MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Flow-based Brute-force Attack Detection in Large and High-speed Networks

PH.D. THESIS

Jan Vykopal

Brno, 2013

Declaration

Hereby, I declare that this thesis is my original authorial work. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Advisor: doc. RNDr. Václav Račanský, CSc.

Acknowledgement

There are many people to whom I owe sincere thanks for their various input to my thesis. Foremost, I show my gratitude to my supervisor Václav Račanský for his unwavering support, enthusiasm for network security and guidance throughout the last five years. I also owe thanks to Jiří Novotný, head of the Liberouter project, who introduced me to my supervisor. They both gave me an opportunity to do research on real network data.

Many thanks go to my colleagues at Institute of Computer Science and CESNET. I have had the privilege to work and discuss with many talented and enthusiastic people who have made contributions to my research and work experience. Especially, I thank my colleagues and students who have contributed to the development of software related to this thesis.

Finishing this thesis would not have been possible without the Pomodoro technique developed by Francesco Cirillo. I am very grateful to all people who survived the ticking of a kitchen timer and understood what does "I'm in a Pomodoro" mean.

Finally, I am deeply thankful to my dear parents and beloved fianceé for their sustained patience, support and encouragement.

Abstract

Network-based intrusion detection is traditionally bound to deep packet inspection, i. e. searching for known signatures of attacks in the packet payload. With the rise of connected users, devices and offered services, the speed of computer networks is increasing from hundreds of megabits to tens of gigabits per second. As a result, the traditional approach to the intrusion detection is infeasible and a new approach for high-speed networks based on the concept of network flows emerged. Network flows provide an aggregated view of network traffic, which significantly reduces the amount of data that need to be processed by detection methods.

In this thesis, we focus on flow-based detection of online brute-force and dictionary attacks against network services. These attacks represent a ubiquitous security threat to weak passwords that is often omitted by vendors and developers of existing applications. At present, a typical detection and prevention is done in the login process of the given application, if at all. Although network-based detection is capable to capture even distributed attacks, we are not aware of any network-based detection mechanism that addresses this type of attack in large and high-speed networks. We therefore propose two different flow-based approaches to the brute-force attack detection: signature-based approach, an analogy to the pattern matching in deep packet inspection, and generic similarity-based approach using clustering. To show that flow-based detection is possible with encrypted protocols, we evaluate these two approaches on detection of SSH and RDP attacks. Next, we tackle the problem of lowering the false positives with respect to the attack mitigation. We propose two flow-based methods of identification of the attacker that detect whether a given IP address is used by a single host at the same time. This is particularly important for making decision whether to filter traffic of the address. Injudicious decision may negatively affect benign users. Then we study behaviour of attackers to propose general methods for eliminating the false positives because every detection method is vulnerable to them. We describe the sequential detection using the fact that the attacker very often do network reconnaissance, the cumulative detection that employs more various methods to improve the overall detection and methods that use honeypots and external data sources.

All proposed methods are evaluated on real network traffic of the campus network of Masaryk University. Some of them have been already deployed in routine operation by Computer Security Incident Response Team of Masaryk University, transferred to a university spin-off company and became a part of a successfully sold product.

Keywords

brute-force attack, dictionary attack, network behaviour analysis, flow-based intrusion detection, NetFlow, bidirectional flow, clustering, DBSCAN, network address translation, honeypot, false positive, external data sources

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Similarly to electricity, computer networks are an essential part of the critical infrastructure. Both public and private sectors rely on available and *secure* networks. However, hosts connected to the public Internet are under continuous attacks and private networks are vulnerable to insider attacks. Computer and network security is also related to national security and *cyber warfare*. There are many cases of politically motivated cyber attacks such as denial of service or a website defacement in recent years [35]. While these attacks used to be attributed to professionals, other attacks may be conducted by unskilled activists that promote their political opinions [61]. As a response to a growing number of all sorts of attacks, several countries including USA and European Union members recently issued national cyber strategies[1].

All of these attacks are possible mainly due to the following facts: i) security aspects were omitted in the design phase of applications or protocols (such as in case of the TCP/IP protocol suite), ii) users often do not follow guidelines or best practices (for example, they choose weak passwords), and iii) legal enforcement authorities are not adequately prepared for investigation of cyber crime.

Network attacks can be detected and even prevented by specialized systems deployed as an element of a network infrastructure or application running at networked hosts. The goal of a *network-based intrusion detection* is to identify attacks or malicious behaviour by observing network traffic, preferably in real time. In comparison to a *host-based detection* performed at particular hosts, this approach scales well, is transparent for users (it is not necessary to install any software on hosts), and capable to capture even *distributed attacks* which are becoming more and more popular in these days. Next, it is the only possibility of intrusion detection in large networks without direct access to particular hosts (namely, customers of an Internet service provider or *eduroam*[2] users at university).

Traditional network intrusion detection systems (NIDS) inspect packet payload for known signatures of attacks. However, this is not feasible in high-speed (multigigabit) networks. Other limitations of traditional NIDSs are: i) a high rate of false positives that overwhelm security operators, and ii) an inability to process encrypted traffic.

In contrast, *network behaviour analysis* (or *flow-based intrusion detection*) relies on information and statistics of network flows which are commonly identified by a 5-tuple key consisting of the source and destination IP addresses, the source and destination ports, and the protocol of the network or transport layer. Statistics related to the network flows (such as numbers of transferred packets and bytes) are computed in predefined time windows. As a result, the flow acquisition provides an aggregated view of network traffic. Although flows do not carry any information about the packet payload, they are sufficient for the detection

1. A selection of links to documents on national cyber and information security maintained by NATO Cooperative Cyber Defence Centre of Excellence and is available at `http://www.ccdcoe.org/328.html`.
2. `https://www.eduroam.org/`

of many types of attacks such as denial of service, network scans, worms and botnets [74]. In addition, the use of flows significantly reduces the amount of data that need to be processed by detection methods.

Flow acquisition and storage, two essential parts of flow-based intrusion detection, have been addressed by both academia and industry since 1990s whereas flow data analysis is still in the early phase. Hence, our work is focused on flow-based intrusion detection, particularly detection of online brute-force and dictionary attacks on authentication in large and high-speed networks.

Dictionary and brute-force attacks against weak passwords are serious security threats that are often omitted by vendors and developers of existing applications. A recent report [20] and our operational experience show that these attacks are steady in time: we have been observing these attack in our /16 campus network every day for last few years. Attackers try to break in computer systems that allows remote access to: i) abuse compromised system as a *stepping stone*, which can hide they malicious activities, ii) gain unauthorized access to user's data, or iii) infect other systems by their worm or botnet that use this attack vector for its self-propagation. At present, a typical detection and prevention of brute-force attacks is done in the login process of the given application, if at all. In the following section, we present research questions related to flow-based detection of brute-force attacks and describe our approach to its design and evaluation.

1.1 Research Questions and Approach

In 2008, we started this research since we were not aware of any *network-based* detection mechanism that addressed this type of attack in the context of large, high-speed and heterogeneous networks. However, we believed that brute-force attacks against various network services share similar characteristics since the attackers repeatedly attempt to guess the correct username and password. We therefore asked a question *Is it possible to detect brute-force attacks at flow-level in real-time?* To answer this question, we first propose the flow-level taxonomy of brute-force attacks and reconnaissance probes. Then we propose flow-based detection approaches and evaluate them using the taxonomy in the campus network of Masaryk University. The evaluation in this modern, dynamic and complex network is more realistic than using obsolete and limited data sets which are still commonly used by some researchers.

Next, we attempt to answer a question *What are techniques of lowering false positives of detection methods?* We start by methods of identification of the attacker since we suppose that the majority of detection methods output the IP address of the attacker. However, the address can be used by more than by a single host (user) at the same time. That means that consecutive attack mitigation (such as traffic filtering) may negatively affect benign users. In fact, we propose flow-based NAT detection by adapting existing network-based methods that inspect the packet payload for high-speed networks. Further, we study behaviour of attackers and try to find common attack patterns and scenarios that enable design of generic techniques of lowering false positives. We also study correlation with other detection methods and data sources about ongoing attacks, both internal and external, since it appears as another promising way. The application of all these methods of lowering false positives is demonstrated on the proposed flow-based brute-force detection and, again, using real data from the campus network.

