FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Stream-wise Parallel Anomaly Detection in Computer Networks

by

Tomáš Čejka

A dissertation thesis submitted to
the Faculty of Information Technology, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of Doctor.

Dissertation degree study programme: Informatics
Department of Digital Design

Prague, May 2018

Supervisor:
doc. Ing. Hana Kubátová, CSc.
Department of Digital Design
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Abstract

This dissertation thesis is a collection of author's works from the areas of the flow-based network monitoring and network security that were elaborated in the last five years. The main feature of all included papers is a so-called stream-wise approach of processing flow data, which is described in this thesis. The stream-wise processing is a suitable principle of security analysis for large-scale computer networks since flow records are being processed on-the-fly when they reach a flow collector. As a proof-of-concept, we have developed an open source NEMEA framework and a set of NEMEA modules for a stream-wise analysis of flow data.

There are several included papers in this thesis that show benefits of *extended flow records* containing information from headers of application protocol (L7). Such extended flow records can increase the reliability of detection algorithms and allow for detection of suspicious traffic that is invisible for traditional flow-based detection methods. The detection modules capable of processing the L7 information are called application-aware.

Since the data volume from monitoring systems grows, and it is expected to increase further in the future as well, the next focus of this dissertation thesis is a scalable infrastructure for parallel flow-based analysis. Our experiments show the importance of a correct algorithm for data distribution among multiple computing nodes. Usage of an algorithm that does not respect semantic relations in the flow data has a strong negative influence on the detection results. Therefore, the dissertation thesis shows a method of constructing a proper Flow Scatter that distributes flow data without breaking these semantic relations.

Besides the described contributions, there was an extensive experimental evaluation of all works included in the papers. The experiments were performed with data sets from real backbone traffic of Czech national academic network. Additionally, the created flow-based NEMEA modules were deployed in the monitoring infrastructure of CESNET2 network.

Keywords:

network security, flow data, witness, anomaly detection, parallel processing, data splitting, NEMEA.

Acknowledgements

I would like to thank my colleagues from CESNET, especially Ing. Jan Kořenek, Ph.D. and Ing. Martin Žádník, Ph.D., for their support and inspiration. I am also very grateful for the support of my supervisor doc. Ing. Hana Kubátová, CSc., who was fighting for me when the academical/political environment required that. Doc. Kubátová motivated me to finish my thesis "sooner than later." Doc. Ing. Petr Fišer, Ph.D. helped me to improve this dissertation thesis by his valuable feedback and proofreading.

Many thanks belong to my students, former or current, who helped me with my intention to build a strong research&development team that can achieve excellent results and can successfully represent our work, e.g., at conferences or in contests. I believe I couldn't get so far (and so quickly) without these co-workers (most of them became my friends). The list would be very long, but I would like to mention Ing. Marek Švepeš, Ing. Zdeněk Rosa, and Bc. Tomáš Jánský for their long-term merit for the team.

Speciální poděkování patří mé rodině a přátelům; když maminka říkala, že tuto práci dokončím, měla pravdu!

This work was partially supported by internal grant No. SGS17/212/OHK3/3T/18 funded by CTU in Prague, and by the "CESNET E-Infrastructure" (LM2015042) project funded by Ministry of Education, Youth and Sports of the Czech Republic.

Contents

List of Algorithms

Abbreviations

Networking Abbreviations

API	Application Programming Interface
C&C	Command and Control
DNS	Domain Name System
DoS	Denial of Service
DDoS	Distributed Denial of Service
DRDoS	Distributed Reflection Denial of Service
HTTP	Hyper-Text Transfer Protocol
IP	Internet Protocol
ISP	Internet Service Providers
L3	Network layer (3th layer) of the ISO/OSI model
L4	Transport layer (4th layer) of the ISO/OSI model
L7	Application layer (7th layer) of the ISO/OSI model
NEMEA	Network Measurements Analysis
SIP	Session Initiation Protocol
URL	Uniform Resource Locator

Flow Records

bytes	number of bytes
dstip	destination IP address
dstport	destination port of transport protocol
packets	number of packets
proto	transport protocol
srcip	source IP address
srcport	source port of transport protocol
timefirst	timestamp of the first packet in the flow
timelast	timestamp of the last packet of the flow

basic flow record: a record identified by srcip, dstip, srcport, dstport, proto, timefirst, timelast and containing bytes and packets
extended flow record: a basic flow record extended by application layer information

Chapter 1

Introduction

Network security plays an essential role in the modern world of digital communication. It is a research field of many researchers and security companies around the world. Many topics in the network security area have been elaborated by various researchers. However, there are still many open research issues. This dissertation thesis focuses on several challenges related to network security and parallel processing of high amounts of data of network traffic. The dissertation thesis is a collection of peer-reviewed research papers on the topic of anomaly detection in large computer networks.

Network technology continuously evolves, network stack is implemented in many hardware chips, and software implementation of communication protocols can be easily integrated into any system or device. Therefore, lots of devices, even small sensors, are being equipped with a network interface, and according to the current trend, the number of network-enabled devices will grow. Devices can communicate via network interfaces not only with humans, as it used to be at the beginning of the computer networking, but also with other "things" or machines connected to the network. Due to automation, machines are becoming both the primary consumers and producers of data. This trend aims to help people to make their life/work easier and more efficient. We are living in the era of the Internet of Things.

Since small devices have low capacity of memory and storage, they are usually designed to access servers via the Internet — so-called Cloud — where the processing power, memory, and storage capacity are much more available. In practice, the idea of collecting data to some central place is valuable because it allows easier data access from anywhere and various types of advanced data processing. The more devices are connected to the Internet, the more significant data volume is transferred via network links, and thus it makes the data processing and storage harder in the central place.

From the topological view of computer networks, the data travel from end-point networks to the target servers via high throughput backbone links and transit networks of Internet Service Providers (ISP). Such network links are usually designed with much higher capacity, so most of the attacks do not affect the infrastructure of ISPs. However, significant attacks can have a crucial impact on end-users' networks. Such attacks are easily

detectable in a victim's network, where they are difficult to mitigate. The attacks are worse detectable in a backbone network; however, their mitigation is more efficient there. The described relation is illustrated in Fig. 1.1.

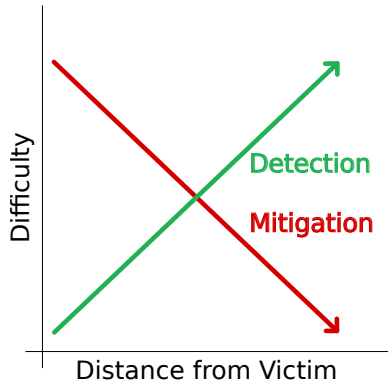


Figure 1.1: Observation of a distributed volumetric attack. The further from victim an observation is, the more difficult detection. On the other hand, mitigation processes are more efficient further from a victim, i.e., closer to a source.

Despite the efforts of many network security initiatives nowadays, lots of manufacturers do not take care of the security of smart devices. Security updates are published with delay or never, and if a security patch is not installed automatically, users usually leave the devices without updates at all. As a result, the number of vulnerable devices connected to the Internet increases, and this is an opportunity for attackers and developers of malicious software to create a piece of code that can be executed on the vulnerable devices. Such devices become a source of malicious traffic that participates in massive volumetric attacks where it is enough to generate just a few packets by each infected device. Therefore, it is difficult to recognize malicious traffic in the source network.

The infected devices (bots) can be used by attackers to create powerful botnets — the devices under control of an attacker. Attackers can use botnets to produce various types of malicious traffic, and since the bots can be synchronized in time, lots of data can be sent against any victim from multiple places at the same time. In 2016, vast attacks of over 600 Gb/s were observed from surveillance systems infected with malware that created the Mirai botnet. These attacks paralyzed even large infrastructures of international commercial organizations (example was analyzed in [1] by B. Krebs).

It is necessary to perform comprehensive analysis of network traffic to discover the infected devices and other sources of attacks. The aim is to monitor the activity and status of the whole infrastructure to preserve its operability and keep so-called situational awareness. Monitoring systems observe network traffic and gather data about communication between connected devices. Monitoring systems produce such a high data volume that it is challenging to do the analysis and detection manually. Additionally, it is a challenging task to store all the data due to slow storage or to keep a long history of data from large

networks. A logical used approach is to aggregate or sample data to decrease the volume of data.

There are many sources of information utilized in network monitoring. According to the source of information, we can recognize two main types of monitoring: *host-based*, which uses information from each device (e.g., system log files), and *network-based*, which uses information exported from network devices such as routers, switches or specific monitoring probes. This work focuses on the network-based monitoring which is a useful and feasible approach in large network infrastructures where operators usually do not have full control over connected devices.

Current state-of-the-art systems use IP Flows, which are defined in [2] as follows:

> *An IP Flow, also called a Flow, is defined as a set of IP packets passing an Observation Point in the network during a certain time interval. All packets that belong to a particular Flow have a set of common properties derived from the data contained in the packet and from the packet treatment at the Observation Point.*

For the sake of simplicity and our purposes, information of IP Flows is represented by data structures *flow records*, whereas each one is usually a unidirectional communication between two network addresses. Such flow records are identified by an n-tuple consisting of: source IP address (*srcip*), destination IP address (*dstip*), source port of transport protocol (*srcport*), destination port of transport protocol (*dstport*), transport protocol (*proto*), timestamp of the first packet in the flow (*timefirst*), timestamp of the last packet of the flow (*timelast*). Besides these fields for identification, flow records contain some additional fields — counters that represent volume of the transferred traffic: number of transferred bytes (*bytes*), number of transferred packets (*packets*). The described list of fields of a flow record is referred as a *basic flow record* in the later text of this thesis. Flow data (sequence of flow records) is sent from monitoring probes, where the flow records are exported, to a flow collector, where they are processed and stored.

The published papers, which are included in Appendix A of this dissertation thesis, deal with detection of malicious traffic based on a stream-wise analysis of the flow records. The stream-wise approach itself is described in Section 3.2. It is based on an on-the-fly analysis of flow data because the input data come from monitoring probes continuously. The described concept allows processing flow records from even large networks. Monitoring probes export the flow records that are collected by a flow collector where the stream-wise processing can run.

The published papers also show the feasibility of the stream-wise approach and describe detection of several different types of malicious or suspicious traffic even in a large network infrastructure. Most of the presented detection methods analyze extracted application information for detection of threats that cannot be reliably detected using just traditional flow data (i.e., without additional application information). The practical experiments evaluated the presented approach. The experiments were performed thanks to the designed and developed NEMEA system (described in Section 3.1),

The dissertation thesis also presents a methodology for splitting a continuous stream of flow data for parallel processing. The main feature of the methodology is the preservation of semantic relations among flow data, i.e., among flow records. Such semantic relations have a significant impact on the performance of the detection algorithms. Sections 3.4–3.6 describe the usage of this methodology using the published detection algorithms as the examples to show characteristics of the relations. The goal is to show relations that should not be broken to preserve the detection results in parallel processing.

The aim is to present a designed system for processing large amount of flow data in parallel. Such processing is intended for security analysis and anomaly detection in large infrastructures.

1.1 Motivation

The complexity of network infrastructures, the throughput of network links and volume of transferred data grow. There are many reasons to deploy and run a network monitoring system that watches the activity of connected devices and status of the network links. However, one of the most important reasons is network security. An analysis of the network traffic may help to **discover many security issues** such as infected devices that are attacking or scanning, devices under attack, data exfiltration, breaking or avoiding security policies, breaching an owned device by an attacker or many others. Many scientific papers show that analysis of flow data may discover lots of security threats. Monitoring systems are usually the primary source of information for security analysis.

There are hundreds of gigabytes of flow records exported per day in large networks. On average, about 285 GB of flow data per day is measured by 11 monitoring probes at the perimeter of CESNET2, the national academic network in the Czech Republic. Altogether, **network monitoring becomes a Big Data processing**, and operators must consider some distributed approach for scalability and sustainability in the future.

In practice, a network infrastructure is being monitored from several different places. It is useful to deploy multiple monitoring probes because it allows tracking routes where the traffic flows, i.e., from where the packets originate and which routers or other devices forward the packets. Analysis of the data from all probes together helps **to create a global view** of different parts of the network infrastructure. The centralized processing can discover events that are not visible from a local perspective, especially when some malicious traffic goes via multiple links like in case of distributed attacks. Transferring information to one place increases the issue of data volume, and it goes against the distributed nature of Big Data approaches, which work with distributed data storage. It is a challenging task to process all data in one place because of resource limits of a single machine, and some distributed computing must be considered for scaling solution.

The weakness of centralized collecting of information about network traffic was identified and solved in a Czech innovation project called Security Cloud[1], funded by Technology

[1]SecurityCloud, TAČR ALFA project no. TA04010062,
https://github.com/CESNET/SecurityCloud/wiki

Agency of the Czech Republic in 2014–2017. This project aimed to design and develop a distributed flow storage that is efficient enough to receive high volumes of data from multiple monitoring probes. The developed system allows users to query the stored data from a longer time period with much shorter delay. However, this project did not cover **distributed on-the-fly anomaly detection**, and so it remains a weak part of the monitoring and analysis pipeline.

The detection delay plays an important role. For instance, detection of enormous Denial of Service (DoS) or distributed DoS (DDoS) attacks should be fast enough to allow network operators to perform some mitigation mechanisms. The total delay is a sum of many factors such as a delay of flow data export at probes, time of the decision of an algorithm, a delay of an alert creation and a communication delay caused by infrastructure. Other used systems also cause some delay (flow collector, detection system and system for incident handling). The delay of the decision of an algorithm is usually the biggest one since it takes time until the algorithm gathers enough data to identify malicious or anomalous traffic and, additionally, advanced algorithms need a significant time to process all the gathered data. To minimize this delay, the aim of this research is a **real-time or near real-time analysis**, i.e., to process data on-the-fly immediately.

However, the sooner an alert is reported, the higher probability of a false positive result. It is not feasible to run a detection system with high false alert rate, especially when human incident response teams process the alerts. There are many reasons why detection algorithms produce false alerts, which identify legitimate traffic as a malicious one. Since detection algorithms usually use some approximate model of legitimate or malicious traffic (such as "an IP address that establishes lots of connections behaves badly"), false alerts are the cases where the ideal model does not fit to the real character of the legitimate traffic. Machine learning (ML) detection algorithms work according to a learned model, which is created in their training phase [3]. Therefore they are weak in the identification of legitimate or malicious traffic that was not included in the training dataset. Statistical detection algorithms [3] suffer from the similar issue because there is currently no perfect statistical model of benign or malicious traffic for the computer networks. On the other hand, increasing thresholds and tuning the algorithm to be more tolerant and not to detect "small" anomalies lowers false alert rate with the price of longer delay and a higher probability of missed (undetected) anomalies. Therefore, some **trade-off between the detection delay and false alert rate** must be considered.

Offline analysis, which was traditionally used, requires high capacity storage for data files that contain flow records from time windows with a fixed length. Contrary, the network traffic is a continuous stream of data by nature and, consequently, monitoring probes export information about the traffic also as a stream of data. Therefore, this research focuses on **processing a continuous stream of data** without splitting into separate time windows. This approach needs a particular design, where no big storage is required.

Stream-wise analysis approach, which is presented in this dissertation thesis, can avoid the need of storing data. The goal of the development of stream-wise detection methods is to save resources and, therefore, to allow for running multiple detection methods at once on the same machine.

Compared to packet-based approaches, it is usual to use only information up to the transport layer (L4) of packet headers (i.e., a basic flow record) for flow-based monitoring and analysis. The reason is lower required processing power in monitoring probes. The L4 information is generally sufficient for detection of many security threats. However, there are lots of attacks that target against application layer (L7). Such traffic cannot be distinguished from legitimate traffic with high reliability, and so the malicious traffic is hardly detectable. Therefore, it is important to find information that can improve the detection results and use it for the detection. This work develops so-called **application-aware detection methods**, which work not only with basic flow records but also with information from the L7 — *extended flow records*. Our research shows that usage of the extended data increases the reliability of detection and allows for detection of some anomalous traffic, which would remain undetected.

An essential motivation of this research is to design a system that can run in a parallel distributed environment to handle big data. It is crucial to bear in mind the stream-wise concept, especially for large networks. It minimizes delays of detection and keeps the storage requirements low. To detect even L7 security issues, some additional information, which can increase reliability and precision of detection, is needed in the analyzed flow records. Since the computing power increases, mainly using some hardware acceleration, it is possible to export some L7 headers, which can help detection algorithms. Altogether, it was needed to focus on research and development of a **detection system that can overcome the current issues** regarding the big volume of data, limited resources of a single machine and invisible malicious traffic on L7.

It was necessary to **develop new methods** to cover needs for continuous stream-wise detection with the use of L7 information that increases the reliability of detection results.

1.2 Problem Statement

Let us have a *set of monitoring probes* and a *central flow collector* that receives data from the probes. Let us have a computing node that can receive a continuous stream of data

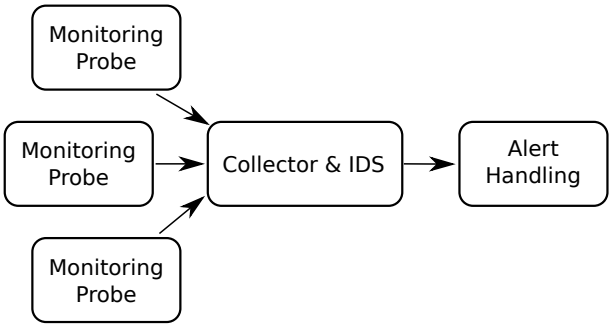


Figure 1.2: Infrastructure for flow monitoring usually consists of multiple monitoring probes and a central flow collector.

via a flow collector. A simple example of the infrastructure is shown in Fig. 1.2, where the Collector&IDS block represents the computing node. The computing node runs *various detection algorithms* which process a *stream of input data* and detect malicious traffic. The node can be easily cloned and started multiple times on separate machines to get more processing power. The challenging part is a *distribution of a single stream of data among the multiple nodes*, whereas each node should process just a subset of the original stream.

This approach does not require a complete redesign of the deployed detection or analysis algorithms. However, a suitable mechanism of data distribution must be designed to preserve results of detection. This work proposes an approach to scalable flow data processing using independent computing nodes.

Another challenge is related to the reliability of detection algorithms when they are used to detect L7 threats. Traditional flow records with basic information are not sufficient for *detection of advanced threats* that are similar to legitimate traffic.

Finally, there is a lack of existing (open source) tools to develop a prototype of a detection and analysis tools. Such tools are essential for academic experiments and evaluation.

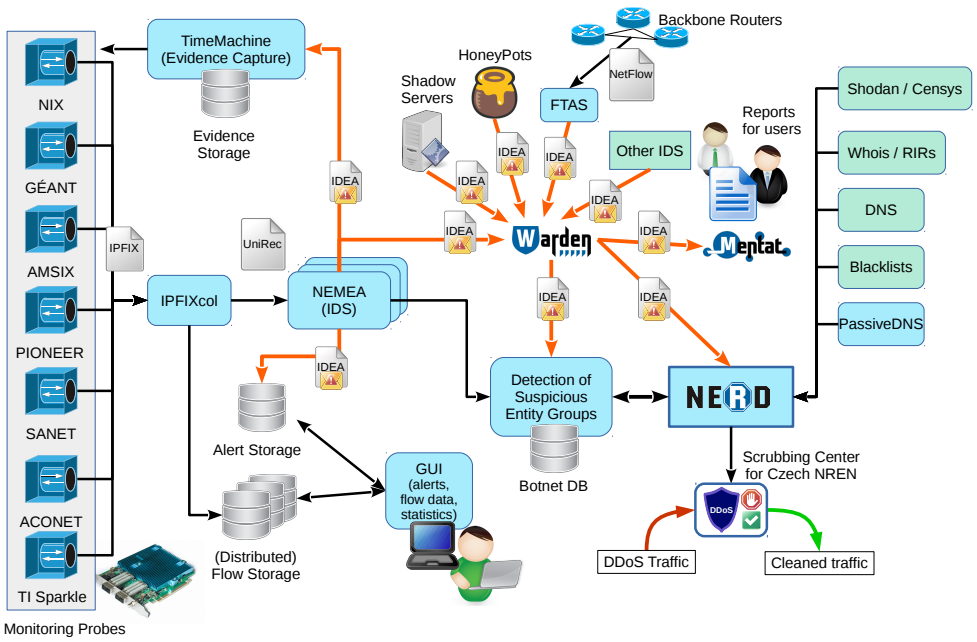


Figure 1.3: This work in the context of security tools in the CESNET2 academic network.

Our work was practically evaluated in the CESNET2 infrastructure, so it is a part of the big set of tools and systems. Figure 1.3 shows several systems that are deployed in CESNET2 to monitor the infrastructure and detect suspicious traffic. The systems also address incident reporting and handling. This dissertation thesis is primarily about the *NEMEA (IDS)* block that process flow data from the flow data collector (*IPFIXcol*). Parallelism of NEMEA is represented by the duplicated blocks. Many other blocks in the figure were designed and implemented as parts of bachelor and diploma theses supervised by

the author of this dissertation thesis ([P.32, P.29, P.28, P.27, P.26]). However, description of the all blocks is out of the scope of this document.

1.3 Goals and Contributions of the Dissertation Thesis

The main goals of this thesis are focused on processing the flow data in large backbone networks. The aim is to design mechanisms to process **huge volume** of flow data as soon as possible especially for network security purposes. It was necessary to design new detection and processing algorithms that could work in a **stream-wise** way, i.e., to process data immediately **without long-term storage**. This approach allows processing in **real-time**. The research presented in this thesis also shows that some application layer information might help to identify advanced security threats. Therefore, the next goal was to extend the monitoring and analysis infrastructure to support additional information, i.e., to make the monitoring "aware of application" (**application aware**). Because of the growing volume of the flow data, the processing on just one machine would not be scalable enough. Therefore, the next goal of the thesis was **parallel processing** based on splitting a stream of flow data into independent subsets that can be processed on separate machines. There was a related requirement about the **balanced load** of each machine, which is the reason why the sizes of the split subsets should be as similar as possible. Splitting the stream of flow data affects the amount of information that is processed by a single machine. The goal was to design **an algorithm for splitting** that ensures the machine receives enough data to identify security threats, i.e., **to preserve detection results**.

Contributions of this thesis can be generalized and divided into the five main topics that are covered in the following chapters.

1. Description of a new **stream-wise approach** to network traffic analysis and anomaly detection using flow data without long-term storage (discussed in Section 3.2).

2. Design and development of **stream-wise** and **application-aware** (discussed in Section 3.3) **detection methods** that can discover threats on application layer. Additionally, design and development of a framework that allows rapid prototyping of the detection methods (discussed in Section 3.1 about **NEMEA**, the open source detection system).

3. The methodology based on *witnesses* to **split a stream of flow data** without breaking semantic relations that affect results of detection algorithms (discussed in Section 3.4). and the design and development of the **scalable architecture** for **parallel processing** the flow data with respect to *witnesses* (discussed in Section 3.6.1).

4. **Practical experiments** with real backbone traffic using the developed detection algorithms (published in the included papers in Appendix A).

5. **Formal symbolic description** of the algorithms implemented as the modules for the NEMEA system are contained in Section 3.5. Some of the algorithms were

published in [P.2, P.6, P.7, P.8] (the papers are included in Appendix A). However, the formal descriptions were not in the original papers.

1.4 Structure of the Dissertation Thesis

The thesis is organized into five chapters as follows:

1. *Introduction* (Chapter 1): Describes our motivation, defines the targeted issues and lists the goals of this dissertation thesis.
2. *State-of-the-Art* (Chapter 2): Describes the current state-of-the-art and lists related existing works in the area of network security, anomaly detection and parallel processing using flow data.
3. *Topics and Contributions in Details* (Chapter 3): Presents the designed and developed NEMEA system for flow-based analysis, explains its essential features (*application-awareness* and *stream-wise* processing), defines terminology that is important for this work, such as *witness* — an abstract representation of semantic relations in the flow data, which is significant for parallel processing. Section 3.5 shows the application of the defined terms on the developed detection algorithms. Additionally, the section contains details not included in the published papers due to the limited space.
4. *Conclusions* (Chapter 4): Summarizes this dissertation thesis and suggests possible topics for further research.
5. *Included Papers* (Appendix A): Contains eight of the published reviewed papers of the author of this thesis. The included papers present details about design and implementation of the stream-wise and application-aware detection algorithms and the NEMEA system. The papers also contain various observations from experiments with real network traffic in CESNET2. Each paper is introduced by a short summary. The described detection algorithms were the proof-of-concept of the parallel analysis.

Figure 1.4 shows the structure of this dissertation thesis visually. The thesis is built upon published papers regarding detection of suspicious and anomalous traffic. Each block in the figure represents a topic covered in this thesis. The blocks are listed and described below:

1) Detection of known features (KF in the figure) in Appendix A.2.

2) Detection of covert channels using DNS (DNS in the figure) in Appendix A.3.

3) Detection of threats against signaling protocol Session Initiation Protocol (SIP) for Voice over IP (SIP in the figure) in Appendix A.5 and Appendix A.4.

4) Detection of vertical scanning, also known as port scans, (VS in the figure) in Appendix A.6.

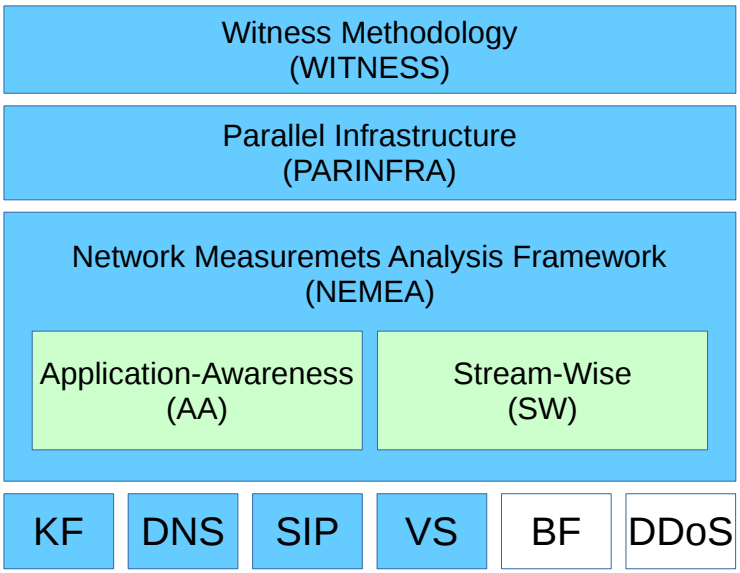


Figure 1.4: The visual outline of this thesis that represents published and described parts. Blue boxes represent topics that were published in the included papers. Green boxes represent novel approaches described as a part of a published paper. White boxes represent related topics that are described in this thesis for the sake of completeness although they are not published elsewhere.

5–6) Simple examples of detection Distributed Denial of Service attacks (DDoS in the figure) and Brute Force attacks (BF in the figure) are mentioned for completeness in Section 3.5 even though they are not published as an original work by the author of this thesis.

7) All described detection methods (KF, DNS, SIP, VS, BF, DDoS) were implemented as modules of the Network Measurements Analysis (NEMEA) system built upon NEMEA Framework that was published in the paper included in Appendix A.1.

8–9) NEMEA was designed with respect to approaches Application-Awareness (AA in the figure) and Stream-Wise processing (SW) that were far from common in the time of the start of the project. However, the deployment of NEMEA in practice proved that they are feasible and useful.

10) Besides published paper on single instance of NEMEA for flow data processing, there is another paper included in Appendix A.7 that shows an infrastructure for processing a single stream of flow data that is being split into parallel substreams (PARINFRA in the figure). Such substreams can be processed in parallel so that it allows higher overall throughput of the detection system.

11) Splitting the stream of flow data from the theoretical point of view (WITNESS in the figure) is covert in a separate paper included in Appendix A.8.

Chapter 2

State-of-the-Art

2.1 Monitoring Approaches

Various research papers (such as [4, 5]) divide approaches to network monitoring and anomaly detection into two groups according to the source of information that is used: *host-based* and *network-based*. The first group, *host-based*, uses information from a "host," i.e., a device. Such a monitoring and detection system analyzes system or application logs, running processes and their behavior and network connections. Host-based detection systems usually have access to the content of a communication, which allows for more precise analysis (e.g., for Anti-SPAM systems). Nowadays, this functionality is usually done by security systems, developed by many commercial companies, which contain the functionality of firewall, auditing tools, and file system scanners. A disadvantage of the *host-based* detection system is that it requires control over a monitored host or at least access to its system logs.

The scope of interest of this dissertation thesis is a *network-based* approach that is focused on an analysis of network traffic at the level of network infrastructure. This approach can work without prior comprehensive information about running servers and services. Additionally, the monitoring does not require any control over the devices of users. Therefore, it is more suitable for ISP and backbone networks and their operators.

Network-based monitoring can be divided into two subgroups from the perspective of the processed data units: *packet-based* and *flow-based*. Generally, they differ in the level of detail that is analyzed and the level of information aggregation. *Packet-based* approaches work with raw packet data and with no or minimal aggregation; meanwhile, *flow-based* methods operate with *flow records*, i.e., aggregated information about packets (already introduced in Chapter 1).

Packet-based detection methods can operate with various information contained in network packets. Methods can use knowledge of the whole content of messages from the data link layer (L2) to the application layer (L7). This approach allows for a pattern or signature matching. A packet-based analysis up to the application layer is sometimes called deep packet inspection (DPI).

Generally, intensive DPI can be used in networks with lower speeds because of the performance issues with transfer of a large volume of data presented in [6]. However, as modern technologies evolve, the data transfer between hardware and software components and bandwidth issues were significantly shifted up to the processing of 100 Gbps per one Ethernet line by [7, 8]. This allows analyzers to perform various algorithms such as pattern matching at high speed.

Flow-based detection methods are commonly used because of a higher level of aggregation with a small loss of information about communicating network hosts as it is described in [9, 10]. Many types of malicious traffic such as brute-force attacks, Denial of Service (DoS), amplification attacks are observable using flows records. For instance, detection of brute-force attacks against SSH using flow data was published in [11].

Flow records can be represented in various data formats. It is usually a sequence of data messages with some fixed set of information fields. There are many data formats that are used by researchers. For instance, JSON format provides enough flexibility and is used mainly for easy integration with Big Data platforms (like the authors of [12] did). However, JSON is very inefficient data format because of size and binary data formats are preferred. The well-known and mostly used flow data representations are NetFlow [2] and more recent IPFIX [13] format. This thesis describes a new efficient data format for stream-wise flow data processing in Section 3.1.

Hardware acceleration can be used for analysis of high-speed network traffic for both packet-based and flow-based monitoring. Hardware acceleration allows to partially or entirely perform some processing tasks in a special hardware monitoring device (e.g., [14]). Development of a hardware design is still more difficult than development of software prototype, despite there are many efforts to create a high-level language to describe the functionality of a hardware device [P.11]. Therefore, a trade-off between efficiency of computation in hardware and speed of development must be chosen. A suitable approach that is also used in [7] is a hardware-software co-design, where the hardware part can efficiently pre-process data at wire speed or compute necessary statistics, and the rest is implemented as a software part.

Network-based monitoring, no matter if packet-based or flow-based, can be further classified as *passive* or *active* [10, 15]. Active monitoring is based on active checks that inject traffic into a network to get the state of infrastructure. The examples of active monitoring systems are [16, 17, 18] or just simple tools like `ping(8)` or `traceroute(8)`. Contrary, passive monitoring is usually done using duplicated (mirrored) traffic besides the monitored infrastructure so it cannot affect it. Passive monitoring can efficiently use dedicated monitoring devices (probes) (such as [19]) that receive the mirrored traffic of the observed link. It is a common practice that such monitoring probes are placed at the perimeter of the network (i.e., bordering links to other networks) or at the core links inside the infrastructure.

2.2 Detection Methods

There are many surveys and taxonomies of detection methods or tools [20, 21, 22]. Naturally, each type of a network attack or a traffic anomaly is elaborated in many other papers, e.g., a DDoS topic presented in papers such as [23, 24, 25]. Types of detection methods are highly related to the specific types of attacks or the types of traffic anomaly that ought to be detected. In addition, the detection mechanisms are usually presented in combination with defense techniques to mitigate the traffic or at least to allow post-mortem investigation.

An anomaly in network traffic usually appears at unpredictable points in time. The start of an anomaly can be observed as a change of statistical distribution of the random variable that represents some characteristic of network traffic [26]. The choice of a set of the observed characteristics depends on a detection method and on an anomaly type that should be detected. Characteristics are usually monitored and analyzed as a time series that is defined as a sequence of measured values in time. The first insight to an anomaly detection using time series can be based on simple thresholds and limits where an alert is raised when the current value reaches some predefined value.

There are many types of network attacks or other anomalies that appear in computer networks. The anomalies differ in many ways such as a volume of traffic, or a duration of anomaly. For successful anomaly detection, a model of the network traffic is needed. It is a complicated issue to create a working and precise model of a computer network traffic as it is described in [27]. The absence of such model can lead to false alerts signaled from detection methods. Based on long-term observation, some similarities and trends can be seen in network traffic. More sophisticated detection methods are based on this knowledge and use various kinds of smoothing such as moving averages or prediction models to normalize measured characteristics.

Some types of a network anomaly are significant enough to be found by manual analysis of the network traffic. The most significant anomaly that can be observed in the network traffic is a DoS attack or its distributed version (DDoS) attack. They are observable as a huge increase in the total traffic. However, from the ISP network point of view, even a high increase of traffic at a link of a user can be below the resolution of the global view and capacity of ISP's backbone links. In addition, a manual analysis is not always possible due to a huge volume of data and the efficiency of such analysis. Therefore, there is a need for automatic detection. Besides trends and variations of the network traffic that regularly appear on a network, the network traffic contains several irregular peaks of legitimate traffic. The example of the legitimate traffic can be a flash crowd effect that means a temporary increase of traffic due to users legitimate activity. These facts make anomaly detection with low false alert rate very challenging.

SYN flood attacks[1] have been studied for more than ten years, and various detection approaches were published in many papers. However, this issue is currently still relevant

[1] a well-known type of flood attacks that is based on sending of several TCP packets with SYN flag, these packets are used for establishment of a TCP session.

because of the increase of network traffic volumes, the growth of networks and the higher level of sophisticated attacks. The detection based on NP-CUSUM is used in [28], where the authors present their observation about SYN-FIN pairs in network traffic under the normal condition:

> *(1) there is a strong positive correlation between the SYN and RST packets;*
>
> *(2) the difference between the number of SYN and FIN packets is close to the number of RST packets.*

The authors bring an experimental evaluation of flood detection using NP-CUSUM and counts of TCP flags observation; however, they mention a possible disadvantage of aggregated counting of packets that can be spoofed by an emission of mixed packet types by an attacker.

The authors of [29] compare a straightforward *adaptive threshold algorithm*, which can bring satisfactory performance for attacks with high intensity, and an *algorithm based on CUSUM*. The adaptive threshold algorithm uses a deviation from a moving average computed by exponentially weighted moving average (EWMA) algorithm. An alert is signaled when a measured value is higher than moving average in last k consecutive intervals. The authors propose to use some prediction method such as Holt-Winters to remove non-stationary behavior before applying the CUSUM algorithm. However, because of time-consuming calculations with minor gains compared to more straightforward approaches, the authors used an approach based on CUSUM and the difference between measured value and result of EWMA algorithm.

The idea to use EWMA for smoothing, Holt-Winters for prediction of seasonal trends of traffic and CUSUM for change point detection was published in [9]. The authors designed a prototype of a monitoring probe extended by anomaly detection as a software plug-in for an existing monitoring probe. The paper proposed two algorithms with fixed parameters evaluated on a dataset from backbone network.

The Holt-Winters method was proposed as a suitable approach to detection in [30]. The authors described the formula of the method as well as the way of its usage for a detection. The detection is based on a prediction of a confidence band and a comparison of current measured value with the confidence band.

