach of labelling traffic provides more

context when deciding if a label was assigned correctly or not. This can be seen in Section 4.1.3, where we successfully used these additional descriptions to further reason as to why attack patterns were recognised. We can thus reason that the enriched traffic-labels provided by FlowBreaker can be used to perform a deeper evaluation of the analysed traffic, as well as provide a basis for judging detection algorithms.

6.3 Contributions and Outlook

Our approach of filtering traffic and describing it from a host perspective, rather than just labelling flows directly, serves as a new addition to the NIDS field, providing FlowBreaker as a new tool that focuses on the ease of use while improving upon existing measurements provided by Zeek. It thus contributes to the science community as a new way to manually, as well as automatically, detect anomalies and network attacks in traffic captures, while also providing a more human-understandable description of network phenomena. Furthermore, it has been evaluated and validated on existing datasets and traffic captures, where it was able to provide comparable and descriptive labels. This could prove to be useful when working with ML-based approaches, as our more descriptive labels could aid in decision-making, feature selection, and potentially reveal hidden patterns or parameters that influence the algorithm's performance.
Future improvements of this approach could include providing flow-based labels in addition to host or IP-based labels, making it easier for researchers to cross-reference results with existing NIDS datasets, potentially improving upon their quality. To further the usability provided by FlowBreaker, planned future versions include a more streamlined detection process, extensions of the limit-based approach making it possible to recognise more sophisticated attacks, as wells as enriching its output offer through the use of graphical and plotting environments.