In this thesis, we extensively use honeypots, i. e. network traps that produce, by definition, no false positives. We utilize them in three diverse ways: i) in the design phase of the flow-based detection approaches, ii) in one technique of lowering false positive as an additional data source which contains information solely about attacks, and iii) in the study of attackers' behaviour.

To sum it up, the main research questions of this thesis are:

1. Is it possible to detected brute-force attacks at flow-level in real-time?
2. What are the methods of attacker identification in high-speed networks?
3. Is there any common attack scenario?
4. What are the techniques of lowering false positives of detection methods?

1.2 Thesis Structure

This thesis is divided into seven chapters. Chapter 2 gives background information about all fields related to the thesis: brute-force and dictionary attacks, network behaviour analysis, data sources related to attacks and attackers, clustering analysis and visualization. Chapter 3 summarizes the state of the art in detection of brute-force attacks, detection of Network Address Translation, and processing external sources storing security-related data. Our contributions with respect to the state of the art are presented in Chapter 4. Two novel flow-based approaches to detection of brute-force attacks (published in [87, 88, 86, 89]) are described in Chapter 5 as well as a flow-based taxonomy (published in [84]) of the attacks and probes that is used in design and evaluation of these approaches. Chapter 6 discusses six various methods and techniques of lowering false positives using analysis of attackers' behaviour from both internal and external data sources (relevant publications are [46, 88, 84]). Chapter 7 concludes the thesis and summarizes areas of future work.

Chapter 2

Background

In this chapter, we introduce basic definitions and components used in the whole thesis. First, we start with brute-force and dictionary attacks. Second, we define notions related to network behaviour analysis (NBA), one of the primary approaches to network intrusion detection. Other approaches are comprehensively described in [68]. Then we briefly describe four layers of a NBA system. Third, we introduce available data sources related to attacks and attackers that allow elimination of false positives. Fourth, we focus on cluster analysis that is relevant to the similarity-based approach to brute-force attack detection.Finally, we explain parallel coordinate plots, a visualization technique used in the thesis and software prototype.

2.1 Brute-force and Dictionary Attacks

Both brute-force and dictionary attacks are aimed at a wide-spread knowledge-based authentication. In both cases, attackers suppose that users choose their passwords from a small subset of the full password space, e. g., short passwords, dictionary words, proper names, and lowercase strings [54]. The attackers attempt to login to user accounts by trying possible passwords until they find the correct one. If the attackers use a predefined list of common passwords, they conduct *dictionary* attack, otherwise they systematically search entire space of passwords using *brute-force* attack. In this thesis, we use the term *brute-force attack* for both types of attacks since we cannot distinguish them at flow level because it does not provide any information about the packet payload.

The attacks can be divided to two distinct classes according to attack interactivity:

- online – the attackers can verify whether a password is correct or not only by interacting with the login server,
- offline – attacks that enable the attacker to check all possible passwords without requiring any feedback from the server.

Brute-force attacks could be *simple* or *distributed*. In case of the simple attacks, the attacker uses only one host that sends the authentication requests. In contrast, during distributed attacks, many attackers send relatively small numbers of requests at once. As a result, they are stealthy and harder to detect than simple ones.

Although there are many studies [71, 3, 82] showing that brute-force attacks on SSH is a common type of network attacks, there are cases [91] of attacks on various services and applications. We support this assertion by our own analysis of a new worm (botnet) exploiting default remote access to small office home office devices (e. g., ADSL routers, set-top boxes) around the world [95]. This example shows that brute-force attacks are not limited only to computers, they can hit other Internet-enabled appliances.

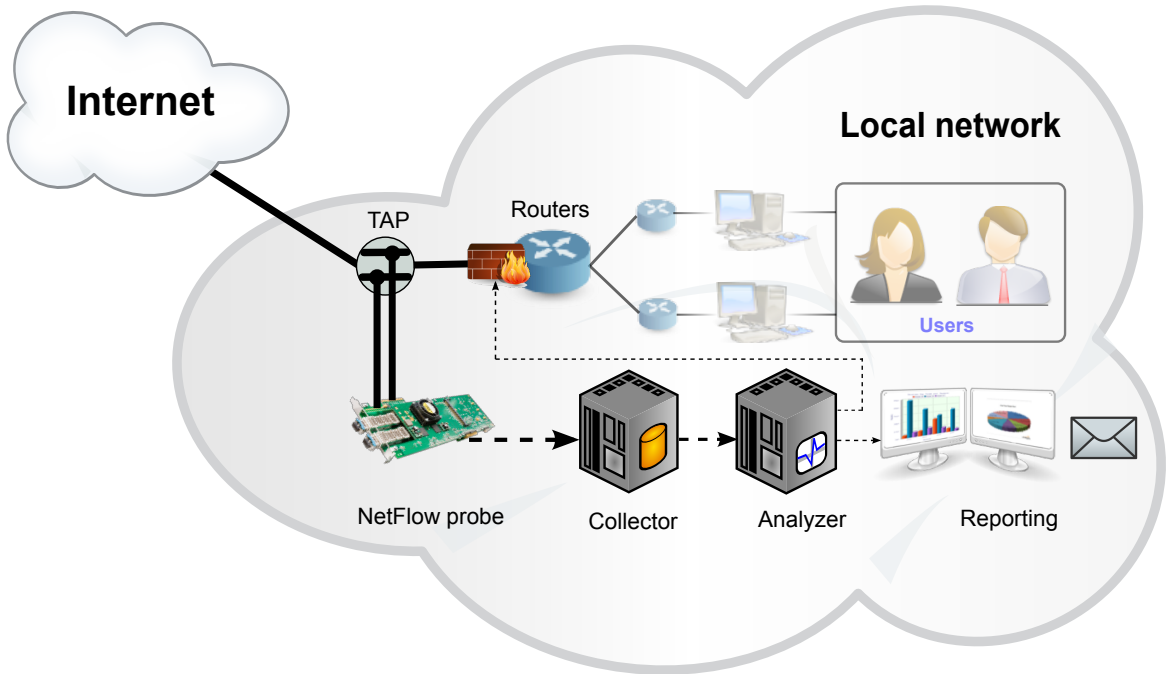

Figure 2.1: Architecture of a network behaviour analysis system and deployment in a network. A line width represents the amount of data transferred between particular layers.

2.2 Network Behaviour Analysis

Scarfone and Mell [68] proposed the following definition of *network behaviour analysis* (NBA):

> A network behaviour analysis system examines network traffic or statistics on network traffic to identify unusual traffic *flows*, such as distributed denial of service (DDoS) attacks, certain forms of malware (e.g., worms, backdoors), and policy violations (e.g., a client system providing network services to other systems).

The definition of a *flow* from RFC 3954 [17] follows:

> A flow is a unidirectional sequence of packets with some common properties that pass through a network device. These collected flows are exported to an external device, the *NetFlow collector*. Network flows are highly granular; for example, flow records include details such as IP addresses, packet and byte counts, timestamps, Type of Service (ToS), application ports, input and output interfaces, etc.

A typical architecture of the NBA system consists of four layers that are described in the following sections. Figure 2.1 depicts the NBA system deployed in a network.

2.2.1 Flow Acquisition

This layer acquires statistics about network flows and export them to a data storage for further analysis. NetFlow is a wide-spread format for IP flow monitoring and export. Originally, it was developed by Cisco Systems as a proprietary format (version 1–8) supported by their routers and switches. The Internet Engineering Task Force (IETF) standardized it as an open protocol (version 9) in 2006 [17]. The most common NetFlow versions are 5 and 9, other versions are not currently widely used.

NetFlow was followed by sFlow [58], another standard for flow monitoring that is supported by various vendors including 3Com, D-Link, Hitachi, Hewlett-Packard and NEC. The key difference between these standards is that sFlow provides only a sampled view of network flows, i. e. some packets are intentionally discarded and thus not considered in the flows acquisition process. Other vendors also develop their proprietary flow monitoring, namely Juniper Networks uses Jflow[1] and Huawei Technology has NetStream[2].

A NetFlow collection and expiration is driven by two values that can be set by user: the *active* and the *inactive (passive) timeout*. The active timeout is applied to regularly export information about long-lasting flows. If the flow has been inactive for the inactive timeout or the end of the flow is detected, flow statistics are exported from the probe to the collector.