The study of detection methods of DDoS attacks was published in [25]. The authors discuss open issues and challenges. The example of the challenges is a development of new detection methods that can detect even sophisticated attacks generated by tools that are being rapidly improved. The paper uses the taxonomy of DDoS attacks and DDoS defense mechanisms published by [24]. According to the taxonomy, DDoS detection methods can be classified as a) *statistical* b) *knowledge-based* c) *soft computing* d) *data mining* and *machine learning*.

The group of *statistical* detection methods contains various methods based on Change-Point Detection (CPD) algorithms such as CUSUM or NP-CUSUM [31, 26] (described in [P.17]), Change aggregation trees (CATs) [32] or D-WARD [33] based on continuous

monitoring of bidirectional traffic flows between the network and the rest of the Internet with analysis of deviation from the typical flow patterns.

Knowledge-based methods are based on predefined rules and patterns of attacks. Detection methods exploit various heuristics and data structures. *Soft computing* methods are based on learning paradigms. Methods from this group use neural networks, radial basis functions, and genetic algorithms. The *data mining and machine learning* algorithms are also exploited for network attacks detection. However, the training and the deployment of these methods are challenging because of lack of suitable datasets.

The authors of [25] propose some requirements on detection methods to be ideally usable in real networks. Detection should be as fast as possible with acceptable resource consumption. Detection should be accurate with low false alarms. Detection methods should be prepared for real network traffic that means they should be scalable for high-speed networks and the performance should be extensively evaluated. Detection methods should be prepared for up-to-date sophisticated network attacks. Detection methods should accurately segregate high-rate attack traffic from legitimate traffic such as flash crowds with minimum resource consumption and low false alarm rate. Detection methods should be able to handle an increased volume of data for processing and persist operable. Detection methods should be able to handle spoofed IP addresses. Detection methods should be dependent on a minimum number of input parameters if not independent of parameters and should also be based on a minimum number of traffic parameters or characteristics.

The issue of estimation of parameters of detection methods is partially solved in several different ways. Ozcelik et al. show in [34] that the current detection methods do not entirely solve the variation of the utilization of the network. It can be partially solved using multiple precomputed values of optimal parameters for different levels of utilization and different network infrastructures. Operating parameters of detection methods can be found using an exhaustive search as in [34].

In [35] the authors use the EWMA algorithm for a detection. They propose to estimate parameters from training data that is free of intrusions or attacks. Important parameters for detection are upper and lower control limits — thresholds.

The detection methods listed in this section have usually been applied on time series of some aggregated counters such as the number of packets with TCP SYN flag. The aggregation of characteristics does not allow to trace back the suspicion flows. Therefore, it is complicated to identify the real reason of an anomaly. The authors of [5] propose an architecture that uses a multi-stage Bloom filter or a sketch structure. The multi-layer reversible sketch (MLRS) and the count-min sketch (CMS) that were used in [5] are briefly described in [P.17].

Additionally, the authors of [5] use multi-chart CUSUM (MNP-CUSUM) proposed in [36] with the input taken from the sketch structure. The sequential MNP-CUSUM over a sketch for anomaly detection allows for detection of changes with a small delay and a low false alarm rate. An alert is raised when at least one of the test statistics reaches the threshold h_i.

Currently, the sketches are used in various detection systems, sometimes, in combination with different sophisticated mechanisms. Callegari et al. have proposed an architecture of

a detection system that is based on reversible sketches and neural networks in [37]. The paper is focused on short-term anomalies detection (without seasonal effect). However, the authors publish the comparison of neural networks and training algorithms that can be used for anomaly detection in computer networks. Another example of usability of the sketches is in [38] where the detection is based on Principal Component Analysis (PCA).

2.3 Big Data Processing

Big Data (described, e.g., in [39]) is a term describing a particular character of complex data that are challenging for storage, processing, and visualization. Big Data are significant for its three characteristics: velocity, variety, and volume. According to the definition, network traffic can be considered as the Big Data problem.

There are many papers about the parallel processing of network data. We focus on flow-based processing that is the current state-of-the-art. There are several software solutions for Big Data processing; one of the most popular is Hadoop. Technical report [40] shows that existing Big data frameworks are not efficient enough for some applications. However, Hadoop tools that support MapReduce algorithm have been used by many researchers.

It is necessary to split data for parallel processing, but this problem is usually handled just by general mechanisms of the frameworks. For instance, the authors of [41, 42] use a distributed database (like Hive or HBase) or a distributed filesystem (HDFS) to store data files. Authors of [43] analyzed IP, TCP and HTTP traffic stored in offline data files. Paper [44] presents experiments with several types of MapReduce jobs to analyze campus network traffic (e.g., computing volume of traffic per subnet).

Authors of [45] use Spark with Netmap to extract traffic features for detection of different types of DDoS attacks in real-time. The detection uses machine learning methods and relies on a distributed storage and an abstraction of objects called Resilient Distributed Dataset (RDD). The paper notes that the use of sampled data produces many false-positives.

Detection of DDoS and SYN flood is presented in [46]. The processing is based on HDFS and HBASE that uses a Bloom filter with several hash functions and a KEY that consists of the fixed n-tuple (src and dst IP addresses and ports, protocol and TCP flags). The authors do not explicitly explain the reason they chose the KEY, but they mention a condition of detection of an attack with 100 % accuracy.

In 2016, Cermak et al. in [47] presented a benchmark of stream processing systems (such as S4, Spark, Samza, Storm, and Flink) for parallel processing. The chosen operations (Identity, Filter, Count, Aggregation, TOP N, SYN DoS) were proposed for performance comparison.

Jirsik et al. in [12] presented the performance of the proposed system for stream processing based on Apache Big Data analytics tools as well. The benchmark was focused on a set of typical analysis queries, and according to the results, the systems were able to process up to 2 million flows/s on a cluster with 32 CPU cores in total. Contrary, the solution of the author of this dissertation thesis is obviously less resource consuming since

it can handle approximately the same flows/s with only half number of CPU cores (16) (described in [P.3]).

The dissertation thesis by Garofalo [48] deals with anomaly detection using a traditional Big Data approach. It describes existing Big Data analytics tools such as Apache Hadoop, Apache Storm, Apache Kafka, and Apache Spark. The author proposes anomaly detection using these tools. However, the datasets are currently out-of-the-date, and the work focuses only on a few types of anomaly (DoS, DDoS, horizontal scan, vertical scan).

Papers [47, 12, 48] do not explicitly mention evaluating the system with extended flow records with some L7 information (they only state that the flow records might be extended). Therefore, we can assume that they used only traditional flow data with information up to L4.

2.4 Semantic Relations in Data

Semantic relations in data and adverse effects of data splitting were mentioned in [49]. The authors show experiments with Hashdoop, an improved Hadoop, that splits data using CRC hashes of *srcip* and *dstip*. The authors chose a packet counting and ASTUTE algorithm for parallel processing. If more different algorithms were used, this hashing would not be efficient enough. The splitting based on IP addresses is just a particular case of the methodology described in this dissertation thesis.

The authors of [41] face the topic of data with semantic relations which require being processed together. Accurately, the paper describes an analysis of TCP connection using stitching flow parts (into one bidirectional connection). It is done by putting the whole traffic of the same IP addresses together.

The authors and their works that were mentioned in Section 2.3 use the existing frameworks for Big Data processing and distributed storage. Hence, they rely on the internal mechanisms of data distribution by the frameworks. The algorithms must handle this state, and it is usually solved by additional communication between the computing nodes to exchange data or results. That is the reason why the authors need not care about the semantic relation, even though the processing is not optimally efficient.

Chapter 3

Topics and Contributions in Details

The contributions of this dissertation thesis are all related to flow-based processing network traffic of high-speed networks. A practical and experimental part of this research is a developed NEMEA framework for a real-time flow-based analysis of the network traffic. NEMEA is introduced in Section 3.1 and it is described in more detail in the paper included in Appendix A.1. The main novel features of NEMEA are a *stream-wise flow data processing*, which is described in the Section 3.2, and an *application-awareness*, which is described in Section 3.3. Both features are used in the designed and developed detection modules to evaluate the concept and they were presented in the published reviewed papers. The theoretical part of this research is explained in Section 3.4 that defines important terminology and elaborates on the idea of semantic relations — *witnesses*. Some examples of the semantic relations are explained in Section 3.5. A feasible architecture for the parallel flow data processing is based on a Flow Scatter component is described in Section 3.6.

Some of the following sections refer to the overall structure of the dissertation thesis shown in Fig. 1.4. The name of the related block in the figure is mentioned in the name of a section (in the bracket).

3.1 Network Measurements Analysis Framework (NEMEA)

Before this research has started, the existing tools worked mostly in a traditional way of batch processing, where the flow data are stored into files on a file system before an analysis can start. Each file contains flow data within a fixed time window. Since each file must be closed before the analysis, there is a necessary delay before the processing starts. Also, one file may contain flow records that belong to a different time window due to delayed exporting. This variability complicates the development of detection algorithms. However, it is usually not an issue in practice. The need for data storage and delayed processing are the significant limits that we avoided using a *stream-wise approach* (described in Section 3.2).

To support this research and to allow experiments, there was a need for a stream-wise framework for the development of prototypes of detection algorithms. The aim was to test and evaluate new stream-wise detection algorithms with real data. Because of lack of such

tools in 2013, development of a new universal framework has started. Even though the batch approach of processing is still commonly used in practice, it is not efficient enough for processing a continuous stream of data from large networks anymore.

As a result of years of research and development, Network Measurements Analysis (NEMEA) Framework was released. NEMEA is deployed and analyzes flow data from the perimeter of the CESNET2 network, the Czech national academic network. It is an open source project that is publicly available at GitHub[1] for a world-wide community of network operators and researchers. Since the existence of NEMEA, several research activities were done, and there are published papers that show some of the developed detection methods and their feasibility in large-scale networks. Besides that, NEMEA itself was successfully presented in Canada in 2016 [P.4]. The included conference paper in Appendix A.1 describes NEMEA in more detail.

The main advantages of NEMEA are:

1. high modularity — each NEMEA module is an independent system process that receives and transmits flow data and runs in an operating system[2],

2. stream-wise approach of data processing (described in Section 3.2) that aims at low memory consumption and decreases a delay of detection,

3. application-awareness (described in Section 3.3), which is an ability to analyze any information field even from application layer headers and any NEMEA module may define new fields without affecting the rest of the system.

NEMEA modules work with data in a proprietary binary data format UniRec that was developed specially for fast in-memory processing and multiple access to data. UniRec specifies data representation (i.e., how and where to obtain information from the data messages). There is a UniRec API for C and Python languages that makes writing new NEMEA modules easier.

UniRec defines two main types of information fields: *fixed-length* and *variable-length*. Fixed-length fields are used mainly for frequently used fields of *basic flow* records, i.e., IP addresses, ports, timestamps, TCP flags, number of packets and bytes. Variable-length fields are more interesting for the application-awareness of NEMEA because a text string value can represent information of the L7 headers. Variable-length fields contain starting pointer and size of the "array of characters." Therefore, a UniRec message with some variable-length fields additionally contains a special table of offsets for them. The data access is designed to have minimal overhead.

The NEMEA framework also contains a unified API for communication between NEMEA modules. Developers of NEMEA modules do not need to solve interconnection of the modules at the time of development. The interconnection between NEMEA modules is handled by the framework at run-time. Such approach makes modules more flexible because they can be easily run locally (e.g., for experiments) or in a production deployment

[1] `https://github.com/CESNET/NEMEA`

[2] GNU/Linux CentOS 7 is primarily supported

without any change of the source codes. NEMEA modules might also run on different machines, and messages for processing are transferred via a computer network. Therefore, developers of a module may focus only on the development of a detection algorithm.

In 2017, NEMEA was used for educational purposes in the Network security course at the Faculty of Information Technology, Czech Technical University in Prague for the first time. Students can quickly do hands-on experiments with the stream-wise flow data analysis. Besides just usage of the existing NEMEA modules, it is possible to improve them or invent own modules with novel detection algorithms.

In the time of writing of this dissertation thesis, NEMEA contains 11 detection modules and about 22 general purpose modules for manipulation with UniRec messages. The detection modules can detect malicious traffic of the following categories:

- DNS tunnels,
- amplification attacks,
- anomalous volume of traffic per observed host,
- brute-force attacks (guessing passwords, user accounts, open SIP URI prefixes),
- communication with blacklisted servers,
- horizontal scanning,
- traffic from miners of cryptocurrency,
- vertical scanning (port scan),
- volumetric DDoS attacks characterized by a significant increase of traffic volume and the number of its sources.

Some of the detection algorithms are described or referenced from this thesis. Since the NEMEA system is a community project, there are detection modules that are not authored by the author of this dissertation thesis.

The main contribution of the author of this dissertation thesis is in the design concept of the overall architecture of the current version of the whole NEMEA system. Additionally, the author of the thesis was the essential developer of the communication interfaces of NEMEA modules and he was the supervisor and designer of the NEMEA modules created in the bachelor or master theses [P.30, P.31, P.34, P.35, P.36, P.37, P.38, P.39, P.40, P.41, P.42]. Besides, he was an initiator of an idea to write down the published papers that followed some of the mentioned theses and the essential author of the text of the papers [P.2, P.7, P.8, P.13]. The `vportscan_detector` NEMEA module (especially observations gained using the module) was published in [P.6] and it was not covered by any thesis. Finally, the author of the dissertation thesis is a leader of the team of over ten NEMEA developers located in Prague at the Faculty of Information Technology, CTU in Prague.

3.2 Stream-Wise Approach to Flow-Based Analysis (SW)

Traditional approaches of flow-based processing worked in batches, i.e., datasets are split into time intervals and processed when the time interval elapses. On the other hand, the stream-wise approach, which is preferred in this work, is based on an on-the-fly analysis of the flow data without any long-term storage. The input data come from monitoring probes continuously, and it is analyzed immediately. It is not intended to store data permanently but all detection methods store only necessary state information temporarily.

It is worth noting that even this stream-wise approach works with some potential time delays due to a standardized and commonly used process of flow records exporting. So-called inactive and active timeouts cause a delay in flow exporters. Both timeouts affect a particular time when a flow record is "exported," i.e., sent from the flow exporter to flow collector(s). The *inactive* timeout is used to recognize a closed connection, and it is measured as the maximal elapsed time since the timestamp of the last captured packet of the flow record. The *active* timeout is useful for exporting long active connection. A flow record is exported (split) when the elapsed time since the first packet of the flow record is longer than the *active* timeout.

Modern flow exporters send flow records continuously because the active timeout is checked on every update of a flow record in the flow cache and the inactive timeout is checked regularly. Therefore, the approach of processing (batch or stream-wise) depends mainly on a flow collector, where the records are being processed. This work focuses on stream-wise processing (security analysis) in a flow collector.

Stream-wise approach is inspired by traditional pipeline processing, where the input data are also processed continuously, i.e., as it is received without long-term storage. Therefore, it is possible to process a stream of Big Data using computing nodes with insufficient resources. To imagine the situation, let t_0 be the time of starting the detection system that processes data observed at monitoring probes. For practical reasons, the detection system should run without any interruption, so ideally, processing runs to infinity (t_∞).

There are flow records F_i received by the detection system at t_i. Naturally, for $i \in (-\infty, 0)$ the flow records were not processed because they were exported before the start of the detection system.

The detection system usually consists of various detection algorithms that are implemented as independent modules. Stream-wise approach of each module can be represented by the following equation:

$$S_t = A(F_t, S_{t-1}), \quad S_{\leq 0} = s_0, \tag{3.1}$$

where A is the algorithm that expects the current flow record F_t in time t and keeps its previous state S_{t-1} internally as an input. The algorithm returns a current state S_t that is related to the observed traffic. The algorithm aims to identify malicious traffic. When the detection module that runs the detection algorithm is started, the state S_0 is set to some initial constant s_0.

Detection modules should depend only on a currently received flow record and the internal state of the algorithm. However, the internal state can be a complex structure

of information. Some detection methods that analyze anomalous changes of behavior of network entities in time need information about history, which can be the previous state.

Additionally, the information about entities might be updated by appending new data into the structure. This causes an increase of memory consumption. Therefore, it is necessary to use some aging and cleanup of the current state when designing a stream-wise module. Some approximation should be considered to sustain low resource consumption. For instance, there can be some maximal limit of stored data per each analyzed entity (e.g., *srcip* or *dstip*).

3.3 Application-Aware Detection Methods (AA)

Various security threats can be detected using aggregated flow data with a low level of detail. Many methods can recognize attacks like network scanning, volumetric attacks or even brute-force password guessing, as it was described in [11]. However, many types of malicious traffic have very similar characteristics as the legitimate traffic, so it is hard to distinguish malicious and legitimate traffic reliably. There are two issues: either malicious traffic is not recognized, or some legitimate traffic is detected by mistake (false positive alerts). For example, multiple unsuccessful attempts of registration to the SIP device looks like a normal TCP or UDP communication (many short IP flows or, sometimes, only two longer IP flows) between two IP addresses. Meanwhile, it should be identified as a brute-force attack.

The included papers present several detection algorithms that use extended flow records with L7 information. The paper in Appendix A.2 describes the most straightforward usage of domain names and URLs in a filter that checks information from the flow records. For this purpose, flow records were enriched by `DNS_NAME` — a domain name extracted from a DNS request or response, `HTTP_HOST` — a hostname extracted from a `Host:` header of an HTTP request, `HTTP_URL` — URL from an HTTP request. These information fields are checked in publicly available blacklists, which are related to the identifiable suspicious activity of known malware samples. For instance, a connection to a host that is listed as a C&C server might indicate some malware infection.

It is known that users can encode and transfer user data via any communication protocol. Various scientific papers describe methods of communication tunnels also known as covert channels. The paper in Appendix A.3 presents another usage of information from DNS messages to detect a covert channel over DNS. This type of covert channel can be established using tools such as `iodine`[3]. The developed detection module that can detect iodine's tunnels uses information extracted from `DNS_RDATA` — an information field containing RDATA part of DNS answers.

The scope of some VoIP security events was elaborated in the papers in Appendix A.4 and Appendix A.5. The papers show detection algorithms that analyze `SIP_CALLED_PARTY` — called SIP URI that is contained in the SIP *To:* header. Analysis of the values of this

[3]http://code.kryo.se/iodine/

information field disclosed very suspicious traffic generated by penetration testing tools (according to `SIP_USER_AGENT` field) that tries to exploit a victim's SIP device. In addition, `SIP_CSEQ` field (*CSeq* header of SIP) is needed to match SIP requests and responses correctly. It was discovered that brute-force guessing the SIP account names or their password is observable as a high number of unsuccessful SIP requests, whereas the `SIP_STATUS_CODE` field represents the return value.

There is also an unpublished work described in the diploma thesis [P.34] on detection of spoofing the victim's system time using NTP (Network Time Protocol). The developed detection module analyzes time series of `NTP_ORIG` (origin timestamp) and `NTP_RECV` (received timestamp) and looks for suspicious changes.

Even though the described approach based on extended flow records increases the precision and reliability of detection, it must be used very carefully. This improvement of security (reliable detection of advanced application layer attacks) might conflict with the privacy of users. It is worth noting the described detection algorithms work with unencrypted traffic only, whereas all private data should be transferred encrypted according to the best practices. Nevertheless, many types of unencrypted traffic might disclose information about users. It is essential to design monitoring and detection systems so they can reliably protect users against malicious traffic instead of spying them or disrupt their activities.

Various documents deal with privacy issues and data protection. IETF community created a document RFC7258 [50] that claims a pervasive monitoring as a privacy issue. In addition, there is a General Data Protection Regulation (GDPR) of the European Union about protection of user data and restricting service providers in frivolous storing and processing user data. Naturally, many exceptions allow providers to process user data (e.g., because of security purposes). However, the scope of information and purpose of processing must always be kept in mind to avoid breaking users' privacy.

Sometimes, it is possible to avoid the danger of privacy issues by using transformed "anonymized" or "pseudonymized" data[4]. Processing such data is harmless since it is not possible to identify users. The application-aware approach described in this thesis is based on the extraction of particular headers from packets, whereas the user content (payload) is completely skipped. This set of information fields is selected according to the requirements of the detection algorithm. The fields can be trivially transformed so that the algorithm is not affected and the privacy of users is still preserved.

Concerning stream-wise approach, the detection methods do not store any piece of information permanently. In addition, information about IP addresses is deleted after a given time interval to keep the memory consumption low.

Having the detection methods that are described in later sections, it was possible to detect several application layer threats. The detection algorithms are described in more

[4]Both anonymized or pseudonymized data alone do not allow identification of users. However, pseudonymized data are usually retrieved by some encryption mechanism so they can be transformed back to the original data using a secret key. It is assumed that the detection system does not have access to the secret key and decryption must be done during the alert reporting or incident handling.

detail in Section 3.5 to explain principles of parallel processing presented in the following section.

3.4 Semantic Relations in Flow Data (WITNESS)

Section 3.4.1 and Section 3.4.2 contain several definitions that were written for this dissertation thesis. Afterward, the terms are used in Section 3.5 to describe selected detection algorithms symbolically.

3.4.1 General Terms

Definition 3.4.1. Network traffic: A sequence of all packets[5] that were sent/received via a computer network in a time window.

Definition 3.4.2. Malicious traffic: A subset of network traffic that is unwanted, either it aims to spend capacity of links, or it threatens a target device or service.

Generally, malicious traffic might be generated by any device or software that can send packets into the network. The source of malicious traffic can be, e.g., attackers, malware, botnets.

Examples of malicious traffic can be a flood (TCP SYN, UDP), general DoS or DDoS, communication of infected device with command and control server, unsolicited e-mail messages (SPAM, phishing, ...), communication with a server hosting malware/ransomware. In some network infrastructures especially in a business environment, an information exfiltration can be a severe issue. Therefore, covert channels can also be classified as malicious traffic.

Definition 3.4.3. Detector: An algorithm that identifies malicious traffic in the network traffic and generates alerts with information about the detected malicious traffic.

3.4.2 Witness types and witnesses

Definition 3.4.4. Witness type: A characteristic or feature of network traffic that is analyzed by a detector to identify particular malicious traffic. The witness type is dependent on the detector, its parameters, and a type of malicious traffic that ought to be detected.

Examples of witness types are *SYN packet rate* or for more precise detection *ratio of SYN, FIN, RST packets in time* for TCP SYN flood detection, number of unsuccessful authentication requests, number of different prefixes of SIP URI, or domain names.

[5] A *packet* is a basic transferred data unit, a sequence of bytes that represents protocol headers and payload.

Definition 3.4.5. Witness: A particular instance of a network traffic subset that was identified (detected) as the malicious traffic by a detector and that resulted in an alert created by the detector.

Since the malicious traffic is detectable using one or more witness types, many different witnesses can exist for one alert. Some examples of witness types and witnesses are listed below:

- Let us have communication of a piece of malware with a Command and Control (C&C) server — head of botnet. If the content of the communication is known (e.g., using reverse engineering), the witness type can be a *sequence of bytes* of a packet, and the witness is any *reassembled message* containing the sequence of bytes discoverable by some pattern matching algorithm.
- Having a malware sample with the fixed IP address as a C&C server, the witness type can be an *IP address* and the witness is any *packet* with this IP address as source or destination.
- Similarly, a C&C server can be specified as a domain name. The witness type can be the *domain name* in such case, and the witness is any *DNS query* from a client or any *HTTP request* containing this domain name as a hostname.
- When a C&C server uses specific communication patterns (e.g., deterministic timing such as once per hour of requests and responses with the known characteristics like number of packets, number of bytes, port numbers), the witness can be any *sequence of packets* that represents the communication and shows the sought characteristic. The same information might be equivalently contained in a *sequence of flow records* aggregating the information about the packets.

It should be noted that a witness need not to be a complete communication between two addresses. Statistic based detectors with thresholds may report an alert at any time after processing a minimal subset of network traffic that contains enough data (bytes, packets, flow records, ...) to reach the threshold. In such case, an alert is reported for every randomly selected subset of the network traffic that contains a witness.

Example 3.4.6. Minimal witness: Let us have a detector of port scans that is based on observation of the number of unique destination ports for each pair of source and destination addresses, but only TCP flow records with set SYN flag are processed. A threshold is set to 50 unique port numbers. The detector looks for the witness type: *any 50 unique destination TCP ports of a pair of source and destination addresses.* The minimal witness would be any *set of 50 flow records of destination and source addresses with different destination TCP ports.* It is clear that the pair of addresses may use much more TCP ports and any set of more than 50 flows with at least 50 unique ones results in an alert from the detector. However, if the detector observes less unique destination TCP ports than 50, the port scan is not detected.

Example 3.4.7. A complex example of terminology: Let us have network traffic targeted against a server. The network traffic consists of legitimate traffic as well as malicious traffic represented by an increased number of TCP SYN packets (TCP SYN flood or port scan). There is a general detector that observes a ratio of TCP SYN and TCP FIN packets (i.e., this ratio is the witness type for this detector) and a maximum threshold T of the ratio. The ratio is computed from flow records that are exported by a monitoring system. The witness is a *set of flow records* which causes that the detector reaches the given threshold of the ratio. Unfortunately, such a witness can be a "liar," e.g., when the threshold is too low, or the algorithm does not work correctly. An alert is a false positive in such cases.

3.5 Selected Detection Algorithms in Details

This section describes in more detail some of the developed detection algorithms that were published in the reviewed papers. The detection methods were designed concerning the stream-wise approach and are usually application-aware. The methods were implemented as modules of the presented NEMEA system, and they were deployed in the Czech national academic network.

This section aims to show examples of the stream-wise detection algorithms (detectors) that can be used in a parallel environment discussed in Section 3.6 and Section 3.6.1 based on the methodology about witnesses described in Section 3.4. It is possible to split a continuous stream of flow data into subsets that can be processed in parallel. When the splitting works according to the witness types, the malicious traffic in a traffic subset is still detectable. This approach allows for scalable processing that is feasible for large networks.

Some helpers are defined to describe and analyze algorithms in the following sections. A *flow record* is an n-tuple of information fields. It can be understood as a structure known in programming languages. To access an information field in flow records, we will use a notation with the operator [ ]. For example, the access to the *dstport* field (destination port number) in the flow record F_i can be written as follows:

$$F_i[dstport]. \tag{3.2}$$

.

Stream-wise algorithms work with a sequence of flow records F (or its subsets)

$$F = \{F_0, \ldots, F_n\},$$

where n goes to infinity.

Let `DISTINCT()` be an indicator, which is equal to 1 for the first occurrence i of a tracked attribute *attr* in the set of the observed flow records F:

$$\mathrm{DISTINCT}(F_i[attr]) = \begin{cases} 1 & \text{for } i = 0, \\ 1 & \text{for } F_i[attr] \notin \{F_0[attr], \ldots, F_{i-1}[attr]\}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

Sometimes, it is useful to use a concatenated value of multiple attributes at once. For instance, getting concatenated values of source and destination IP addresses (*srcip* and *dstip*) of the flow record F_i is represented by:

$$F_i[srcip/dstip]. \tag{3.4}$$

To compute the number of unique destination ports in the set of flow records F, let us define a function `COUNT()` as follows:

$$\text{COUNT}(F[dstport]) = \sum_{0}^{n} \text{DISTINCT}(F_i[dstport]). \tag{3.5}$$

Detection algorithms are usually specialized in the detection of some particular malicious traffic. Therefore, only a subset of flow data with some characteristic is interesting for detection. For instance, let us select a subset I of the set F, where I contains all flow records with the same source IP (*srcip*). Let us define a new symbol

$$\underset{srcip}{\subset}$$

that is used to describe this relation between I and F. Let F and I are sets of flow records, where I is a subset of F and I contains only flow records with the equal value of the attribute *srcip*. This relation can be described as follows:

$$\begin{aligned} F &= \{F_0, \ldots, F_n\}, \\ I &= \{I_0, \ldots, I_m\}, \\ I \underset{srcip}{\subset} F &: I \subset F, \quad \forall i, j : I_i[srcip] = I_j[srcip]. \end{aligned} \tag{3.6}$$

Naturally, F can contain multiple different subsets I (i.e., with different values of the attribute), whereas each flow record of the subset I has the same value of attribute *attr* used in the operator $\underset{attr}{\subset}$. Given that a detection algorithm analyzes a subset I to detect an anomaly, it is necessary to analyze all existing subsets I separately. This will be described using a loop over all existing subsets in the following algorithms.

3.5.1 Detection of Known Features (KF)

Detection of malicious traffic can be based on searching for known patterns or traffic features. There are lots of lists of suspicious IP addresses, domain names or URLs. These lists are usually called as blacklists. Flow-based detection can easily check the presence of the traffic features on any blacklist.

For instance, there are blacklists of Command and Control (C&C) servers, which are publicly available and maintained. Since any communication with an IP address that is blacklisted as a C&C server is very suspicious and may be an indication of infected device, it is helpful to perform online checking of the observer network entities (IP addresses, hostnames, URLs) against such blacklists. Cooperation with the forensic laboratory to retrieve

and use information from malware samples was described in [P.15], which is included below in this section.

A use-case similar to C&C server blacklists is a detection filter that was used for finding communication with so-called booter services. The topic of booters is discussed in detail in [51] by J. Santanna et al. A real-time detection of the communication with booter sites was presented at TF-CSIRT meeting in Valencia 2017 during the Network Monitoring Training session[6].

Algorithm 3.1 Detection of Communication with Known Feature

$F = \{F_0, \ldots, F_n\}$
for all $F_i \in F$ **do**
  if $F_i[feature] \in Blacklist$ **then**
    $\text{Alert}_{\text{feature}}$
  end if
end for

The detection algorithm is described in Alg. 3.1 and represents straightforward filtering the observed flow records F. Every time a flow record contains a *feature* that is listed in any blacklist, an alert is reported. The *feature* can be for example an IP address, a domain name looked up using DNS, a hostname used in HTTP traffic. Naturally, there are many different blacklists, and the equation shows just one general *Blacklist* for simplicity.

According to Alg. 3.1, we can derive a witness characteristic as follows: an alert is reported when at least one flow record with some blacklisted feature is observed. As it will be discussed in Section 3.6, this kind of detection can be parallelized using the splitting flow data stream into separate substreams because the detector does not check any relations between flow records.

Application of the described filter on L7 was presented in the included paper in Appendix A.2.

3.5.2 Detection of DNS Misusage — Covert Channel (DNS)

Domain Name System (DNS) [52] is one of the most important communication protocols for the functionality of modern networks. This protocol is used to translate human-readable domain names and IP addresses. It is usually allowed in security policies so that any connected device can perform translation either directly or via a local DNS server.

The paper included in Appendix A.3 describes a scenario of a user that uses DNS messages to establish a communication tunnel from a restricted area. The paper also shows a detection method that is based on the analysis of a sequence of extended flow records D with L7 information from DNS messages. The reason is that the DNS requests and responses can carry any encoded user data. The main part of the detection algorithm is a computation of the number of different subdomains for each domain name of the higher level since subdomains are used for data transfer.

[6] https://www.first.org/events/symposium/valencia2017/program

To find the number of distinct requested subdomains, an indicator

DISTINCT_SUBDOMAIN()

is needed. In the implemented detection module, there is a prefix tree data structure, which identifies distinct subdomains and computes their number. For this text, it is not necessary to describe DISTINCT_SUBDOMAIN() in more detail.

The detection algorithm is described in Alg. 3.2, where D is a sequence of flow records that contain DNS information. Let us have a subset I of the flow records D, which belong to the same IP address. The detection module computes the number of distinct subdomains. Therefore, we can derive the witness type as follows: an alert is reported when at least Thr_1 different subdomains are observed per source IP address.

Algorithm 3.2 Detection of Communication Tunnels over DNS

for all $I \underset{srcip}{\subset} D$ **do**
  $S_1 = \text{COUNT}(\text{DISTINCT\_SUBDOMAIN}(D[domain]))$
  if $S_1 \geq Thr_1$ **then**
    $\text{Alert}_{\text{dnstun}}$
  end if
end for

The algorithm was implemented as a NEMEA module. The implementation and results of deployment are presented in the included paper in Appendix. A.3.

3.5.3 Detection of Attacks Against VoIP Infrastructure (SIP)

Voice over IP (VoIP) is a successor of Public Switched Telephone Network (PSTN). It uses standard computer network infrastructure. One of the well-known signaling protocols, which is used for establishment or termination of phone calls, is Session Initiation Protocol (SIP). Even though there are many best practices and specification for securing SIP infrastructure, in practice, it is usually operated without any encryption.

The SIP messages in plain text allow for security analysis that can be used for successful detection of malicious activity. Using the NEMEA framework, several detection methods were developed to analyze information about SIP traffic. As it was mentioned in Section 3.3, exporting some L7 headers helps to reliably discover attempts of attackers to find a vulnerable setting of SIP devices.

Scanning and Password Guessing

There is a NEMEA module that can detect brute-force password guessing. This module (`sip_bf_detector`) analyzes the number of SIP messages that represent unsuccessful authentication attempts. The algorithm is described by Alg. 3.4, where we have a subset I of the set F containing all observed flow records. I contains SIP extended flow records

with the same n-tuple source IP, destination IP and To SIP URI. The algorithm analyzes mainly the failed attempts, which are identified by exported information *sipstatus*. The number of failed attempts is compared with Thr_2 to trigger an alert. Therefore, we can derive the witness type as follows: an alert is reported when at least Thr_2 unsuccessful authentication attempts are observed per any n-tuple of the destination IP (which is a SIP client), the source IP (which is a SIP server) and the callee SIP URI (To:).

$$\text{FAILURE}(F_i) = \begin{cases} 1 & \text{for } F_i[sipstatus] = 401, \\ 0 & \text{otherwise.} \end{cases} \tag{3.7}$$

Algorithm 3.3 Detection of SIP Brute Force

for all $L \underset{dstip/sipto}{\subset} I$ **do**
  $S_2 = \sum_0^n \text{FAILURE}(I_i)$
  if $S_2 \geq Thr_2$ **then**
    $\text{Alert}_{\text{sipbf}}$
  end if
end for

In addition, the same `sip_bf_detector` module can detect scanning user accounts. The scanning is based on analysis of responses from a SIP device to a user/attacker, whereas an attacker tries a sequence of different SIP URI. The algorithm is described in Alg. 3.4, whereas I is a set of SIP extended flow records of the same pairs of source IP and destination IP. The algorithm computes the number of unique callee SIP URI.

Algorithm 3.4 Detection of SIP Scanning

for all $M \underset{(srcip/dstip)}{\subset} I$ **do**
  $S_3 = \text{COUNT}(\text{DISTINCT}(M[sipto]))$
  if $S_3 \geq Thr_3$ **then**
    $\text{Alert}_{\text{sipscan}}$
  end if
end for

Finally, the detection module is capable of detection of distributed brute-force attack. When the number of unique source IP observed for any pair of destination IP and callee SIP URI is higher than Thr_4, it is assumed to be a distributed scan.

All three detection algorithms were implemented as one (`sip_bf_detector`) module for NEMEA. The paper included in Appendix A.4 describes the algorithm and its implementation. There are also results and observation from the deployment of the detection module on the real backbone network.

Algorithm 3.5 Detection of SIP Distributed Bruteforce

for all $N \underset{(dstip/sipto)}{\subset} I$ **do**
  $S_4 = \text{COUNT}(\text{DISTINCT}(I[srcip]))$
  if $S_4 \geq Thr_4$ **then**
    $\text{Alert}_{\text{sipdistbf}}$
  end if
end for

Dial Scheme Guessing

The paper included in Appendix A.5 focuses on detection of another malicious traffic via SIP. There are SIP devices with a weak and vulnerable configuration that allows an unknown user (without authentication) to establish a phone call based on knowledge of a dial scheme. In practice, it means when an attacker knows or guesses a special prefix, it is possible to make a call to some phone number.