APPENDIX A

Appendix

A.1 Configuration Files

A.1.1 Standard Configuration

```
# List of enabled scanning modules
Enabled_Modules = ["Scanning", "BruteForce", "DDoS"]

# Variables for Preprocessing conn.log
[BasicParameters]
# The values below are multipliers applied
# to the average number of IPs/Ports
# Example: if a host has more than twice the
# average of connections to unique IPs
# the highOutIP Flag is set.

# TCP

# highOutPort is set if exceeded
Threshold_Outliers_Outgoing_Unique_Port_TCP = 1.0
# highOutIP is set if exceeded
Threshold_Outliers_Outgoing_Unique_IP_TCP = 1.0
# highInPort is set if exceeded
Threshold_Outliers_Incoming_Unique_Port_TCP = 1.0
# highInIP is set if exceeded
Threshold_Outliers_Incoming_Unique_IP_TCP = 1.0
# highOutConn is set if exceeded
Threshold_Connections_Per_Destination_IP_TCP = 1.0
# highInConn is set if exceeded
Threshold_Connections_Per_Source_IP_TCP = 1.0

# UDP

# highOutPort is set if exceeded
Threshold_Outliers_Outgoing_Unique_Port_UDP = 1.0
# highOutIP is set if exceeded
Threshold_Outliers_Outgoing_Unique_IP_UDP = 1.0
# highInPort is set if exceeded
Threshold_Outliers_Incoming_Unique_Port_UDP = 1.0
# highInIP is set if exceeded
Threshold_Outliers_Incoming_Unique_IP_UDP = 1.0
# highOutConn is set if exceeded
Threshold_Connections_Per_Destination_IP_UDP = 1.0
# highInConn is set if exceeded
Threshold_Connections_Per_Source_IP_UDP = 1.0

# ICMP

# highOutPort is set if exceeded
Threshold_Outliers_Outgoing_Unique_Port_ICMP = 1.0
```

```
# highOutIP is set if exceeded
Threshold_Outliers_Outgoing_Unique_IP_ICMP = 1.0
# highInPort is set if exceeded
Threshold_Outliers_Incoming_Unique_Port_ICMP = 1.0
# highInIP is set if exceeded
Threshold_Outliers_Incoming_Unique_IP_ICMP = 1.0
# highOutConn is set if exceeded
Threshold_Connections_Per_Destination_IP_ICMP = 1.0
# highInConn is set if exceeded
Threshold_Connections_Per_Source_IP_ICMP = 1.0

# Port Scan Detection Settings
[PortScan]
Connection_Threshold = 20
Unique_Port_Threshold = 20

# Host Discovery Scan Detection Settings
[HostDiscoveryScan]
Unique_IP_Threshold = 1000

# Protocol-Specific Scan Detection Settings
[ProtocolSpecificScan]
SYN_Scan_Threshold = 1000

# Version Scan Detection Settings
[VersionScan]
Connection_Threshold = 5 # Number of connections per port
Min_Port_Number = 8 # Number of unique Ports
Max_Bytes_Transferred = 10
Common_Ports = [
    20, 21, 22, 23, 25, 53, 80, 110, 111,
    135, 137, 138, 139, 143, 161, 389,
    443, 445, 464, 500, 513, 514, 515, 623,
    636, 1433, 1521, 2049, 3306, 3389, 5432,
    5900, 5985, 5986, 6379, 8080, 8443, 9200, 27017
]

# Service Enumeration Detection Settings
[ServiceEnumeration]
Connection_Threshold = 5 # Number of connections per port
Min_Port_Number = 8 # Number of unique Ports
Min_Bytes_Transferred = 10
Common_Ports = [
    20, 21, 22, 23, 25, 53, 80, 110, 111,
    135, 137, 138, 139, 143, 161, 389,
    443, 445, 464, 500, 513, 514, 515, 623,
    636, 1433, 1521, 2049, 3306, 3389, 5432,
    5900, 5985, 5986, 6379, 8080, 8443, 9200, 27017
]
```

```toml

[CommonPortsAttack]
CommonPorts = [22, 23, 3389, 21, 5900]  # SSH, Telnet, RDP, FTP, VNC
MinConnectionsPerPort = 50

[PasswordSpraying]
CommonPorts = [22, 23, 3389, 21, 5900]  # SSH, Telnet, RDP, FTP, VNC
PasswordSprayingThreshold = 50

[SSHBruteForce]
MinConnections = 50

[SSLBruteForce]
MinConnections = 50

[HTTPBruteForce]
MinConnections = 50

[SYNFlood]
SYNThreshold = 1200

[UDPFlood]
UDPThreshold = 1000

[ICMPFlood]
ICMPThreshold = 500

[DNSAmplification]
DNSThreshold = 100
MaxDomainRepetitions = 1

[NTPAmplification]
NTPThreshold = 100

[SSDPAmplification]
SSDPThreshold = 100

[ConnectionExhaustion]
ConnectionThreshold = 1000
MaxBytes = 200
MinDuration = 20.0

[Slowloris]
HalfOpenThreshold = 100
MinDuration = 30.0
```