Network flows are observed by routers, switches or specialized devices called *probes*. Originally, NetFlow export was implemented as an optional feature of routers and switches and acquired data were used for traffic accounting and billing. High-end devices that handle large amounts of data use *packet sampling* to reduce the flow acquisition overhead. However, packet sampling can negatively influence results of anomaly detection based on such flows. Namely, Mai et al. [52] demonstrated that "sampling distorts various traffic features and degrades the performance of three different port scan detection algorithms in terms of success detection ratio and false positives". Next, Brauckhoff et al. [10] concluded that sampling produces grossly inaccurate estimates of flow counts that makes the Blaster worm invisible at higher sampling rates. As opposed to this *deterministic sampling* – where exactly nth of every n packets is sampled – *random sampling* guarantees that each packet is sampled with a probability of $\frac{1}{n}$. This is secure against an eventual adversary that crafts packets to evade deterministic sampling [33]. Other types of sampling are motivated by decreasing data processing time. For example, Canini et al. [14] introduced *per flow packet sampling* to enable a traffic real-time classification of high-speed network traffic. Besides the sampling, another important factor that influences exploitation of flow data for intrusion detection is a quality of their acquisition and export. Hofstede et al. [39] thus analyzed several measurement artifacts occurring in flow data exported by six different devices, mainly routers.

Acquiring *non-sampled* and *high-quality* flow data in high-speed networks is possible with a standalone probe. The probe is commonly run on a dedicated PC server with a specialized application that captures packets from network interface card, extracts key features of the packets, maintains flow cache and sends expired NetFlow records to another host called *NetFlow collector*. On one hand, some setups in small networks with a load up to hundreds of megabits per second do not need the dedicated server: an exporting applications

1. `http://www.juniper.net/techpubs/software/erx/junose60/swconfig-routing-vol1/html/ip-jflow-stats-config2.html`
2. Technical White Paper for NetStream: `http://www.huawei.com/products/datacomm/pdf/view.do?f=65`

(e. g., *nProbe*[3], *fprobe*[4], *softflowd*[5] or *YAF*[6]) running on a monitored host or middlebox is sufficient. On the other hand, our measurements [85] showed that it is necessary to deploy packet capture acceleration such as *PF_RING*[7] to capture all packets on fully loaded 1 gigabit networks without any loss. For higher speeds, it is necessary to use a hardware-accelerated network interface cards (produced, e. g., by Endace[8], Napatech[9] and Invea-tech[10]) or cards specialized to the NetFlow acquisition (such as those developed in the Liberouter project[11]).

In general, there are two alternatives how to connect the standalone probes to network. Network traffic that should be monitored is: i) forwarded via mirror port of a router/switch to the probe or ii) copied by *test access port* (TAP) – an inline hardware device providing real-time copy of all data passing through the observed link. Zhang and Moore [93], and our experience show that port mirroring performed by routers or switches distort collected traffic in terms of timing difference, packet reordering and losses.

The IETF IPFIX working group[12] puts effort into unification of protocols and applications of IP flow monitoring. RFC 3917 [62] defines requirements for exporting traffic flow information out of routers, middleboxes (such as firewalls) or traffic measurement probes for further processing by applications located on other devices. Next, candidate protocols for a new standard called IP Flow Information Export (IPFIX) were evaluated in RFC 3955 [51]. As a result, NetFlow version 9 was chosen for extension to IPFIX. The main advantage over previous NetFlow versions is that users can flexibly define the flow key. They are not constrained by fixed properties (5-tuple) such as in case of NetFlow version 5. The IPFIX working group published suite of RFCs that specifies basics of this protocol as well as guidelines for implementation [18, 63, 9]. In addition, they extended an unidirectional definition of the flow to bidirectional and introduced the notion of *biflow* including description of an efficient method for exporting biflows information using the IPFIX protocol [83]:

> A Biflow (Bidirectional Flow) is a Flow as defined in the IPFIX Protocol document [18], composed of packets sent in both directions between two endpoints. A Biflow is composed from two Uniflows such that:
>
> 1. the value of each Non-directional Key Field of each Uniflow is identical to its counterpart in the other, and
> 2. the value of each Directional Key Field of each Uniflow is identical to its reverse direction counterpart in the other.
>
> A Biflow contains two non-key fields for each value it represents associated with a single direction or endpoint: one for the forward direction and one for the reverse direction.

3. http://www.ntop.org/nProbe.html
4. http://fprobe.sourceforge.net/
5. http://www.mindrot.org/projects/softflowd/
6. Yet Another Flow sensor: http://aircert.sourceforge.net/yaf/
7. http://www.ntop.org/PF_RING.html
8. http://www.endace.com/dag-network-monitoring-cards.html
9. http://www.napatech.com/products/network_adapters.html
10. http://www.invea-tech.com/products-and-services/overview
11. http://www.liberouter.org
12. http://datatracker.ietf.org/wg/ipfix/charter/

2.2.2 Data Storage

The next layer of the NBA system is responsible for receiving NetFlow records from the previous layer and their storage for further analysis. A device dedicated for this task is called a *NetFlow collector*. It provides software tools for data querying and simple analysis.

In this thesis, we rely on *nfdump*[13], an open-source CLI[14] suite of tools for receiving/replaying, storing and filtering NetFlow data (version 5, 7 and 9). Collected data are stored in proprietary binary files organized in a directory structure. Although this allows easy manipulation with the data, query performance is not satisfactory for large datasets. *NfSen*[15] is a graphical web-based front-end for the nfdump tools. It periodically plots time series of aggregate statistics (numbers of flows, bytes and packets) in the scope of user-defined filters. Moreover, NfSen can be extended by external plug-ins that process collected NetFlow data in various ways.

Besides nfdump, there are also other tools for flow processing. *flow-tools*[16] is an outdated suite of tools similar to nfdump. *SiLK*[17], the System for Internet-Level Knowledge, consists of two parts: the *packing system* is similar to the nfdump tools and supports NetFlow version 5, 9 and IPFIX (collected data are stored as a compressed binary flat files), but the *analysis suite* is a very part of the next layer of NBA. For enterprise users preferring GUI[18] with data visualization, charts and reports are available commercial tools such as *Caligare Flow Inspector*[19] or *IBM AURORA*[20]. A comprehensive list of both free and commercial tools is maintained by SWITCH (Swiss national and research network) on its website[21].

Generally, there are three issues related to collectors: i) NetFlow data are exported via unreliable UDP protocol that is also easier to forge and distort monitored data (hence, TCP and STCP [79] are introduced as other transport protocols in the IPFIX protocol specification), ii) exported NetFlow data transmitted via monitored links increase traffic on the links, so it may be desirable to build a dedicated link between the exporter (probe) and the collector, and iii) due to flow aggregation, the acquired data are available in *near* real-time – there is a trade-off between delay and the desired flow aggregation driven by timeout settings.

2.2.3 Data Analysis

This layer is represented by methods that process stored flow data in terms of intrusion detection and provide output to the *reporting and prevention layer*. If any attack is detected, the output usually contains details of a suspicious flow (at least timestamp and attacker's IP address) and optionally other attack details.

Although many collectors provide some basic flow data analysis functionality, namely simple visualization, lists of top talkers and conversations, port scan detection or other basic statistics computed from raw NetFlow data, there is still a lack of advanced flow-based intrusion detection methods and algorithms. In recent years, new methods for both generic and even very specialized intrusion detection emerged. Even though flows do not carry any

13. `http://nfdump.sourceforge.net/`
14. Command-line interface
15. `http://nfsen.sourceforge.net/`
16. `http://www.splintered.net/sw/flow-tools/`
17. `http://tools.netsa.cert.org/silk/`
18. Graphical user interface
19. `http://www.caligare.com/netflow/caligare_flow_inspector.php`
20. `http://www.zurich.ibm.com/aurora/`
21. `http://www.switch.ch/network/projects/completed/TF-NGN/floma/software.html`

information about payload, they are sufficient for the detection of many types of attacks such as denial of service, network scans, worms and botnets [74].

Complex methods are usually proposed and evaluated by research community on packet traces captured from real networks. Only some of methods address incorporation into the whole system. They often do not consider other layers under and above the data analysis layer. The state of the art related to the brute-force attack detection is described in Chapter 3.