Therefore, it is very usual that network traffic contains a lot of requests, mainly the INVITE messages, with various prefixes. Such attempts try to find the right prefix. Since there is a danger that a vulnerable device establishes a call to some premium rate number that can lead to a significant financial loss, it is useful to focus on detection of this kind of malicious traffic.

The detection algorithm is based on the analysis of the SIP requests from the same source IP. The analyzed header field is the *SIP To:* URI. To find the number of distinct requested prefixes,

DISTINCT_PREFIXES()

indicator is needed. In the implemented detection module, there is a prefix tree data structure, which identifies distinct subdomains and computes their number. However, for this text, it is not necessary to describe DISTINCT_PREFIXES() in more detail. The detection algorithm is described in Alg. 3.6. The detection module computes the number of distinct prefixes in the *SIP To:* URI. Therefore, we can derive the witness type as follows: an alert is reported when at least Thr_5 different prefixes are observed per source IP address.

Algorithm 3.6 Detection of SIP Dial Scheme Guessing

for all $I \underset{srcip}{\subset} F$ **do**
  $S_5 = \text{COUNT}(\text{DISTINCT\_PREFIXES}(I[sipto]))$
  if $S_5 \geq Thr_5$ **then**
    $\text{Alert}_{\text{dialscheme}}$
  end if
end for

The algorithm was implemented as a NEMEA module, and it was presented in the paper included in Appendix A.5.

3.5.4 Vertical and Horizontal Port Scan (VS)

Network scanning is a widespread activity observable in every network. It is generally an innocent activity performed by, e.g., network operators. However, network scanning can also be a useful source of information for attackers, who want to find potentially vulnerable targets. Therefore, it makes sense to track such behavior in the network.

There are many approaches for detection of network scanning. Included paper in Appendix A.6 presents a simple method based on the computation of statistics per each host (IP address). The value, which is computed from flow records of specific characteristics, is then compared with defined threshold Thr_6. The detection algorithm is described in Alg. 3.7 and it was implemented as a stream-wise module of the NEMEA system.

Algorithm 3.7 Detection of vertical scanning

for all $K \underset{srcip/dstip}{\subset} I$ **do**
  $S_6 = \text{COUNT}(K[dstport])$
  if $S_6 \geq Thr_6$ **then**
    $\text{Alert}_{\text{vscan}}$
  end if
end for

For **vertical scans**, the number of unique destination TCP ports is counted for each pair of source and destination IP but only flows with a low number of packets and only SYN flag set are considered. Therefore, we can derive a witness characteristic as follows: an alert is reported when at least Thr_6 unique destination ports are observed per one pair of source and destination IP addresses.

Considering witnesses, we should ideally keep flow records of each pair of source and destination addresses together. However, it is sufficient to analyze at least Thr_6 flow records of a pair with different destination ports together in one detector to identify a scan.

The detection method of **horizontal scanning** works similarly. In this case, the number of different destination IP addresses is counted per each source address and destination port. Therefore, we can derive a witness characteristic as follows: an alert is reported when at least Thr_7 unique destination addresses are observed per source address and destination port.

3.5.5 Brute-Force Password Guessing (BF)

To get users' credentials, attackers can use various approaches such as social engineering, phishing, however, these approaches are out of the scope of this work. A brute-force password guessing that is performed via the network is more interesting for this analysis. It is a well-known attack which is based on checking various combinations of the username and password. Having valid knowledge of the user's credentials, an attacker may gain access to the resources and services of the targeted victim.

Usually, attacks use some dictionaries, which list popular usernames and passwords. Therefore, the guessing checks all entries of a dictionary one by one.

Detection of brute-force attacks in encrypted data is complicated. However, there are studies (like in [11]) that show successful detection based on analysis of both directions of communication between the two hosts. The detection algorithms assume that characteristics of traffic of successful and unsuccessful authentication differ.

An indicator CHECKBRUTEFORCE() used in Algorithm 3.8 is dependent on the analyzed communication protocol and the detection approach. The subset K represents all flow records between two IP addresses in both directions. $K \underset{biflow}{\subset} F$ can be described as a subset with the following condition:

$$K \underset{biflow}{\subset} F : K \subset F, \quad \forall i,j : \begin{cases} K_i[srcip] = K_j[srcip] \text{ and } K_i[dstip] = K_j[dstip], \\ K_i[srcip] = K_j[dstip] \text{ and } K_i[dstip] = K_j[srcip]. \end{cases} \tag{3.8}$$

Algorithm 3.8 Detection of Brute Force Guessing

```
for all K ⊂ F (biflow) do
    if CHECKBRUTEFORCE(K) then
        Alert_bf
    end if
end for
```

3.5.6 Distributed Denial of Service (DDoS)

DDoS attacks are frequent and very powerful. They are based on depletion of a victim's resources such as computing power, memory, or network bandwidth. When a victim receives too many requests from attacking devices, it is no longer able to respond to legitimate users, so the service seems to be unavailable.

There are many ways how to achieve this state. Generally, botnets are being used for this purpose. In such case, each device from the botnet creates some small number of requests against the victim. Since there are plenty of devices in botnets (more than thousands), the overall volume of traffic against the victim is higher than the victim can handle. The attacks may generate over hundreds of gigabits per second traffic.

A straightforward detection algorithm can analyze the number of bytes and the number of unique clients observed within a time window. Alg. 3.9 shows a detection algorithm, where Thr_8 is a threshold for the number of unique sources of traffic, and Thr_9 is a threshold for the total number of bytes transferred to the destination IP.

The supervised bachelor thesis [P.30] has studied DDoS attacks and their detection. The result of the thesis was a NEMEA module for detection of DDoS attacks.

Algorithm 3.9 Detection of DDoS

for all $J \underset{dstip}{\subset} F$ **do**
  $S_8 = \text{COUNT}(\text{DISTINCT}(J[srcip])$
  $S_9 = \text{COUNT}(J[bytes])$
  if $S_8 \geq Thr_8$ and $S_9 \geq Thr_9$ **then**
    $\text{Alert}_{\text{ddos}}$
  end if
end for

3.6 Construction of Flow Data Scatter (PARINFRA)

This thesis presents a parallel approach to flow data processing at (near) real-time. The infrastructure for parallel processing that was built for our experiments is shown in the high-level view in Fig. 3.1. The Flow Scatter box splits a single stream of flow records (incoming flow data) into subsets and distributes the subsets among multiple equivalent computing nodes. Equivalent means that every node has the same configuration of the detection modules. Each node processes independent set of flow records without additional communication with other nodes. The results are alerts containing detected security events.

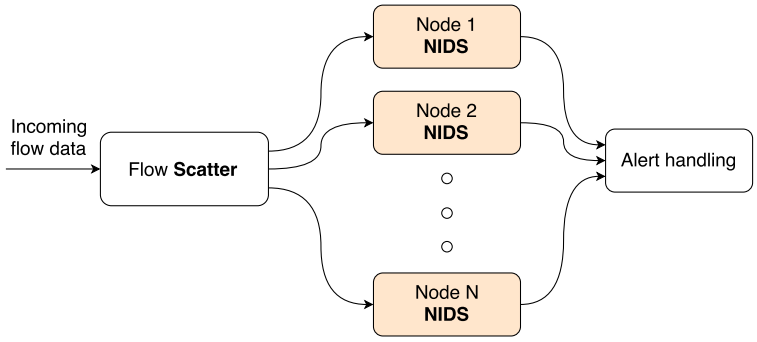


Figure 3.1: A high-level view of the infrastructure for parallel stream-wise processing using a Flow Scatter. This figure was published in the paper included in Appendix A.7.

The heart of the infrastructure is the Flow Scatter that must decide which computing node processes which flow record. The decision must be made with minimal delay; otherwise, the Flow Scatter is a bottleneck of the whole infrastructure. In addition, the selection of a node must be made concerning semantic relations in data, whereas the importance of such relations was explained in Section 3.4 and in the paper included in Appendix A.8. The experiments about splitting the stream of flow records were described in the paper included in Appendix A.7. The experiments showed that disrupting the semantic relations affects the performance of the detection algorithms, i.e., a security event might be undetected if there is not enough data together on one computing node.

The detection algorithms that were described in Alg. 3.1 Alg. 3.2, Alg. 3.3, Alg. 3.4, Alg. 3.5, Alg. 3.6, Alg. 3.7, Alg. 3.8, Alg. 3.9, show characteristics for the observed traffic

that must be fulfilled to successfully identify the anomalous traffic by an algorithm with particular parameters.

Construction of the Flow Scatter that respects semantic relations depends on a set of detection algorithms that are used. The practical experiments with the flow data captured from a real backbone network and the set of detection modules (the NEMEA system) showed that a Flow Scatter can distribute flow records uniformly among computing nodes to the algorithms of the same group, and there are different groups of algorithms (scattering groups). Each scattering group is identified by common characteristics that define *"what data should stay together."* Therefore, the first challenge is to find the scattering groups, whereas each group is then implemented as a hashing method in the scatter.

Algorithm 3.10 shows a straightforward way how to find such groups of algorithms. Let us have a set of the scattering groups S, which is empty at the beginning, and the list A of the detection algorithms, which are formally described. The set S contains descriptions of subsets of flow records that contain complete non-broken witnesses. Every scattering group S_i contains algorithms with similar witness types, and therefore it defines a scattering function of the scatter that chooses the computing node number for processing a particular flow record by the group of algorithms. A description of the witness type of the algorithm A_i is represented by $\delta(A)$.

Algorithm 3.10 Construction of scatter

$S = \emptyset$
for all $A_i \in A$ **do**
  if $\delta(A_i) \in S$ **then**
    do nothing, witness is already contained
  else if $S_j \subset \delta(A_i)$ for any j **then**
    change S_j to contain witnesses of A_i
  else
    add $\delta(A_i)$ into S.
  end if
end for
S contains at least one scattering group

More clearly, 3.10 puts all algorithms into scattering groups. During the grouping, descriptions of the witness types δ are compared and the algorithm A_i is put into the existing scattering group S_j if and only if the $\delta(A_i)$ is a subset of $\delta(A_k)$, whereas A_k already belongs to S_j. Assuming the set S is implemented optimally, the algorithm must iterate over all n algorithms only once. Let us also assume that the time of testing whether an A_i belongs into S_j is constant. As a result, the computational complexity of Alg. 3.10 is $O(n)$.

The experiments from Appendix A.7 identify three different scattering groups as it is shown in Fig. 3.2. Generally, some algorithms analyze flow records of the same *srcip*, *dstip*, and the last group contains algorithms that expected bi-directional flow records, i.e., traffic between two IP addresses in both directions. Therefore, the Flow Scatter has three

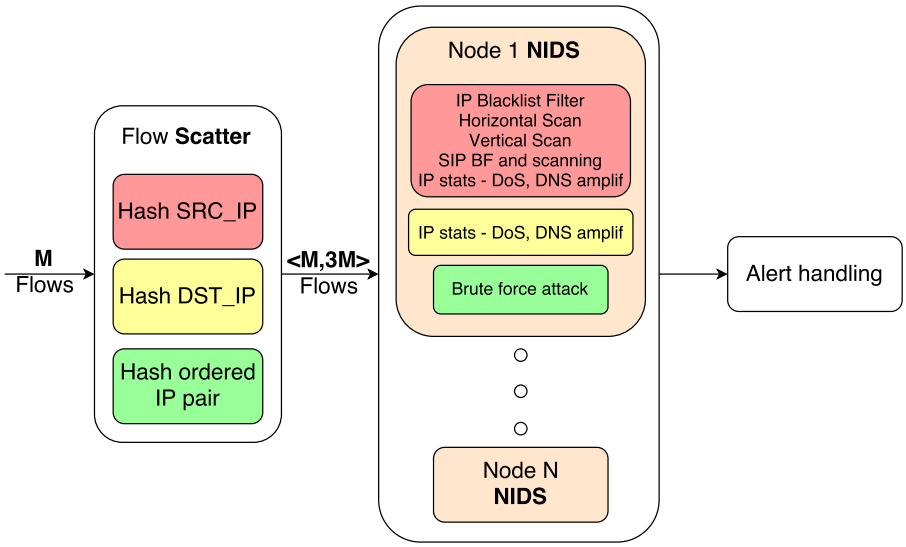


Figure 3.2: Example of three groups of algorithms that are controlled by a Flow Scatter. This figure was published in the paper included in Appendix A.7.

scattering functions inside. Since all scattering functions are computed for each incoming flow records, it is possible that the Flow Scatter duplicates a flow record and resends it to at most three computing nodes. However, the flow record is processed only once by all detection algorithms (even though it might be on different nodes).

3.6.1 Scalable Infrastructure for Processing with Flow Scatters

To overcome a possible throughput limit of a single Flow Scatter that would affect the performance of the whole infrastructure, an extended scheme with multiple Flow Scatters was designed. Figure 3.3 presents a set of computing nodes known from Fig. 3.1. In addition, there is a set of Flow Scatters that receive the flow data. In the figure, there is also a Round-Robin Scrubber, which simply distributes the single stream of flow data among the Flow Scatters, however, it is also possible that each Flow Scatter receives a separate flow data stream (e.g., from flow exporters). The point is that all Flow Scatters have completely same configuration and so the resulting node number is always the same no matter which Flow Scatter computed it. Naturally, the number of Flow Scatters is not limited.

3.7 Recapitulation of the Main Contributions

This chapter presented the main contributions of this dissertation thesis. It contains a feasible approach to the stream-wise analysis of flow data, usage of extended flow records with L7 information, real examples of the developed detection algorithms and, finally, the

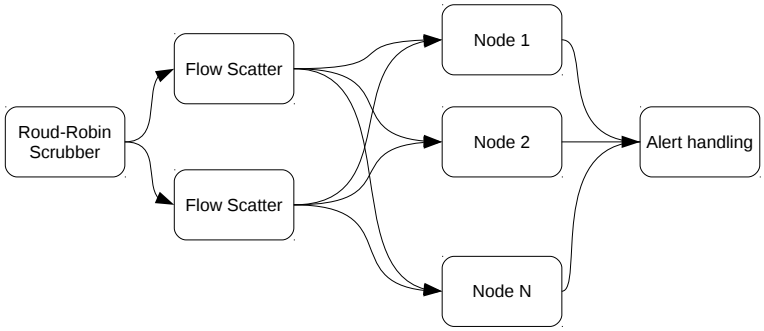


Figure 3.3: Scaling infrastructure contains multiple computing nodes with Flow Scatter. It is possible to duplicate the Flow Scatter to increase the maximal throughput of one scatter, i.e., to scale up. It is assumed that all used Flow Scatters have the same configuration.

scalable infrastructure for parallel analysis of flow data. The parallel analysis uses a Flow Scatter that is created according to witness types of the described detection algorithms.

This work is built on the NEMEA system that was developed as a proof-of-concept during the last years. It became an important platform for our research, development, and experiments.

The following paragraphs summarize the contributions from a higher perspective.

Processing high data volume for anomaly detection at real-time

Modern network infrastructures, especially backbone networks, produce high data volume, and it is complicated to store and analyze even aggregated data — flow records. Stream-wise processing (Section 3.2) was proposed as a feasible approach to process such data.

The stream-wise approach was implemented as a Network Measurements Analysis (NEMEA) framework that was published in [P.4]. Several detection methods were developed in the last years as a proof-of-concept for this work. NEMEA has been successfully deployed in the Czech national academic network (NREN) and helps to discover suspicious traffic.

The NEMEA system contains several detection methods that are analyzing extended flow records with carefully selected information from headers of application protocols that are used unencrypted. This application aware approach (described in Section 3.3) allows for detection of attacks that are hidden from classical analysis based on basic flow data.

Modules of the NEMEA system can be easily distributed among multiple computing nodes because of the modular nature of the system's design. However, the main power of distributed processing is in the possible splitting the data into substreams of flow data that can be processed in parallel.

Data distribution for parallel processing with preserved detection results

The part of this research that was published in [P.1] defines semantic relations in the flow data that are necessary for successful anomaly detection. Moreover, the paper [P.3], which

was created in cooperation with my master student, shows these relations in practice using experiments with real data. Both papers show the impact of breaking the relations during some careless splitting of a stream of flow data into subsets.

Section 3.5 contains several case studies, whereas most of them are also described in Appendix A in the included papers that were published at international conferences. The papers cover some detection abilities of the NEMEA system that was used as the platform for experiments with processing high-speed network data. Besides the included papers, the section describes formal descriptions of the detection algorithms to show so-called witnesses that were defined in Section 3.4.2 as well as in [P.1].

In Section 3.6, there is an algorithm that can be used to take information from the formal descriptions of the detection algorithms to design a unique Flow Scatter for splitting a stream of flow data concerning witnesses. The constructed scatter can be used to create subsets of data that can be processed separately in parallel without affecting detection results. The benefit of the Flow Scatter is the higher granularity of the flow data subsets, so even massive volumetric attacks targeted against one destination do not affect only one computing node of the monitoring and analysis system. Therefore, the witness-based scatter protects the detection system against overload.

The proposed witness-based scatter splits the single stream of flow data into multiple separate substreams. Generally, the approaches of splitting data are based on fixed basic characteristics of the traffic such as IP addresses. Contrary to this traditional approach, the Flow Scatter in this thesis goes further in the splitting the data because it takes the witnesses into account. A witness, or more general witness type, is defined using many factors such as the type of the traffic to detect, chosen detection algorithm and its parameters, values of thresholds.

Naturally, the scatter is designed concerning the stream-wise approach too, so the set of rules, which are used for splitting, are evaluated at real-time and every flow record is passed to the right computing node immediately. The current prototype of the Flow Scatter is based mostly on hash functions that compute the identifier of the computing node.

Since the decision of the scatter takes a non-zero time due to the computational complexity of the rules evaluation, the throughput of one scatter is limited in the real deployment. To overcome this limit, we have designed the architecture of a scatter cascade in the diploma thesis of M. Švepeš [P.33] that was supervised by the author of this dissertation thesis.

Chapter 4

Conclusions

Network monitoring, as well as anomaly detection, are essential parts of every well-running network infrastructure. Since the network infrastructures grow and the bandwidth increases, it is necessary to adapt monitoring and analysis systems to handle high data volumes. Additionally, many security threats are difficult to detect using traditional approaches based on *basic flow data* (NetFlow). Therefore, this dissertation thesis deals with several challenges to overcome the current state and prepare the systems for the future.

The contributions of this thesis, as they were described in the previous chapters, might be divided into several areas. From the practical point of view, a NEMEA framework for the analysis of IP flows has been designed and developed. The framework allowed for performing practical experiments with real network traffic monitoring as well as starting this research in multiple scopes: *stream-wise processing*, *application-aware detection algorithms*, *semantic relations in the flow data*, *parallel processing the flow data*.

Monitoring systems of large networks produce high data volumes of flow data. Operators ought to store the flow data by law; however, only basic flow records with information up to transport layer (L4) are being stored. Additionally, the traditional approach to an analysis of the basic data is based on processing the fixed length time windows. The fixed time windows necessarily cause higher detection delays. Therefore, this research on stream-wise processing started. The stream-wise algorithms process data immediately without any need of waiting for the end of a time window, i.e., they work *online*. Besides, the stream-wise algorithms do not store data permanently by design. They are designed to process infinite stream of flow data on-the-fly.

This research showed that some advanced security threats are invisible for the traditional NetFlow tools. Such security events look like legitimate flow records. The application-aware detection methods were studied and designed to overcome the lack of visibility. The application-aware approach is based on extended flow records containing information from higher protocol layers (up to application layer — L7). Application headers from unencrypted network traffic allow for reliable detection of some application threats such as brute-force dictionary attacks. Naturally, this approach requires capable tools that can

process extended flow records. This was the reason for developing the NEMEA framework.

Years of studying and development of detection algorithms brought experiences about semantic relations in the flow data. Practical experiments with the real network traffic from the Czech academic network showed that breaking the relations in the flow data has a significant impact on the detection results. However, it is useful to split a stream of flow data into independent substreams. It was hence necessary to find a methodology that would preserve those semantic relations during the splitting. A published paper describes the semantic relations as witnesses in the flow data that must remain complete to keep the security events detectable.

The use witnesses allows for parallel processing that is essential for analysis of the increasing volume of flow data. This dissertation thesis describes a Flow Scatter module that is constructed according to the witness types. The Flow Scatter splits the stream of flow records among multiple computing nodes. As a result, the monitoring and detection system can scale up.

Finally, the valuable achievement of this many-years work is the deployment of NEMEA in the Czech national research and education network (NREN) — CESNET2, in a significant commercial data center & ISP network in the Czech Republic, and in NREN of Switzerland. The author of this dissertation thesis has successfully started to build a community of researchers and developers that contribute to this project. NEMEA was also successfully used for educational purposes, and it is planned to continue with this effort because NEMEA is a suitable tool for students and researchers to study network traffic and detection algorithms.

This thesis is a collection of published papers on anomaly detection in computer networks. Each paper represents a particular contribution in the network security area, particularly, in the detection of suspicious traffic using flow data from monitoring systems. This work focused on large high-speed networks, especially the backbone ones. The topics and contributions of this dissertation thesis were described as parts of the published papers. The papers also contain some observations and statistics about the real network traffic that could be gathered using the developed tools. All works have been tested and deployed in the Czech national academic network and therefore the papers usually contain results and statistics from our experiments with the real network traffic that was observed.

4.1 Future Work

As a future research, there are several unsolved challenges related to this dissertation thesis:

- automatic adaptation of the Flow Scatter based on, e.g., Machine learning algorithms,
- automatic mitigation triggered by the detected security events,
- granularity of the Flow Scatter — find minimal subsets and sample data among the computing nodes without affecting the detection results.

Bibliography

[1] Krebs, B. DDoS on Dyn Impacts Twitter, Spotify, Reddit. Available at: `https://krebsonsecurity.com/2016/10/ddos-on-dyn-impacts-twitter-spotify-reddit/`, last checked on: 2018-05-30.

[2] Claise, B. Cisco Systems NetFlow Services Export Version 9. http://www.ietf.org/rfc/rfc3954.txt, Oct. 2004.

[3] Bhuyan, M. H.; Bhattacharyya, D. K.; Kalita, J. K. *Network Traffic Anomaly Detection and Prevention: Concepts, Techniques, and Tools*. Springer, 2017.

[4] Hu, J.; Yu, X.; Qiu, D.; et al. A simple and efficient hidden Markov model scheme for host-based anomaly intrusion detection. *Network, IEEE*, volume 23, no. 1, 2009: pp. 42–47.

[5] Salem, O.; Vaton, S.; Gravey, A. A scalable, efficient and informative approach for anomaly-based intrusion detection systems: theory and practice. *International Journal of Network Management*, volume 20, no. 5, 2010: pp. 271–293.

[6] Santiago del Rio, P. M.; Rossi, D.; Gringoli, F.; et al. Wire-speed Statistical Classification of Network Traffic on Commodity Hardware. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC '12, New York, NY, USA: ACM, 2012, ISBN 978-1-4503-1705-4, pp. 65–72, doi:10.1145/2398776.2398784.

[7] Kekely, L.; Pus, V.; Korenek, J. Software defined monitoring of application protocols. In *INFOCOM, 2014 Proceedings IEEE*, IEEE, 2014, pp. 1725–1733.

[8] Puš, V. Hardware acceleration for measurements in 100 gb/s networks. In *Proceedings of the 6th IFIP WG 6.6 international autonomous infrastructure, management, and security conference on Dependable Networks and Services*, AIMS'12, Berlin, Heidelberg: Springer-Verlag, 2012, ISBN 978-3-642-30632-7, pp. 46–49, doi:10.1007/978-3-642-30633-4_7. Available at: `http://dx.doi.org/10.1007/978-3-642-30633-4_7`

[9] Hofstede, R.; Bartos, V.; Sperotto, A.; et al. Towards real-time intrusion detection for NetFlow and IPFIX. 2013.

[10] Hofstede, R.; Čeleda, P.; Trammell, B.; et al. Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX. *IEEE Communications Surveys & Tutorials*, volume 16, no. 4: pp. 2037–2064.

[11] Hellemons, L.; Hendriks, L.; Hofstede, R.; et al. *SSHCure: A Flow-Based SSH Intrusion Detection System*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN 978-3-642-30633-4, pp. 86–97, doi:10.1007/978-3-642-30633-4_11. Available at: http://dx.doi.org/10.1007/978-3-642-30633-4_11

[12] Jirsik, T.; Cermak, M.; Tovarnak, D.; et al. Toward Stream-Based IP Flow Analysis. *IEEE Communications Magazine*, volume 55, no. 7, 2017: pp. 70–76, ISSN 0163-6804, doi:10.1109/MCOM.2017.1600972.

[13] Claise, B.; Trammell, B.; Aitken, P. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011 (INTERNET STANDARD), Sept. 2013. Available at: http://www.ietf.org/rfc/rfc7011.txt

[14] CESNET, a.l.e. Liberouter / Cards. Available at: https://www.liberouter.org/technologies/cards/, last checked on: 2018-05-30.

[15] Chaudet, C.; Fleury, E.; Lassous, I. G.; et al. Optimal positioning of active and passive monitoring devices. In *Proceedings of the 2005 ACM conference on Emerging network experiment and technology*, ACM, 2005, pp. 71–82.

[16] GÉANT et al. PerfSONAR. 2017. Available at: https://www.perfsonar.net, last checked on: 2018-05-30.

[17] Enterprises, N. Nagios. 2017. Available at: https://www.nagios.org, last checked on: 2018-05-30.

[18] NCC, R. RIPE Atlas. 2017. Available at: https://atlas.ripe.net, last checked on: 2018-05-30.

[19] Flowmon Networks. Flowmon ADS. 2017. Available at: https://www.flowmon.com/, last checked on: 2018-05-30.

[20] Hoque, N.; Bhuyan, M. H.; Baishya, R. C.; et al. Network attacks: Taxonomy, tools and systems. *Journal of Network and Computer Applications*, volume 40, 2014: pp. 307–324.

[21] Ahmed, M.; Mahmood, A. N.; Hu, J. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, volume 60, 2016: pp. 19 – 31, ISSN 1084-8045, doi:https://doi.org/10.1016/j.jnca.2015.11.016. Available at: http://www.sciencedirect.com/science/article/pii/S1084804515002891

[22] Bhuyan, M. H.; Bhattacharyya, D. K.; Kalita, J. K. Network Anomaly Detection: Methods, Systems and Tools. *IEEE Communications Surveys Tutorials*, volume 16, no. 1, First 2014: pp. 303–336, ISSN 1553-877X, doi:10.1109/SURV.2013.052213.00046.

[23] Zargar, S. T.; Joshi, J.; Tipper, D. A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. *IEEE Communications Surveys & Tutorials*, volume 15, no. 4, 2013: pp. 2046–2069, ISSN 1553-877X, doi:10.1109/SURV.2013.031413.00127.

[24] Mirkovic, J.; Reiher, P. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, volume 34, no. 2, 2004: pp. 39–53.

[25] Bhuyan, M. H.; Kashyap, H. J.; Bhattacharyya, D. K.; et al. Detecting Distributed Denial of Service Attacks: Methods, Tools and Future Directions. *The Computer Journal*, 2013: p. bxt031.

[26] Tartakovsky, A. G.; Rozovskii, B. L.; Blažek, R.; et al. A Novel Approach to Detection of Intrusions in Computer Networks via Adaptive Sequential and Batch-Sequential Change-Point Detection Methods. *IEEE Transactions on Signal Processing*, volume 54, no. 9, 2006: pp. 3372–3382.

[27] Willinger, W.; Paxson, V. Where mathematics meets the Internet. *Notices of the AMS*, volume 45, no. 8, 1998: pp. 961–970.

[28] Wang, H.; Zhang, D.; Shin, K. Detecting SYN flooding attacks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, 2002, ISSN 0743-166X, pp. 1530–1539, doi:10.1109/INFCOM.2002.1019404.

[29] Siris, V. A.; Papagalou, F. Application of anomaly detection algorithms for detecting SYN flooding attacks. *Computer communications*, volume 29, no. 9, 2006: pp. 1433–1442.

[30] Brutlag, J. D. Aberrant Behavior Detection in Time Series for Network Monitoring. In *LISA*, 2000, pp. 139–146.

[31] Blazek, R. B.; Kim, H.; Rozovskii, B.; et al. A novel approach to detection of denial-of-service attacks via adaptive sequential and batch-sequential change-point detection methods. In *Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance Workshop, West Point, NY, USA*, 2001.

[32] Chen, Y.; Hwang, K.; Ku, W.-S. Distributed change-point detection of DDoS attacks over multiple network domains. In *Proc. of Int'nl Symp. on Collaborative Technologies and Systems*, 2006, pp. 543–550.

[33] Mirkovic, J.; Prier, G.; Reiher, P. Attacking DDoS at the source. In *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*, IEEE, 2002, pp. 312–321.

[34] Ozcelik, I.; Fu, Y.; Brooks, R. DoS Detection is Easier Now. In *Research and Educational Experiment Workshop (GREE), 2013 Second GENI*, March 2013, pp. 50–55, doi:10.1109/GREE.2013.18.

[35] Ye, N.; Borror, C.; Zhang, Y. EWMA techniques for computer intrusion detection through anomalous changes in event intensity. *Quality and Reliability Engineering International*, volume 18, no. 6, 2002: pp. 443–451.

[36] Tartakovsky, A. G.; Rozovskii, B. L.; Blažek, R. B.; et al. Detection of intrusions in information systems by sequential change-point methods. *Statistical Methodology*, volume 3, no. 3, 2006: pp. 252–293.

[37] Callegari, C.; Giordano, S.; Pagano, M. Neural Network based Anomaly Detection. In *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD) (CAMAD 2014)*, Athens, Greece, Dec. 2014.

[38] Callegari, C.; Gazzarrini, L.; Giordano, S.; et al. A novel PCA-based network anomaly detection. In *Communications (ICC), 2011 IEEE International Conference on*, IEEE, 2011, pp. 1–5.

[39] Sagiroglu, S.; Sinanc, D. Big data: A review. In *2013 International Conference on Collaboration Technologies and Systems (CTS)*, May 2013, pp. 42–47, doi:10.1109/CTS.2013.6567202.

[40] Zadnik, M.; Krobot, P.; Wrona, J. Experience with big data frameworks for IP flow collector. In *Technical report*, 2016.

[41] Lee, Y.; Lee, Y. Toward scalable internet traffic measurement and analysis with hadoop. *ACM SIGCOMM Computer Communication Review*, 2013.

[42] Lee, Y.; Kang, W.; Son, H. An internet traffic analysis method with mapreduce. In *IEEE/IFIP Network Operations and Management Symposium Workshops (NOMS)*, 2010.

[43] Ibrahim, L. T.; Hassan, R.; Ahmad, K.; et al. A study on improvement of internet traffic measurement and analysis using Hadoop system. In *International Conference on Electrical Engineering and Informatics (ICEEI)*, IEEE, 2015.

[44] Bumgardner, V. K.; Marek, V. W. Scalable Hybrid Stream and Hadoop Network Analysis System. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering (ICPE)*, 2014.

[45] Karimi, A. M.; Niyaz, Q.; Sun, W.; et al. Distributed network traffic feature extraction for a real-time IDS. In *IEEE International Conference on Electro Information Technology (EIT)*, 2016.

[46] Zhang, J.; Zhang, Y.; Liu, P.; et al. A Spark-Based DDoS Attack Detection Model in Cloud Services. In *Proceedings of 12th International Conference on Information Security Practice and Experience (ISPEC)*, 2016.

[47] Čermák, M.; Tovarňák, D.; Laštovička, M.; et al. A performance benchmark for NetFlow data analysis on distributed stream processing systems. In *IEEE Network Operations and Management Symposium (NOMS)*, 2016.

[48] Garofalo, M. *Big Data Analytics for Flow-based Anomaly Detection in High-Speed Networks*. Dissertation thesis, 2017.

[49] Fontugne, R.; Mazel, J.; Fukuda, K. Hashdoop: A MapReduce framework for network anomaly detection. In *IEEE Conference on Computer Communications Workshops (INFOCOM)*, 2014.

[50] Farrell, S.; Tschofenig, H. Pervasive Monitoring Is an Attack. http://www.ietf.org/rfc/rfc7258.txt, May 2014.

[51] Santanna, J. J.; van Rijswijk-Deij, R.; Hofstede, R.; et al. Booters — An analysis of DDoS-as-a-service attacks. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, ISSN 1573-0077, pp. 243–251, doi: 10.1109/INM.2015.7140298.

[52] Mockapetris, P. Domain names - implementation and specification. RFC 1035 (Standard), Nov. 1987.

Publications of the Author

Reviewed Relevant Publications of the Author

[P.1] T. Cejka, M. Zadnik *Preserving relations in parallel flow data processing* 11th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2017, Zurich, Switzerland, 2017.

[P.2] T. Jánský, T. Čejka, V. Bartoš *Hunting SIP Authentication Attacks Efficiently* 11th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2017, Zurich, Switzerland, 2017.

[P.3] M. Švepeš, T. Cejka *Making flow data analysis parallel* 11th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2017, Zurich, Switzerland, 2017.

[P.4] T. Cejka, V. Bartoš, M. Svepes, Z. Rosa, H. Kubatova *NEMEA: A Framework for Network Traffic Analysis* 12th International Conference on Network and Service Management (CNSM 2016), Montreal, Canada 2016.

[P.5] Z. Rosa, T. Cejka, M. Zadnik, V. Puš *Building a Feedback Loop to Capture Evidence of Network Incidents* 12th International Conference on Network and Service Management (CNSM 2016), Montreal, Canada 2016.

[P.6] T. Cejka, M. Svepes *Analysis of Vertical Scans Discovered by Naive Detection* Management and Security in the Age of Hyperconnectivity: 10th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2016, Munich, Germany, 2016.

[P.7] T. Cejka, V. Bartos, L. Truxa, and H. Kubatova *Using Application-Aware Flow Monitoring for SIP Fraud Detection* Intelligent Mechanisms for Network Configuration and Security: 9th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2015, S. Latré, M. Charalambides, J. François, C. Schmitt, and B. Stiller, Eds. Ghent, Belgium: Springer International Publishing 2015.

[P.8] T. Cejka, Z. Rosa, H. Kubatova *Stream-wise Detection of Surreptitious Traffic over DNS* 19th IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (IEEE CAMAD 2014), Athens, Greece 2014.

The paper has been cited in:

- Nuojua, V., David, G., Hämäläinen, T.: *DNS tunneling detection techniques – Classification, and theoretical comparison in case of a real APT campaign.* In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 10531 LNCS, pp. 280-291, 2017. DOI: 10.1007/978-3-319-67380-6_26

[P.9] P. Benacek, R. B. Blazek, T. Cejka, H. Kubatova *Change-Point Detection Method on 100 Gb/s Ethernet Interface* Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, New York, USA 2014.

The paper has been cited in:

- Nakamura, K., Hayashi, A., Matsutani, H.: *An FPGA-based low-latency network processing for spark streaming.* In Proceedings of the IEEE International Conference on Big Data, Big Data 2016, art. no. 7840876, pp. 2410-2415, 2016. DOI: 10.1109/BigData.2016.7840876

[P.10] T. Cejka, L. Kekely, P. Benacek, R. B. Blazek, H. Kubatova *FPGA Accelerated Change-Point Detection Method for 100Gb/s Networks* MEMICS proceedings, 9th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2014), Telč, Czech Republic 2014.

[P.11] P. Benáček, V. Puš, H. Kubátová, T. Čejka *P4-To-VHDL: Automatic generation of high-speed input and output network blocks*, Microprocessors and Microsystems, Vol. 56, Elsevier, pp. 22–33, 2018.

The paper has been cited in:

- Garcia, Luis Fernando Uria, et al.: *Introdução à Linguagem P4-Teoria e Prática.* Minicursos do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (Minicursos_SBRC) 36 (2018).

Other relevant publications

[P.12] T. Čejka and R. Krejčí *Configuration of open vSwitch using OF-CONFIG* IEEE/IFIP Network Operations and Management Symposium, Istanbul, Turkey,

2016.