Listing A.1: standard.toml Configuration File used in FlowBreaker

A.1.2 Zero Limits Configuration

```
# List of enabled scanning modules
Enabled_Modules = ["Scanning", "BruteForce", "DDoS"]

# Variables for Preprocessing conn.log
[BasicParameters]
# The values below are multipliers applied
# to the average number of IPs/Ports
# Example: if a host has more than twice the
# average of connections to unique IPs
# the highOutIP Flag is set.

# TCP

# highOutPort is set if exceeded
Threshold_Outliers_Outgoing_Unique_Port_TCP = 1.0
# highOutIP is set if exceeded
Threshold_Outliers_Outgoing_Unique_IP_TCP = 1.0
# highInPort is set if exceeded
Threshold_Outliers_Incoming_Unique_Port_TCP = 1.0
# highInIP is set if exceeded
Threshold_Outliers_Incoming_Unique_IP_TCP = 1.0
# highOutConn is set if exceeded
Threshold_Connections_Per_Destination_IP_TCP = 1.0
# highInConn is set if exceeded
Threshold_Connections_Per_Source_IP_TCP = 1.0

# UDP

# highOutPort is set if exceeded
Threshold_Outliers_Outgoing_Unique_Port_UDP = 1.0
# highOutIP is set if exceeded
Threshold_Outliers_Outgoing_Unique_IP_UDP = 1.0
# highInPort is set if exceeded
Threshold_Outliers_Incoming_Unique_Port_UDP = 1.0
# highInIP is set if exceeded
Threshold_Outliers_Incoming_Unique_IP_UDP = 1.0
# highOutConn is set if exceeded
Threshold_Connections_Per_Destination_IP_UDP = 1.0
# highInConn is set if exceeded
Threshold_Connections_Per_Source_IP_UDP = 1.0

# ICMP

# highOutPort is set if exceeded
Threshold_Outliers_Outgoing_Unique_Port_ICMP = 1.0
# highOutIP is set if exceeded
Threshold_Outliers_Outgoing_Unique_IP_ICMP = 1.0
```

```
# highInPort is set if exceeded
Threshold_Outliers_Incoming_Unique_Port_ICMP = 1.0
# highInIP is set if exceeded
Threshold_Outliers_Incoming_Unique_IP_ICMP = 1.0
# highOutConn is set if exceeded
Threshold_Connections_Per_Destination_IP_ICMP = 1.0
# highInConn is set if exceeded
Threshold_Connections_Per_Source_IP_ICMP = 1.0

# Port Scan Detection Settings
[PortScan]
Connection_Threshold = 0
Unique_Port_Threshold = 0

# Host Discovery Scan Detection Settings
[HostDiscoveryScan]
Unique_IP_Threshold = 0

# Protocol-Specific Scan Detection Settings
[ProtocolSpecificScan]
SYN_Scan_Threshold = 0

# Version Scan Detection Settings
[VersionScan]
Connection_Threshold = 0 # Number of connections per port
Min_Port_Number = 0 # Number of unique Ports
Max_Bytes_Transferred = 1000000
Common_Ports = [
    20, 21, 22, 23, 25, 53, 80, 110, 111,
    135, 137, 138, 139, 143, 161, 389,
    443, 445, 464, 500, 513, 514, 515, 623,
    636, 1433, 1521, 2049, 3306, 3389, 5432,
    5900, 5985, 5986, 6379, 8080, 8443, 9200, 27017
]

# Service Enumeration Detection Settings
[ServiceEnumeration]
Connection_Threshold = 0 # Number of connections per port
Min_Port_Number = 0 # Number of unique Ports
Min_Bytes_Transferred = 1
Common_Ports = [
    20, 21, 22, 23, 25, 53, 80, 110, 111,
    135, 137, 138, 139, 143, 161, 389,
    443, 445, 464, 500, 513, 514, 515, 623,
    636, 1433, 1521, 2049, 3306, 3389, 5432,
    5900, 5985, 5986, 6379, 8080, 8443, 9200, 27017
]

[CommonPortsAttack]
```

```
CommonPorts = [22, 23, 3389, 21, 5900]  # SSH, Telnet, RDP, FTP, VNC
MinConnectionsPerPort = 0

[PasswordSpraying]
CommonPorts = [22, 23, 3389, 21, 5900]  # SSH, Telnet, RDP, FTP, VNC
PasswordSprayingThreshold = 0

[SSHBruteForce]
MinConnections = 0

[SSLBruteForce]
MinConnections = 0

[HTTPBruteForce]
MinConnections = 0

[SYNFlood]
SYNThreshold = 0

[UDPFlood]
UDPThreshold = 0

[ICMPFlood]
ICMPThreshold = 0

[DNSAmplification]
DNSThreshold = 0
MaxDomainRepetitions = 0

[NTPAmplification]
NTPThreshold = 0

[SSDPAmplification]
SSDPThreshold = 0

[ConnectionExhaustion]
ConnectionThreshold = 0
MaxBytes = 10000000
MinDuration = 0.0

[Slowloris]
HalfOpenThreshold = 0
MinDuration = 0.0
```