2.2.4 Reporting and Prevention

Finally, the highest layer of the NBA system presents outputs of the data analysis layer to user or sends them to other systems such as Security Information and Event Management (SIEM). If the analysis layer contains more detection methods, this layer should aggregate and even correlate their outputs to avoid overwhelming user with too many events.

For better understanding by users (network operators and administrators), the reported attacks are often presented in some visual form, but effective visualization is still a big challenge. It is traditionally implemented as a virtual *dashboard* that summarizes detected attack in various time perspectives and forms (tables, charts, graphs, listings etc.).

Other ways of reporting are sending a message to: i) a request tracker of CSIRT[22] or ii) to other devices of the network infrastructure. Although there are several standards for exchanging such messages, namely the Intrusion Detection Message Exchange Format [23] or the Incident Object Description Exchange Format [22], most of the messages are still sent via e-mail or published as new rules or signatures for the specific IDS. An automated early warning among different sites is in the beginning of development. An Extensible Format for Email Feedback Reports [72] is intended as a machine-readable replacement for various existing report formats currently used in Internet e-mail.

The accurate detection is a key prerequisite of a (semi)automated *intrusion prevention*. The NBA system, as described, processes a copy of the network traffic. It is not an in-line device so it cannot directly block malicious connections. So the only way to mitigate detected attacks is to send commands to other device of the network infrastructure that is capable of traffic filtering, e. g., firewalls and routers with access control lists or Remotely-Triggered Black Hole Routing [48].

2.2.5 Summary

The network behaviour analysis (NBA) is a near real-time, passive flow-based intrusion detection at the network and transport layer of the TCP/IP network model. It is capable of detecting active attacks such as (D)DoS, worm spreading or port scanning. The flow aggregation lowers the amount of processed data as well as processing time that is crucial for high-speed networks. Acquired and stored flows can be analyzed by various methods ranging from simple specialized heuristics to complex generic statistical methods developed for both backbone and local networks.

The generic NBA system consists of four interfacing layers: flow acquisition, data storage, data analysis, and reporting and prevention. Actually, there are also systems (mainly commercial ones) where is difficult to clearly distinguish separate layers such as *Arbor Peak-*

22. Computer Security Incident Response Team

Flow[23] or *Plixer Scrutinizer*[24]. Some SIEM act as a NBA system as well, namely *QRadar SIEM*[25] process and correlate NetFlow data to other data sources.

2.3 Data Sources Related to Attacks and Attackers

In this section, we introduce other data sources that may be useful for studying attackers behaviour as well as for more accurate detection of an attack. While so-called *honeypots* come with an advantage of no false positives since legitimate users should not access them, external data sources allows us to utilize information that is not available in our network nor directly present in network traffic, i. e. geographical location of an IP address.

2.3.1 Honeypots

There are plenty of definitions, we chose the general definition proposed by Spitzner in 2003: A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource. [77] Generally, we distinguish two classes of honeypots: low-interaction and high-interaction. The former *emulates* operating systems and services, i. e. it is easy to deploy and install, bears a minimal risk because the emulated services control activities of attackers and captures limited amounts of information. In contrast, the latter *are real* operating systems and provides *real* services. That means they can capture substantially more information, including new tools, communications, or attacker keystrokes, can be complex to install or deploy, and pose an increased risk since attackers are provided real operating systems to interact with. [77]

Current state of the art of honeypots from the operational perspective is comprehensively reported in a study by ENISA[26] [28]. Over 30 freely available standalone honeypot solutions were evaluated to create a large inventory of honeypots. The evaluation criteria included quality of collected data, scalability and performance, ease of use and setting up and deployment costs.

In this thesis, we use virtual machines complemented by the low-interaction honeypot *honeyd*[27] that was identified by the study as being most useful and recommended for immediate deployment by security teams.

2.3.2 External Data Sources

We designate all services hosted outside the administered network and providing information about a given global identifier as *external data sources*. Most of them use an IP address of a host/network or URL as the global identifier. The type of provided information varies from source to source, e. g., host name, location of the address; presence in various blacklists, results of intrusion detection or honeynet monitoring.

There are an abundance of publicly available external data sources, several known closed sources and possibly private or undocumented sources serving a limited group of users. Thirty well-recognized sources both public and with restricted access are described and dis-

23. `http://www.arbornetworks.com/`
24. `http://www.plixer.com/products/netflow-sflow/scrutinizer-netflow-sflow.php`
25. `http://www.q1labs.com/products/407/qradar-nsm/`
26. European Network and Information Security Agency works for the EU institutions and Member States.
27. `http://www.honeyd.org/`

cussed in another study by ENISA [27]. In this thesis, we use five types of external data sources:

1. DNS servers,
2. databases of regional Internet registries (RIR),
3. a blacklist,
4. a reputation system,
5. an early warning system.

First, we query the Domain Name System, a distributed database, for a host name assigned to a given IP address. Second, using the WHOIS protocol [21], we search for an autonomous system[28] (AS) that a given address belongs to and country of an organization that registered network containing the address. Third, we search for a presence of an address in a blacklist generated by the OpenBL project[29]. It aggregates reports from geographically distributed hosts that monitor network ports 21 (FTP), 22 (SSH), 110 (POP3), 143 (IMAP), 993 (IMAPS) and 995 (POP3S) for brute-force login attacks as well as scans on ports 80 (HTTP) and 443 (HTTPS) for vulnerable installations of phpMyAdmin and other web applications. Fourth, we query BGP Ranking[30], a reputation system that computes for a given AS number its rank that is based on malicious activities of addresses of the AS and size of the AS. In fact, BGP Ranking aggregates results of detections outputted by other external organizations and services. The formula used for the rank calculation and other details can be found in [25, 26]. Finally, we query Warden[31], a Czech academic early warning system, for all events that contain a given IP address. The events are detected and sent to Warden by organizations (networks) connected to CESNET2, the Czech academic network[32]. The events describe a security incident or threat such as network port scanning, brute-force attacks or access to honeypots.

2.4 Clustering

First of all, we start by the general definition and characteristics of clustering by Tan et al. [80], then we introduce clustering algorithms discussed in this thesis and conclude by a description of software for clustering analysis.

> Cluster analysis groups data objects based only on information found in the data that describes the objects and their relationships. The goal is that the objects within a group be similar (or related) to another and different from (or unrelated to) the objects in other groups. The greater the similarity (or homogeneity) within a group and the greater the greater the difference between groups, the better or more distinct the clustering.

28. An Autonomous System is a collection of connected IP routing prefixes under the control of one or more network operators that presents a common, clearly defined routing policy to the Internet. [36]
29. Formerly known as the SSH blacklist; see `http://www.openbl.org`.
30. `http://bgpranking.circl.lu/`
31. `https://warden.cesnet.cz/`
32. `http://www.ces.net/network/`

2.4.1 Taxonomy

There are several types of clustering that we distinguish: a *partitional clustering* is simply a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset. If we permit clusters to have subclusters, then we obtain a *hierarchical clustering*, which is a set of nested clusters that are organized as a tree. From another perspective, we distinguish between *exclusive*, *overlapping* and *fuzzy* clustering. The first one assign each object to a single cluster whereas the second one is used to reflect the fact that an object can simultaneously belong to more than one group. The third put every object to every cluster with a membership weight between 0 (absolutely does not belong) and 1 (absolutely belongs). The last important characteristics is whether the clustering is *complete* or *partial*. The former assigns every object to a cluster, whereas the latter does not. The motivation for a partial clustering is that some objects in a data set may not belong to well-defined groups since they may represent noise or outliers.

The cluster itself may be defined differently with respect to the application domain, i. e. typical characteristics of analyzed data. A *well-separated* cluster is a set of objects in which each object is closer (or more similar) to every other object in the cluster than to any object not in the cluster. A *prototype-based cluster* is a set of objects in which each object is closer (more similar) to the prototype that defines the cluster than to the prototype of any other cluster. A *density-based* cluster is a dense region of objects that is surrounded by a region of a low density.

2.4.2 Algorithms

Clustering has a long and rich history in a variety of scientific fields. One of the most popular and simple clustering algorithms, K-means, was first published in 1955. Since then thousands of clustering algorithms have been published. An overview of clustering including well known clustering methods, the major challenges and issues in designing clustering algorithms is summarized in [42].