The paper has been cited in:

- P. Bellavista, et al. *Multi-domain SDN controller federation in hybrid FiWi-MANET networks*, EURASIP Journal on Wireless Communications and Networking 2018.1 (2018).
- P. Jha, and B. Tech. *End-to-End Quality-of-Service in Software Defined Networking*, 2017.

[P.13] T. Čejka and A. Robledo *Detecting Spoofed Time in NTP Traffic* The 4th Prague Embedded Systems Workshop, Roztoky u Prahy, Czech Republic, 2016.

[P.14] M. Svepes and T. Cejka *Overload-resistant Network Traffic Analysis* The 4th Prague Embedded Systems Workshop, Roztoky u Prahy, Czech Republic 2016.

[P.15] T. Čejka, R. Bodó, and H. Kubátová *Nemea: Searching for Botnet Footprints* The 3th Prague Embedded Systems Workshop, Roztoky u Prahy, Czech Republic 2015.

[P.16] V. Bartoš, M. Žádník, T. Čejka *Nemea: Framework for stream-wise analysis of network traffic* CESNET Technical Report 2013.

The paper has been cited in:

- Y.K. Lai, T. Wellem, H.P. You. *Hardware-assisted estimation of entropy norm for high-speed network traffic*, Electronics Letters, 2014.

[P.17] T. Čejka *Hardware Accelerated Anomaly Detection in Computer Networks*, A Doctoral Study Report submitted to the Faculty of Information Technology, Czech Technical University in Prague Prague, December 2014.

[P.18] T. Cejka, R. B. Blazek *Real-time Detection of Anomalies in High-speed Computer Networks* The 1st Embedded Systems Workshop - ESW 2013, Temešvár, Czech Republic 2013.

[P.19] T. Cejka *Systém pro detekci anomálií v počítačových sítích* Počítačové architektury a diagnostika - PAD 2013, Teplá, Česká republika 2013.

[P.20] T. Cejka *Hardwarově akcelerovaná detekce anomálií v počítačových sítích s využitím FPGA* Počítačové architektury a diagnostika - PAD 2012, Milovy, Česká republika 2012.

Projects

[P.21] deputy task leader, member of team *GN4 JRA2T6* GÉANT

[P.22] member of team *CESNET e-Infrastructure* Operational Program Research and Development for Innovations (OP VaVpI in Czech) project, CESNET, a.l.e

[P.23] member of team *Large Infrastructure* Operational Program Research and Development for Innovations (OP VaVpI in Czech) project, CESNET, a.l.e

[P.24] member of team *DMON100* TAČR alfa project, CESNET, a.l.e

[P.25] member of team *SGS* Czech Technical University in Prague

Selected Relevant Supervised Theses

[P.26] L. Stejskalová *System for grouping suspicious network addresses*, (in Czech), Diploma thesis, FIT, CTU in Prague, 2018.

[P.27] M. Tomáš *Extension of reputation database with information from Passive DNS*, (in Czech), Bachelor thesis, FIT, CTU in Prague, 2018.

[P.28] J. Jančička *Automatic classification of network entities*, (in Czech), Bachelor thesis, FIT, CTU in Prague, 2017. *Honored as an excellent thesis*

[P.29] A. Plánský: Automatic Analysis of Security Incident Alerts, (in Czech) Diploma thesis, FIT, CTU in Prague, 2017.

[P.30] O. Hollmann *Detection of network attacks of Denial of Service type*, (in Czech), Bachelor thesis, FIT, CTU in Prague, 2017.

[P.31] Jiří Havránek *Network flows exporter supporting application information*, (in Czech), Bachelor thesis, FIT, CTU in Prague, 2017. *Honored as an excellent thesis*

[P.32] Z. Rosa *Automatic data capture for detected events*, (in Czech) Diploma Thesis, FIT, CTU in Prague, 2017.

[P.33] M. Švepeš *Extension of the NEMEA system for deployment in a distributed environment*, (in Czech), Diploma Thesis, FIT, CTU in Prague, 2017. *Honored as an excellent thesis*

[P.34] A. Robledo *Network Time Protocol attacks detection*, Diploma thesis, FIT, CTU in Prague, 2016.

[P.35] M. Kalina *Traffic monitoring in 100Gb/s network infrastructures*, (in Czech), Diploma thesis, czech, FIT, CTU in Prague, 2016.

[P.36] Z. Kasner *Flow-Based Classification of Devices in Computer Networks*, Bachelor thesis, FIT, CTU in Prague, 2016.

[P.37] T. Jánský *Application for analysis of VoIP/SIP network flows*, (in Czech), Bachelor thesis, FIT, CTU in Prague, 2016. *Honored as an excellent thesis*

[P.38] L. Truxa *Detection of VoIP Exchanges Fraud*, (in Czech), Diploma thesis, czech, FIT, CTU in Prague, 2015.

[P.39] N. Jíša *Detection of Network Attacks to Voice over IP Infrastructure*, (in Czech), Diploma thesis, czech, FIT, CTU in Prague, 2015.

[P.40] Z. Rosa *Detection of tunneling in computer networks*, (in Czech), Bachelor Thesis, FIT, CTU in Prague, 2014, *Honored as an excellent thesis*

[P.41] M. Švepeš *Configuration and monitoring system for distributed system NEMEA*, (in Czech), Bachelor Thesis, FIT, CTU in Prague, 2014, *Honored as an excellent thesis*

[P.42] J. Neužil *P2P Botnet Detection in Computer Networks*, Bachelor Thesis, FIT, CTU in Prague, 2014.

Appendix A

Included Papers

A.1 NEMEA: A Framework for Network Traffic Analysis

Ing. Tomáš Čejka (20 %), Ing. Václav Bartoš (20 %), Ing. Marek Švepeš (20 %),
Ing. Zdeněk Rosa (20 %), doc. Ing. Hana Kubátová, CSc. (20 %)
In *Proceeding of the 12th International Conference on Network and Service Management (CNSM 2016).*
Montreal, Canada, 2016
DOI: 10.1109/CNSM.2016.7818417

The following included paper ([P.4]) presents a unique stream-wise NEMEA system for network flow analysis and anomaly detection. It was originally developed for research purposes. It has been deployed in the CESNET2 network infrastructure for analysis of the traffic at the perimeter of the network since 2013. NEMEA (besides honeypots) is one of the most significant sources of detected security events in CESNET2. NEMEA was firstly introduced in the CESNET technical report [P.16] written in 2013

The NEMEA framework is the core of the NEMEA system. It was used to implement several algorithms for analysis and detection. Also, the system was used for experiments with splitting a stream of flow data into separate substreams based on witnesses as it was described in the previous chapters.

The paper was written in cooperation with Zdeněk Rosa, Marek Švepeš, Václav Bartoš, and Hana Kubátová. The work of Václav Bartoš was focused on the efficient data format UniRec, and his primary interest is an incident handling and processing security events detected by NEMEA and other IDS. The dissertation thesis of Václav Bartoš (not finished yet) focuses on computation of reputation for the "evil" network entities. Marek Švepeš worked on the module for configuration and management of the NEMEA system. Additionally, Marek Švepeš and Zdeněk Rosa helped with implementation of the NEMEA modules and proofreading and improvement of the text of the paper. My authorship on NEMEA and this paper was summarized at the end of Section 3.1.

NEMEA: A Framework for Network Traffic Analysis

Tomas Cejka*, Vaclav Bartos*, Marek Svepes*, Zdenek Rosa* and Hana Kubatova†

*CESNET, a.l.e.
Zikova 4, 160 00 Prague 6, Czech Republic
{cejkat,bartos,svepes,rosa}@cesnet.cz

†CTU in Prague, FIT
Thakurova 9, 160 00 Prague 6, Czech Republic
kubatova@fit.cvut.cz

***Abstract*—Since network attacks become more sophisticated, it is difficult to discover them using traditional analysis tools. For some kinds of attacks, it is necessary to analyze Application Layer (L7) information in order to detect them. However, there is a lack of existing tools capable of L7 processing and manipulation. Therefore, we propose a flow-based modular Network Measurements Analysis (NEMEA) system to overcome the situation. NEMEA is designed with respect to a stream-wise concept, *i. e.* data are analyzed continuously in memory with minimal data storage. NEMEA is developed as an open-source project and is publicly available for world-wide community. It is designed for both experimental and operational use. It is able to process offline traffic traces as well as live network flows. The system is very flexible and can be easily extended by new modules. The modules are developed within a NEMEA framework that is a key component of the project. NEMEA thus represents a unified platform for research and development of new traffic analysis methods. It covers several important topics not limited to analysis and detection. Originally, NEMEA has been developed for the purposes of Czech National Research and Education Network operator. Therefore, it is focused on handling high speed network traffic with links working at 100 Gbps.**

I. Introduction

Monitoring computer networks belongs to important tasks of every network operator. Monitoring systems can provide valuable information about status and utilization of a network infrastructure. Network security must be kept in mind due to the importance of computer networks, safety of users and their data. Due to the huge volume of data that is transferred via modern network infrastructures, monitoring systems usually aggregate information about traffic into smaller *flow records*. These traditionally consist of network addresses, ports, timestamps and volume information. This data can be used for accounting, statistical analysis to improve situational awareness or for anomaly detection. An overview of concepts of the flow-based network monitoring can be found in [1], [2].

Since network attacks are becoming more sophisticated and hidden, it is sometimes very difficult to recognize them in normal benign traffic. There are several methods for detection of malicious traffic based on traditional flow records. The records can even be used for detection of some attacks on Application Layer (L7) (*e. g.* SSH bruteforce [3]). However, for some kinds of malicious traffic, traditional flow records are not sufficient and information from L7 headers is needed for reliable detection. L7 information is however not well supported in current flow analysis tools.

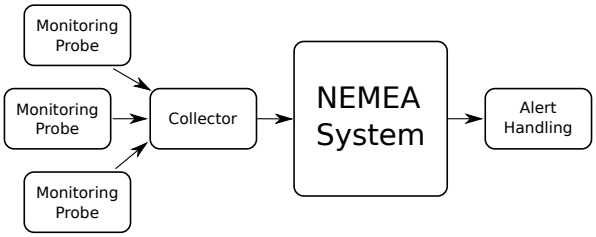


Fig. 1. Monitoring system infrastructure is based on exporting flow records (by Monitoring probes) that are passed for storage (Collector) and analysis (NEMEA). Detection modules of NEMEA produce alerts in a unified format suitable for subsequent processing and storage.

To overcome a lack of existing tools, we developed a new platform for online stream-wise traffic analysis and anomaly detection, capable of L7 data processing – Network Measurements Analysis (NEMEA). Fig. 1 shows a typical flow monitoring infrastructure with traffic analysis done by NEMEA. The network is monitored by monitoring probes (flow exporters) which send flow data to a central collector in form of flow records. The probes contain plugins to parse selected information from L7 protocols and extend the flow records by this information. The IPFIX protocol is used to transfer such data. In our deployment at CESNET2 network (Czech NREN), we use FlowMon [4] probes based on special devices with hardware acceleration in order to process backbone traffic (up to 100 Gbps) without sampling. However, any exporter capable of parsing L7 protocols can be used. Collectors usually store received flow records and in our case the collector (IPFIXcol [5]) also resends them for analysis to the NEMEA system. The probes with the collector are the source of data for analysis. The results of the analysis are statistics of traffic and alerts produced by various detection mechanisms.

NEMEA was firstly introduced in our technical report [6] in 2013 but since that time, it is still being improved. This paper presents main features of NEMEA. The system is composed of independent interconnected modules, it is extensible and it can run in distributed environment. Every module has its own task that can be, for example, anomaly detection, filtering or statistics computation. Each module is built using a set of libraries that create a common framework. Everything is developed as an open-source project and is available at github.com[1].

[1]https://github.com/CESNET/NEMEA

978-3-901882-85-2

This paper is organized as follows. Section II compares the NEMEA system with existing related projects. Section III describes architecture and shows main features of the NEMEA system. Section IV lists real use-cases that we target and Section V presents results that we have achieved using the NEMEA system. Finally, Section VI concludes the paper.

II. Related Work

This section describes related existing systems for traffic monitoring and analysis. The analysis and anomaly detection is often done using Intrusion Detection Systems (IDS) or Intrusion Prevention Systems (IPS). Such systems analyze the network based on packets or network flows. The most popular packet based systems are Bro [7] and Snort [8]. They process every packet and use pattern matching, possibly enhanced by scripting, to search for suspicious traffic and perform actions when a predefined rule matches. Since these systems operate on packets, they can see more details than flow based systems. In contrary, flow based systems usually analyze data on a higher level of abstraction and are thus able to detect different kinds of events. Also, since they process less detailed data, they usually have better performance. These two approaches may complement each other in many scenarios.

Nfdump [9] is a set of tools for storage and processing of flow records. It receives data from the network and stores them into files corresponding to time windows, typically 5 minutes long. Nfsen [10] is a graphical frontend for nfdump that visualizes the stored data in form of graphs and reports. It also allows manual analysis of the data and it can be configured to automatically generate alerts based on simple rules. More advanced data analysis can be done using plugins. The disadvantage of this approach is that the data must be stored to a disk and then read by all the plugins. This is often a performance bottleneck in large networks. Moreover, nfdump does not support flow records extended by L7 information.

The Network Situational Awareness (NetSA) group at CERT created Analysis Pipeline [11] based on SiLK [12]. SiLK is a set of tools for manipulation of records in a flexible data format containing information from flow records. Analysis Pipeline processes the data according to a configuration file that describes a sequence of operations in three stages — filtering, evaluation or statistics computation, and alerting. There is a set of predefined options for each stage. By combining them into a pipeline a complex query can be assembled. Building a complex analysis task from simpler modules is an approach very similar to that of NEMEA. The latest version of Analysis Pipeline also adds a support for L7 data. However, functionality of the building blocks of the pipeline and their possible interconnections are very limited in comparison to NEMEA.

III. NEMEA System

A. Overview

The NEMEA system is designed as a heterogeneous modular system. Modules are independent processes interconnected by unidirectional interfaces for communication. The interfaces transfer data in the form of streams of messages — flow records, results of some analysis etc. A simple example of an instance of the NEMEA system is shown on Fig. 2. Each

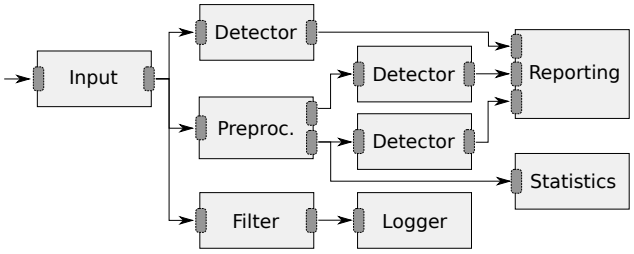


Fig. 2. Example of several NEMEA modules and their interconnection.

module is basically a program which performs a specific task, such as flow data preprocessing, filtration, anomaly detection, or logging and reporting results.

Each instance of the NEMEA system can be put together out of different sets of modules, interconnected in various ways. Different deployments of the NEMEA system may use completely different configurations of modules and thus perform different tasks. Therefore, NEMEA is very flexible. In a typical configuration, modules are interconnected into a tree or directed acyclic graph with a single module acting as main input of the whole system. This module gathers or creates flow records and sends them to other modules which process them. On the other end of the system, there are usually modules for logging the results to log files, a database or for sending alerts via email. In our deployment, we use a plugin for IPFIXcol collector as an input of the system — the collector receives and parses IPFIX data from our monitoring probes, the plugin transforms them to a format used by NEMEA and forwards them to NEMEA modules. This was shown in Fig. 1. However, for smaller deployments or for testing it is possible to use NEMEA module *flow meter* as an input. It can read packets from a network interface or a PCAP file, generate flow records and send them directly to the rest of the NEMEA system, so no external exporters and collector are needed.

NEMEA is not only easily reconfigurable, it can also be easily extended by new functionality. The NEMEA framework is designed to allow quick and easy implementation of new modules. Although NEMEA can be used in production environment for analysis of live traffic, it is also designed to serve as a common platform for researchers in the area of network security and monitoring. It allows for fast prototyping of new traffic analysis methods, testing them on both offline and online data and comparing them with existing methods. Therefore, although we already provide a number of modules for common tasks and several detectors of malicious traffic, we hope a community will develop much more in the future.

The framework used by the NEMEA system is developed in C language and brings support for implementation of NEMEA modules in C, C++ or Python (with possibility to add more languages later). The system should run on any UNIX-like operating system.

B. Architecture

Figure 3 shows a simplified architecture of the NEMEA system. There is a set of running modules (interconnected by interfaces, which is not shown here). The set of modules can be controlled and monitored by a tool called Supervisor. All

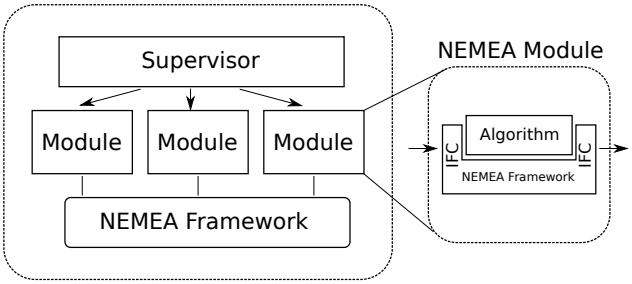


Fig. 3. High level look at the NEMEA system.

the modules are programs built upon NEMEA framework, *i. e.* the modules use functionality that is implemented in shared libraries of the NEMEA framework.

The right side of Fig. 3 points out usual inner organization of a module. Every module implements an algorithm or method for traffic analysis or detection and it uses features of the NEMEA framework for communication with other modules and for access to information contained in data records.

The most important part of the framework is *TRAP library* (Traffic Analysis Platform), which implements the communication interfaces and other basic functions needed by every NEMEA module. Another library – *UniRec* – implements a data format for binary representation of flow records and other information. UniRec is the default data format used for communication of modules. There is a Python wrapper around these two libraries that supplies API for modules written in Python. The last part of the NEMEA framework is the *common* library that provides a number of functions and data structures commonly used in network data analysis algorithms, such as various hash functions, hash tables, Bloom filter, prefix tree, or B+ tree.

The NEMEA system follows stream-wise concept of data processing. That means it is designed to process data continuously at real-time or near real-time without a need of a data storage. Writing and reading flow data to/from hard-drive is often a performance bottleneck in other systems for flow data analysis. In NEMEA, all the data remains in operation memory (unless some module intentionally stores them to disk or database), allowing real-time processing of data even from large networks using a single server. However, if needed, NEMEA can be distributed to more servers since each module can run separately and communicate with others via network.

C. Communication interfaces

TRAP Communication Interfaces (IFC) allow modules to communicate with each other. The IFCs are unidirectional, each one represents either input or output of the module. Each module can use multiple input and output IFCs. An output IFC of a module (*sender*) can be connected to one or more input IFCs of other modules (*receivers*). All receivers connected to the same sender get the same data.

The data are sent over an IFC as a potentially infinite stream of short messages (max 64 kB each). A message may be a flow record, result of a detection algorithm (alert), statistics computed from flow records in some time window or anything else. Formats of the messages are described later.

The IFCs are in fact an abstraction of several different underlying methods of interprocess communication. The two main ones are based on UNIX domain sockets and TCP sockets. The former one is used for communication between modules on the same system, the latter one for communication over a network. There are also two special types of IFCs – *file* and *blackhole*. The *file* type allows to store a stream of data into a file (when used as output IFC) and replay it later (when used as input IFC). The *blackhole* type can be used as output IFC only. It simply discards all messages sent to it.

The key point is that a data processing algorithm inside a module is completely abstracted from the type and parameters of module's IFCs. It just calls functions to receive data from or send to a specified interface. The types of interfaces to use, together with parameters specifying where they should be connected (socket name, IP address and port, file name), are passed as command-line parameters when the module is started. The parameters are processed by the TRAP library so the developer of the module do not need to care about it. The library also handles most of the IFC related errors that may occur, for example if a connection to the other module breaks (*e. g.* because the other module is restarted) the library automatically tries to reconnect.

Generally, NEMEA is designed in such a way that the developer of a module can focus on data processing algorithm only, leaving all the integration work up to the TRAP library. It significantly shorten the time needed to develop, test and deploy a new traffic analysis method. It also opens the system to less experienced programmers, *e. g.* researchers focusing rather on traffic analysis methods than on programming.

D. Data formats

Data are exchanged over IFCs in one of three supported formats – unstructured data, JSON and NEMEA's own binary format UniRec. The first two are rarely used, flow data as well as most of other data records are transferred in UniRec format.

UniRec is an efficient binary format for storage and transfer of simple data records similar to plain C struct. In addition to the C struct it supports fields with variable length. UniRec itself is a generic data structure, a particular format is given by *template*, *i. e.* a set of fields in a record.

In comparison to other formats for transfer of simple records like IPFIX, JSON or its binary equivalents BSON or MessagePack, UniRec has two key differences. First, it is designed to allow very fast access to fields of a record. While other formats have to be parsed before fields can be accessed, UniRec fields can be read directly, with access speed almost equal to plain C struct[2]. But contrary to C struct, an UniRec template can be defined at run-time. Second, all records sent over a single IFC have the same template. This is not a problem in most use cases (when it is, JSON can be used instead) and it significantly simplifies data processing.

The data format compatibility is checked automatically by IFCs. Each output IFC specifies the data format it is able to send, while each input IFC specifies a set of required fields. These formats can be specified at run time, which

[2]There is one additional memory access to a small table which easily fits into L1 cache.

adds to flexibility of the system. When two IFCs are about to be connected, their formats are checked. If the output IFC contains all the fields required by the input IFC, the connection is established and data transfer begins. Otherwise the connection is refused.

Therefore, if a module for processing HTTP traffic needs flow data with URL field and a user tries to connect it to an IFC providing only basic flows with no L7 data, the connection fails.

This mechanism is useful not only for error checking. For example, it allows to create a generic logging module which automatically recognize what data it receives. Using this information, the module knows how to interpret messages and how to log them.

We want to explicitly point out that NEMEA natively supports flow records extended by L7 information. UniRec is a generic format whose records can contain any fields, therefore, flow records can be naturally extended by any new information elements, even at run-time. In general, the system allows to add or remove modules at run-time without an interruption of other modules. This property is important for production deployment of complex configurations of the NEMEA system.

E. Central Configuration and Monitoring

Modules of the NEMEA system can be run manually as any other set of UNIX processes. However, the system can also be controlled and monitored centrally by a tool called *nemea-supervisor*, which is usually a better option, especially for instances composed of a large number of modules. Nemea-supervisor can run as a system daemon or in an interactive mode. It also supports configuration via the standard NETCONF protocol [13].

Nemea-supervisor takes care of modules according to an XML configuration file. The file defines modules, their parameters and a grouping to profiles – groups of modules that can be started or stopped together, *e. g.* for an experiment. The configuration can be changed at run-time using provided thin client or any NETCONF client.

As a monitoring tool, nemea-supervisor periodically retrieves state information of every module and, with respect to the configuration, performs actions needed to keep the modules running or stopped. Besides the module's status, nemea-supervisor reads some statistics about the resource consumption of modules (CPU and memory usage) and also about their interfaces. Every module's IFC automatically updates counters of received or sent messages and the counters are read by nemea-supervisor via a special *service IFC* opened in every module. All statistics about modules can be periodically exported into the Munin system [14].

F. System Performance

Overall performance of the NEMEA system depends mainly on a set of deployed modules and their resource requirements. However, there is a limitation given by maximal throughput of IFCs. Every output IFC uses a buffer to optimize utilization of data transfer in order to increase throughput.

Maximal number of messages per second (MPS) that can be sent depends on a message size. Fig. 4 shows a relation

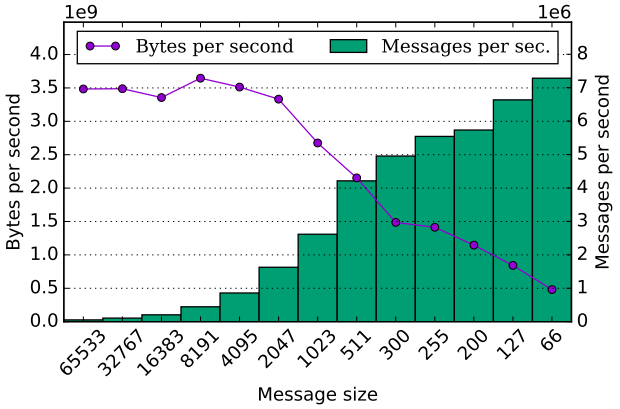


Fig. 4. Impact of message size on throughput of the socket based IFC.

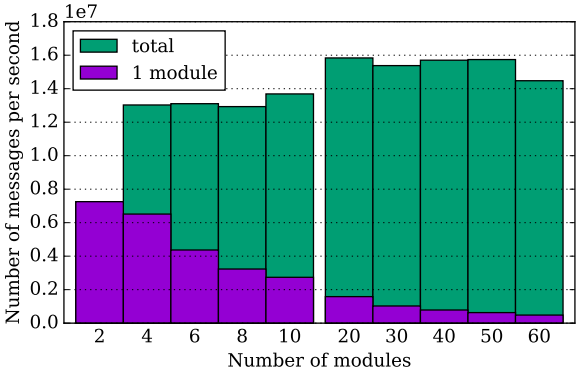


Fig. 5. Impact of number of running modules to IFC throughput. The number of modules (x-axis) contains all running modules (senders and receivers).

between MPS and message size measured between a pair of test modules (one sender and one receiver) on the same machine, whereas, messages were generated in memory. The green bars in the figure show the number of MPS that can be sent/received via IFC. The highest number of MPS (over 7 million per second) was measured for 66 B message size, which is the size of a basic flow record that we use. Due to significant overhead for this size of message, the transmission speed in bytes per second (purple line in the figure) is the lowest (over 480 MB/s). L7 information consumes additional memory besides the basic flow information. For the biggest measured messages (65 KB messages), the number of MPS is low (53 thousand MPS) but the transmission speed is about 3.5 GB/s.

Philosophy of NEMEA is based on running several modules at once. However, the more running modules, the less resources are available for each module. Figure 5 shows throughput of IFCs when running multiple pairs of modules. All the modules were sending messages of a fixed size (66 B was chosen). The purple bars represent the average number of MPS that was received by a single receiver, the green bars represent the sum of MPS from all receivers. For two modules (one pair), the total number equals to the MPS of one receiver.

The real-world performance of a complete system heavily depends on a particular set of instantiated modules and

complexity of algorithms they implement. However, as shown in Sec. V, it is not a problem to process flow data from a medium-sized NREN by tens of modules using only a single server.

IV. Targeted Real Use-Cases

NEMEA already contains a number of modules for different purposes, mostly for detection of various kinds of malicious traffic. This section describes main topics that are covered by the modules. Most of them are currently in use in CESNET2 network.

A. Detection up to L4

Network scanning belongs to the most common activities that occur in the Internet. It is usually harmless, however, it can be used by attackers to gather information. There are NEMEA modules for scans detection (port scans were analyzed in [15]).

Denial of Service (DoS) and Distributed DoS (DDoS) belongs to the most powerful types of attack. According to various reports (*e. g.* [16]), number of DoS/DDoS attacks increases and volume of traffic that attackers can generate gets higher. Detection and mitigation of these attacks is a challenging task, since it is very difficult to reliably and timely recognize malicious and benign traffic. The NEMEA system tries to get into the topic by providing several detection modules based on different analysis methods. For example, we implemented a detection method based on MULTOPS tree [17]. There is also a specialized NEMEA module for detection of amplification attacks.

B. Detection using L7

Session Initiation Protocol (SIP) can be used as a signalling protocol for Voice over IP (VoIP) that is a modern successor of telephone services. In specific circumstances, a misconfigured SIP Private Branch Exchange (PBX) allows to call to a PSTN phone number just by using the correct prefix, added to the number. Attacks trying to guess the prefix and make calls to premium-rate numbers are quite common and if successful, they can lead to a significant financial loss for the owner of the vulnerable PBX. A NEMEA module detecting this kind of attack using flow records extended by selected SIP headers was proposed in [18].

DNS protocol is usually not restricted by security policies and firewall settings because it belongs to indispensable services. It can sometimes be used to circumvent a network connection restrictions or escape from a secured network by encapsulating data into DNS messages and thus creating a communication tunnel. There is a NEMEA module for detection of such tunnels based on statistical analysis of DNS messages, further described in [19].

Heartbleed is a critical bug discovered in the OpenSSL library in 2014. It gives an opportunity for attackers to remotely read random chunks of memory from a server that uses the vulnerable version of OpenSSL. A NEMEA module for Heartbleed exploit detection was developed in a few days after the bug was published. It analyzes information from SSL protocol headers which are extracted by a special plugin for flow exporter. Within the first two months of operation, the module discovered more than a thousand vulnerable hosts in our network that were attacked (or probed) from the outside. The detection mechanism is described in detail in [20].

NEMEA can be used to detect devices infected with malware as well. The paper [21] presents that, having samples of malware, it is possible to retrieve valuable information for the detection of infected devices connected to the network. A generic *filter* module can be used to find a communication with suspicious servers. In such a scenario, extended flow records with domain names and URLs of HTTP are the input of the filter. The filtering condition can contain the suspicious addresses, domain names and URLs. Every matching connection is immediately reported, since its source is probably infected by the malware.

C. Alert Handling

The detection modules generate records about detected security incidents (alerts). To abstract the detectors from tasks related to alert handling, such as logging or reporting, NEMEA provides a means to handle the alerts in a unified way. This is represented by a set of modules called *reporters* that convert alerts from detectors into a unified format and then they can: log alerts into files, store alerts into database, send e-mails containing information from alerts, send alerts into the Warden[3] system.

Since these modules are implemented as Python scripts, it is easy to extend them to support any output data format or to export alerts to any other system.

D. Offline testing

NEMEA does not have to be used just in a production deployment processing live data. It can work offline and process stored data as well. There are modules for reading flow data from files in nfdump format, fastbit database used by IPFIXcol or CSV files. It is also possible to store and replay a stream of data generated by any NEMEA module directly in UniRec format.

This allows to repeatedly send the same data into a set of processing modules, which is useful for testing modules (and therefore processing methods) as well as for research, *e. g.* for comparison of different methods or parameter settings using the same input data.

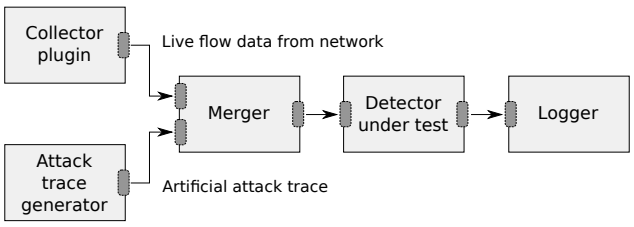


Fig. 6. Testing a detector by injecting artificial attack traces into a live stream of flow data

An interesting use-case is also a possibility to mix two or more streams of data using the *merger* module. For example a

[3]Warden is a system for sharing information about detected incidents within a community of security teams, developed at CESNET, https://warden.cesnet.cz/

stream of flow records got from normal traffic can be merged with a stream generated by an attack simulator or with a trace of an attack stored previously. Thus, attack traces can be mixed into normal traffic to test abilities of detection modules, as shown in Fig. 6.

V. Real world deployment and results

An instance of NEMEA system is deployed at CESNET since early 2014. It receives and analyzes flow data from probes deployed on the perimeter of CESNET2 network, that is ten lines with capacity ranging from 10 Gbps to 100 Gbps connecting CESNET2 to other backbone networks. The data are not sampled in any way. On average, the probes generate 120,000 flow records per second during peak hours.

At the time of writing (June 2016), our instance of the NEMEA system consists of 28 modules. All the modules run on a single server with 6 CPU cores and 12 GB of memory and are able to process all incoming data without any loss. In fact, no more than 40 % of server's resources (both CPU and memory) are utilized in normal circumstances.

The system is able to detect port scans, DDoS attacks (simple SYN floods as well as DNS and NTP amplification attacks), dictionary attacks on SSH, and watches for connections to several malicious IP addresses and URLs. Also, various statistics about the traffic are computed and stored. At last, one of the modules performs on-the-fly anonymization of the flow data and re-sends them to a server with less restricted access. This is used for development and testing of new modules by network security students and other academic people who can not get access to real data due to privacy concerns.

On average, every day the NEMEA system detects and reports the following events in the CESNET2 network:

- 110,000 horizontal port scans[4] (76 every minute).
- 12,000 dictionary or bruteforce attempts to log in to SSH (8.3 per minute).
- 2,400 DDoS attacks (mostly DNS and NTP amplification).

The number of DDoS attacks may seem very high. This is partly caused by the fact that a single attack may be reported by two modules and that long attacks may be reported several times due to properties of methods used for detection. Therefore, it is rather a number of alerts generated by detectors than real number of attacks. Nevertheless, even if the alerts are aggregated, the number of attacks is still in the order of hundreds. This is because most of the DDoS attacks today use thousands of DNS or NTP servers across the world for reflection and amplification of the attack traffic. If traffic to just one of the servers passes through the CESNET2 network, the attack can be detected. In fact, we often observe use of a single server for several separate attacks at the same time.

We also have several modules for analysis of L7 data present in our extended flow records, for example a detector of attacks on VoIP servers or detector of DNS tunnels. They are still considered experimental and are not running on the main NEMEA instance, but for example, the VoIP detector reports around 1,100 attacks per day if it gets data from all the probes.

We also often use L7 data for ad-hoc filtering by URL in HTTP or hostname in DNS requests based on current needs of security management. That means ad-hoc addition of modules for filtering and logging the traffic of interest. For example we recently acquired a sample of a new malware and analyzed it in cooperation with our forensic laboratory [22]. This resulted in a list of IP addresses and URLs used to control a botnet. A set of filter modules looking for those IP addresses and URLs in L7-extended flow data was immediately added to our NEMEA system. It revealed 11 infected devices in our network during 2 weeks. The detection was done continuously and alerts were sent at near real-time.

VI. Conclusion

Flow-based measurement and analysis became a standard approach for network monitoring. Traditional flow records provide visibility to transport layer (L4) only. However, for detection of some kinds of problems, application layer (L7) information is necessary. Although there exist exporters capable of parsing L7 information, it is hard to process them with current tools. We therefore created a new platform for analysis of L7-extended flow data – NEMEA.

The NEMEA serves for both experimental and operational use. One of its most important features is the capability of L7 processing. At the same time, it is a flexible modular system that allows researchers and network operators to extend its functionality by implementing new NEMEA modules. It can also be viewed as a common platform for development of traffic analysis algorithms, which allows to easily test them on both offline traces and live data, compare to other algorithms and eventually deploy them operationally. The modularity of NEMEA also makes the system scalable to handle even large numbers of flow records. In case one machine does not have enough resources, it is possible to distribute the computation, *i. e.* start NEMEA modules on multiple hosts.

In this paper, we have shown a complex deployment combining NEMEA system with a set of high performance exporters and an open-source collector IPFIXcol. However, since NEMEA contains its own minimalistic exporter, it can also be used independently on smaller networks.

Deployment at CESNET2 network have proven that NEMEA can be successfully used to monitor large networks and detect various kinds of malicious traffic. Thousands of incidents have been detected thanks to NEMEA.

Acknowledgment

This work was partially supported by the "CESNET E-Infrastructure" (LM2015042), CTU grant No. SGS16/124/OHK3/1T/18 both funded by the Ministry of Education, Youth and Sports of the Czech Republic and by the Technology Agency of the Czech Republic under No. TA04010062 *Technology for processing and analysis of network data in big data concept.*

[4]To avoid false alerts, we set thresholds quite high. Port scan is reported when more than 200 connections on different destinations are attempted within 5 minutes.

REFERENCES

[1] R. Hofstede, P. Celeda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014.

[2] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of IP flow-based intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 3, pp. 343–356, 2010.