Listing A.2: zero_limits.toml Configuration File used in FlowBreaker

A.2 Setup Files

A.2.1 Docker Compose File

```
# This stack saves to /home/user/docker/zeek
# - this can be changed by the user,
# as long as it is consistent!
# In order for this script to function properly,
# the folder permissions need to
# be set to 766 - otherwise the container won't have write access!

# PLEASE DON'T RUN THIS SCRIPT WITHOUT ANY ACCESS CONTROL AS
# IT ENABLES ACCESS TO YOUR LOCAL DIRECTORIES THROUGH DOCKER!!!
# BEST PRACTICE WOULD BE USING A REVERSE PROXY
# INSTEAD OF FORWARDING THE 80 PORT!

# Before deploying, either change the /home/user/docker/zeek
# folder to a custom location, or change user to your own username.
# Make sure the permissions are set before launching

services:
  zeek:
    image: zeek/zeek:latest
    volumes:
      - /home/user/docker/zeek/zeek-logs:/zeek-logs
      - /home/user/docker/zeek/uploads:/uploads:ro
    working_dir: /zeek-logs
    user: "root"
    command: tail -f /dev/null
    restart: always

  webserver:
    image: nginx:latest
    volumes:
      - /home/user/docker/zeek/uploads:/usr/share/nginx/html/uploads
      - /home/user/docker/zeek/zipped-logs:/usr/share/nginx/html
      /zipped-logs:ro
      - /home/user/docker/zeek/nginx/nginx.conf:/etc/nginx/
      nginx.conf:ro
      -/home/user/docker/zeek/nginx/upload.php:/usr/share/
      nginx/html/upload.php:ro
    ports:
      - "9005:80"
    restart: always
    networks:
      - zeek

  php:
    image: php:7.4-fpm
```

```
46      volumes:
47        - /home/user/docker/zeek/uploads:/usr/share/nginx/html/uploads
48        - /home/user/docker/zeek/zipped-logs:/usr/share/nginx/html/
49        zipped-logs
50        - /home/user/docker/zeek/nginx/upload.php:/usr/share/nginx/
51        html/upload.php:ro
52        - /home/user/docker/zeek/nginx/php.ini:/usr/local/etc/php/
53        php.ini:ro
54      restart: always
55      networks:
56        - zeek
57
58    zeek_manager:
59      image: debian:latest
60      volumes:
61        - /home/user/docker/zeek/uploads:/uploads
62        - /var/run/docker.sock:/var/run/docker.sock
63        - /home/user/docker/zeek/scripts:/scripts
64        - /home/user/docker/zeek/zeek-logs:/zeek-logs
65        - /home/user/docker/zeek/zipped-logs:/zipped-logs
66        - /home/user/docker/zeek/scripts/zeek_manager.sh:/
67        zeek_manager.sh:ro
68      depends_on:
69        zeek:
70          condition: service_started
71      environment:
72        - DOCKER_HOST=unix:///var/run/docker.sock
73      command: ["/bin/bash", "/zeek_manager.sh"]
74      restart: always
75
76  networks:
77    zeek:
78      external: false
```

Listing A.3: docker-compose.yml for deploying Zeek Frontend

Table of Abbreviations

AL Active Learning

ML Machine Learning

DDoS Distributed Denial of Service

LAN Local Area Network

OSI Open Systems Interconnection

IP Internet Protocol

IoT Internet of Things

TCP Transmission Control Protocol

UDP User Datagram Protocol

ICMP Internet Control Message Protocol

DNS Domain Name System

SSH Secure Shell

API Application Programming Interface

PCAP Packet Capture

PCAPNG Packet Capture Next Generation

SSL Secure Sockets Layer

NID Network Intrusion Detection

NIDS Network Intrusion Detection System

NIPS Network Intrusion Prevention System

IDS Intrusion Detection System

IPS Intrusion Prevention System

GUI Graphical User Interface

IoC Indicator of Compromise

CLI Command Line Interface

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

PHP PHP: Hypertext Preprocessor

LINQ Language Integrated Query

FTP File Transfer Protocol

MAC Medium Access Control (Address)

QUIC Quick UDP Internet Connections

MAWI Measurement and Analysis on the WIDE Internet

WIDE Widely Integrated Distributed Environment

RAM Rapid Access Memory

TLS Transport Layer Security

URI Unique Resource Identifier

Bibliography

[1] Bernard, J., Hutter, M., Zeppelzauer, M., Fellner, D., & Sedlmair, M. (2018). Comparing Visual-Interactive Labeling with Active Learning: An Experimental Study. IEEE Transactions on Visualization and Computer Graphics, 24(1), 298-308.