In the following subsections, we introduce the widely used K-means algorithm and DBSCAN that is used further in this thesis. K-means differs from DBSCAN and hence it is mentioned for an easier comparison to DBSCAN.

K-means

This is a prototype-based, partitional clustering technique that attempts to find a user-specified number of clusters (K), which are represented by their centroids, i. e. the mean of a group of points. [80] Even though K-means was first proposed almost 60 years ago, it is still one of the most widely used algorithms for clustering. Ease of implementation, simplicity, efficiency, and empirical success are the main reasons for its popularity. [42] Here, we describe the basic algorithm, its various extensions are introduced in [42].

We first choose K initial centroids, where K is a user-specified parameter, namely, the number of clusters desired. Each point is then assigned to the closest centroid, and each collection of points assigned to a centroid is a cluster. The centroid of each cluster is then updated based on the points assigned to the cluster. We repeat the assignment and update steps until no point changes clusters, or equivalently, until the centroids remain the same. [80]

To illustrate the result of clustering by K-means, we use points in two-dimensional space that represents numbers of transferred packets in both forward and reverse directions in

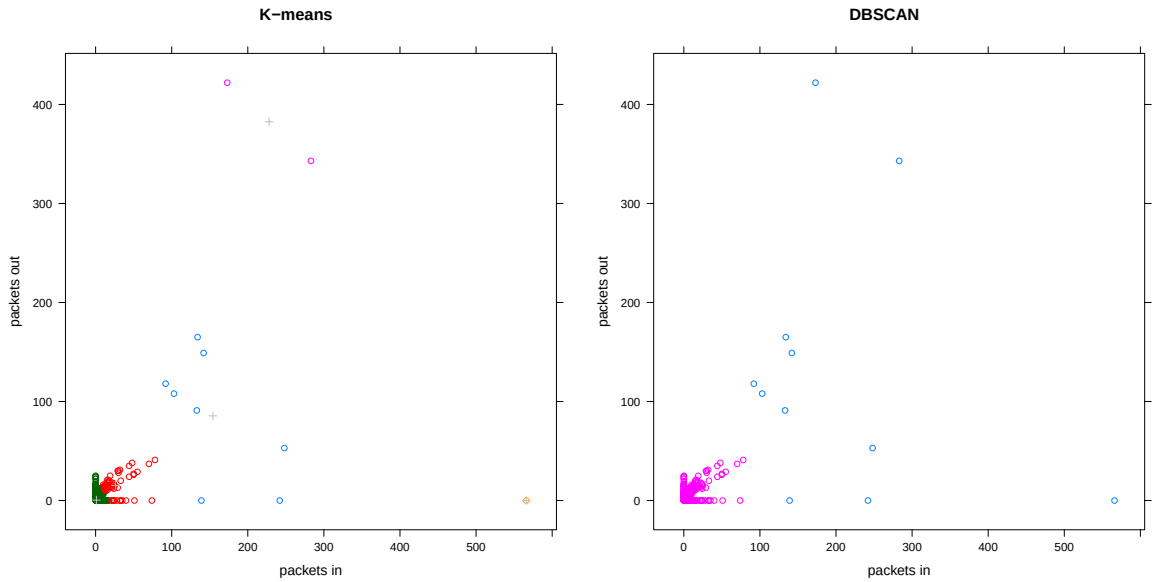

Figure 2.2: Results of clustering biflows captured during a DoS attack by K-means and DBSCAN. Points are individual biflows represented by numbers of transferred packets in both directions.

ID	Colour	Size
1	green	9927
2	red	62
3	blue	8
4	violet	2
5	orange	1

Table 2.1: Number of members of each cluster obtained by K-means

10 000 biflows captured during a DoS attack. We ran K-means with $K = 5$ and therefore the left plot in Figure 2.2 that depicts results contains 5 clusters drawn by different colours. While green and red points near the $[0, 0]$ coordinate form two biggest clusters, violet and blue points form sparser clusters and the single orange point forms the last one cluster. The numbers of points in each cluster is reported in Table 2.1. The five centroids are plotted by grey crosses.

DBSCAN

DBSCAN was proposed by Ester at el. in 1996 [30]. As opposed to K-means, this is a density-based clustering algorithm that produces a partitional clustering, in which the number of clusters is automatically determined by the algorithm. Points in low-density regions are classified as noise and omitted; thus, DBSCAN does not produce a complete clustering. [80]

The key idea is that for each point of a cluster the neighborhood of a given radius Eps has to contain at least a minimum number of points, i. e. the density in the neighborhood has

to exceed the threshold $MinPts$. The shape of a neighborhood is determined by the choice of a distance function for two points p and q, denoted by $dist(p, q)$. For instance, when using the Manhattan distance in 2D space, the shape of the neighborhood is rectangular. [30]

The center-based approach to density allows us to classify a point as being [80]:

- in the interior of a dense region – a *core* point,
- on the edge of a dense region – a *border* point, or
- in a sparsely occupied region – a *noise* or background point.

Any two core points that are close enough – within a distance Eps of one another, i. e. $dist(p, q) \leq Eps$ – are put in the same cluster. Likewise, any border point that is close enough to a core point is put in the same cluster as the core point. Noise points are discarded.

The basic time complexity of DBSCAN is $O(m \times t)$, where m is the number of points and t is time to find points in the Eps-neighborhood. In the worst case, this complexity is $O(m^2)$. However, in low-dimensional spaces, there are data structures, such as k-d trees [8], that allow efficient retrieval of all points within a given distance of a specified point, and the time complexity can be as low as $O(m \log m)$. The space requirement, even for high-dimensional data, is $O(m)$. [80]

We illustrate DBSCAN clustering on the same example as for K-means. DBSCAN was run with $Eps = 50$ and $MinPts = 5$. As result, the right plot in Figure 2.2 depicts one cluster that contains 9989 violet points and 11 blue noise points. In contrast to K-means, the points near the $[0, 0]$ coordinate form one cluster.

2.4.3 Software

Numerous tools for cluster analysis are available in various forms such as a part of statistics and analytics software suites (SAS[33], STATISTICA[34], The R project), specialized data-mining software (KNIME[35], RapidMiner[36], Weka, ELKI), or stand-alone libraries or packages for a large variety of programming languages (Apache Mahout[37], Perl Algorithm::KMeans[38], Python scikit-learn[39]).

In the following subsections, we briefly describe software tools relevant to this thesis, i. e. with respect to the DBSCAN implementation.

R project

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. [64] According to 5th Annual Rexer Analytics Data Miner Survey[40], R continued its rise in 2011 and was being used by 47 % of 1 319 surveyed data miners from over 60 countries.

33. http://www.sas.com/
34. http://www.statsoft.com/
35. http://www.knime.org/
36. http://rapid-i.com/
37. http://mahout.apache.org/
38. http://search.cpan.org/~avikak/Algorithm-KMeans/lib/Algorithm/KMeans.pm
39. http://scikit-learn.org/stable/
40. http://www.rexeranalytics.com/Data-Miner-Survey-Results-2011.html

R comes with thousands of packages available at the Comprehensive R Archive Network (CRAN)[41]. One of the packages, *fpc* [38], provides procedures for clustering including an implementation of DBSCAN algorithm. We used R and this package for the illustration of both K-means and DBSCAN clustering in the previous subsection. Unfortunately, the package provides a simplified version of the original algorithm (no k-d trees used) with time complexity $O(n^2)$.

ELKI

Environment for Developing KDD[42]*-Applications Supported by Index-Structures* (ELKI) has been developing since 2008 by Ludwig Maximilian University of Munich, Germany [1]. ELKI is intended for data mining and database research community and is free for scientific usage. It is written in Java and has a modular architecture that allows to plug in new implementations of data mining algorithms, user-defined distance functions, index structures, visualization and evaluation modules, etc.

The framework is steadily developed and improved. The last stable version 0.5.5 was released in December 2012 [2] and is used in this thesis. This version contains implementation of various clustering algorithms including K-means and DBSCAN, anomaly detection algorithms, spatial index structures such as R-, R*- or M-trees and various evaluation methods. Regarding efficiency, ELKI benefits from using advanced index structures such R*-tree [6]. This is documented by a rigorous performance benchmark of all stable versions of ELKI and other software made by authors [69].