[3] L. Hellemons, L. Hendriks, R. Hofstede, A. Sperotto, R. Sadre, and A. Pras, "SSHCure: A flow-based SSH intrusion detection system," in *Dependable Networks and Services*. Springer, 2012, pp. 86–97.

[4] Flowmon Networks, "The Most Powerful NetFlow Probes in the World." [Online]. Available: https://www.flowmon.com/en/products/flowmon/probe

[5] CESNET, "IPFIXcol." [Online]. Available: https://github.com/CESNET/ipfixcol/

[6] V. Bartoš, M. Žádník, and T. Čejka, "Nemea: Framework for stream-wise analysis of network traffic," CESNET, a.l.e., Tech. Rep., 2013. [Online]. Available: http://www.cesnet.cz/wp-content/uploads/2014/02/trapnemea.pdf

[7] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23, pp. 2435–2463, 1999.

[8] M. Roesch and *et. al.*, "Snort: Lightweight intrusion detection for networks," in *LISA*, vol. 99, 1999, pp. 229–238.

[9] P. Haag, "NFDUMP – Netflow processing tools." [Online]. Available: http://nfdump.sourceforge.net/

[10] ——, "NfSen – Netflow Sensor." [Online]. Available: http://nfsen.sourceforge.net/

[11] CERT/NetSA at Carnegie Mellon University, "Analysis Pipeline." [Online]. Available: {http://tools.netsa.cert.org/analysis-pipeline}

[12] ——, "SiLK (System for Internet-Level Knowledge)." [Online]. Available: http://tools.netsa.cert.org/silk

[13] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," RFC 6241, Internet Engineering Task Force, Jun. 2011.

[14] "Munin." [Online]. Available: http://munin-monitoring.org

[15] T. Cejka and M. Svepes, *Analysis of Vertical Scans Discovered by Naive Detection*. Munich, Germany: Springer International Publishing, 2016, pp. 165–169. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-39814-3_19

[16] Kaspersly Lab, "Kaspersky DDoS Intelligence Report Q3 2015," November 2015. [Online]. Available: https://securelist.com/analysis/quarterly-malware-reports/72560/kaspersky-ddos-intelligence-report-q3-2015/

[17] T. M. Gil, "MULTOPS: A data structure for denial-of-service attack detection," Ph.D. dissertation, Vrije Universiteit, 2000.

[18] T. Cejka, V. Bartos, L. Truxa, and H. Kubatova, "Using Application-Aware Flow Monitoring for SIP Fraud Detection," in *Intelligent Mechanisms for Network Configuration and Security (LNCS 9122)*. Springer International Publishing, 2015, pp. 87–99.

[19] T. Cejka, Z. Rosa, and H. Kubatova, "Stream-wise detection of surreptitious traffic over DNS," in *19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE, 2014, pp. 300–304.

[20] V. Bartoš, "Heartbleed Detection at CESNET using Extended Flow Monitoring," in *Proceedings of 8th International Scientific Conference on Security and Protection of Information*, 2015.

[21] T. Cejka, R. Bodó, and H. Kubatova, "Nemea: Searching for Botnet Footprints," in *The 3rd Prague Embedded Systems Workshop (PESW2015)*, 2015.

[22] CESNET, "FLAB – Forensic laboratory." [Online]. Available: https://flab.cesnet.cz/

A.2 Nemea: Searching for Botnet Footprints

Ing. Tomáš Čejka (33 %), Ing. Radoslav Bodó (33 %),
doc. Ing. Hana Kubátová, CSc. (33 %)
In proceedings of the *3th Prague Embedded Systems Workshop, PESW2015*
Roztoky u Prahy, Czech Republic, 2015
ISBN: 978-80-01-05776-6, pp. 11-16

Botnets are dangerous groups of infected devices (bots) that can perform various malicious activities. All bots use some communication channel for their synchronization. There are botnets with a central control server, usually referred as command and control servers (C&C). The address of a C&C server must be known to bots, so that it is either contained in the source codes of the malware or it is generated by some deterministic algorithm at runtime.

The following included paper ([P.15]) describes usage of addresses of C&C servers gained from samples of malware by a forensic laboratory. Application-aware approach of detection allows for identification of such a communication with the C&C servers. The detection is based on filtering extended flow records that contain a domain name or IP address that is contained in some malware sample.

The paper was written in cooperation with Radoslav Bodó and Hana Kubátová. The idea of stream-wise filtering the L7 extended flow records as well as most of writing the paper are the work of the author of this dissertation thesis. Radoslav Bodó was cooperating as a member of forensic laboratory and provided us with features from real malware samples.

The algorithm and its possibility of parallel processing were described in Section 3.5.1.

Nemea: Searching for Botnet Footprints

Tomas Cejka[1], Radoslav Bodó[1], Hana Kubatova[2]

[1] CESNET, a.l.e.
Zikova 4, 160 00 Prague 6, Czech Republic
cejkat@cesnet.cz, bodik@cesnet.cz
[2] CTU in Prague, FIT
Thakurova 9, 160 00 Prague 6, Czech Republic
kubatova@fit.cvut.cz

Abstract. Malicious network traffic originated by malware means a serious threat. Current malware is designed to hide itself from the eyes of victim users as well as network administrators. It is very difficult or impossible to discover such traffic using traditional ways of flow-based monitoring. This paper describes a network traffic analysis of a backbone network as an attempt to discover infected devices. Cooperation with forensic laboratory and analysis of samples of malware allow to gain information that can lead to find unwanted traffic. Special tailored Nemea framework with high speed monitoring pipeline was used to discover infected devices on the network.

1 Introduction

This paper is focused mainly on network traffic monitoring and analysis on a backbone networks. Backbone networks are very specific because of the volume of traffic that flows through the network infrastructure. The growth of network speed and bandwidth makes monitoring and analysis complicated. There are several issues related to this area [10].

Standard monitoring mechanisms (well described e.g. in [8]) exploits various mechanism of traffic aggregation or sampling that allow storage of information about traffic. Many security threats are still detectable even though the used aggregation looses precise information about transferred packets. Information about communication over network is represented as so called flow record, i.e. tuple of network addresses, ports and protocol that uniquely identify "one connection" between two hosts.

Basic flow records contain volume statistics such as number of packets and number of bytes. Analysis of the flow records is used to detect Denial of Service attacks (DoS), scanning or sweeping of addresses or ports, brute-force attacks etc. However, there are malicious applications that generate traffic very similar to normal benign traffic. The malicious applications (usually called malware), forged by attackers, usually exploit standard protocols and services. Additionally, this traffic is composed of a few packets in long time periods. Basic volume information about malware traffic is almost useless for detection of infected devices.

As we will explain in following sections, malware sometimes generate specific patterns that appear in packets. The patterns are e.g. URL in the HTTP protocol or DNS lookups containing suspicious domain names. Detection of such traffic could be based on application layer analysis of transferred packets, sometimes called as deep packet inspection [14]. The examples of detection of suspicious traffic using application layer information can be found e.g. in [2, 3].

Application layer analysis is very difficult or impossible for large networks without any hardware acceleration. The main obstacle is the volume of data that must be processed. On current high-speed networks, it is needed to analyze traffic at speed over 10 Gbps. Such traffic is very difficult to be transferred into software for processing by standard tools. For this task, a special tailored hardware card [7] can be used for hardware accelerated preprocessing in monitoring probes. Using the monitoring probes it is possible to process headers of application layers export information for additional analysis.

The rest of this paper describes real use case of cooperation of monitoring system, network traffic analysis tools and forensic laboratory that is able to analyze malware and provide valuable information to discover infected devices on the network.

2 Malware Sample Analysis

Currently, most of devices that are connected into internet can be infected by a malicious software. When malware gets into a device, it usually works as a downloader and starts to download additional applications such as backdoor [13].

Malicious codes spread through the internet via many means, but so far most used channel is a simple e-mail. The most typical scenario includes the obfuscated executable sent as an e-mail attachment, latter upon delivery the user itself is forced (perhaps by an urgently composed text) to open an attachment and thus executing the malicious program on one of his devices. Our analyzed samples were acquired as an suspicious incoming e-mail and were securely sent to forensic laboratory [4] for extended analysis.

This section briefly describes a process of malware analysis performed in the forensic laboratory.

At first, malware samples are passed to several simple tools for extracting basic data and metadata from the samples. This phase includes `file`, `strings`, antivirus software, PE Explorer, ... Finaly samples are disassembled to gain basic overview of its functionality. The data, metadata and assembler code of the executable file supplies the first insight and sometimes it can provide e.g. domain names or IP addresses or other interesting keywords in readable text format.

The second step is execution of the malicious code in specially prepared virtualized environment. Isolated virtual machine contains tools that monitors system calls (e.g. using procmon [12], `strace`, ...) and state of the system. It is needed to be able to take snapshots of disk and memory in a different phases

of malware execution (starting with the state before execution). Execution of malware can bring valuable information about the infiltration into the operating system as well as the activity of malicious process. The activity of the process include disk operations (writing data) or network communication (downloading other binaries, contacting command and control servers).

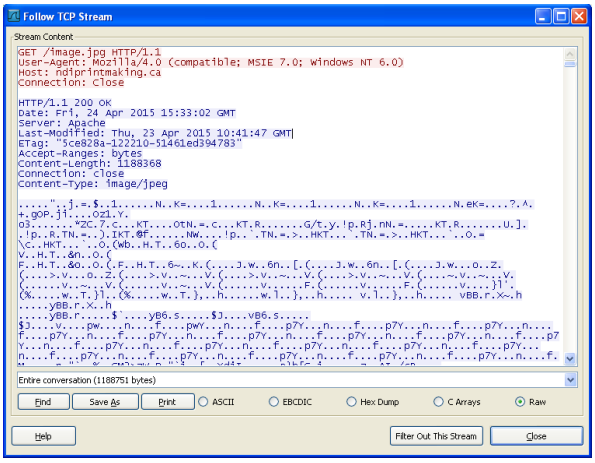


Fig. 1: Network communication of infected computer with attacker's server.

Traces of process monitor and network traffic dump (e.g. using Wireshark [11]) can be visualized by ProcDOT [15]. This application can generate call graph that represents behaviour of the analyzed sample. Network dump is useful source of addresses and domain names that should be traced and monitored in real network traffic. Communication with gained addresses can expose infected hosts on the network.

Forensic analysis discovered several characteristics of malware behavior. The interesting characteristics for our purposes are related to the network traffic generated by malware. Fig. 1 shows the example of captured traffic from testing virtual environment.

Fig. 2 shows visualisation of malware activity. It summarizes information of process monitoring and network monitoring. The figure contains part of the graph that was created by ProcDOT [15].

The laboratory analysis found the list of suspicious domain names and IP addresses that is shown in the following list and those pieces of information can be used to locate other infected nodes in monitored network:

- globalgateone.com
- globalgatetwo.com
- lac-fessenheim.org
- ogarape.com

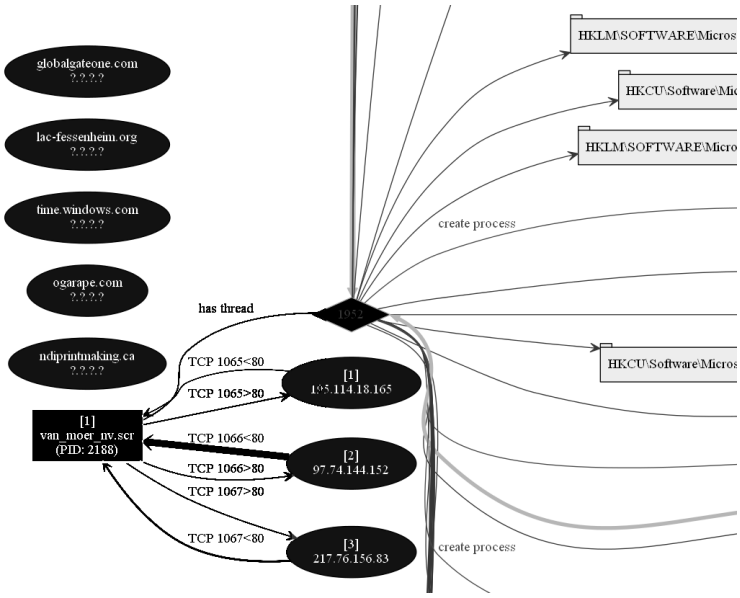


Fig. 2: Part of the diagram of malware activity gained using ProcDOT in forensic analysis.

– ndiprintmaking.ca

– 195.114.18.x
– 97.74.144.x
– 217.76.156.x

3 Real Network Monitoring – Infrastructure

Our monitoring infrastructure consists of monitoring probes [9] with the COMBO cards [7] for hardware acceleration, the open-source IPFIXcol collector [5] for collecting information about network traffic and finally the Nemea system [6] for stream-wise data analysis. The monitoring infrastructure is shown in Fig. 3.

The Nemea system a modular system for network traffic analysis and anomaly detection. It is composed by independent modules developed using the Nemea framework [1]. The framework tries to make development of Nemea modules easier and faster due to implementing common tasks in form of shared libraries.

Nemea contains several modules that work as a source of data for the Nemea system. An important example of such modules is a plug-in for IPFIXcol. This plug-in can export all flow records that collector receives from monitoring probe and pass them in format that Nemea modules understand. Using IPFIXcol plug-in, it is possible to get flow records extended by application layer information at

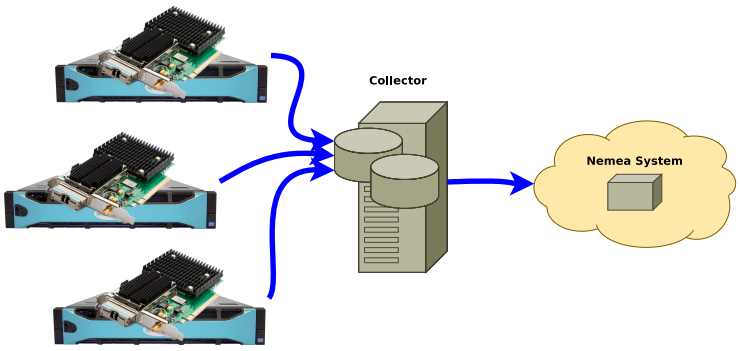


Fig. 3: Monitoring infrastructure consisting of monitoring probes, collector and Nemea system.

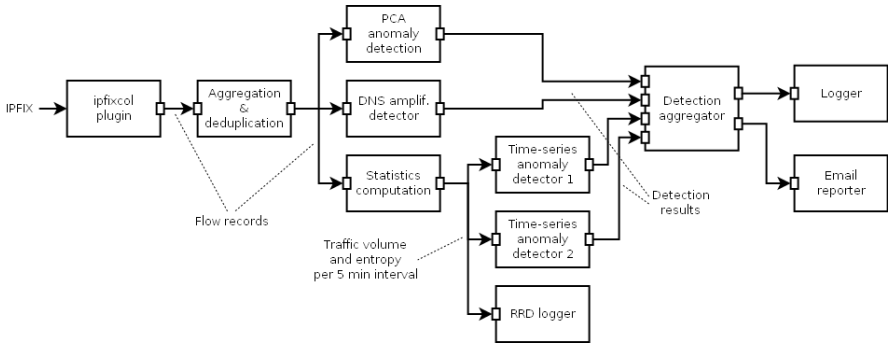


Fig. 4: Example of interconnected modules of the Nemea system.

near real-time into detection modules in the Nemea system. The example of the Nemea system configuration is shown in Fig. 4.

For the purposes of tracking of malicious traffic, a special filter module was used. The filter is able to take user-defined condition and apply it on incoming messages with information about flow. Messages that do not satisfy the condition are dropped, the rest of messages is passed for next processed.

The module supports many operators that can be used in condition. We have used mainly matching regular expressions with values of fields extracted from application layer. The following listing shows the example of condition that filter messages of DNS traffic:

```
:DNS_NAME=~".*globalgate.*\.com$"||DNS_NAME=~".*ogarape\.com$"||
DNS_NAME=~".*lac-fessenheim\.org$"||
DNS_NAME=~".*ndiprintmaking\.ca$"
```

It is possible to filter traffic of discovered addresses that are probably command and control servers as follows (addresses are anonymized):

```
:SRC_IP==195.114.18.x||DST_IP==195.114.18.x
```

This condition was applied on HTTP data and we can discovered other potentially infected computers on our network infrastructure that communicate with the given server.

4 Results

After few days of monitoring, we have discovered 11 devices connected into the monitored network infrastructure that tried to resolve domain names used by malware samples or to communicate with discovered servers via the HTTP protocol.

Except domain names listed in previous section, we found that infected devices tried to query another suspicious domain name `dqwdwqwqdqwddqw.cn`. In addition, we discovered URLs that appeared in malicious HTTP queries that probably work as keep-alive ping: `/so/` and `/hr/`.

Most frequent source and destination IP address that appeared in our observation is from France. The second most frequent addresses belong to subnets of the Czech Republic, to the academic networks.

5 Conclusion

Discovering of infected computers on computer network infrastructure is a non-trivial but critically important task for every network operators. Attackers and authors of malicious code try hard to hide not only the running malware in the victim's system but also the traffic that malware produces. It is very difficult maybe impossible to find malicious traffic or traffic originated from infected devices using basic flow records.

This paper described the case study of cooperation of department of tools for monitoring and configuration with the forensic laboratory to discover infected devices on the monitored network infrastructure. As a result, 11 infected computers were discovered communicating with suspicious servers. In addition, it could be observed how infected devices communicated with command and control servers to let them know that they are still alive (infected).

The whole monitoring was allowed by monitoring infrastructure based on hardware accelerated monitoring probes that can operate at link speed (10 Gbps). Exported information about network traffic was extended by some headers of application protocols such as DNS and HTTP. The resulting flow records were processed by modular Nemea system that is designed for stream-wise traffic analysis at near real-time.

After 10 days of monitoring, the most active command and control server that communicated from domain name `globalgateone.com` stopped responding,

however, infected computers still continued to send their TCP SYN packets. The infected addresses discovered by our work were reported to members of CSIRT team.

Acknowledgments This work was partially supported by the "CESNET Large Infrastructure" (LM2010005), CTU grant No. SGS15/122/OHK3/1T/18 funded by the Ministry of Education, Youth and Sports of the Czech Republic.

References

1. Bartos, V., Zadnik, M., Cejka, T.: Nemea: Framework for stream-wise analysis of network traffic. Tech. rep., CESNET (2013)
2. Cejka, T., Bartos, V., Truxa, L., Kubatova, H.: Using Application-Aware Flow Monitoring for SIP Fraud Detection. In: Proc. of 9th International Conference on Autonomous Infrastructure, Management and Security (AIMS15) (2015)
3. Cejka, T., Rosa, Z., Kubatova, H.: Stream-wise detection of surreptitious traffic over dns. In: Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2014 IEEE 19th International Workshop on. pp. 300–304. IEEE (2014)
4. CESNET, a. l. e.: FLAB forensic laboratory, `https://flab.cesnet.cz/`
5. CESNET, a. l. e.: IPFIXcol, `https://github.com/CESNET/ipfixcol/`
6. CESNET, a. l. e.: Nemea, `https://www.liberouter.org/nemea/`
7. CESNET, a. l. e.: Programmable Hardware, `http://www.liberouter.org/technologies/cards/`
8. Hofstede, R., Celeda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A., Pras, A.: Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. IEEE Communications Surveys Tutorials 16(4), 2037–2064 (2014)
9. INVEA-TECH a.s.: FlowMon Probe – High-performance NetFlow Probe up to 10 Gbps, `http://www.invea-tech.com/products-and-services/flowmon/flowmon-probes`
10. Kekely, L., Pus, V., Korenek, J.: Software defined monitoring of application protocols. In: INFOCOM, 2014 Proceedings IEEE. pp. 1725–1733. IEEE (2014)
11. Orebaugh, A., Ramirez, G., Beale, J.: Wireshark & Ethereal network protocol analyzer toolkit. Syngress (2006)
12. Russinovich, M.: Process Monitor v3.1, `https://technet.microsoft.com/en-us/library/bb896645.aspx`
13. Skoudis, E.: Malware: Fighting malicious code. Prentice Hall Professional (2004)
14. Velan, P., Celeda, P.: Next generation application-aware flow monitoring. In: Monitoring and Securing Virtualized Networks and Services, LNCS, vol. 8508, pp. 173–178. Springer (2014)
15. Wojner, C.: ProcDOT — Visual Malware Analysis, `http://www.procdot.com/`

A.3 Stream-wise Detection of Surreptitious Traffic over DNS

Ing. Tomáš Čejka (33 %), Ing. Zdeněk Rosa (33 %),
doc. Ing. Hana Kubátová, CSc. (33 %)
In *IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*
Athens, Greece, 2014
DOI: 10.1109/CAMAD.2014.7033254

This section includes the paper ([P.8]) that describes a particular example of detection of so-called covert channels. Some tools can be used to encapsulate user data into packets of standard operational protocols such as DNS or ICMP. The aim of such encapsulation can be, e.g., avoiding security policies in networks where communication is forbidden for users. For instance, there are hotspots, which allow users to connect to the Internet only after paying some fees. Another example can be blocking outgoing traffic to protect secret data of an organization.

DNS can be misused to create a communication tunnel and to avoid the security policies. The following paper describes a principle of DNS tunnels and presents a stream-wise detection algorithm. The algorithm was implemented as an application-aware module for the NEMEA system.

The paper was written in cooperation with Zdeněk Rosa, who worked on the development and evaluation of the NEMEA module in his bachelor thesis [P.40] (supervised by the author of this dissertation thesis), and Hana Kubátová. The idea of the flow-based detection of the DNS tunnels using extended flow records was the contribution of the author of this dissertation thesis.

The algorithm and its possibility of parallel processing were described in Section 3.5.2.

Stream-wise Detection of Surreptitious Traffic over DNS

Tomas Cejka
CESNET, a.l.e.
Zikova 4, 160 00 Prague 6, Czech Republic
cejkat@cesnet.cz

Zdenek Rosa
and Hana Kubatova
CTU in Prague, FIT
Thakurova 9, 160 00 Prague 6, Czech Republic
{rosazden,kubatova}@fit.cvut.cz

***Abstract*—The Domain Name System (DNS) belongs to crucial services in a computer network. Because of its importance, DNS is usually allowed in security policies. That opens a way to break policies and to transfer data from/to restricted area due to misusage of a DNS infrastructure. This paper is focused on a detection of communication tunnels and other anomalies in a DNS traffic. The proposed detection module is designed to process huge volume of data and to detect anomalies at near real-time. It is based on combination of statistical analysis of several observed features including application layer information. Our aim is a stream-wise processing of huge volume of DNS data from backbone networks. To achieve these objectives with minimal resource consumption, the detection module uses efficient extended data structures. The performance evaluation has shown that the detector is able to process approximately 511 thousand DNS flow records per second. In addition, according to experiments, a tunnel that lasts over 30 seconds can be detected in a minute. During the on-line testing on a real traffic from production network, the module signalized on average over 60 confirmed alerts including DNS tunnels per day.**

I. Introduction

Domain Name System (DNS) [1], [2] is prevalently utilized for translation of domain names to network addresses and vice versa. Misconfiguration or limitation of the DNS service (e.g. by successful attacks on DNS infrastructure) can cause collapse of the most of all network services. Therefore, it is useful to concentrate on DNS traffic analysis. Because of importance of this protocol, it is usually allowed in security policies and firewall settings. That opens a possibility for a covert communication channel.

DNS is a client-server protocol based on the well-known principle of request-response where clients send requests and servers answer with appropriate responses. With a special tailored tools, it is possible to encapsulate arbitrary data into DNS messages on one side and decapsulate it on the other side. The resulting DNS messages seem to be legitimate and are normally processed by DNS servers. Local DNS servers are usually permitted to communicate with the Internet. Principle of communication tunnels over DNS protocol is shown in Fig. 1 and well-described in [3], [4]. Exploiting DNS mechanisms, it is possible to misuse the local DNS server to establish a tunnel with the user's tunneling server.

The contribution of this paper is a proposal of the software module for near real-time detection of suspicious DNS communication. The module is able to signalize alerts about communication tunnels in DNS traffic. In addition, used data structures and detection procedure allows for analysis of huge volumes of network traffic of Internet Service Providers (ISP). The module is built on known and published detection and analysis principles adapted for high speed networks. The module is prepared and deployed as a stream-wise component to process huge volumes of data from the Czech National Research and Education Network (NREN).

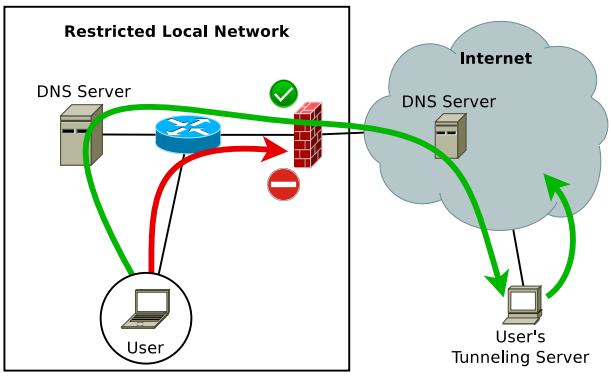


Fig. 1. Communication tunnel via DNS servers.

This paper is organized as follows: Sec. II shows the related work and the difference of the proposed realization. Sec. III describes the implemented and tested detection module. Sec. IV describes chosen data structures for storage of information from DNS traffic and for stream-wise analysis. Sec. V summarizes the parameters of the detection module. Sec. VI presents the monitoring infrastructure and results of the running detection module. Sec. VII concludes the paper.

II. Related Work

The area of surreptitious communications detection has been studied for many years and there are several detection methods that are more or less related to our work. In this section, we try to list the most relevant works of other authors. The main type of surreptitious communications, that we are focused on, is data tunneling over the DNS protocol. Dns2tcp [5], Iodine [6], OzymanDNS [7], and tcp-over-dns [8] are some examples of tools for establishment of the tunnel. There are also free or commercial services (e.g. Wi-Free [9]) that supply a server service for easier tunnel setup. The server plays the role of the second side of the tunnel for users.

Some of the listed tools generate regular patterns in the network traffic. This type of communication can be detected by

well-known detection systems such as Bro [10] or Snort [11]. However, pattern matching is out of our focus.

The detection method that uses matching of regular expressions with domain names is presented by Farnham in [12] as a part of detection system. Farnham's system also uses a basic statistical evaluation of DNS traffic. The implementation was based on a commercial Capture and Parse system. In addition, Farnham uses the MySQL database to store DNS data that are later queried for the source bytes and the destination bytes and their ratio is compared with a limit value.

Bilge et al. proposed a detection procedure in [13] that exploit the CUSUM method for change point detection of distributions of several features. Farnham and Bilge use longest meaningful substring (LMS) analysis based on meaning of domain name parts. The idea of LMS is based on the generation of benign domain names that usually consist of meaningful words (e.g. words from dictionary) or at least human readable easy-to-remember text strings.

The difference from our approach is the view on domain name strings. Bilge et al. look for the similarities and meaningful substrings, whereas we focus on number of differences between each DNS requests. Even though these two approaches seem very similar, the advantage of our approach is that the module needs no dictionaries, knowledge of natural language, nor domain names generation algorithm.

Paxson et al. presented a complex detection procedure for communication tunnels over DNS detection in [4]. Paxson's module is designed for a great volume of a DNS traffic processed with a day period. The performance of the module is based on sophisticated massive filtering of the traffic and removing of a benign traffic without any communication tunnel over DNS. One criterion for removing is the number of look-ups of the same domain name. Repeated domain names represent a traffic without tunneled data.

Our detection module differs mainly in the size of the time window that is processed at once because we concentrate on stream-wise near real-time detection. The volume of data in a short time period is much lower so that the module is able to process all the data. To spare the computer's memory, domain names are stored and analyzed only for suspicious addresses. The processed domain names are stored separately for each network address. However, repeating domain names may appear only once per one network address. This way of analysis allows the module to detect not only tunnels over DNS but also other anomalies such as Denial of Service (DOS) attacks, distributed DOS (DDOS) attacks, or DNS amplification attack.

III. Detection Module

The proposed detection module is a software application written in the C language. It is based on a stream-wise concept of the Nemea system [14] that will be briefly explained in Sec. VI. The detection module is built using the Nemea framework [15] that simplifies the connection with a data source. In our case, the data source is a special module that collects data with DNS information from monitoring probes. The module currently receives basic flow records extended with DNS fields such as domain name, type and class of DNS messages and fragments of RDATA field.

The detection module contains 4 types of detection mechanisms based on the set of features that is used: 1) tunnel in DNS requests, 2) tunnel in DNS responses, 3) other anomalies in DNS requests, and 4) other anomalies in DNS responses. The four detection mechanisms perform analysis of domain names, and fields of DNS messages (CNAME, TXT, MX and NS) that are potential places for encoded data of a tunnel.

The module contains two detection modes: *basic* and *advanced*. Every observed address is analyzed in *basic* mode. When an address is claimed as suspicious by the *basic* mode, the mode is switched to the *advanced* detection for the address. The decision is made according to preset thresholds and limits.

The *basic* and *advanced* modes work with several **observed features**. The *basic* mode is based on statistics of flow information such as the number of DNS messages per host and size of packets of DNS messages. Therefore, the *basic* mode has lower resource and time consumption.

The *advanced* mode is the heart of the detection module and it is used for all suspicious addresses. It is based on extended information about application protocol gained by payload analysis in a monitoring probe. The additional **observed features** are the number of similar looked up domain names per host, the number of different subdomains of the n^{th} level, the length of domain names, the number of digits and unique characters in domain name, the ratio of letters and digits in domain names.

There are many cases of benign DNS traffic with suspicious statistical characteristics. The example of such traffic can be a DNS query for Content Delivery Network (CDN) services, antivirus software communication, or reverse address translation (using *arpa* top level domain). To avoid false alerts signaling well-known benign traffic, the module supports whitelisting of addresses and domain names.

The proposed detection module can signalize various interesting alerts. The alerts can be classified as follows: a) tunneling data over DNS, b) CDN and load balancing, and c) other anomalies. The first group contains alerts of probable tunnels over DNS. This kind of traffic is significantly similar to traffic of tunneling tools that were evaluated in simulation environment of virtual network and that were observed during our experiments in the real network traffic. The second group represents valid traffic of CDN and load-balancing services and probably benign communication of antivirus software systems. The last group represents other detected anomalies such as DOS, DDOS, or DNS amplification attacks, that are characterized by increased number of DNS messages targeted to some host or set of hosts.

IV. Data Structures

Based on the study of the nature of monitored data from a network traffic, we encountered an issue of storage of all data needed for analysis and detection. For a detection of tunnels over DNS and other surreptitious communications, the detection module needs to store flow information. Additionally, the content of DNS messages is stored for the *advanced* mode.

There are several ways to store this kind of data. The naive approach would use arrays or more sophisticated hash maps. Because of hierarchical structure of domain names and

canonical transcription of network addresses, we decided to use tree data structures that decrease the size of required memory without loss of information for duplicate subset of data.

Storage of all possible addresses would theoretically need an array of almost 2^{32} resp. 2^{128} elements. Unfortunately, in practice, this approach would consume much resources. Using hash maps, the memory consumption decreases, however, collisions that can occur in hash maps are unwanted. We need precise information about communicating addresses to identify tunnel origin and unresolved collisions would group data from different addresses together. Joining different data would lead to false alerts. Therefore, we decided to use B+ tree to store data with a network address as a key.

Network addresses are used as keys in B+ trees [16] and are stored as 32 b resp. two 64 b numbers for IPv4 resp. IPv6. The module uses separate B+ tree for each protocol. The structure of B+ tree is shown in Fig. 2 where the numbers represent keys (network addresses). The stored values (*val* in the figure) consist of continuously updated information needed for detection purposes.

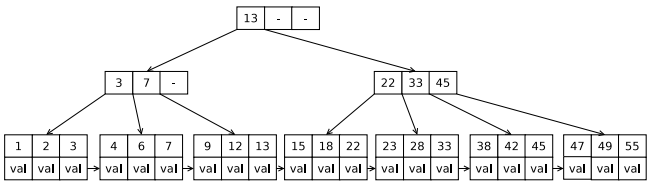


Fig. 2. B+ tree structure

A communication tunnel over DNS needs to use some registered domain name in order to route data from the user to the tunneling server. In addition, the subdomain part of the domain name is a suitable place for transported data. The same domain name is common to all messages of the tunnel. Therefore, it is advantageous to group domain names by suffixes (top level domain – TLD) and examine the number of different prefixes (subdomains) per domain and per address. Domain names can be efficiently stored in prefix tree (well-known data structure based on [17]), where TLD represents root element. Used data structure is shown in Fig. 3. Domain names are processed and stored backwards because of faster matching of common parts.

The implementation of the prefix tree is enriched with special nodes containing metadata (represented by the square with dot in Fig. 3). These nodes are used as a separator of common and different parts of a domain name. In addition, the separator nodes accumulate statistical information about subtrees in the prefix tree. It is useful for fast evaluation of all stored domain names of one network address.

The performance of the implemented prefix tree was compared with a simple array. Both data structures were used for storage of domain names that appear on the real network. The comparison was focused mainly on the memory consumption. The results show that the prefix tree can save about 60 % of memory that would be needed for simple array because due to the data deduplication of common parts of domain names.

Combination of B+ trees and prefix trees gives us an opportunity to analyze traffic of each network address separately. Computational complexity of operations on B+ trees is

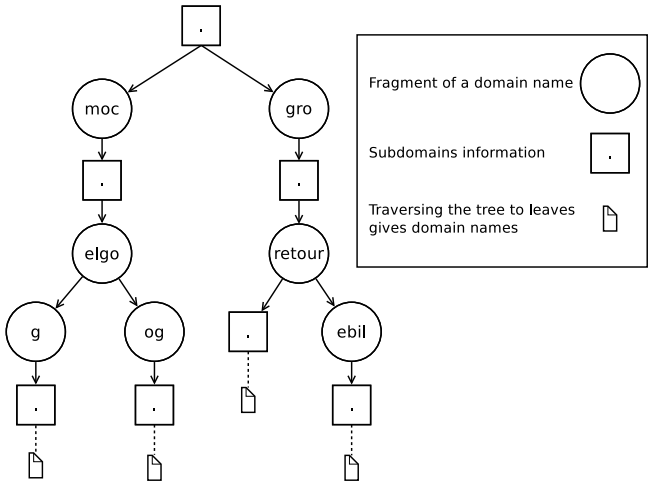


Fig. 3. Domain names stored backwards in the prefix tree: circle nodes contain domain name substring and square nodes represent separators after common substrings. Separators contain metadata about the subtree.

$O(\log_m n)$, where m is the number of node's children. The mostly used operation during the processing of real data is a look-up of network addresses, it is performed about 80.7 % of computation (other operations – insert and remove constitute the remaining 19.3 %).

V. Module Configuration

The module works with various limits and thresholds that influence the detection, especially the transition between the *basic* and the *advanced* modes. The number of suspicious addresses highly influences the resources consumption. The module is fully tuneable via a set of parameters of the module. There are limits and thresholds for all observed features that were introduced in Sec. III. Some thresholds represent mean value or variance that should not be exceeded for the feature. The detection works in cycles with given size of a time window in seconds.

The module has default values of all limits and thresholds that were found as optimal values for the traffic on the CESNET2 network. The values were estimated using observations of average behavior of addresses that use DNS. We used annotated data set with tunnels to find the optimal values. The experiments that tested detection capability of the module will be described in Sec. VI. Some of the thresholds (such as the number of unique characters in domain names) were set and evaluated with respect to values found in [12].

Some of the default thresholds of a legitimate traffic are: the interval of the mean values of the size of DNS requests in bytes – $\langle 70, 100 \rangle$, the interval of the variance of the size of DNS requests in bytes – $\langle 30, 150 \rangle$, the interval of the mean values of the size of DNS responses in bytes – $\langle 70, 600 \rangle$, the interval of the variance of the size of DNS responses in bytes – $\langle 200, 5000 \rangle$, the maximal count of unique letters in request – 24 and response – 30, the maximal number of digits in domain names – 12, and the default size of the time window is 60 s.