[2] Guerra, J. L., Catania, C., & Veas, E. (2022). Datasets are not enough: Challenges in labelling network traffic. Computers & Security, 120, 102810.

[3] Lemay, A., & Fernandez, J. M. (2016). Providing SCADA network data sets for intrusion detection research. In 9th Workshop on Cyber Security Experimentation and Test (CSET 16).

[4] Shiravi, A., Shiravi, H., Tavallaee, M., & Ghorbani, A. A. (2012). Toward developing a systematic approach to generate benchmark datasets for intrusion detection. Computers & Security, 31(3), 357-374.

[5] Catillo, M., Del Vecchio, A., Pecchia, A., & Villano, U. (2021). A Critique on the Use of Machine Learning on Public Datasets for Intrusion Detection. In A. C. R. Paiva, A. R. Cavalli, P. V. Martins, & R. Pérez-Castillo (Eds.), Quality of Information and Communications Technology (pp. 253-266). Springer International Publishing.

[6] Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., & Stiller, B. (2010). An Overview of IP Flow-Based Intrusion Detection. IEEE Communications Surveys & Tutorials, 12(3), 343-356.

[7] Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A., & Pras, A. (2014). Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX. IEEE Communications Surveys & Tutorials, 16(4), 2037-2064.

[8] Kim, M., & Lee, I. (2022). Human-guided auto-labelling for network traffic data: The GELM approach. Neural Networks, 152, 510-526.

[9] McElwee, S. (2017). Active learning intrusion detection using k-means clustering selection. In SoutheastCon 2017 (pp. 1-7). IEEE.

[10] Fan, X., Li, C., Yuan, X., Dong, X., & Liang, J. (2019). An interactive visual analytics approach for network anomaly detection through smart labelling. Journal of Visualization, 22, 955-971.

[11] Kenyon, A., Deka, L., & Elizondo, D. (2020). Are public intrusion datasets fit for purpose characterising the state of the art in intrusion event datasets. Computers & Security, 99, 102022.

[12] Flood, R., Engelen, G., Aspinall, D., & Desmet, L. (2024). Bad Design Smells in Benchmark NIDS Datasets. In 2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P) (pp. 658-675). IEEE.

[13] Velarde-Alvarado, P., Gonzalez, H., Martínez-Peláez, R., Mena, L. J., Ochoa-Brust, A., Moreno-García, E., Félix, V. G., & Ostos, R. (2022). A novel framework for generating personalized network datasets for nids based on traffic aggregation. Sensors, 22(5), 1847.

[14] Cordero, C. G., Vasilomanolakis, E., Wainakh, A., Mühlhäuser, M., & Nadjm-Tehrani, S. (2021). On generating network traffic datasets with synthetic attacks for intrusion detection. ACM Transactions on Privacy and Security (TOPS), 24(2), 1-39.

[15] Herzalla, D., Lunardi, W. T., & Andreoni, M. (2023). TII-SSRC-23 Dataset: Typological Exploration of Diverse Traffic Patterns for Intrusion Detection. IEEE Access, 11, 118577-118594.

[16] Iglesias, F., & Zseby, T. (2016). Time-activity footprints in IP traffic. Computer Networks, 107, 64-75.

[17] Iglesias, F., & Zseby, T. (2019). Pattern Discovery in Internet Background Radiation. IEEE Transactions on Big Data, 5(4), 467-480.

[18] Meghdouri, F., Vázquez, F. I., & Zseby, T. (2020). Cross-Layer Profiling of Encrypted Network Data for Anomaly Detection. 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA), 469-478.