Weka

Similarly to ELKI, Weka is an open-source collection of machine learning algorithms for data mining tasks written in Java. It contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes[43]. Weka has been developing since 1990s by University of Waikato, New Zealand. As opposed to ELKI, it is possible to obtain a commercial licence of Weka for commercial projects.

The DBSCAN implementation was contributed to Weka by authors of ELKI and should not be used as reference implementations for runtime comparisons [70].

2.5 Visualization Using Parallel Coordinate Plots

Parallel coordinate plots are used for visualization points in N-dimensional space R^N using N parallel vertical lines, x_i axes, in two-dimensional space, where N is typically greater than 3. A point C with coordinates $(c_1, c_2, ..., c_N)$ is represented by the polygonal line whose N vertices are at $(i-1, c_i)$ on the x_i-axis for $i = 1, .., N$. In effect, a 1-1 correspondence between points in R^N and planar polygonal lines with vertices on $x_1, x_2, ..., x_n$ is established. [40].

Figure 2.3 shows an example of parallel coordinate plot of four features of biflows acquired during a DOS attack[44]. A biflow is here represented by the source address, the source

41. `http://cran.r-project.org/`
42. KDD stands for Knowledge Discovery in Databases.
43. `http://www.cs.waikato.ac.nz/ml/weka/`
44. This is the same data set used in Subsection 2.4.2.

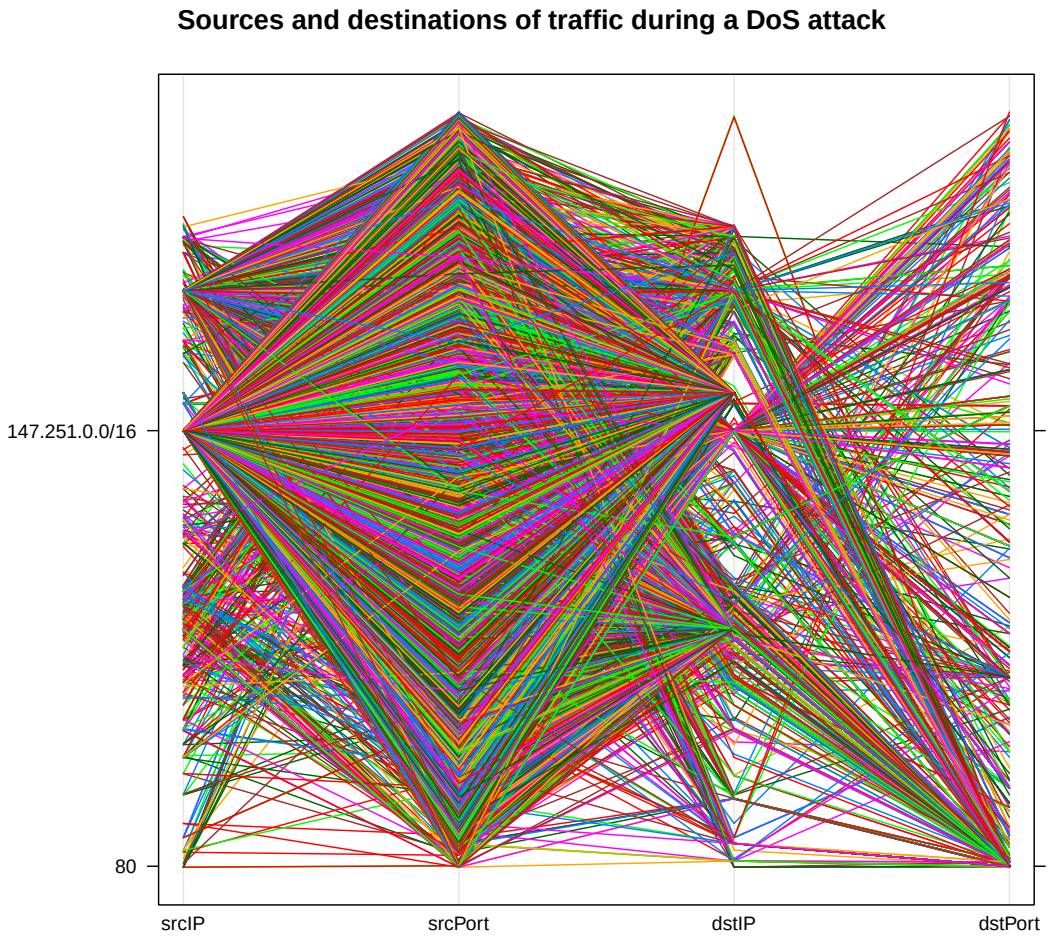

Figure 2.3: Biflows acquired during a DOS attack

port, the destination address and the destination port. While the first and the third parallel axis, x_1 and x_3, depicts /16 source and destination subnets (not individual addresses), respectively, the second and fourth axis, x_2 and x_4, shows individual source and destination ports, respectively. The plot provides an overview of the DOS attack, particularly its sources and destinations. We can clearly see that there are:

- two main sources of biflows, i. e. the majority of vertices of the polygonal lines (biflows) are in two points (two /16 source subnets) on the x_1 axis,
- almost all sources ports used, i. e. vertices of the lines are spread through the entire x_2 axis; this is typical for ephemeral source ports,
- several destinations, i. e. vertices of the lines are in several points (several /16 destination subnets) on the x_3 axis,
- one main destination port, i. e. the vast majority of vertices of the line is in one point (port 80) on the x_4 axis.

In conclusion, the parallel coordinate plots allows to find and understand traffic patterns more efficiently than standard plain text output containing one (bi)flow per one text line.

Chapter 3

State of the Art

First, we summarize available host-based and network-based detection of brute-force attacks with focus on flow-based detection. Second, we report the state of the art in NAT detection that is tightly connected to detailed identification of the attacker and attack resilience. Finally, we briefly introduce existing systems which process external sources storing security-related data.

3.1 Brute-Force Attack Detection

3.1.1 Host-based Detection

Currently, detection and prevention of brute-force attack is done nearly always at host level. Besides common host-based intrusion detection systems, there are other applications available (mainly for Unix operating systems) that parse application log files and report attacks if a number of unsuccessful login attempts exceeds a predefined threshold. Examples of such applications are *fail2ban*[1], *LogSurfer*[2] or *logwatch*[3] with specific rules, or *DenyHosts*[4], a popular SSH log analyzer, that not only block incoming attacks but also report attackers to the central blacklist; furthermore, it can block attackers that were reported by other DenyHosts instances deployed in other networks.

Brute-force attacks often aim at web applications, namely their login interface. Some web applications prevent to break-in attempts by constraints on user passwords, e. g., Twitter does not allow users to choose the most popular weak passwords at registration[5]. Some other (such as Google) try to mitigate brute-force attacks by CAPTCHA[6] after several unsuccessful login attempts. Some services including web applications (such as internet banking) or remote access (such as RDP) are more strict: they lock user account and user has to contact the administrator or authenticate by other channel to unlock the account.

Last but not least, brute-force attacks may be detected using honeypots. For SSH, there are two low-interaction honeypots available: Kojoney[7] and Kippo[8] (inspired by Kojoney). Both honeypots are evaluated in [28]: Kippo was identified as being most useful and recommended for deployment. For RDP, we are not aware of any low-interaction honeypot, there are only guidelines for setting up a virtual RDP server and capturing incoming traffic[9].

---

1. `http://www.fail2ban.org/`; It provides even prevention capabilities.
2. `http://www.crypt.gen.nz/logsurfer/`
3. `http://www.logwatch.org/`
4. `http://denyhosts.sourceforge.net/`
5. `http://www.whatsmypass.com/370-banned-twitter-passwords`
6. Completely Automated Public Turing test to tell Computers and Humans Apart
7. `http://kojoney.sourceforge.net/`
8. `https://code.google.com/p/kippo/`
9. For instance, see `http://samsclass.info/123/proj10/rdp-honeypot.htm`.

To sum it up, host-based detection does not scale well in large networks since maintenance costs grow linearly with the number of hosts. Next, it effectively detects simple attacks but cannot capture stealthy distributed attacks because the detection is done separately for each source IP address.