The size of the time window is the most important parameter that influences the resources consumption. The size of time windows also controls the duration of the module cycle and a

detection delay. The increase of the time window size causes the increased size of stored data in memory.

VI. Evaluation

We have used two different environments to test the proposed detection module. The environments were used for *offline* and *online* testing. The first and simpler *offline* environment consists of the standalone detection module that is able to load file with data set. The data set contains annotated traffic with benign traffic and communication tunnels over DNS.

This *offline* environment allowed us to examine the memory consumption and computational performance of the module. We have evaluated the memory consumption depending on the size of the time window. The graph with results is shown in Fig. 4. The figure shows the increase of the memory consumption with increasing time window that is caused by higher number of different domain names. The figure also shows that the configuration of the limits and thresholds influences the memory consumption. The blue line with circle points shows the default configuration of parameters. The line with triangles and the line with squares represent different configurations (decrease of the interval of the legitimate domain name sizes).

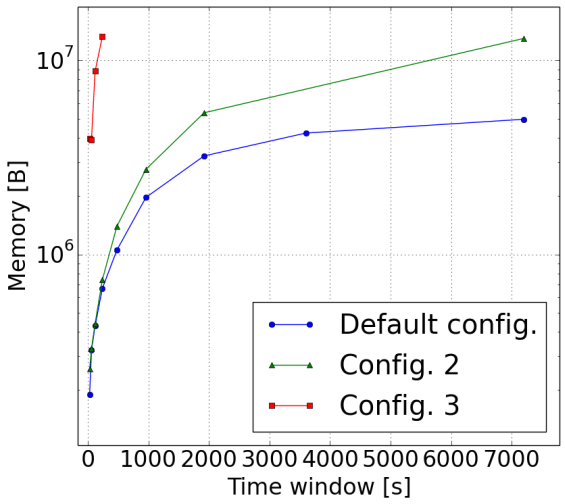


Fig. 4. Memory consumption in logarithmic scale depending on the time window size and different module configuration.

For the performance evaluation, a large data set was replayed and sent into the module. This way, it was possible to simulate high flow records rate. The module successfully handled the load of 90 millions of flow records in 176 seconds that corresponds to approximately 511 kflows/s[1].

The *offline* evaluation of the module configuration was performed on a machine with Intel Core2 Duo E8500 3.16 GHz and 8 GB of an operation memory.

For the *online* tests, we have used the existing monitoring infrastructure of the CESNET2 network (Czech NREN) with the Nemea system. The environment allowed us to examine the usability of the module for near real-time detection in the real traffic of the backbone network.

[1] 1 kflow/s means one thousand flow records per second

In ordinary way, monitoring probes export data in preset regular time intervals (usually 5 minutes) into a monitoring system (e.g. NfSen / nfdump). Contrary, the Nemea system is based on transfers of flow information without fixed time intervals. The simplified scheme of monitoring of a network is shown in Fig. 5. The advantage of used monitoring probes and the Nemea concept is a stream-wise and near real-time approach. Special tailored monitoring probes [18] can export data into the Nemea system continuously and much sooner. Additionally, it is possible to export information about application protocols that is needed by the detection module.

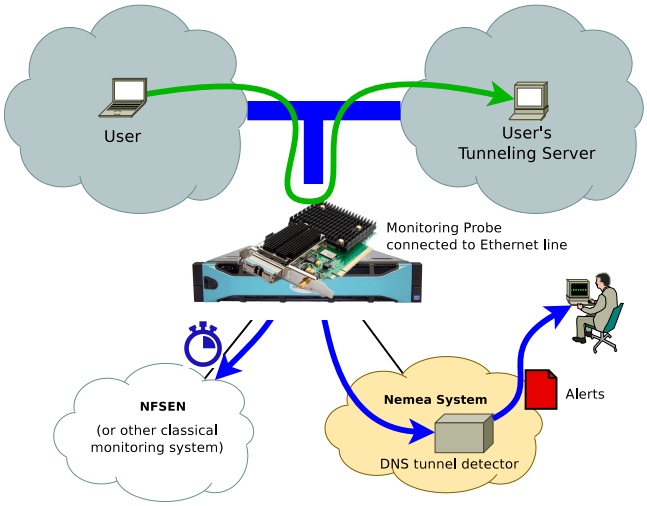


Fig. 5. Monitoring probe exporting application information as soon as possible.

The *online* experiments were based on establishment of communication tunnels using various tunneling tools. We used data from monitoring probes on the perimeter of the CESNET2 network. During preparations of the experiments, we put the server application (one end of tunnel) inside the CESNET2 network and we started the client application placed outside the network. This configuration causes that the tunnel goes through the perimeter of the network and it can be monitored and detected as it is shown in Fig. 5.

During the tests, various tunneling tools over DNS were used for establishment of tunnels. The used tunneling tools for experiments were *iodine* and *dns2tcp*. The time course of one experiment is shown in Fig. 6. After a tunnel establishment, *iperf(1)* was executed in order to generate traffic through the tunnel and to measure the throughput of the tunnel. The experiments show that tunnels can operate at about 600 kb/s (the client application was connected to the Internet via 2 Mb/s link). The detection module found all tunnels that were manually established in real traffic network. The results show that communication tunnels that last over 30 seconds are detected in about 1 minute.

The overall information about traffic of the CESNET2 network and the results of the detection module on non-annotated data can be described as follows. Sum of all address ranges that are connected to the Internet via CESNET2 is approximately one million. The network is mainly academic and therefore it is possible to see seasonal differences in volumes of the network traffic. However, the total number

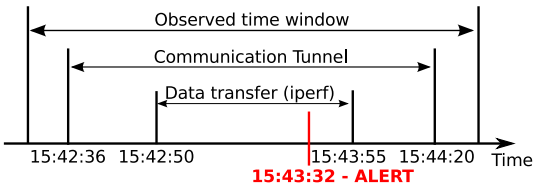


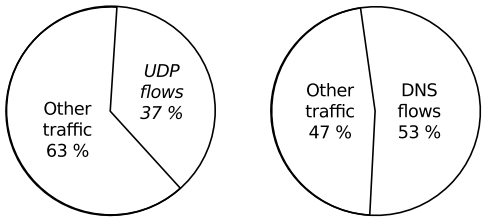
Fig. 6. Time diagram of an experiment with detection of a short tunnel established over DNS. The traffic transferred through the tunnel was generated by *iperf(1)*.

of flows moves in an interval 33–119 kflows/s as it is shown in Tab. I. During the experiments, we observed traffic of approximately 78 kflows/s, one tunnel transferred about 3 MB of data and generated about 0.1 % of all packets.

TABLE I. TRAFFIC ON THE PERIMETER OF CESNET2

[kflows/s]	all	TCP	UDP	DNS
Maximal	119	77	39	17
Minimal	33	16	17	9

Fig. 7a and Fig. 7b show average distribution of the traffic on the network. There is approximately 37 % of UDP flows and over a half of that flows has source or destination port 53. About 0.01 % of about 95 millions of analyzed flow records of the DNS protocol was analyzed in *advanced* mode. 3 % of the suspicious addresses was reported by alerts, 7 % of alerts was classified and manually verified as tunnels over DNS (the rest alerts were recognized as DOS attacks and other anomalies).



(a) The ratio of the UDP and other traffic.

(b) The ratio of the DNS and other UDP traffic.

VII. CONCLUSION

This paper presented the known misusage of the DNS protocol that allows users to establish a communication tunnel. We have shown the monitoring infrastructure for experiments with near real-time detection of such surreptitious traffic.

We have presented a comprehensive detection module that detects communication tunnels over the DNS protocol and other anomalies such as Denial of Service. The module is a combination of well-known traffic and payload analysis methods adapted for huge volume of DNS data processing. The module was designed to be deployed as a stream-wise near real-time detector based on the principle of the Nemea system [14], [15].

Performance of the module was tuned for huge volumes of network traffic. The used suitable data structures for storage of domain names allowed for saving memory space. Due to continuous computation and the extensions of the data structures, the module processes a DNS traffic of 511 kflows/s.

The functionality of the module was tested on annotated and partially annotated data sets containing benign traffic as well as a tunneled communication over DNS. The evaluation of the module showed promising results of the detection. The module was deployed into the running instance of the Nemea system in the existing infrastructure of the CESNET2 network.

ACKNOWLEDGMENT

The authors would like to thank co-workers from CESNET for helpful advices. This work was partially supported by the "CESNET Large Infrastructure" (LM2010005) and CTU grant No. SGS14/107/OHK3/1T/18 funded by the Ministry of Education, Youth and Sports of the Czech Republic.

REFERENCES

[1] P. Mockapetris, "Domain names - concepts and facilities," RFC 1034 (Standard), Internet Engineering Task Force, Nov. 1987. [Online]. Available: http://www.ietf.org/rfc/rfc1034.txt

[2] ——, "Domain names - implementation and specification," RFC 1035 (Standard), Internet Engineering Task Force, Nov. 1987. [Online]. Available: http://www.ietf.org/rfc/rfc1035.txt

[3] M. Dusi, , and *et. al.*, "Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting," *Computer Networks*, vol. 53, no. 1, pp. 81–97, 2009.

[4] V. Paxson and *et. al.*, "Practical comprehensive bounds on surreptitious communication over dns," in *Presented as part of the 22nd USENIX Security Symposium*. USENIX, 2013, pp. 17–32.

[5] O. Dembour, "Dns2tcp." [Online]. Available: http://www.hsc.fr/ressources/outils/dns2tcp/

[6] Ekman, E. and *et. al.*, "Iodine, tunnel IPv4 over DNS," 2011. [Online]. Available: http://code.kryo.se/iodine/

[7] D. Kaminsky, "Ozymandns."

[8] "TCP-over-DNS." [Online]. Available: http://analogbit.com/software/

[9] Wi-Free Ltd., "WI-Free." [Online]. Available: http://wi-free.com/

[10] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23, pp. 2435–2463, 1999.

[11] M. Roesch and *et. al.*, "Snort: Lightweight intrusion detection for networks." in *LISA*, vol. 99, 1999, pp. 229–238.

[12] Farnham, G., "Detecting DNS Tunneling," 2013. [Online]. Available: http://www.sans.org/reading_room/whitepapers /dns/detecting-dns-tunneling_34152

[13] L. Bilge and *et. al.*, "Exposure: Finding malicious domains using passive dns analysis." in *NDSS*, 2011.

[14] V. Barto and *et. al.*, "Nemea: Framework for stream-wise analysis of network traffic," CESNET, a.l.e., Tech. Rep., 2013. [Online]. Available: http://www.cesnet.cz/wp-content/uploads/2014/02/trapnemea.pdf

[15] CESNET, a.l.e., "Nemea." [Online]. Available: https://www.liberouter.org/nemea/

[16] D. Comer, "Ubiquitous b-tree," *ACM Comput. Surv.*, vol. 11, no. 2, pp. 121–137, Jun. 1979. [Online]. Available: http://doi.acm.org/10.1145/356770.356776

[17] E. Fredkin, "Trie memory," *Communications of the ACM*, vol. 3, no. 9, pp. 490–499, 1960.

[18] INVEA-TECH a.s. [Online]. Available: http://www.invea-tech.com/products-and-services/flowmon/flowmon-probes

A.4 Hunting SIP Authentication Attacks Efficiently

Bc. Tomáš Jánský (33 %), Ing. Tomáš Čejka (33 %), Ing. Václav Bartoš (33 %)
In *Security of Networks and Services in an All-Connected World: Proceedings of the 11th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2017*
Zurich, Switzerland, 2017
Publisher: Springer International Publishing
ISBN: 978-3-319-60774-0, pp. 125–130
DOI: 10.1007/978-3-319-60774-0_9

This section includes the published paper ([P.2]) related to the detection of SIP attacks. This paper focuses on detection of password guessing (e.g., dictionary attack) and scanning user accounts on a SIP server.

Scanning and password guessing are frequent and well-known attacks that can target even SIP infrastructure. The success of attackers in the case of password guessing can lead to financial loss. Additionally, the attacker gains access to the SIP account and, generally, it is possible to establish any call that could be established by the legitimate user.

The stream-wise and application-aware detection module, which is presented in the following paper, deals with a trade-off of resource consumption and precision of the detection algorithm. Since this single detection module can detect more attack types, the gathered data are reused. As a result, the module is more efficient and requires lesser resources.

The paper was written in cooperation with Tomáš Jánský and Václav Bartoš. The idea about analysis of the extended flow records with information from SIP headers to detect the mentioned suspicious SIP traffic was the contribution of the author of this dissertation thesis, and it was successfully developed as a bachelor thesis [P.37] (supervised by the author of this dissertation thesis). Václav Bartoš participated on the writing of the paper as a consultant and a proofreader, and he significantly improved the text of the paper.

The algorithm and its possibility of parallel processing were described in Section 3.5.3.

Hunting SIP Authentication Attacks Efficiently

Tomas Jansky[1], Tomas Cejka[2], Vaclav Bartos[2]

[1] CTU in Prague, FIT, Thakurova 9, 160 00 Prague 6, Czech Republic
janskto1@fit.cvut.cz
[2] CESNET, a.l.e., Zikova 4, 160 00 Prague 6, Czech Republic
cejkat@cesnet.cz, bartos@cesnet.cz

Abstract. Extended flow records with application layer (L7) information allow for detection of various types of malicious traffic. Voice over IP (VoIP) is an example of technology that works on L7 and many attacks against it cannot be reliably detected using just basic flow information. Session Initiation Protocol (SIP), which is commonly used for VoIP signalling, is a frequent target of many types of attacks. This paper proposes and evaluates a novel algorithm for near real time detection of username scanning and password guessing attacks on SIP servers. The detection is based on analysis of L7 extended flow records.

1 Introduction

Voice over IP (VoIP) is a technology that replaces classic telephone services and is used to transfer multimedial data such as voice or video over common packet switched networks. One of the core protocols used in VoIP services is Session Initiation Protocol (SIP), which is used for signalling between communicating parties.

There are many types of attacks against SIP infrastructure. The most dangerous attacks often compromise Private Branch Exchange (PBX) devices and cause a significant financial loss to the owner of PBX. According to [3], a total worldwide loss due to VoIP hacking and calling to premium rate services goes to billions of dollars per year.

Even though there are standards that describe security considerations and extensions of the SIP protocol, it is still often observed unencrypted in real network traffic. This allows for security analysis of SIP traffic at a network level using a network passive monitoring. The analysis may detect malicious SIP traffic so that a network operator can inform owners of the target device about a potential threat or take appropriate actions to mitigate malicious traffic.

Network traffic monitoring in large networks is usually done using so called flow records, i.e. aggregated information about communicating hosts that is computed from observed packets. A typical flow record consists of information from packet headers up to the transport protocol. This approach is feasible and it allows for detection of various types of malicious traffic. However, as it was presented in [2], many types of attack at application protocol (L7) cannot be reliably detected using just the basic flow records. This paper shows usage of

application layer flow records [6], in this case flows extended by L7 information about SIP traffic, for detection of brute-force password guessing and scanning for user accounts (called *extensions* in SIP terminology) on PBX. This work is a continuation of [2] and an improvement of detection abilities of the previous detection mechanism.

2 SIP attacks

This work focuses on two types of network attacks by an unauthenticated external attacker against a SIP server – extension scanning (i.e. finding valid usernames) and password guessing.

Both are based on sending large amount of requests (usually `REGISTER`) to the server. When a client sends the request requiring authentication, server challenges it with a response code `401 Unauthorized`. Normally, the client sends valid credentials and server responds with `200 OK`. If the username is not valid, server responds with `404 Not Found` or `401 Unauthorized`, depending on configuration[1]. In case of correct username but wrong password, `401 Unauthorized` is returned.

Therefore, both types of attacks are characterized by a high number of `REGISTER` requests and `401 Unauthorized` (or `404 Not Found`) responses, using either different extensions (extension scanning) or a single extension but different passwords (passwod guessing). Combination of both is also possible. More details about these SIP attacks can be found in [4].

3 Detection algorithm

In line with the L7 flow monitoring approach, our monitoring probes use a plugin which is able to extract necessary SIP information from traffic (*response code*, *To* and *CSeq*). As it is shown in Fig. 1, flow records are sent from probes to a collector in the IPFIX format and afterwards analyzed by the detection algorithm which is implemented as a part of the NEMEA [1] system.

The detection method is designed to work without any prior knowledge of VoIP infrastructure or existing extensions. It is based on an analysis of `401` responses from SIP servers. By aggregating these responses by a PBX IP address, an extension (username) and a client IP address, the detection algorithm can detect non-standard and potentially malicious traffic.

The algorithm shifts between two stages. In the first stage, it receives data and stores it into data structures. For each SIP server (i.e. IP address sending SIP responses), the following data is stored – a list of client IPs, a list of usernames, and a mapping between them that tells which clients tried which usernames and a number of such attempts.

[1] The former is considered insecure since it eases the extension scanning as it immediately discloses existence of the extension on the server.

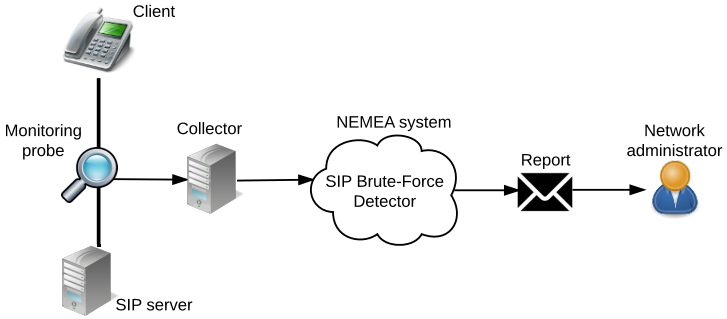


Fig. 1: Monitoring infrastructure.

After a certain time period, the algorithm gets to the second stage where it evaluates the stored data. First a type of (potential) attack is determined. If a single client attempts to register one certain extension, it is classified as a brute-force attack. This attack can be reclassified as a distributed brute-force attack if more clients attempt to register the particular extension on the same server. When a client tries to register more than one extension, the behavior is classified as a scan. When the number of attempts exceeds a threshold, the attack is reported. If `200 OK` response code is detected as part of the communication, the attack is considered successful. If no communication between the server and the client is observed for a certain amount of time, the corresponding structures are released from memory.

The algorithm was implemented as a module for open-source NEMEA system and published at GitHub[2].

4 Evaluation

Since the algorithm is threshold based, it was necessary to estimate some key values based on the behavior on a real network. We temporarily captured SIP traffic from CESNET2 network[3].

After the analysis of the captured data, we discovered that more than **99.9 %** of all successful register attempts use **20** messages or less. We therefore set 20 attempts as a threshold for deciding whether the communication is malicious or not.

We also examined the frequency of malicious requests in individual attacks and discovered that only **0.01 %** have more than **30 minutes** delay between individual requests. Therefore an information about a communication is released from the program memory if no new message is observed for 30 minutes. It also

[2] `https://github.com/CESNET/Nemea-Detectors/`

[3] CESNET2 network is monitored at all its 7 peering links at the 10 and 100 Gbps wire speeds. Average total amount of traffic: 110,000 flows/s, average SIP traffic: 1,500 flows/s.

means that an elapsed attack is reported after this delay since the last observed message.

Finally, we counted unique extensions attempted by every client in 30 minute windows. Most observed clients attempted to register as less than **10** unique extensions on a certain server. This value is surprisingly high, but it is possible that the client is actually a proxy server or there are multiple SIP clients hidden behind NAT. We used 10 distinct extensions as a threshold for extension scanning detection.

First, the detection module was tested on a real network with generated malicious traffic using auditing tool SIPVicious [5]. All generated attacks were successfully distinguished from other SIP communication and reported.

Then, the module was run for one week to capute real attacks in the CESNET2 network. Total number of 7,008 events were reported. Table 1 shows some statistics about reported events. One of the most interesting findings is that **46.3 %** of all `200` and `401` SIP responses to `REGISTER` requests are a malicious traffic and are directly related with one of reported alerts.

Tab. 1: Statistics after one week of flow detection

Brute-force events	6,488 (92.6 %)
Extension scanning events	520 (7.4 %)
Successful brute-force events	7
Strongest brute-force	6,930,911 attempts
Largest scan	9,360 extensions
SIP flows observed	718,627,758
SIP flows analyzed (401 & 200 responses)	40,909,352 (5.7 %)
Number of malicious flows	18,945,291 (46.3 %)

Detection results were stored to a log file during the the week. Thorough examination showed that most attackers perform either brute-force attacks or extension scanning. However, some of the attackers combine these two attacks to one, usually trying a small number of password guesses (between 20 to 100) to a large number of extensions. This behavior indicates that these attackers use some sort of a set of common and frequently used passwords.

To confirm that the detection module is working correctly, we manually analyzed traffic of some of the reported attacks. Most of them are certainly scanning or brute-force attempts. In just a few cases were the traffic did not look like any of the attacks and can be viewed as false positive (we estimate total FP rate to 0.1%), however, it was still an unusual traffic, probably caused by misconfiguration of some devices, which is worth inspecting. To prove practical usefulness of the detection, we chose one of the attacks marked as successful and contacted the administrator of the attacked PBX. He confirmed that, indeed, the account was compromised and informed us that appropriate steps to fortify the PBX will be taken.

5 Conclusion

We designed a method for detection of SIP attacks, namely username scanning and password guessing, based on an analysis of SIP headers in extended flow records. The algorithm works without any prior knowledge of VoIP infrastructure. Its key parameters and thresholds can be adjusted by network administrators in accordance to the characteristics of their network to reach optimal detection results. It is efficient and it is able to process data from an NREN-sized network (several 10 and 100 Gbps links) in real time.

Using the algorithm, we were able to detect thousands of scanning and password guessing against SIP infrastrucutre. The software is also capable of detecting distributed guessing of user's password, however, this type of attack was not observed in our network yet. Some of the attacks, which were identified as successful, were reported to network administrators who subsequently confirmed the attacks. Analysis of detection results showed only a small amount of false positive reports with frequency around 0.1 % of all reported events. Most of the false positives are caused by a few clients that communicate in an unusual way and can be easily filtered using a whitelist.

6 Acknowledgments

This work was supported by *Packet analysis based network diagnostics (DISTANCE)* project No. TH02010186 granted by Technology Agency of the Czech Republic, project Reg. No. CZ.02.1.01/0.0/0.0/16_013/0001797 co-funded by the MEYS of the Czech Republic and ERDF and the CTU grant No. SGS17/212/OHK3/3T/18 funded by the MEYS of the Czech Republic.

References

1. Cejka, T., Bartos, V., Svepes, M., Rosa, Z., Kubatova, H.: NEMEA: a framework for network traffic analysis. In: 12th International Conference on Network and Service Management (CNSM 2016). Montreal, Canada (Oct 2016)
2. Cejka, T., Bartos, V., Truxa, L., Kubatova, H.: Using application-aware flow monitoring for sip fraud detection. In: Intelligent Mechanisms for Network Configuration and Security (AIMS 2015). pp. 87–99. Springer (Jun 2015)
3. Communication Fraud Control Association: Global fraud loss survey (2015), `http://www.cfca.org/pdf/survey/2015_CFCA_Global_Fraud_Loss_Survey_Press_Release.pdf`
4. Dwivedi, H.: Hacking VoIP: protocols, attacks, and countermeasures. No Starch Press (2009)
5. Gauci, S.: SIPVicious. Tools for auditing sip based voip systems (2012), `https://code.google.com/p/sipvicious/`
6. Velan, P., Celeda, P.: Next generation application-aware flow monitoring. In: Monitoring and Securing Virtualized Networks and Services, LNCS, vol. 8508, pp. 173–178. Springer (2014)

A.5 Using Application-Aware Flow Monitoring for SIP Fraud Detection

Ing. Tomáš Čejka (25 %), Ing. Václav Bartoš (25 %), Ing. Lukáš Truxa (25 %),
doc. Ing. Hana Kubátová, CSc. (25 %)
In *Intelligent Mechanisms for Network Configuration and Security: 9th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2015.*
Publisher: Springer International Publishing
Ghent, Belgium, 2015
ISBN: 978-3-319-20034-7, pp. 87–99
DOI: 10.1007/978-3-319-20034-7_10

As the modern technologies evolve, they replace the traditional ones because of flexibility and efficiency. Telephone services are one of such examples. Nowadays, it is possible to establish a phone call (even with various multimedia functions) over the Internet. There are various standards and specification of protocols and mechanisms that can be used. In our work, we focused on the Session Initiation Protocol (SIP), which is a signaling protocol.

The following paper ([P.7]) presents a stream-wise and application-aware detection algorithm that analyzes extended flow records with information about SIP headers. The aim is to automatically detect suspicious traffic that was observed in a real network. The anomalous and suspicious traffic can be characterized as multiple attempts to establish a phone call using many different combinations of prefixes of the callee phone number.

According to the experts on the telephone technologies, the aim of attackers, in this case, is probably to establish an unauthorized phone call via a SIP server owned by the victim. In case the attack is successful, this fraud can cause a significant financial loss. Unfortunately, the attackers try to exploit a bad configuration that can be set in a victim's SIP server.

The idea about analysis of the extended flow records with SIP headers to detect suspicious SIP traffic was the contribution of the author of this dissertation thesis, and it was successfully developed by Lukáš Truxa as a diploma thesis [P.38] (supervised by the author of this dissertation thesis). It was the intention of the author of this dissertation thesis to extend the results of the diploma thesis and to transform it into the conference paper. Václav Bartoš participated on the writing of the paper [P.2], he generalized the idea of the processing application information (i.e. application-awareness), and he significantly improved the text quality of the paper.

The algorithm and its possibility of parallel processing were described in Section 3.5.3.

Using Application-Aware Flow Monitoring for SIP Fraud Detection

Tomas Cejka[1], Vaclav Bartos[2], Lukas Truxa[1], Hana Kubatova[3]

[1] CESNET, a.l.e.
Zikova 4, 160 00 Prague 6, Czech Republic
cejkat@cesnet.cz
[2] Faculty of Information Technology, Brno University of Technology
Bozetechova 2, Brno, Czech Republic
ibartosv@fit.vutbr.cz
[3] CTU in Prague, FIT
Thakurova 9, 160 00 Prague 6, Czech Republic
kubatova@fit.cvut.cz

Abstract. Flow monitoring helps to discover many network security threats targeted to various applications or network protocols. In this paper, we show usage of the flow data for analysis of a Voice over IP (VoIP) traffic and a threat detection. A traditionally used flow record is insufficient for this purpose and therefore it was extended by application-layer information. In particular, we focus on the Session Initiation Protocol (SIP) and the type of a toll-fraud in which an attacker tries to exploit poor configuration of a private branch exchange (PBX). The attacker's motivation is to make unauthorized calls to PSTN numbers that are usually charged at high rates and owned by the attacker. As a result, a successful attack can cause a significant financial loss to the owner of PBX. We propose a method for stream-wise and near real-time analysis of the SIP traffic and detection of the described threat. The method was implemented as a module of the Nemea system and deployed on a backbone network. It was evaluated using simulated as well as real attacks.

1 Introduction

Computer networks are a multifunctional communication channel used by various different applications. The example of such an application that is studied in this paper is the telephone service – the Voice over IP (VoIP) technology. This, as well as many other applications, is often considered to be critically important for users. It is therefore important to have effective ways for quick detection of any problems, including security threats. This often means a necessity for monitoring and analysis of the traffic. A common approach allowing situational awareness even in high speed networks is the usage of flow monitoring.

Traditional flow monitoring provides data extracted from packet headers up to the transport layer. Therefore, it provides information about IP addresses,

TCP/UDP ports, TCP flags or ICMP message types in form of flow records. The flow records also contain statistics such as number of packets, number of transferred bytes and information about observation time. However, there is no detailed information about application layer protocols in the flow records.

The traditional flow record is sufficient for many purposes, including detection of several types of malicious traffic. For example, port scanning and SYN flood attacks are easy to detect using only these basic flow data, since these attacks have clearly distinguishable characteristics on network and transport layers. Even some attacks on application layer, such as dictionary attacks on SSH, can be detected using basic flow data with proper algorithms [7]. However, this is not always possible or it may be very difficult and unreliable. For some kinds of malicious traffic, knowledge of additional information from the application layer is necessary for reliable detection.

Fortunately, application awareness has been implemented into some flow exporters in the last years, usually in the form of plugins [9]. Such exporters inspect packet payload, extract information from headers of application layer protocols and add this information into flow records. The *extended flow records* can contain e.g. URLs, response codes from HTTP, or domain names from DNS requests along with common features of the flow record. The flow records are then transferred to a collector using the IPFIX protocol [3].

The extension of flow records by application layer information was added mainly to allow more detailed statistics about traffic or to support application performance monitoring. However, it can be used for detection of security threats as well. In this paper, we show an example of such a usage.

We focus on monitoring of the VoIP traffic, in particular the Session Initiation Protocol (SIP). Our goal is to detect one of the most common VoIP frauds – the one in which an attacker tries to misuse a poorly secured gateway to the public switched telephone network (PSTN) to make unauthorized calls. Such calls are often made to premium-rate numbers (operated by the attackers) causing significant financial losses to operators of the misused gateway.

According to [4], worldwide losses due to VoIP hacking and calling to premium rate services go to billions of dollars per year. It is one of the most costly fraud types in the telecommunication industry. Therefore, even though successful attacks are not very usual, it is highly important to detect them as soon as possible, before a significant damage is caused.

The rest of this paper is organized as follows. The next section describes the related work. Sec. 3 describes details about the attack on which we focus. Sec. 4 describes the detection method and our implementation of it, followed by Sec. 5 where it is evaluated. Sec. 6 concludes the work.

2 Related Work

Attacks and security threats are an ordinary part of network traffic. There are several different types of attacks that are targeted to VoIP infrastructure. Some possible attacks and vulnerability exploits are shown in [5] by El-Moussa et

al. The paper describes denial of service attacks, which are very common in computer networks, or SPAM over internet telephony. Furthermore, the authors mention brute-force attacks against authentication mechanism, which are somewhat similar to the prefix-guessing attacks described in this paper. They however do not present any countermeasures or a way of detection of such attacks.

Another work enumerating possible attacks against VoIP technology is a survey by A. Keromytis [12]. Although it summarizes hundreds of papers from the area, there is no mention about the kind of attack we deal with, nor any work using flow measurement to detect security threats in VoIP traffic.

To our best knowledge, the only paper providing a way of detection of toll fraud attempts is [8] by Hoffstadt et al. It is focused on monitoring of VoIP threats using honeypots. The authors describe principle of toll fraud based on hijacking of a SIP account. An attacker that gains a user's identity is able to establish a phone call that can be charged. Discovery of a fraud is based on an off-line analysis of honeypot logs. Even though the authors stated that they observed manual attempts of toll fraud, we are observing automatic brute-force guesses of dialing prefixes. Also, our flow-based approach allows to monitor traffic going to real gateways, not only honeypots, therefore we are able to detect real attacks and possibly raise alerts on the successful ones.

Besides common hardware and software VoIP phones, there are several tools that allow users to communicate over SIP in unusual ways. For example, they allow to craft a request with any values of headers or to perform brute-force attacks automatically. Examples of the tools are [6, 11, 13, 15].

A detailed description of the flow monitoring technology can be found in the article [9] by Hofstede et al. It also contains an overview of available software. The paper [14] by Velan and Celeda introduces the concept of application-aware flow monitoring.

3 Principle of the Phone Call Fraud

The following subsection provides a brief introduction to VoIP telephony and SIP. We focus only on aspects needed to understand the type of fraud discussed in this paper and the proposed detection method; we knowingly leave out many otherwise important details for brevity.

3.1 Short Introduction to SIP

Voice over IP is a technology for transferring voice and multimedial data over computer networks. Session Initiation Protocol (SIP) is the well-known protocol used for initiation, control and termination of VoIP sessions (or calls). The multimedial data are transferred in a separate channel using Real-time Transport Protocol (RTP).

SIP is a protocol based on a request-response transaction model. Every device can act both as a client or as a server. The client creates and sends requests, the

server receives and processes them, and generates one or more replies. The high-level architecture consists mostly of end devices (hardware or software phones) and SIP proxy servers. The proxies receive requests for calls, localize called parties and route the requests to them (possibly via other proxies). They also route replies on their way back to the callers. The proxies also provide authentication services and several other tasks, which are out of scope of this short introduction.

There are several types of requests in SIP. The one which is crucial for this work is `INVITE`. It is used to initiate a new call. As any request in SIP, it carries several headers describing parameters of the request. The most important headers of `INVITE` request are:

- *Request-URI:* Used for addressing the called party. It is usually in the form `sip:user@host`, although more complex forms are possible. The *user* part usually consists of user name or phone number and the *host* part is the destination server where the request should be sent to (domain name or IP address).
- *To:* Called party identification.
- *From:* Identification of the caller.
- *Call-ID:* Unique identifier of a call, usually a long random string.

When `INVITE` is sent by a client to a proxy server, the request is propagated to the destination, possibly via other proxy servers. Responses such as `TRYING` and `RINGING` are returned and when the called party eventually indicates that it is ready to establish the call, the `OK` response is sent to the caller and the multimedia transfer begins. When the connection can not be established for some reason, a reply with corresponding error code is returned.

If a SIP proxy server is deployed in some private organization to serve as a central hub for internal phone communication and as a proxy for communication with outer world, it is usually called a Private Branch Exchange (PBX). These PBXs usually operate with both VoIP as well as classic telephone networks (PSTN), acting as gateways between those two technologies. The following text focuses on misuse of such gateways.

3.2 Principle of the Fraud

Depending on configuration a PBX may allow users to make VoIP calls to PSTN numbers by setting the destination user ID in an `INVITE` request to the called number prepended by a special prefix. For example, to call a PSTN number 555-555-0123 a SIP call to `995555550123@example.com` needs to be performed, assuming that the gateway is located at `example.com` and it has "99" configured as the prefix for PSTN calls.

The attack is based on finding poorly secured PBXs and using them to make fraudulent calls to PSTN. Motivations to make such calls may differ, but the common one is to gain money by calling to paid services. This is outlined in Fig. 1. The attacker first loans a premium-rate phone number[4], usually in a

[4] *I. e.* a number which is charged at a high rate in favour of the line operator.

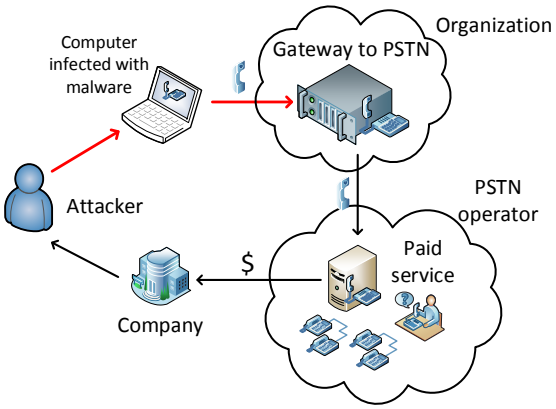


Fig. 1: Principle of the SIP fraud.

foreign country and via some intermediary company. Any calls to that number generate revenue for the attacker. Computers controlled by the attacker are then instructed to find open PBXs and make fake calls via them. When a call is successful, the PSTN operator charges the organization operating the PBX. The money goes to the company operating the premium line and to the attacker.

In order to make calls to PSTN via a PBX, the attacker needs to know the prefix which must be prepended in front of the number. Since this prefix depends on a particular PBX and its configuration, attackers usually do not know it. They must therefore guess it by trying various possibilities until the correct prefix is found and the call is successful or until all possibilities from a dictionary are used and the attacker moves on to another victim.

Such guessing can be recognized as a large number of `INVITE` requests from the same source, all trying to call the same number but with different prefixes. A typical sequence of URIs in the `INVITE` requests is shown in Fig. 2.