[19] Wichmann, P., Marx, M., Federrath, H., & Fischer, M. (2021). Detection of Brute-Force Attacks in End-to-End Encrypted Network Traffic. Proceedings of the 16th International Conference on Availability, Reliability and Security, Article 56, 1-9.

[20] Aizuddin, A. A., Atan, M., Norulazmi, M. M., Noor, M. M., Akimi, S., & Abidin, Z. (2017). DNS amplification attack detection and mitigation via sFlow with security-centric SDN. Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication, Article 3, 1-7.

[21] Biagioni, E. (2019). Preventing UDP Flooding Amplification Attacks with Weak Authentication. 2019 International Conference on Computing, Networking and Communications (ICNC), 78-82.

[22] Kührer, M., Hupperich, T., Rossow, C., & Holz, T. (2014). Hell of a handshake: abusing TCP for reflective amplification DDoS attacks. Proceedings of the 8th USENIX Conference on Offensive Technologies, Article 4.

[23] Cheng, J., Yin, J., Liu, Y., Cai, Z., & Wu, C. (2009). DDoS Attack Detection Using IP Address Feature Interaction. 2009 International Conference on Intelligent Networking and Collaborative Systems, 113-118.

[24] Gadge, J., & Patil, A. A. (2008). Port scan detection. 2008 16th IEEE International Conference on Networks, 1-6.

[25] Trassare, S. T., Beverly, R., & Alderson, D. (2013). A Technique for Network Topology Deception. MILCOM 2013 - 2013 IEEE Military Communications Conference, 1795-1800.

[26] Chapple, M., & Nijim, S. (2024). Database Security. In CompTIA DataSys+ Study Guide: Exam DS0-001 (pp. 231-263).

[27] Rios, V., Inacio, P., Magoni, D., & Freire, M. (2024). Detection of Slowloris Attacks using Machine Learning Algorithms. Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, 1321-1330.

[28] The Tcpdump Group. libpcap. GitHub. Retrieved February 20, 2025, from https://github.com/the-tcpdump-group/libpcap

[29] IETF OPSAWG WG. PCAP Next Generation (pcapng) Capture File Format. GitHub. Retrieved February 20, 2025, from https://github.com/IETF-OPSAWG-WG/draft-ietf-opsawg-pcap

[30] Wireshark Wiki. Development/LibpcapFileFormat. Retrieved February 20, 2025, from https://wiki.wireshark.org/Development/LibpcapFileFormat

[31] P. Phaal. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. Retrieved March 28, 2025, from http://datatracker.ietf.org/doc/html/rfc3176

[32] B. Claise, Ed. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. IETF. Retrieved March 28, 2025, from http://datatracker.ietf.org/doc/html/rfc5101

[33] Lua. About Lua. Retrieved March 28, 2025, from https://lua.org/about.html

[34] Wireshark. Wireshark · Go Deep. Retrieved February 20, 2025, from https://www.wireshark.org/

[35] Snort. Snort - Network Intrusion Detection & Prevention System. Retrieved February 21, 2025, from https://www.snort.org/

[36] Snort. Snort Rules. Retrieved February 13, 2025, from https://docs.snort.org/rules/

[37] TShark.dev. TShark - The Wireshark Network Analyzer on the CLI. Retrieved February 16, 2025, from https://www.wireshark.org/docs/wsug_html_chunked/AppToolstshark.html

[38] Zeek. Zeek Scripting. Zeek Documentation. Retrieved February 16, 2025, from https://docs.zeek.org/en/master/scripting/basics.html

[39] Suricata. Suricata | Open Source IDS / IPS / NSM engine. Retrieved February 21, 2025, from https://suricata.io/

[40] Zeek. Zeek: An Open Source Network Security Monitoring Tool. Retrieved February 21, 2025, from https://zeek.org/

[41] Zeek. About Zeek. Zeek Documentation. Retrieved February 21, 2025, from https://docs.zeek.org/en/current/about.html