3.1.2 Network-based Detection

Various *secured* protocols, services and applications became more and more popular in recent years. Besides services such as SSH or RDP, even web applications provided by Google or Facebook are currently accessible over HTTPS. Furthermore, user authentication via secured communication channels is becoming a standard these days. With the rise of encrypted traffic, the traditional approach to network-based intrusion detection is becoming ineffective. Packet payload, which is searched for signatures of known attacks by deep packet inspection, is opaque, only packet headers can be analyzed. Therefore, a flow-based detection is one of the possible ways to deal with encrypted traffic.

A survey of current flow-based intrusion detection is reported in [74]. Although it provides an overview of the state of the art with focus on DoS attacks, scans, worms and botnets, it does not discuss any method of detection of brute-force attacks.

Main ideas of some statistical methods mentioned in the survey were adopted by CAMNEP [66], a collaborative agent-based system that combines:

- the outlier detection algorithm from MINDS[10] [29] that assign the *local outlier factor* (LOF) [11] (i. e. an anomaly score) to each network connection,
- behaviour models for profiling Internet backbone traffic by Xu et al. [90],
- *Origin-Destination flow analysis* by Lakhina et al. [49] aimed at volume anomalies; it uses matrices and *Principal Component Analysis* [43] to separate the space of traffic measurements into normal and anomalous subspaces, and
- detection based on *feature entropy* by Lakhina et al. [50] that observes changes in distributional aspects of packet header fields, namely source and destination addresses and ports.

Although CAMNEP is a general intrusion detection system, we validated that it is capable to detect SSH brute-force attacks that generate at least one hundred flows per minute.

In the following text, we describe in more detail two different methods of flow-based detection of brute-force attacks. While the first method is generic and applicable even to brute-force attack detection (similarly to CAMNEP), the second one is directly tailored to detection of SSH brute-force attacks.

Holt-Winters Forecasting

Brutlag [12] showed that it is possible to use a time series analysis of volume characteristics of network traffic and automatically detect changes between current and forecasted values. The time series analysis is a statistical approach to anomaly detection and has many applications in various domains, e. g., economics or healthcare.

10. Minnesota Intrusion Detection System is, in fact, another example of flow-based intrusion detection system.

The *Holt-Winters method* alias *triple exponential smoothing*, one of the time series analysis method, has been used for forecasting since 1960s. It rests on the premise that the observed time series can be decomposed into three components: a baseline, a linear trend, and a seasonal effect. The algorithm presumes that each of these components evolves over time. This is accomplished by applying exponential smoothing to incrementally update the components. [12]

Let $y_1 ... y_{t-1}, y_t, y_{t+1}, ...$ denote the sequence of values for a time series observed at some fixed temporal interval. Let m denote the period of the seasonal trend (e. g., the number of observations per day). Then the prediction is the sum of the three components:

$$y_{t+1} = a_t + b_t + c_{t+1-m}$$

The update formulae for the three components a, b, c are:

- $a_t = \alpha(y_t - c_{t-m}) + (1 - \alpha)(a_{t-1} + b_{t-1})$, baseline ("intercept"),
- $b_t = \beta(a_t - a_{t-1}) + (1 - \beta)b_{t-1}$, linear trend ("slope"),
- $c_t = \gamma(y_t - a_t) + (1 - \gamma)c_{t-m}$, seasonal trend.

α, β, γ are the adaptation parameters of the algorithm; $0 < \alpha, \beta, \gamma < 1$. Larger values mean the algorithm adapts faster and predictions reflect recent observations in the time series; smaller values means the algorithm adapts slower, placing more weight on the past history of the time series. [12]

We illustrate the method on an example of time series from the campus network. Figure 3.1 shows a decomposition of time series of number of flows (denoted as "observed") in December 2012 to three components: trend, seasonal and irregular. First, the trend component corresponds to a traffic decrement caused by the Christmas holiday in the end of 2012. Second, the season is one day since the time series have a diurnal pattern. Finally, the irregular component contains the residue. Figure 3.2 depicts the same time series (drawn black) with values forecasted by the Holt-Winters method (drawn red). We can see that the forecasting adapts very well to changes in the observed series, i. e. there are no big differences in observed and forecasted values.

Sperotto et al. [75] analyzed time series of various protocols and inspected relevant network traffic. They observed that massive SSH brute-force attacks cause a suspiciously high number of flows, among others. Hence, it follows that the Holt-Winters forecasting applied to suitable time series is capable to detect brute-force attacks.

The Holt-Winters forecasting has been implemented in RRDtool[11] by Brutlag. Gabor Kiss from HUNGARNET[12] patched the NfSen collector and added an aberrant behaviour detection based on the built-in Holt-Winters algorithm of RRDtool. However, this experimental version of NfSen named *NfSen-HW*[13], is no longer developed nor supported. As a consequence, we implemented the similar functionality in a plugin for the NfSen collector[14].

11. The open-source industry standard, high performance data logging and graphing system for time series data. For more details see `http://oss.oetiker.ch/rrdtool/`.
12. HUNGARNET is Hungarian National Research and Education Network.
13. `http://bakacsin.ki.iif.hu/~kissg/project/nfsen-hw/`
14. `http://is.muni.cz/th/208106/fi_m/`

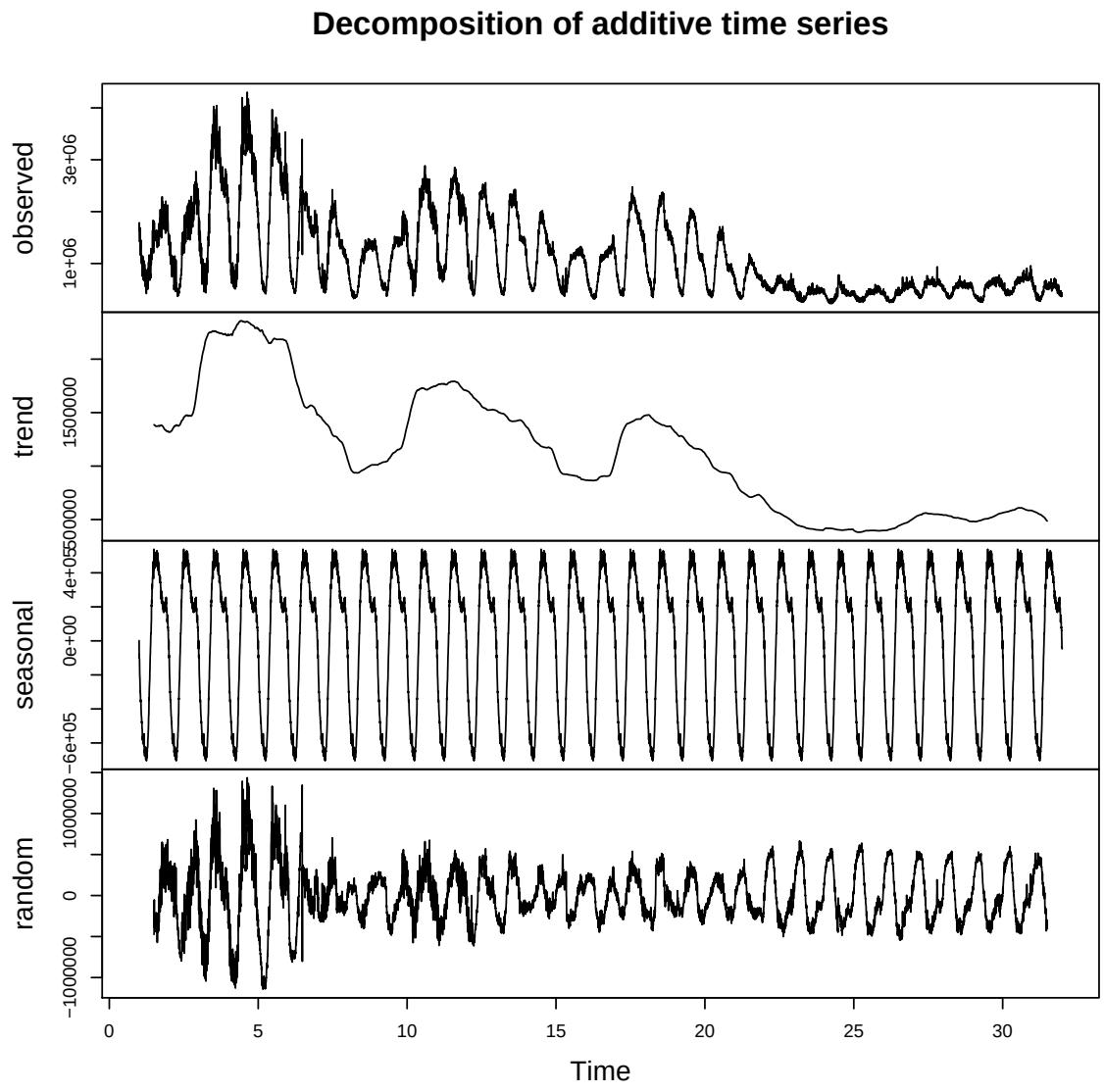

Figure 3.1: Decomposition of time series of numbers of flows in 5-minute time windows in the Masaryk university network in December 2012