Such a sequence typically contains tens of `INVITE` requests with different prefixes. All the prefixes may be tried within a few minutes, but attackers often try to evade detection by putting long intervals between individual trials, so it may take up to several days. Such slow attacks are harder to notice in logs and generally harder to detect by any means.

In some cases, PBXs are configured insecurely and allow to make such calls without a proper authentication. More secured PBXs require an authentication header in the `INVITE` request. In such a case, attackers can perform a dictionary attack first, in order to find login credentials of some user. If some login and password is successfully found, the attacker may impersonate the user and the prefix guessing can be run in the same way as described. This paper focuses solely on the prefix guessing part of attacks. The detection method described in the next section makes no difference between authenticated and non-authenticated users.

00972592577956@A.B.C.D	**9999990**972592577956@A.B.C.D
000972592577956@A.B.C.D	**99999990**972592577956@A.B.C.D
900972592577956@A.B.C.D	**999999900**972592577956@A.B.C.D
+972592577956@A.B.C.D	**9999999900**972592577956@A.B.C.D
972592577956@A.B.C.D	**99999999900**972592577956@A.B.C.D
100972592577956@A.B.C.D	**9000**972592577956@A.B.C.D
800972592577956@A.B.C.D	**0**972592577956@A.B.C.D
600972592577956@A.B.C.D	**0000**972592577956@A.B.C.D
700972592577956@A.B.C.D	**0000000**972592577956@A.B.C.D
400972592577956@A.B.C.D	**00000000**972592577956@A.B.C.D
300972592577956@A.B.C.D	**000000000**972592577956@A.B.C.D
200972592577956@A.B.C.D	**0000000000**972592577956@A.B.C.D
500972592577956@A.B.C.D	**91000**972592577956@A.B.C.D
99900972592577956@A.B.C.D	**9900**972592577956@A.B.C.D
999900972592577956@A.B.C.D	**9100**972592577956@A.B.C.D
9999900972592577956@A.B.C.D	. . .

Fig. 2: An example of URIs called during a typical prefix guessing attack (IP address anonymized).

4 Detection Method

The detection method is designed to work without any prior knowledge of VoIP infrastructure and dialing plans on the network. The assumed deployment is on an ISP level or in a network of a large organization, where an operator of the detection system has no direct control over VoIP equipment but still wants to know about any issues related to it.

The detection is based on an analysis of SIP INVITE requests trying to make calls to PSTN numbers. The goal is to find IP addresses that generate large number of such requests varying only in a prefix of the called number. The detection method works even if prefixes are tried in a very low rate (e.g. one attempt per day). Also, it was needed to design the method to be efficient since we are targeting large networks with high volumes of traffic.

The input data comes from flow monitoring probes. Basic flow records are not sufficient for the detection of the SIP fraud, it is needed to extract additional information from SIP headers. In particular, we extended the flow records by *request/response code*, *Request-URI* and *To*, *From*, *Call-ID* and *User-Agent* headers from SIP messages. We achieved this by using a plugin for FlowMon probes [10]. It is the probe able to monitor high speed networks and parse information from application layer protocols. The whole monitoring infrastructure is shown in Fig. 3. Data from the monitoring probes are passed to a collector in the IPFIX format [3] and then into the Nemea system [1, 2] – a modular framework for network traffic analysis and anomaly detection. The detection method described in the following paragraphs was implemented as a software module for the Nemea system. It receives and analyses extended flow records of SIP traffic and reports detected attacks.

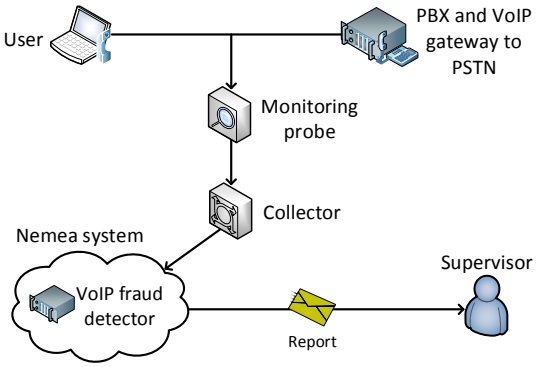


Fig. 3: Infrastructure of the monitoring system with the detection module.

The detection algorithm works as follows. For each incoming flow record carrying information about an INVITE request a called party identification is taken from *Request-URI* (or *To* header, depending on configuration; however, both are usually the same). If its *user* part, i.e. the part before the `@`, contains only digits or some of the allowed special symbols (`+`, `*`, `#`, `-`, `:`) it is further processed. Otherwise, the message is ignored since it is not a call to a phone number.

Responses to the INVITE messages are also processed and are used for determination whether the call was successfully established. In particular, when an OK response is observed after a previous INVITE request and their *Call-ID* headers match, the call is considered to be successful.

A set of URIs observed in INVITE requests is stored for each source IP address. In order to allow efficient storage and analysis of such sets, the URIs are stored into a specially designed data structure based on the suffix tree. Figure 4 shows an example of a set of URIs stored in such a tree. In the suffix tree, the common suffix of two or more URIs is represented by a parent node while the children nodes (or subtrees) represent their different prefixes. There is a rule that none of nodes can have a common part with its sibling. Therefore, in case a newly inserted URI contains an unknown prefix which has a common part with some existing prefix, it can cause a split of an existing node.

Each node represents an URI given by its value concatenated with values of all its ascendants. Each node also contains a number of call attempts to that URI, number of successfully established calls and other information, mostly for optimization of the detection algorithm.

Such a tree is constructed for each source IP address and is continuously updated as new INVITE requests from that address are observed. The trees are periodically analyzed in order to detect prefix guessing attacks. As shown in Sec. 3.2, during such attack a large number of URIs is observed with the same phone number and destination host but many different prefixes. That results in

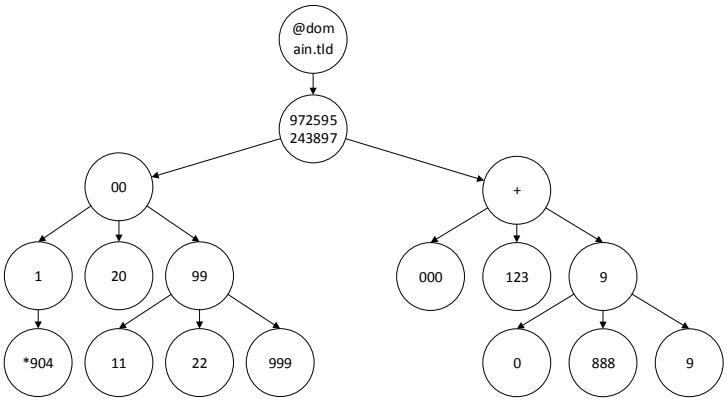


Fig. 4: Suffix tree for analysis of phone number prefixes.

a tree in which there is a single node which contains the phone number and have a large number of descendants.

The algorithm for detection of such a node works with two parameters – maximal prefix length (l_{max}) and a threshold on number of tested unique prefixes (T). At first, the tree is traversed from the bottom to the top (i.e. from leaves to the root). For each leaf node, the algorithm goes up through its ascendants until the total length of numbers stored in the visited nodes exceeds l_{max}. The final node potentially represents the called number. Then, the number of its descendants satisfying the following two conditions is counted: 1) the prefix represented by a node must be shorter than l_{max} and 2) there must be an unsuccessful call attempt made to the node's URI. If the number of such descendants is the same or higher than the threshold T, an attack is reported. Otherwise, the algorithm continues traversing the tree from another leaf node.

After the attack is reported, all related nodes are removed from the tree, so the same attack is not reported in the next run of the algorithm. Basic information about the attack is however kept. Therefore, if the attack continues by trying another prefixes and their number again exceeds the threshold, so it is detected as an attack, it is recognized that it is only a continuation of the attack reported earlier. The new detection is thus reported only as an update of the previous one.

Besides the suffix tree, some other information is stored per IP address, mostly for the purpose of reporting. This information includes the time of the last seen SIP message, the time of the last detected attack or the value of the *User-Agent* header.

The detector is designed for continuous processing of potentially infinite stream of data from the network. Some of the incoming data are stored in memory, but because the capacity of available memory is always limited, old data must be periodically removed. Most of the data removals are based on a simple timeout. If no SIP communication from an IP address has been detected for a

given time period, all information about that address is removed. The default timeout in our implementation is 14 days. Also, if a suffix tree of some address grows into a huge size (we use a threshold of 100,000 nodes), the whole tree is removed. Because such a big tree is often the result of a large attack, the detection algorithm is applied to the tree before removal. Finally, nodes representing prefixes in an attack are removed after the attack is reported, as was described earlier.

5 Evaluation

In order to evaluate the detection method, we prepared a SIP server simulating a PBX with a gateway to PSTN. The server was configured to not require authentication and to allow calls to PSTN using a three-digit prefix. A modified SIPVicious tool [6] (`svwar.py` script) was used to generate attacks to the server from several sources. The simulated attackers tried to call to a number with randomly changing prefixes until they guessed the correct one. The traffic between the attacking machines and the server was monitored by the detector. Both parameters of the detection algorithm, that is the maximal length of a prefix (l_{max}) and the minimal number of call attempts that is considered as a guessing (T), were set to 10.

At first, the tests were performed in a virtual environment with no other traffic than the generated attacks. As expected, all attacks were successfully detected and reported, except a few cases in which the correct prefix was guessed in less than 10 tries (such attacks could be detected as well by decreasing the threshold T, but too low threshold might cause false alerts when deployed on real network).

We continued by tests in the real environment – in the CESNET2 network. CESNET2 is the academic network of the Czech Republic, connecting Czech universities and many other organizations to the Internet (around 1 million IP addresses in total). Its perimeter – 10 peering links, all with wire speed of 10 Gbps – is monitored using FlowMon probes. The total traffic on these links ranges from 5 Gb/s at night to 25 Gb/s during the day (50k to 150k flow records per second). The average amount of SIP traffic is around 50 flows per second, with occasional peaks up to several hundreds.

The probes were running the plugin for extending flow records with values of SIP headers. The detector was deployed into an operational instance of the Nemea system which receives and analyzes flow records from all the probes. The SIP server and the machine simulating attacks were placed so that the traffic between them is observed by one of the monitoring probes. Configuration of the server, attacks and the detector was the same as before.

Despite the generated attacks were hidden in a lot of real traffic now, all the attacks consisting of at least 10 attempts were successfully detected again, no matter how slow or fast they were. A lot of real attacks were detected, too.

In a measurement period of two weeks, the detector received and analyzed 10.5 million flow records corresponding to SIP INVITE messages. There were

Tab. 1: Top-20 prefixes observed in the backbone network traffic.

Prefix	Count	Succ. calls	Prefix	Count	Succ. calls
00	3800	22	9	1946	2
000	3412	9	810	1706	6
900	3273	12	9000	1608	8
+	3072	13	9900	1599	4
(none)	2498	8	9011	1582	7
0000	2464	14	99900	1462	10
011	2286	3	9009	1330	2
800	2248	5	9810	1323	9
0011	2092	4	005	1303	2
009	1982	5	001	1297	8

15,992 prefix guessing attacks reported consisting of 201,438 INVITE messages. That means that on average 12.6 prefixes are tried by a single attacker at a single gateway. Around 1.9 % of all INVITE messages were marked as part of this kind of attacks (although we expect that many of the others are malicious as well, since authentication attacks generate a lot of INVITE messages, too). Approximately 1.1 % of attacks seemed to be successful[5], i.e. a call was successfully established.

During its operation, the detection module consumed about 200 MB of memory and took only about 5 % of CPU on average (Intel(R) Xeon(R) CPU E5-2630 @ 2.30 GHz).

During our testing of the detection module, we gathered a lot of information and statistics about the SIP attacks as well as the SIP traffic in general. The interesting results are summarized in the rest of this section.

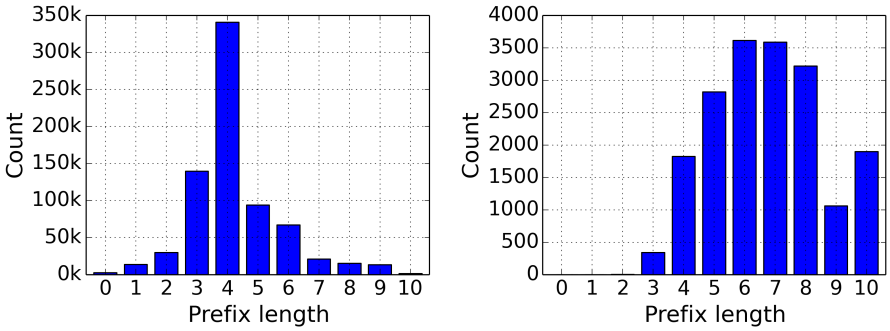
Table 1 shows a list of the 20 most often observed prefixes that were tested by attackers. The data for the table was taken from a two week period. The "Count" column represents the number of times the prefix was used and the "Succ. calls" represents the number of successful calls that were observed with the prefix.

Figure 5a shows the histogram of lengths of all prefixes that were used in attacks reported within a two week period. Figure 5b shows the histogram of the longest prefixes that were used in the reported attacks. That means it shows the maximal length of prefixes used in individual attacks. It can be observed that while most prefixes have 3 or 4 digits, attacks almost always contain some longer prefixes as well.

Table 2 shows the most frequent values of the *User-Agent* header that were observed on the backbone network. The data for the table was taken from a one week interval. There were 384 distinct values of the header. As can be seen, the most often used user agents (*sipcli*, *friendly-scanner*[6]) are scripts that allow users/attackers to craft SIP messages with any values of headers. They can be

[5] Not every successful attempt necessarily means a breach of a real gateway. Some of the gateways may in fact be honeypots.

[6] *friendly-scanner* is the default *User-Agent* value used by SIPVicious tool.



(a) (b)

Fig. 5: Histogram of the lengths of a) all prefixes tried in attacks, b) the longest prefixes tried in attacks.

Tab. 2: The most frequent values of the *User-Agent* header.

User-Agent	**Count**
sipcli/v1.8	643,312
friendly-scanner	424,178
Cisco-SIPGateway/IOS-12.x	6,304
FPBX-2.10.1(1.8.7.1	1,153
Asterisk PBX 11.11.0	570
None or empty	2,440
Other values	7,220

used to generate e.g. phone numbers of the callee in the case of prefix guessing. Of course, the *User-Agent* header cannot be a reliable source of information since a client can fill in any string. However, a legitimate client usually does not have any reason to present itself by the name of another tool and malicious clients apparently do not do that often.

6 Conclusion

This paper presented a possible usage of the emerging technology of application-aware flow monitoring in the area of security threat detection. In particular, a method for detection of VoIP-based toll fraud – a network attack that can lead to a significant financial losses – has been proposed. The detection is enabled by special flow monitoring probes which are able to extend flow records by information from application-layer protocols.

Using exported headers of Session Initiation Protocol (SIP), the proposed detection module is able to analyze SIP transactions and detect attempts to guess a prefix configured on a PBX to allow calls to PSTN. It is also able to

detect whether any of the attempts was successful. A successful call after a previous guessing indicates that the attacker found a way to make unauthorized calls via the PBX. In such a case it is necessary to alert an operator of the PBX immediately.

An implementation of the method was deployed and evaluated on a real backbone network. Some interesting results of the SIP analysis in real network traffic were presented in Sec. 5. The information from the extended flow records allows us to observe statistics about *User-Agent* headers and called phone numbers, for example. The detection capabilities of the proposed method are very good according to our experiments – all simulated attacks were successfully detected, as well as many real attacks. In fact, any attack consisting of at least 10 `INVITE` requests in a time window of 14 days is detected in default configuration. Of course, these thresholds can be tuned to fit requirements of the network operators.

While these attacks may be detected by other methods as well, for example by analysing logs of SIP servers, the flow monitoring approach allows to monitor all SIP servers in the network from one place, without necessity of access to the servers (which are usually operated by other people than those responsible for security). Moreover, if the detector is deployed on backbone links, like in our test scenario, it allows to observe attacks from many different sources to many different destinations, which is impossible with other methods (log parsing or honeypots). It provides us with more complete view on attacks and attackers. For example, by deep analysis of attack characteristics and their sources, it may be possible to detect groups of IP addresses attacking collectively, which can lead to revealing botnets.

Acknowledgments This work was partially supported by the "CESNET Large Infrastructure" (LM2010005), CTU grant No. SGS15/122/OHK3/1T/18 funded by the Ministry of Education, Youth and Sports of the Czech Republic, and BUT grant FIT-S-14-2297. This work was also supported by the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme, as well as Czech Ministry of Education, Youth and Sports via the project Large Research, Development and Innovations Infrastructures (LM2011033).

References

1. Bartos, V., Zadnik, M., Cejka, T.: Nemea: Framework for stream-wise analysis of network traffic. Tech. rep., CESNET (2013)
2. CESNET: Nemea, `https://www.liberouter.org/nemea/`
3. Claise, B., *et al.*: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011 (Sep 2013)
4. Communications Fraud Control Association: 2013 CFCA Global Fraud Loss Survey. Press release (October 2013), `http://www.cfca.org/pdf/survey/CFCA2013GlobalFraudLossSurvey-pressrelease.pdf`

5. El-Moussa, F., Mudhar, P., Jones, A.: Overview of sip attacks and countermeasures. In: Information Security and Digital Forensics, pp. 82–91. Springer (2010)
6. Gauci, S.: SIPVicious. Tools for auditing sip based voip systems (2012), https://code.google.com/p/sipvicious/
7. Hellemons, L., Hendriks, L., Hofstede, R., Sperotto, A., Sadre, R., Pras, A.: SSHCure: A Flow-Based SSH Intrusion Detection System. In: Dependable Networks and Services, LNCS, vol. 7279, pp. 86–97. Springer (2012)
8. Hoffstadt, D., Marold, A., Rathgeb, E.: Analysis of SIP-Based Threats Using a VoIP Honeynet System. In: Proccedings of the 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 541–548 (June 2012)
9. Hofstede, R., Celeda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A., Pras, A.: Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. IEEE Communications Surveys Tutorials 16(4), 2037–2064 (2014)
10. INVEA-TECH a.s.: FlowMon Probe – High-performance NetFlow Probe up to 10 Gbps, http://www.invea-tech.com/products-and-services/flowmon/flowmon-probes
11. KaplanSoft: SipCLI, http://www.kaplansoft.com/sipcli/
12. Keromytis, A.D.: A comprehensive survey of voice over ip security research. IEEE Communications Surveys & Tutorials 14(2), 514–537 (2012)
13. Ohlmeier, N.: SIP Swiss Army Knife (Sipsak), http://sourceforge.net/projects/sipsak.berlios/
14. Velan, P., Celeda, P.: Next generation application-aware flow monitoring. In: Monitoring and Securing Virtualized Networks and Services, LNCS, vol. 8508, pp. 173–178. Springer (2014)
15. VoP Security: SiVuS (SiP Vulnerability Scanner) – User Guide v1.07, http://www.voip-security.net/pdfs/SiVuS-User-Doc1.7.pdf

A.6 Analysis of Vertical Scans Discovered by Naive Detection

Ing. Tomáš Čejka (50 %), Ing. Marek Švepeš (50 %)
In *Management and Security in the Age of Hyperconnectivity: Proceedings of the 10th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2016*.
Publisher: Springer International Publishing
Munich, Germany, 2016
ISBN: 978-3-319-39814-3, pp. 165–169
DOI: 10.1007/978-3-319-39814-3_19

Vertical scanning, sometimes called port scanning, is usually classified as a harmless activity and can be used by administrators, operators or by automatic monitoring tools. Vertical scanning aims to test what ports of transport protocols are open on the target machine. A port is open when there is an application which listens on the port and answers to the packets with that comes.

However, network scanning can be easily used by attackers to gather information. Knowledge about open ports can be useful for various following scans or attacks. Vertical scanning is a normal phase of reconnaissance before an attack.

The following paper ([P.6]) presents a straightforward algorithm for detection of the vertical scans in a stream-wise detection algorithm. The paper also shows several statistics about traffic of the Czech NREN and detected scans.

The paper was written in cooperation with Marek Švepeš. The algorithm was designed by the author of this dissertation thesis, and it was developed and evaluated by Marek Švepeš.

The algorithm and its possibility of parallel processing were described in Section 3.5.4, which also mentions a minor modification of the algorithm to detect horizontal port scans instead of vertical scans.

Analysis of Vertical Scans Discovered by Naive Detection

Tomas Cejka[1], Marek Svepes[2]

[1] CESNET, a.l.e., Zikova 4, 160 00 Prague 6, Czech Republic
cejkat@cesnet.cz
[2] CTU in Prague, FIT, Thakurova 9, 160 00 Prague 6, Czech Republic
svepemar@fit.cvut.cz

Abstract. Network scans are very common and frequent events that appear in almost every network. Generally, the scans are quite harmless. Scanning can be useful for network operators, who need to know state of their infrastructures. Contrary, scans can be used also for gathering sensitive information by attackers. This paper describes a simple detection method that was used to detect vertical scans. Our aim is to show results of long-term measurement on backbone network and to show that it is possible to detect scans efficiently even with a simple method. The paper presents several interesting statistics that characterize network behavior and scanning frequency in a large high-speed national academic network.

1 Introduction

Network scanning is a common and frequent activity that can be observed in almost every network infrastructure. It is a normal benign mechanism used by network operators or automatic tools for monitoring and management. A network scan is based on probing targets to recognize the active ones. That is a scan referred as *horizontal*. Scans can also probe ports of one target. Such scans are called *vertical*. A *block* scan is a combination of horizontal and vertical scans.

Scans are easily performed even by attackers. Attackers can use scanning to search for publicly available services and vulnerable devices in the internet. Even though network scanning is basically harmless, current researches show, that it can be dangerous in some ways. Bartos et al. in [2] show correlation between network scans and attacks (e.g. bruteforce guessing of passwords or DoS attacks) that follow scans. Similarly in [11], Raftopoulos et al. discuss their observation about high probability of malware infection of devices that had been scanned previously. Therefore, it is important not to underestimate a danger of scans.

Unfortunately, there is no universal detection method, that would be suitable for all sizes of networks. According to [13], large transit networks or National Research and Education Network (NREN) infrastructures require a special detection approach. The main issues related to such networks are: high speed, wider diversity of IP addresses, lack of knowledge about end-hosts' configuration, asymmetric routing, coexistence with other monitoring and detection tasks without interference. This paper presents observations from the perimeter of

CESNET2. It is Czech NREN, a backbone and a transit network. Based on observations, we created a straightforward detection method.

2 Related Work

Bhuyan et al. presents a taxonomy of network scanning and a survey of some existing detection approaches in [3]. Using the taxonomy, we can classify the detection method presented in our paper as a threshold-based method.

One of the well-known methods is a Threshold Random Walk (TRW) proposed for scan detection by Jung et al. in [7]. The detection method was implemented as a part of Bro [10]. Sridharan et al. in [13] points out disadvantage of TRW that needs knowledge about the configuration of end-hosts. In backbone networks there are several issues that complicate scan detection. However, it is still useful to perform detection even on backbone level. The paper investigates effectiveness of existing methods and proposes a new method Time-based Access Pattern Sequential hypothesis testing (TAPS). Lee et al. is one of the most closely related works to this paper. Their paper presents a report about observed port scans. The authors analyzed two weeks of traffic at University of California, San Diego (UCSD) using Snort [12]. The paper was written in 2003 and the authors discovered 9,927 vertical scans. Whereas, we have been monitoring network traffic from CESNET2 more recently (2015) for longer time (two months) and average number of discovered vertical scans in two weeks was 203,000.

As it was shown, there are various approaches of scan detection. However, we used a simple flow-based method with thresholds and filtering flow records.

3 Detection Algorithm

The detection algorithm uses information from basic flow records (source and destination IP addresses and ports, protocol, #packets, #bytes). The principle of algorithm was inspired by characteristics of scans generated by nmap [8]. Analysis showed that scans are composed of plenty flow records with small number of packets (≤ 4) transferred between the source IP (potential attacker) and a destination IP (victim).

We have focused on a default scanning technique supported by nmap. It uses Transmission Control Protocol (TCP) packets with set SYN flag. This simulates establishment of a new TCP session and the target should reply with SYN+ACK if the probed port is opened. The detection results of SYN scans are verifiable manually even in unknown network traffic of backbone since TCP normal traffic from a host always contain not only SYN flag and should not imply plenty RST responses. Detection of other scan types is more complicated due to verification of false positives and missing ground truth in the real backbone traffic.

The detection algorithm is based on analysis of the number of destination ports per source IP and uses threshold for number of ports. It is important to remember all unique destination ports for each pair of addresses separately. The source IP is a potential source of scan, meanwhile, the destination IP is a victim.

The algorithm was implemented as a module of the NEMEA system [1, 5] and is described in more detail in [4].

4 Evaluation and Measurements

Our measurement started on 31. 10. 2015 and stopped on 31. 12. 2015. In total, we observed over 388 billion flow records from all monitoring probes. That is on average 76,283 flows per second with over 144,506 flows per second in peak.

In order to find a reasonable threshold for the number of destination ports, we measured average number of destination ports used by a source IP. Moreover, we were interested in maximal number of destination ports per source. These observations were based only on TCP protocol without any consideration of TCP flags. Values were computed in hour intervals. The results are shown in Fig. 1. It is clear that intensive port scans probe a lot of destination ports. Therefore, the maximal number of ports is over fifty thousand. Most of source addresses use only a few destination ports and therefore total average number of destination ports per source IP lies around ten. From this point of view, with respect to memory consumption, the threshold was experimentally set to 50. Distribution function in Fig. 2 shows, that over 99 % addresses use less then 50 destination ports. Therefore, source IP address which has used 50 or more destination ports is considered as a potential attacker.

On average, network scans take about 1.2 % of observed flows. Alerts are aggregated in 10 minute time windows. Using the aggregation, the number of alerts decreases by 94 %. Average length of the aggregated alerts is about 2 minutes 45 seconds and over 2,600 destination ports are being probed. On average, there are over 580 aggregated alerts per hour.

During the analysis of scans targeted to a single target, we found distributed scans as well. Scanning hosts were active for about 10 minutes and each scanner probed about 2,000 ports. Altogether, scanners probed disjoint sets of ports.

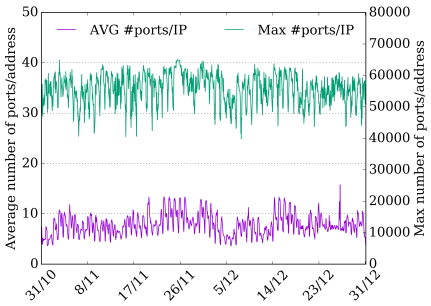


Fig. 1: Average and maximal number of destination ports per source IP.

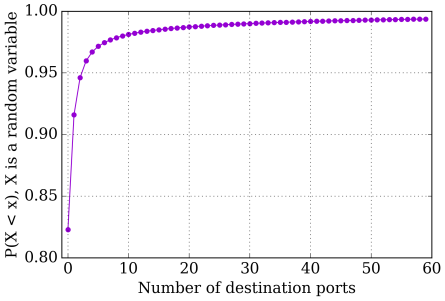


Fig. 2: Distribution function of number of destination ports per source IP.

According to the results, most of scans occur at 8:00 and 13:00 UTC. The rest of scans are spread over all hours. We expect that the distribution is caused by peaks of traffic that we normally see in these hours.

The deployed detector has discovered non-uniform intensity of some scans that was changing in time. For example, 47,156 addresses of Czech university of economics were scanned during 19 minutes by one block scan. The highest intensity (over 10,000 alerts per minute) was in the middle of the scan. Over 50 ports were probed for each target.

Memory consumption of the module was analyzed by valgrind [9]. The measurement was performed for almost two days and the module consumed 576 MiB during the peak. The amount of required memory is decreased due to autoregulation based on removing inactive addresses. It is set by module's threshold.

The used detection algorithm, from its nature, suffers from some limitation. The algorithm skips repeating SYN flows with the same destination port. Such traffic is assumed to be benign traffic. However, this fact can be easily exploited by scanners to avoid the detection. Distributed scans are generally difficult to detect. Large botnets can scan the whole internet using just a few packets generated by each bot [6]. A distributed scan can be detected by our algorithm if and only if at least some of the scanners fulfill conditions to be detected (e.g. number of destination ports threshold).

5 Conclusion

Vertical scans are frequent events that occur in almost every computer network. In this paper, we have proven this fact by observation of the vertical scans in the backbone network. The measurement showed that it is possible to detect scans with a simple straightforward detection algorithm using commodity hardware.

The proposed algorithm is limited to detection of TCP scans, however, it can be deployed in large network infrastructures and analyze huge volume of data. On average, there are about 580 aggregated alerts per hour that are detected by the implemented detection module. Some randomly selected alerts from the two-month measurement were verified manually. This paper presented statistical characteristics and results of scans detected at the perimeter of CESNET2.

Modern network attacks are mostly performed by botnets. Therefore, the importance of detection of distributed attacks (scans in our case) increases. According to our observations, intensive distributed scans became usual. However, the larger botnets are, the harder the detection is because each bot can probe just a few ports and so it is difficult to recognize bot's traffic from benign clients.

Acknowledgments This work was partially supported by the "CESNET E-Infrastructure" (LM2015042) and CTU grant No. SGS16/124/OHK3/1T/18 both funded by the Ministry of Education, Youth and Sports of the Czech Republic.

References

1. Bartoš, V., *et. al.*: Nemea: Framework for stream-wise analysis of network traffic. Tech. rep., CESNET, a.l.e. (2013), http://www.cesnet.cz/wp-content/uploads/2014/02/trapnemea.pdf
2. Bartos, V., Zadnik, M.: An analysis of correlations of intrusion alerts in an nren. In: Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2014 IEEE 19th International Workshop on. pp. 305–309. IEEE (2014)
3. Bhuyan, M.H., *et. al.*: Surveying port scans and their detection methodologies. The Computer Journal p. bxr035 (2011)
4. Cejka, T., Svepes, M.: Vertical Scan Detector README, https://github.com/CESNET/Nemea-Detectors/tree/master/vportscan_detector
5. CESNET, a. l. e.: NEMEA: Network Measurements Analysis Framework, https://github.com/CESNET/Nemea
6. Dainotti, A., *et. al.*: Analysis of a /0 stealth scan from a botnet. In: Proceedings of the 2012 ACM conference on Internet measurement conference. pp. 1–14. ACM (2012)
7. Jung, J., *et. al.*: Fast portscan detection using sequential hypothesis testing. In: Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on. pp. 211–225. IEEE (2004)
8. Lyon, G.F.: Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. Insecure (2009)
9. Nethercote, N., Seward, J.: Valgrind: a framework for heavyweight dynamic binary instrumentation. In: ACM Sigplan notices. vol. 42, pp. 89–100. ACM (2007)
10. Paxson, V.: Bro: a system for detecting network intruders in real-time. Computer networks 31(23), 2435–2463 (1999)
11. Raftopoulos, E., *et. al.*: How dangerous is internet scanning? In: Traffic Monitoring and Analysis, Lecture Notes in Computer Science, vol. 9053, pp. 158–172. Springer International Publishing (2015)
12. Roesch, M., *et. al.*: Snort: Lightweight intrusion detection for networks. In: LISA. vol. 99, pp. 229–238 (1999)
13. Sridharan, A., Ye, T., Bhattacharyya, S.: Connectionless port scan detection on the backbone. In: Performance, Computing, and Communications Conference, 2006. IPCCC 2006. 25th IEEE International. pp. 10–pp. IEEE (2006)

A.7 Making flow-based security detection parallel

Ing. Marek Švepeš (50 %), Ing. Tomáš Čejka (50 %)
In *Security of Networks and Services in an All-Connected World: Proceedings of the 11th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2017.*, Zurich, Switzerland, 2017
Publisher: Springer International Publishing
ISBN: 978-3-319-60774-0, pp. 3–15
DOI: 10.1007/978-3-319-60774-0_14

The following paper ([P.3]) contains practical experiments about parallel flow data processing. It is based on splitting a stream of flow data into subsets based on *witnesses* (described in Section 3.4.2 and in Appendix A.8). The paper presents an infrastructure composed of a *Flow Scatter* and independent computing nodes with single instances of a detection system (NEMEA from Section 3.1 and Appendix A.1 was used for experiments).

The Flow Scatter was tested with different methods of flow data splitting, and the results of the experiments are concluded in the paper. The paper shows the importance of preserving semantic relations in the flow data. The flow data are split by the Flow Scatter into subsets that can be processed/analyzed in parallel (described in Section 3.6 and further extended in Section 3.6.1).

The paper was written in cooperation with Marek Švepeš. The general design concept was created by the author of this dissertation thesis, and it was successfully realized and evaluated by Marek Švepeš in his diploma thesis [P.33] (that was supervised by the author of this dissertation thesis). This diploma thesis was one of the nine finalists in ACM ITSpy2017 contest (out of about 1700 diploma thesis).

Making flow-based security detection parallel

Marek Švepeš[1] and Tomáš Čejka[2]

[1] CTU in Prague, FIT
Thakurova 9, 160 00 Prague 6, Czech Republic
svepemar@fit.cvut.cz
[2] CESNET, a.l.e.
Zikova 4, 160 00 Prague 6, Czech Republic
cejkat@cesnet.cz

Abstract. Flow based monitoring is currently a standard approach suitable for large networks of ISP size. The main advantage of flow processing is a smaller amount of data due to aggregation. There are many reasons (such as huge volume of transferred data, attacks represented by many flow records) to develop scalable systems that can process flow data in parallel. This paper deals with splitting a stream of flow data in order to perform parallel anomaly detection on distributed computational nodes. Flow data distribution is focused not only on uniformity but mainly on successful detection. The results of an experimental analysis show that the proposed approach does not break important semantic relations between individual flow records and therefore it preserves detection results. All experiments were performed using real data traces from Czech National Education and Research Network.

1 Introduction

Flow-based monitoring plays a key role in network management. Not only it provides an overview of the traffic mix, it greatly helps with network security issues such as malicious traffic detection.

There are many types of malicious traffic that should be detected in real networks. As the speed and size of computer networks grow, it is necessary for network operators to process more and more data to be informed about the status of their network. However, with the increasing traffic volume, it is difficult to run lots of detection algorithms at once using just a single machine. The more data, the more computing resources are needed and the longer time the processing takes.

In order to overcome resource limits of a single machine, parallelism plays an important role. Various types of scalable architecture have been invented to process data in parallel. Generally, to be able to process more data, analyzer has to either run parts of its algorithms in parallel or split data for separate processing units.

Since the parallelization of individual detection algorithms is very dependent on the nature of the algorithm and, additionally, according to Amdahl's law,

there are parts of algorithms that can't be run in parallel, we have decided to focus on data distribution for independent processing units. Our aim is to split a continuous stream of network data (more specifically flow records, i.e. aggregated packet headers) into much smaller subsets that are being processed separately in parallel. We also focus on evaluation of the impact of data splitting on the security analysis results.

The contribution of this paper is to present our experiments with processing data traces from the real backbone network. The aim is to use existing algorithms from a single machine processing and deploy them in a distributed environment. This paper shows, that data splitting for such purpose is complicated due to semantic relations in data which should be preserved. Breaking the relations can cause that the obtained detection results are significantly worse than using a single processing machine. The paper also shows a feasible way how to split flow data with respect to semantic relations. Proposed approach preserves detection results and allows a scalable deployment.

This paper is organized as follows. Sec. 2 describes existing related work, i.e. systems for anomaly detection, traffic sampling and network traffic processing in parallel. Sec. 3 describes scattering methods that can be used to split flow records into a separate groups for parallel processing by independent computational nodes. Sec. 4 describes our testing environment that was created for our experiments. The section also presents results of measurement of described scattering methods. Sec. 5 concludes the paper.

2 Related work

This section describes related approaches of parallel network traffic analysis and anomaly detection usually done using Network Intrusion Detection System (NIDS) or Network Intrusion Prevention System (NIPS).