[42] Zeek. Docker Images. Zeek Documentation. Retrieved February 17, 2025, from https://docs.zeek.org/en/master/install.html#docker-images

[43] Zeek. Quick Start Guide. Zeek Documentation. Retrieved February 17, 2025, from https://docs.zeek.org/en/master/quickstart.html

[44] Zeek. local.zeek. GitHub. Retrieved February 17, 2025, from https://github.com/zeek/zeek/blob/master/scripts/site/local.zeek

[45] Margolis, J., Oh, T. T., Jadhav, S., Kim, Y. H., & Kim, J. N. (2017). An In-Depth Analysis of the Mirai Botnet. In 2017 International Conference on Software Security and Assurance (ICSSA) (pp. 6-12). IEEE.

[46] Fielding, R., Ed., & Reschke, J., Ed. HTTP Semantics. IETF. Retrieved February 17, 2025, from https://datatracker.ietf.org/doc/html/rfc9110

[47] Mockapetris, P. Domain Names - Concepts and Facilities. IETF. Retrieved February 17, 2025, from http://datatracker.ietf.org/doc/html/rfc1034

[48] Eddy, W., Ed. Transmission Control Protocol (TCP). IETF. Retrieved February 19, 2025, from https://www.ietf.org/rfc/rfc9293.html

[49] Postel, J. User Datagram Protocol. IETF. Retrieved February 19, 2025, from https://datatracker.ietf.org/doc/html/rfc768/

[50] Postel, J. Internet Control Message Protocol. IETF. Retrieved February 19, 2025, from https://datatracker.ietf.org/doc/html/rfc792

[51] Malkin, G. Traceroute Using an IP Option. IETF. Retrieved May 05, 2025, from https://datatracker.ietf.org/doc/html/rfc1393

[52] Preston-Werner, T. TOML: Tom's Obvious, Minimal Language. Retrieved February 20, 2025, from https://toml.io/en/

[53] Postel, J., & Reynolds, J. File Transfer Protocol (FTP). IETF. Retrieved February 26, 2025, from https://datatracker.ietf.org/doc/html/rfc959

[54] Ylonen, T., & Lonvick, C., Ed. The Secure Shell (SSH) Transport Layer Protocol. IETF. Retrieved March 7, 2025, from https://datatracker.ietf.org/doc/html/rfc4253

[55] Freier A., Karlton P. & Kocher P. The Secure Sockets Layer (SSL) Protocol Version 3.0. IETF. Retrieved April 1, 2025, from https://www.rfc-editor.org/rfc/rfc6101.html

[56] Zeek. ssl.log. Zeek Documentation. Retrieved April 1, 2025, from https://docs.zeek.org/en/master/logs/ssl.html

[57] Postel, J., & Reynolds, J. Telnet Protocol Specification. IETF. Retrieved March 7, 2025, from https://datatracker.ietf.org/doc/html/rfc854

[58] Reschke, J. The 'Basic' HTTP Authentication Scheme. IETF. Retrieved March 7, 2025, from https://datatracker.ietf.org/doc/html/rfc7617

[59] Berners-Lee, T., Fielding, R., & Masinter, L. Uniform Resource Identifier (URI): Generic Syntax. IETF. Retrieved April 2, 2025, from https://datatracker.ietf.org/doc/html/rfc3986

[60] Snort. Alert Logging. Snort Documentation. Retrieved March 5, 2025, from https://docs.snort.org/start/alert_logging

[61] Suricata. Alerting. Suricata Documentation. Retrieved March 5, 2025, from https://docs.suricata.io/en/latest/quickstart.html#alerting

[62] WIDE Project. WIDE Project. Retrieved March 6, 2025, from https://www.wide.ad.jp/index_e.html

[63] WIDE Project. MAWI Working Group Traffic Archive: Sample Point F. Retrieved March 6, 2025, from https://mawi.wide.ad.jp/mawi/samplepoint-F/2025/202501011400.html

[64] Redis. About Redis. Retrieved March 6, 2025, from https://redis.io/about/

List of Figures

List of Tables