There are many existing systems for network traffic analysis and anomaly detection that are modular by design. For instance, TOPAS [1] and NEMEA [2] are flow-based systems that consists of modules that process data. When there is a big volume of flow data, running the systems on a single machine may reach resource limits of the machine. The systems do not support data distribution natively as it is available for various big data frameworks. However, NEMEA modules can be easily run and connected in a distributed environment.

K. Xinidis et al. in [3] presented an architecture with Active Splitter for distributed NIDS aiming for performance optimization of the detection sensors running Snort [4] (packet-based system performing deep packet inspection). The splitter uses hash functions for packet distribution and three techniques to optimize the performance of the sensors. Cumulative Acknowledgements reduce redundant sending of packets between splitter and sensors, Early Filtering in splitter applies Snort rule subset on packet headers (no payload inspection) and finally Locality Buffering reorders packets in a way that improves the locality of sensors memory accesses.

H. Sallay et al. in [5] made the network traffic analysis distributed using switch/router. The architecture contains dedicated sensors for individual services (e.g. FTP) and the incoming traffic is forwarded to them according to switching table of the switch/router. Sensors are running Snort but only with needed rule subset for their service. Since the volume of traffic of individual services can differ significantly, the load of computational nodes wouldn't be uniform. Therefore, this approach is not suitable for us.

Nam-Uk Kim et al. in [6] compare static and dynamic hash-based load balancing schemes and propose dynamic (i.e. adaptive) load-balancing scheme for NIDS. It uses a lookup table which is periodically reorganized according to historical packet distribution and current load of individual nodes. If needed, flows with the smallest volume are reorganized. Proposed method distributes packets in a way that does not break the flow stream, however, they don't take into account relations between individual flow records and the impact of splitting on detection results is not evaluated.

M. Valentin et al. in [7] presented a NIDS cluster for scalable intrusion detection. It consists of frontend nodes that distribute packets between backend nodes running Bro [8] for intrusion detection. Moreover, there are proxy nodes propagating state information of backend nodes and also one central manager node for collecting and aggregating results. Each frontend node distributes data from one monitored line and uses a hashing distribution scheme with a single hash function. The architecture requires backend nodes and proxy nodes to exchange data with detection subresults. In our approach, we are dealing with splitting a stream of flow records instead of packets. Our hashing distribution scheme is adjusted to provide all needed data to the detection methods for correct intrusion detection. Therefore, our computational nodes running intrusion detection are independent and don't communicate with each other. Finally, our proposed hashing distribution scheme represents a general way, how to split flow data with respect to detection results.

Big data frameworks such as Hadoop [9] or Spark [10] are distributed by nature. They are based on storage of data onto some distributed file system. A special designed parallel algorithm can be used to run on many distributed nodes and process all data. A universal and the most popular approach of distributed processing is MapReduce. However, the overall result of this kind of computation usually depends on the Reduce phase that merges local results from all nodes. Therefore, only low attention is paid to any relations or semantics during the data distribution and storage. An improved data distribution in Hadoop was presented as Hashdoop in [11]. Contrary, our approach is more general and it is applicable even on stream-wise processing with multiple different algorithms. Even though the main focus of ours is to make non-distributed system working in parallel, the principle described in our paper can be used for improvement of data distribution in big data frameworks as well.

Sampling has a common goal with parallel processing – capability to handle more data at the same time. J. Mai in [12] shows impact of the packet sampling on detection of portscanning and K. Bartos in [13] deals with flow sampling

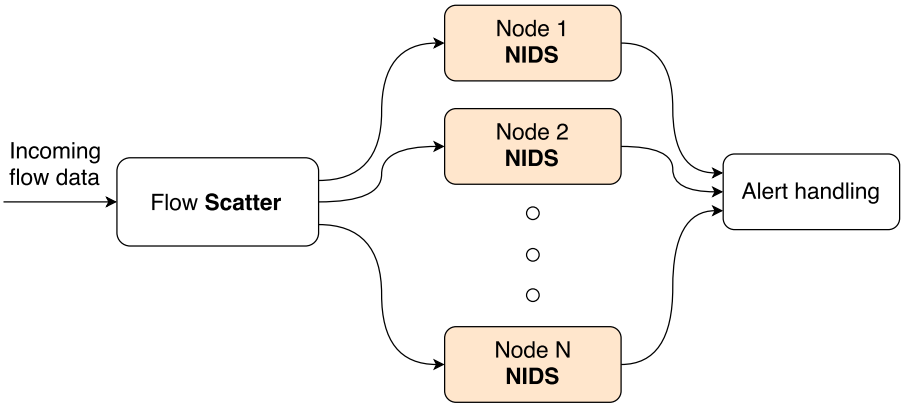


Fig. 1: A high level view of the infrastructure of scalable and distributed network flows analysis using NIDS.

techniques for anomaly detection. However none of these approaches can be applied on data splitting.

3 Flow distribution scheme

When designing a distribution scheme, several aspects have to be taken into account: i) the data should be distributed uniformly between all computational nodes, ii) the distribution algorithm should be as fast as possible in order to process as much data as possible, iii) the impact of splitting the data on detection results should be minimized.

In general, there are two ways how to distribute the data, statically or dynamically (also called static and dynamic load-balancing) and both have some pros and cons when applied in parallel NIDS. Static distribution has immutable rules for splitting the data e.g. a packet with source IP address 1.2.3.4 goes to node 1 and a packet with source IP address 5.6.7.8 goes to node 2. This preserves the data stream with possible security incident. However, it cannot affect the load of individual computational nodes when the distribution is not uniform. On the other hand dynamic distribution can perform some actions in order to make the load uniform (e.g. redirect some packets to less loaded computational nodes). Unfortunately, this behaviour can make the security incident invisible. Therefore, we have decided to use static distribution and focus on uniformity.

Fig. 1 shows high level view of the infrastructure of scalable and distributed network flows analysis using NIDS. The following subsections describe several splitting mechanisms used in the flow scatter.

3.1 Random scattering

Lets assume the detection results are not dependent on any semantic relations between flow data, i.e. scattering mechanism can distribute the data regardless

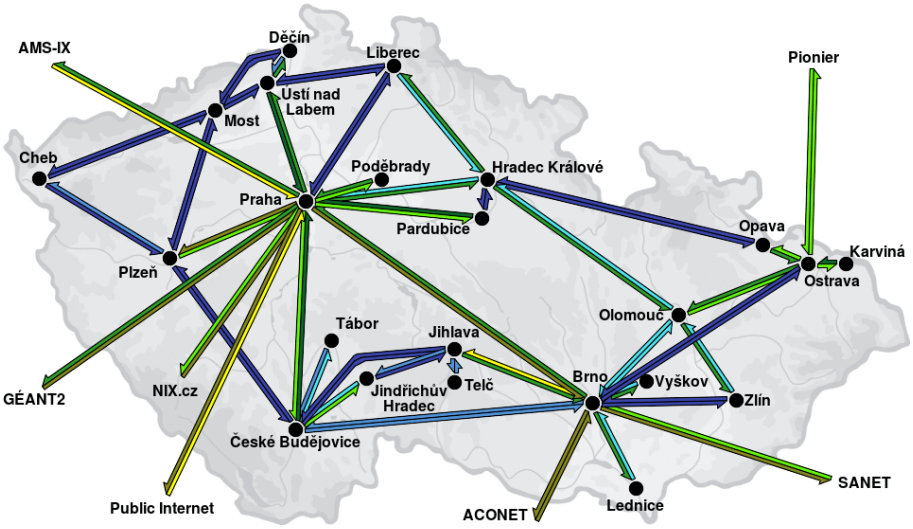


Fig. 2: Topology of Czech national research and education network (NREN) CESNET2, network traffic on the perimeter is analyzed.

of the information from flow records. In that case, the flow scatter can distribute the flow records between nodes using statistical uniform distribution, which is optimal for load-balancing. Received records by flow scatter are forwarded to computational nodes according to random number generator. It is clear that every random distribution splits the flow records into different subsets. However, as we discuss in Evaluation (Sec. 4), breaking semantic relations in flow data using random distribution affects the detection results.

3.2 Scattering based on network topology

Scattering based on network topology is another logical way of distributing the flow data. Since the computer networks are designed using hierarchical model that usually respects geographical and logical division into subnets and network lines.

Fig. 2 shows a high level topology of CESNET2 National Research and Education Network (NREN), which is a backbone academic network and it is also used as a transit network. It is inter-connected with other networks via several lines that are being monitored. The data taken from the monitoring probes contain a line identification — the line number. Flow scatter can easily distribute the data using these line numbers.

Standard monitoring infrastructure collects flow records from monitoring probes onto one central collector. In case of scattering based on network topology, this concept can be changed and it would be more efficient to send exported flow records directly to computational nodes.

3.3 Hash-based scattering

Hash functions are used to transform an input data into an output form with a fixed length. Cryptography expects that the output of an ideal hash function meets requirements such as uniform distribution and missing relation between output and input. In our case, the hash function can be used in the flow scatter to select an appropriate computational node number uniformly. Information from the incoming flow records can be used as an input for the hash function.

The dependency of selected node number on the input data of the hash function leads to divison of flow records into subsets with the same characteristics. The subsets with the same characteristics are then processed together on the same computational node and this can be used to preserve the detection results. For instance, if we use only the source IP address for hashing, all flow records having the same source IP address ends up on the same node. Meanwhile, flow records with different source IP addresses have a high probability to be processed with different nodes.

Let some set of flow records contain a security incident that can be detected using some detection method. Then, there exists a minimal subset of flow records with semantic relations that must be processed by this detection method together to get a correct result. In order to find the semantic relations in flow data, a set of detection methods was studied. The aim is to find a suitable set of information that is used as an input for hash function.

Studied detection methods

- Vertical SYN scanning can be detected using a threshold-based method published in [14]. To successfully detect this type of scanning, the method needs to receive all flow records of the same source IP address which is a possible attacker (scanner). Similar method can be used to detect horizontal SYN scanning. Source IP address is used for hashing.
- Brute-force password guessing against remote management services (SSH, TELNET, RDP etc.) can be detected using a method which needs to inspect all flow records between two IP addresses in both directions. An ordered pair of source and destination IP addresses (i.e. bi-flow) is used for hashing.
- There are many public lists of malicious addresses (black-lists). These addresses were abused due to various reasons like sharing malware, controlling botnets or acting in some anomalous evil way. Communication with a blacklisted IP address can indicate some malware infection and thus it should be reported. The detection is quite easy — every time any blacklisted address appears in a flow record, an alert can be sent. This type of detection is very efficient with a scattered data, because just a single flow record is needed to trigger an alert. Source IP address is used for hashing.
- More complex method based on statistics about IP addresses and matching the rules describing malicious traffic is able to detect DoS, DNS amplification, SSH brute-force password guessing and horizontal scanning. It needs to receive all flow records with the same IP address regardless of whether it is a

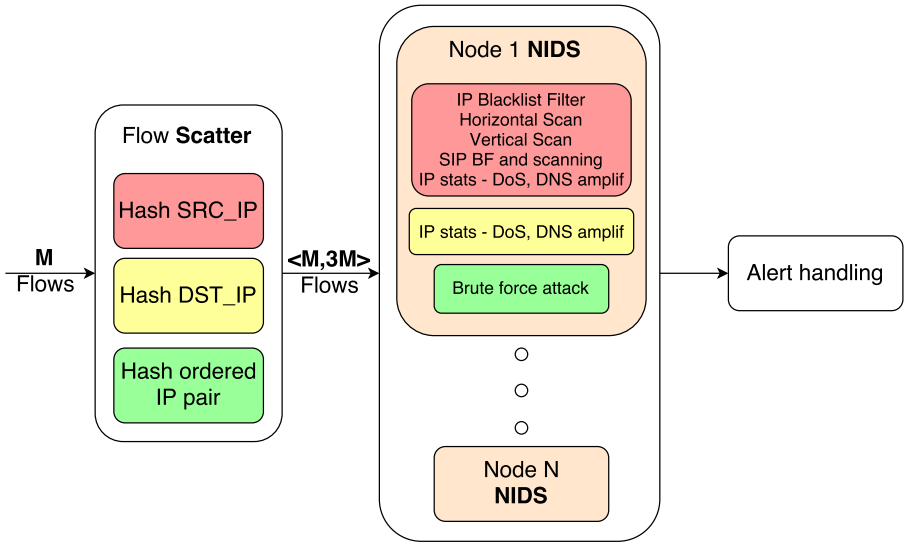


Fig. 3: Flow scatter contains three hash functions, each uses a specific information from flow records. The result of a hash function determines the computational node that processes the flow record with corresponding group of detection methods.

source or a destination IP. Therefore, hashing both source and destination IP addresses separately is needed in this case, which can result in duplication. The flow record can be forwarded to two different computational nodes. The duplication effect will be discussed later in this section.

– One of the detection methods based on application layer can detect brute-force attacks and scanning of user accounts on a Session Initiation Protocol (SIP) device. The detection method analyzes SIP responses from the server so all flows with the same source IP address must be delivered to the same node. Source IP address is used for hashing.

In general, we have recognized three groups of detection algorithms, whereas each group has to process all flow data with the same characteristic (e.g. same source IP address) on a single computation node. Therefore, we have a group of detection algorithms expecting all flow records with the same **source** address, a group expecting flow records with the same **destination** address and a group expecting flow records with the same **ordered pair of source and destination** addresses. Fig. 3 shows all three hash functions of the flow scatter where each hash function has the same color as the corresponding group of detection algorithms.

Since we want to run all detection algorithms in parallel, all three hash functions must be computed for every flow record. Naturally, results of the three hash functions can be different. Therefore, one flow record can be sent to at least one and, in the worst case, up to three computational nodes. This duplication

is caused by the number of different groups of algorithms and it is needed to provide all flow records that should be processed together to the algorithms (to preserve correct detection results).

In fact, the number of duplicates does not affect overall scaling of the parallel processing i.e. higher number of computational nodes does not increase the duplicates. Moreover, each hashing function determines a computational node, which processes the flow record with corresponding group of detection methods. Therefore, each group processes the flow record only on one computational node and every flow record is processed by all groups of detection methods. For example, if the selected nodes are 2 (for the SRC IP red hash) and 5 (for the DST IP yellow hash and for the IP pair green hash), it is processed by red group on `node 2`, yellow and green group on `node 5`. It means, that flow record may be duplicated, but only on a communication level between flow scatter and computational nodes.

To compare our approach with a single hashing function e.g. NIDS cluster [7] uses *hash = md5(srcIP + dstIP)*, we can show, that it would not work for us. Let's take methods for detection of horizontal port scanning and brute-force password guessing discussed in Sec. 3.3. The method for brute-force password guessing needs to see all flow records between source and destination IP addresses in both directions, so this hash function would work (*md5(A + B)* is equal *md5(B + A)*). On the other hand, horizontal scanning has the same source IP address but different destination addresses, so it is possible that two flow records with the same source IP but different destination IP could end up on a different computational node.

Our approach with multiple hash functions can be applied on arbitrary detection method. To do so, it is necessary to determine characteristics of needed flow data for correct detection result, as it was done in Sec. 3.3.

4 Experiments and Evaluation

In order to evaluate all important aspects of the scattering methods (uniformity, speed, impact on detection), the NEMEA system was chosen as a platform for our experiments and evaluation. The system itself has already implemented detection methods, which were studied and discussed in Sec. 3.3 and its efficient libraries allow us to process traffic from high speed backbone network. Overall, we have processed over 5 billions of flow records of real data traces in 10 different (pseudonymised) data sets captured in CESNET2 NREN during August and September 2016 with on average of 60,000 flows/s.

4.1 Testing environment

For our experiments we used a virtual machine with 64b Scientific Linux 7 OS, with the following hardware specification: 16 CPU cores, 24 GB RAM, 2 TB free disk capacity.

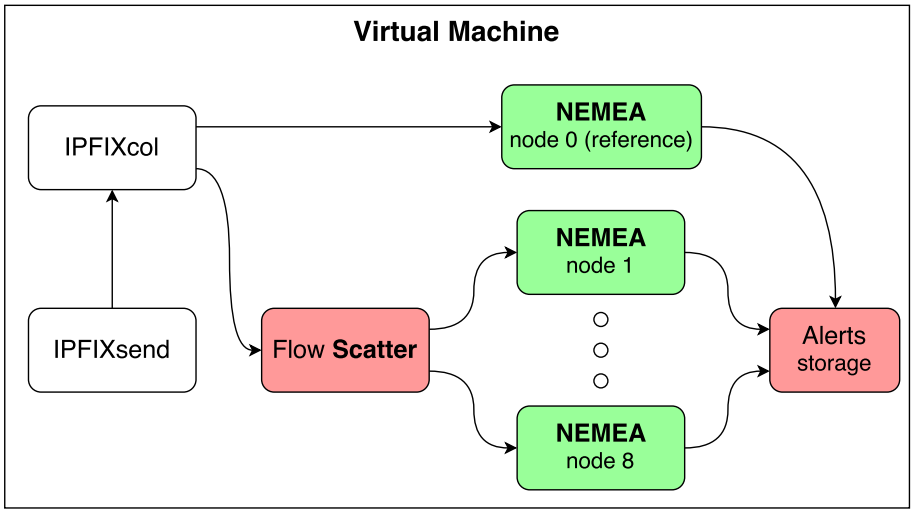


Fig. 4: Testing environment for experiments and evaluation of various methods of flow data distribution between computational nodes running flow-based NIDS NEMEA.

Fig. 4 shows the configuration of our testing environment. `IPFIXsend` and `IPFIXcol` [15] were used for replaying the IPFIX data in real-time. The flow data were received by the `flow scatter` and also directly by the `node 0` which was used as a reference single instance (it processes all flow data without any splitting). The `node 0` was a ground truth for us to evaluate an impact of data splitting on detection results. The flow scatter distributes flow data between nodes 1–8 as it was described earlier. All nodes contain exactly the same set of detection methods. During the experiments, we have collected data from 8 exporting probes that monitor different lines.

4.2 Results

The detection results from all nodes were stored and the analysis is described in this section.

Fig. 5 shows a comparison of an average distribution of flow records based on various scattering methods. The optimal value (red dashed line) is 12.5 % for 8 nodes. Random scattering achieves optimal results because of the used statistical uniform distribution. However, hash-based scattering is not significantly worse than the random (i.e. optimal) one. On the other hand, link-wise scattering is unbalanced because of different speeds of the monitored lines and the volume of traffic[3]. Node 1 even processed no data because there were no data exported

[3] We expect that such unbalanced distribution based on observation points can be observed in every network with lines of different bandwidth.

from the first line. For hash-based scattering, we have compared data distribution using two different hashing algorithms — CRC32 and Jenkins. On average, CRC32 had better results and therefore it was chosen as a final solution.

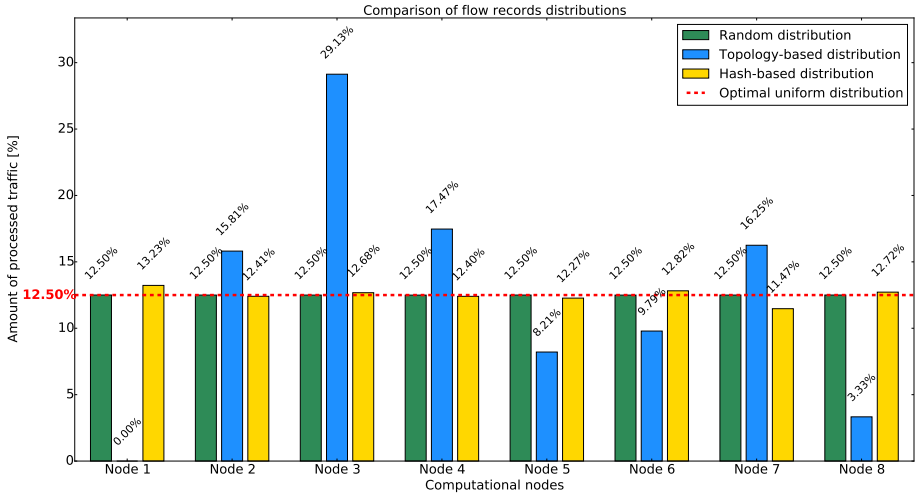


Fig. 5: Comparison of average flow records distributions using various scattering methods.

To analyze the reported alerts, we needed to compare the set of unique events from the reference node 0 with the set of unique events from all distributed nodes. To achieve this, the reported events of each detection method and each node were analyzed separately at first. Subsequently, the unique events were merged together. For example, in the case of horizontal scanning, if an attacker probes 50 or more computers in two different subnets, where 50 is a threshold of the detection algorithm, 2 events should be reported. Hash-based scattering delivers all flow records representing this traffic to the same node due to the source IP address hashing. Using link-wise scattering, the flow records could end up on different nodes because the traffic can go through different lines. Random scattering will split the flow records randomly.

Fig. 6 shows a comparison of the detection results after applying various scattering methods. Note, that *Hoststatsnemea* in the figure legend stands for the method based on statistics about IP addresses, which was discussed in section 3.3. The first column represents the reference instance with 100 % reported events, whereas each type of events has a different color and it is normalized so that the number of different event types are represented equally. Random distribution (the second column) has a huge impact on the detection results because of breaking the semantic relations between flow records. This was an expected result, however, the random distribution is a reference of optimal flow data distribution. Scattering based on the network topology (the third column) caused

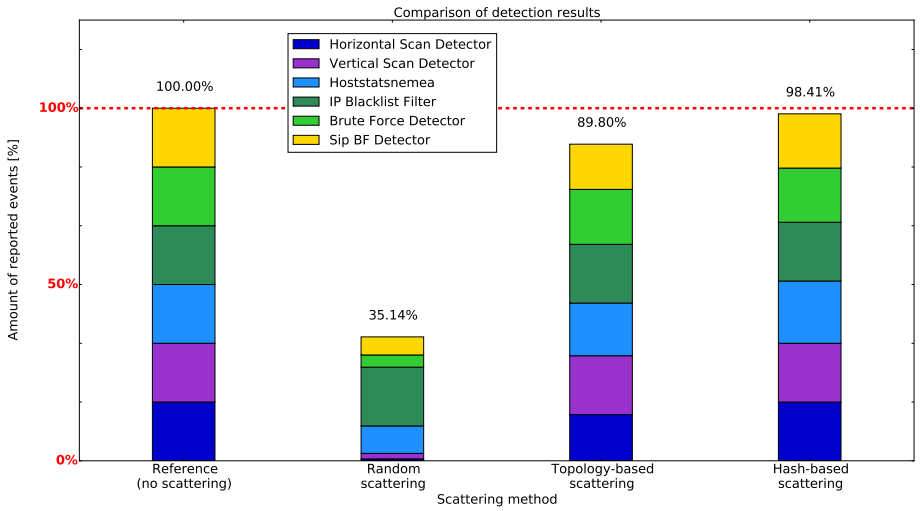


Fig. 6: Comparison of the detection results after applying various scattering methods. Each part of column with different color stands for normalized number of unique events reported by different detection method.

that some of distributed attacks and, in general, N:1 or 1:N attacks (DDoS, horizontal scanning etc.) were not detected. The last column shows that scattering based on hashing specific information from flow data has the best results. The reason of undetected events is probably a periodic clean-up of structures containing information and timing of stream-wise detection algorithms.

After the evaluation of the uniformity and the impact on the detection, we tested a maximal throughput of the hash-based flow scatter as the best method for distribution. A simple NEMEA module was created to generate and send 100 million flow records at full speed to the flow scatter. Measured computation time was focused on the main cycle receiving the flow record, hashing, making decision about number of computational nodes the flow belongs to according to the computed hashes and sending the flow record. The maximal throughput was on average 1.8 million flow records per second.

5 Conclusion

This paper presented the results of practical experiments with different approaches of splitting a stream of network flow data for the purposes of parallel anomaly detection. The aim of our work was to compare not only a uniformity of distribution but also an impact of data splitting on the detection results. Our experiments were performed using real traffic traces from Czech national research and education network (NREN). For simulation of parallel processing, we used an open source detection system NEMEA, however, the analysis results

are general enough and we believe that the proposed distribution approach can be used with any other detection system.

We have recognized three groups of detection algorithms with different requirements on data. Therefore, we have designed a flow scatter that uses three different hashing specific information from flow records (source address, destination address, ordered pair of source and destination address) to provide all needed data to independent computational nodes. The results of our experiment show that our approach preserves semantic relations in flow data that are important for different groups of detection algorithms and therefore the results of parallel detection are similar to reference results without splitting the data.

With the proposed approach of flow data distribution, it is possible to use detection methods that are deployed on a single machine and run them in parallel without changes. As a future work, we want to make more experiments with scaling beyond the measured throughput of the flow scatter by using multiple flow scatters in parallel and distribute incoming flow records between the flow scatters with e.g. round robin.

Acknowledgment

This work was supported by the Technology Agency of the Czech Republic under No. TA04010062 *Technology for processing and analysis of network data in big data concept*, grant No. SGS17/212/OHK3/3T/18 funded by MEYS and the project Reg. No. CZ.02.1.01/0.0/0.0/16_013/0001797 co-funded by the MEYS and ERDF.

References

1. Munz, G., Carle, G.: Real-time analysis of flow data for network attack detection. In: 2007 10th IFIP/IEEE International Symposium on Integrated Network Management. (May 2007) 100–108
2. Cejka, T., Bartos, V., Svepes, M., Rosa, Z., Kubatova, H.: Nemea: A framework for network traffic analysis. In: 2016 12th International Conference on Network and Service Management (CNSM). (October 2016) 195–201
3. Xinidis, K., Charitakis, I., Antonatos, S., Anagnostakis, K.G., Markatos, E.P.: An active splitter architecture for intrusion detection and prevention. IEEE Transactions on Dependable and Secure Computing **3**(1) (January 2006) 31–44
4. Roesch, M.: Snort - lightweight intrusion detection for networks. In: Proceedings of the 13th USENIX Conference on System Administration. LISA '99, Berkeley, CA, USA, USENIX Association (1999) 229–238
5. Sallay, H., Alshalfan, K.A., Fred, O.B., Words, K.: A scalable distributed ids architecture for high speed networks. In: IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.8, Citeseer (2009)
6. Kim, N., Jung, S., Chung, T.: An efficient hash-based load balancing scheme to support parallel NIDS. In: Computational Science and Its Applications - ICCSA 2011 - International Conference, Santander, Spain, June 20-23, 2011. Proceedings, Part I, Springer (January 2011) 537–549

7. Vallentin, M., Sommer, R., Lee, J., Leres, C., Paxson, V., Tierney, B.: The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware. In: Recent Advances in Intrusion Detection: 10th International Symposium, RAID 2007, Gold Goast, Australia, September 5-7, 2007. Proceedings, Berlin, Heidelberg, Springer Berlin Heidelberg (2007) 107–126
8. Paxson, V.: Bro: A system for detecting network intruders in real-time. Comput. Netw. **31**(23-24) (December 1999) 2435–2463
9. Apache: Hadoop http://hadoop.apache.org.
10. Apache: Spark http://spark.apache.org.
11. Fontugne, R., Mazel, J., Fukuda, K.: Hashdoop: A MapReduce framework for network anomaly detection. In: IEEE Conference on Computer Communications Workshops (INFOCOM). (2014)
12. Mai, J., Sridharan, A., Chuah, C.N., Zang, H., Ye, T.: Impact of packet sampling on portscan detection. IEEE Journal on Selected Areas in Communications **24**(12) (December 2006) 2285–2298
13. Bartos, K., Rehak, M.: Towards efficient flow sampling technique for anomaly detection. In: Proceedings of the 4th International Conference on Traffic Monitoring and Analysis. TMA'12, Berlin, Heidelberg, Springer-Verlag (2012) 93–106
14. Cejka, T., Svepes, M.: Analysis of vertical scans discovered by naive detection. In: Management and Security in the Age of Hyperconnectivity: 10th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2016, Munich, Germany, Springer International Publishing (2016) 165–169
15. Velan, P., Krejčí, R.: Flow information storage assessment using ipfixcol. In: Proceedings of the 6th IFIP WG 6.6 International Autonomous Infrastructure, Management, and Security Conference on Dependable Networks and Services. AIMS'12, Berlin, Heidelberg, Springer-Verlag (2012) 155–158

A.8 Preserving relations in parallel flow data processing

Ing. Tomáš Čejka (50 %), Ing. Martin Žádník, Ph.D. (50 %)
In *Security of Networks and Services in an All-Connected World: Proceedings of the 11th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2017.*, Zurich, Switzerland, 2017
Publisher: Springer International Publishing
ISBN: 978-3-319-60774-0, pp. 153–156
DOI: 10.1007/978-3-319-60774-0_14

Network monitoring systems produce lots of data every hour that must be analyzed as soon as possible because the delay is a critical parameter in the field of network security. Analyzing in parallel is a difficult task since usually requires significant adaptation of the existing algorithms for a parallel environment.

There are many existing Big Data frameworks, which can be used for parallel analysis. However, real experiments show that such frameworks have a significant processing and communication overhead. This paper describes a different approach to processing which is based on splitting a stream of flow data into smaller substreams that are being processed in parallel. Naturally, the splitting is the most important and crucial part of the approach because it must prepare independent substreams. The paper explains a witness (a subset of flow data that must remain together) that can be used for the better construction of a flow splitter. The term witness was discussed in more detail in Section 3.4.2.

The paper was written in cooperation with Martin Žádník, who participated as a consultant and a proofreader, and he significantly improved the quality of the text. The idea of the methodology about witnesses and their definitions are the work of the author of this dissertation thesis.

Section 3.4 contains the complete list of definitions and some examples that could not be put into the paper due to the limited space.

Preserving relations in parallel flow data processing

Tomas Cejka[1], Martin Zadnik[2]

[1] CTU in Prague, FIT, Czech Republic, cejkato2@fit.cvut.cz
[2] CESNET, a.l.e., Czech Republic, zadnik@cesnet.cz

Abstract. Network monitoring produces high volume of data that must be analyzed ideally in near real-time to support network security operations. It is possible to process the data using Big Data frameworks, however, such approach requires adaptation or complete redesign of processing tools to get the same results. This paper elaborates on a parallel processing based on splitting a stream of flow records. The goal is to create subsets of traffic that contain enough information for parallel anomaly detection. The paper describes a methodology based on so called witnesses that helps to scale up without any need to modify existing algorithms.

1 Introduction

Common architecture of monitoring large networks contains multiple observation points measured by monitoring probes and a central collector with captured data. This approach creates a global view of the network traffic. In addition, it allows for analysis and detection of global events that are less visible from a local view.

This approach works well on small networks, however, since the network traffic grows, processing all data on one place reaches limits of resources such as memory capacity. In addition, various network events produce data that reach maximal performance of a single machine. Altogether, network monitoring becomes a Big Data processing and some scalable approach must be considered.

As it is described later in Sec. 2, Big Data principles are being studied for last years. However, a general methodology of splitting network data into subsets is missing. The aim of this work is to describe a principle how to split data with respect to internal relations and used processing algorithms in order to analyze balanced data subsets while the needed information still remains together.

The goal is to allow processing huge amounts of flow data using analysis tools that are not designed for a distributed environment. A correct selection of data subsets can improve current mechanisms of data distribution or sampling without loosing information needed by detection algorithms.

2 Related Work

MapReduce was used for network data processing e.g. in [6, 7], however, the authors used a distributed database or a distributed filesystem to store data

files. Paper [4] analyzes IP, TCP and HTTP traffic stored in offline files using Hadoop and MapReduce. Paper [1] analyzes campus network using several types of MapReduce jobs (e.g. measuring volume of traffic per subnet).

Authors of [5] try to use Apache Spark framework with Netmap to extract traffic features for a packet-based detection of different types of DDoS attacks in real-time. The detection uses machine learning methods. The authors rely on a distributed storage and an abstraction of objects called Resilient Distributed Dataset, however, no efficient data splitting is discussed. The paper notes that usage of sampled data produces many false-positives.

Semantic relations in data and possible negative effects of splitting data were mentioned in [3]. The authors present experiments with Hashdoop, an improved Hadoop, that splits data using CRC hashes of src and dst IPs. The authors chose a simple packet counting and ASTUTE algorithm for parallel processing. The splitting based just on IP addresses is a single case in our methodology.

3 Proposed approach

The main requirement is to identify which parts of data must stay together to preserve data relations and which parts can be split into subsets.

A **detection algorithm** can be described as a function with data about network traffic as its input and alerts (detected events) as its output. Generally, the input data is a mixture of benign and malicious traffic. The goal of a detection algorithm is to identify the malicious traffic and to generate an alert that describes the detected event. The algorithm is successful if it observes at least a **minimal subset** of malicious traffic which triggers the alert. Lets call the instance of a minimal subset a **witness**. If a witness gets divided, the malicious traffic is not detectable with the same detection algorithm anymore because there is not enough information for decision. As a result, data can be divided for parallel processing in any way that does not break witnesses.

In practice, there are many different detection algorithms processing the same data to detect various types of malicious traffic. As a result, multiple different witnesses must be preserved at the same time which complicates data splitting.

Data can contain many witnesses that identify the same malicious traffic, while any of them is sufficient for a successful detection. In order to design a data splitter a particular type of witness should be characterized. This kind of characterization describes what data an algorithm analyzes, how the malicious traffic looks like and what is the configuration of an algorithm.

4 Evaluation

To evaluate the witness-based splitting, we analyze data distribution among computation nodes and overall detection results. We need to compare results of a single instance and results of a distributed environment. As the distributed processing generates some alerts multiple times a deduplication based on timestamps, type of events and other information contained in the alerts is necessary.

For the evaluation, we use a NEMEA framework [2] which can be easily run in a single instance as well as in a distributed configuration. There are several detection modules in NEMEA and some of them were presented in our previous work. However, the presented principle can be used with any system that allows to modify an algorithm of data splitting.

The first experiments with splitting flow data with respect to potential witness showed that it is possible to distribute flow data almost uniformly and there is no significant difference between detection results of a single instance and the distributed environment. The measured difference was about 1 % which is caused by inaccurate timing of stream-wise real-time analysis during our experiments.

5 Conclusion

This paper addressed a network traffic analysis in a distributed environment. There are many papers focusing on existing Big Data frameworks but, to our best knowledge, a general methodology of splitting a stream of flow data is missing. This research aims to describe data relations that must be preserved for the parallel analysis. The data relations, types of malicious traffic and used detection algorithms with their parameters define so called witnesses. Since this research is rather a work-in-progress, we have some preliminary results. However, the experiments with real data show that respecting witnesses allow for distributed processing without significant impact on detection results.

As a future work, we are going to explore the principle of witnesses in more detail. Moreover, based on witnesses, an algorithm of real-time reconfiguration of the splitter to scale up the distributed system would be useful.

Acknowledgments This work was supported by the Technology Agency of the Czech Republic under No. TA04010062 *Technology for processing and analysis of network data in big data concept* and grant No. SGS17/212/OHK3/3T/18 funded by Ministry of Education, Youth and Sports of the Czech Republic.

References

1. Bumgardner, V.K., el al.: Scalable Hybrid Stream and Hadoop Network Analysis System. In: Proceedings of the 5th ACM/SPEC ICPE (2014)
2. Cejka, T., et al.: NEMEA: a framework for network traffic analysis. In: Proceedings of CNSM (2016)
3. Fontugne, R., et al.: Hashdoop: A MapReduce framework for network anomaly detection. In: Proceedings of INFOCOM (2014)
4. Ibrahim, L.T., et al.: A study on improvement of internet traffic measurement and analysis using Hadoop system. In: Proceedings of ICEEI (2015)
5. Karimi, A.M., et al.: Distributed network traffic feature extraction for a real-time IDS. In: Proceedings of EIT (2016)
6. Lee, Y., et al.: Toward scalable internet traffic measurement and analysis with hadoop. ACM SIGCOMM Computer Communication Review (2013)
7. Zhang, J., et al.: A Spark-Based DDoS Attack Detection Model in Cloud Services. In: Proceedings of ISPEC (2016)