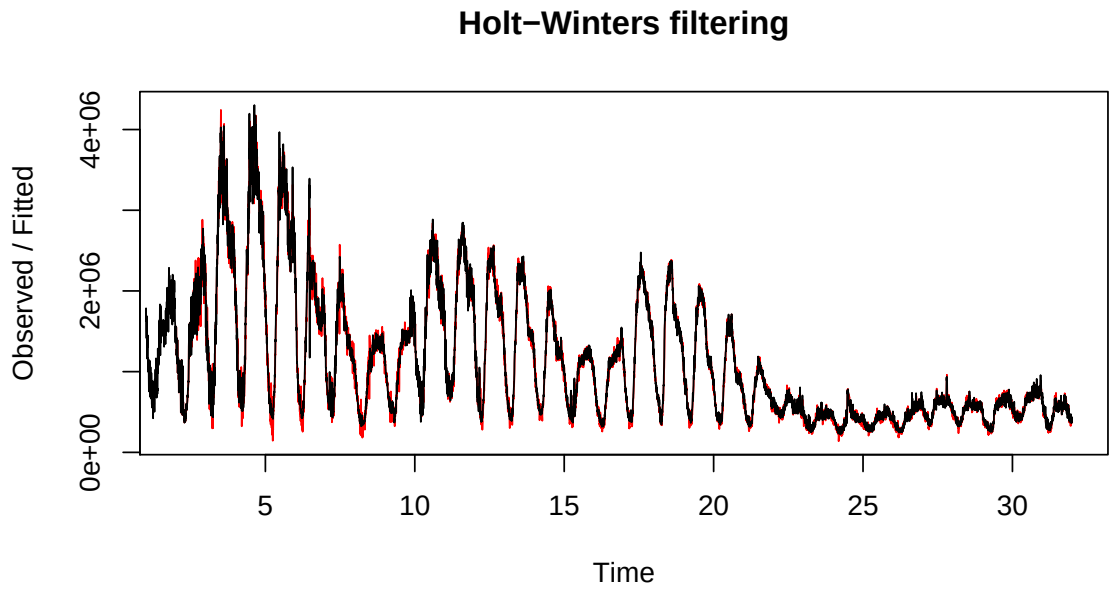

Figure 3.2: Time series of numbers of flows in 5-minute time windows in the Masaryk university network in December 2012 (black) and forecasted values of this time series by the Holt-Winters method (red)

SSHCure

SSHCure is an intrusion detection system which is able to detect SSH intrusion attempts and compromised hosts using flow data. [37] SSHCure detects attacks that consist of three phases: network scanning, brute-force password guessing and die-off phase, i. e. residual traffic between the attacker and compromised host. The detection method stems from the analysis in [76] and identifies the attacks phases by two statistics: packets-per-flow (PPF) and the number of flows per minute. Concrete ranges of the statistics are based on measurements done in 2011 in the campus network of University of Twente, The Netherlands. The authors claims that the evaluation of the detection in their network shown that the number of false positives and false negatives is low. The open-source implementation, plugin for the NfSen collector, is publicly available at SourceForge[15].

3.2 NAT Detection

In this section, we discuss works related to the detection of Network Address Translation (NAT) [78]. There have been published several works how to detect NAT devices in the network. These methods are based on different principles and use different type of data as input, therefore we decided to describe them with respect to the network OSI model [94], which divides particular methods to the distinct layers. There are other possible categorization of existing methods, e. g., passive methods (we observe traffic from the examined host)

15. `http://sourceforge.net/projects/sshcure/`

or active methods (we observe traffic from the targeted host and initiate connections to this host), but for our work the most suitable is the division by the OSI model.

Layer 1 and 2 The host fingerprinting and NAT detection using the lowest layers of the OSI model is represented, for example, by fingerprinting the devices based on their radio signatures in the wireless networks [24]. To use such methods, we need to have an access to the Physical and Data link layer information. Without the physical presence of the monitoring device in the targeted network, we could not use the methods for the analysis, e. g., NAT detection using the MAC address analysis. Therefore, we focus on the methods processing information from the OSI Layer 3 and higher.

Layer 3 The Network layer provides information about Internet Protocol (IP) [59]. Phaal [57] uses the TTL (Time To Live) value from the IP header. The initial TTL values are fixed by each operating system (OS) and corresponding protocol by default. When a packet is forwarded through the network device (router), the TTL value is decreased, as defined in [59]. The detection system presented in [57] assumes the knowledge about inspected network, particularly the hop count, i. e. a number of intermediate routers ("distance") between the observation point and examined host. Therefore, if there are some devices behind the NAT, the hop count differs since it is smaller than it is supposed. This method is easy to implement, but the supposed knowledge of the network infrastructure and a possibility to alter the TTL value at the NAT device are the main limitations.

The IP header's ID field (IP ID) is another indicator for NAT system discovery present at L3 layer of the OSI model. In case of routing with packet fragmentation, this field should be unique for each packet. It is also used for counting hosts behind NAT in [7]. The work is based on the fact that many OSs increment value of the IP ID field permanently, e. g., Windows machines increase this value sequentially. Therefore, it is possible to identify IP ID sequences pointing to various hosts behind the NAT. However, this method cannot be used in the case of OSs that fill the IP ID values with zeros or random numbers, e. g., Linux or BSD.

Layer 4 The Transport layer of the OSI model brings information about the TCP protocol [60]. The key field that is usable for detection of NAT systems is the TCP timestamp value. A method for identifying various hosts behind NAT using this field is presented in [45]. To distinguish various hosts, the authors analyze i) the difference between the current time and TCP timestamp and also ii) the jitter. A similar approach is used in [13]. The main limitation of these method is that the TCP timestamp is not used in Windows OSs, by default.

OS fingerprinting identifies various OS using information not only from Layer 4 such as the TCP initial window size, TCP flags or TCP Initial Sequence Number Selection as described in [31] but also from higher layers.

Layer 7 Concerning higher layers, the main role plays the Application layer. Various artifacts are used for recognizing particular host, namely cookies information or HTTP session IDs. Cohen et al. [19] design an energy function which combines several traffic artifacts introduced from lower layers and add information about HTTP sessions and cookies from Layer 7.

3.3 External Data Sources

The state of the art in exploitation of external data sources is summarized in [44]. This paper by CERT Polska[16] is a follow-up to [27] and includes survey of systems for automated incident processing, where the term *incident* is very generic, i. e. it may denote a full range of security events from "noise" traffic such as network port scanning to severe host compromise. The reviewed systems were:

- Megatron[17], a centralized system built by CERT-SE[18]; it is oriented to batch processing of feeds from external sources and distributing the incident information to the affected organizations.
- Collective Intelligence Framework (CIF)[19] created by REN-ISAC[20]; it collects security-related data from multiple sources and provide mechanisms to effectively query, correlate and share it.
- AbuseHelper[21] developed by CERT-FI, CERT-EE[22] and Clarified Networks; in contrast to Megatron and CIF that use a database centric approach, AbuseHelper is based on distributed agents (bots) that process event streams in real time.
- Network Security Incident eXchange (n6)[23] designed by CERT Polska with a goal to keep the system as simple as possible; it is built around a central repository, a file-oriented archive for all types of collected information.
- an experimental system built by CERT Polska on top of Splunk[24], commercial monitoring and analytics software capable of scaling to large volumes of data.

Another example of using an external data source is DenyHosts mentioned in Subsection 3.1.1. The synchronization mode of DenyHosts allows to send and receive addresses of detected attackers to/from the centralized blacklist. As a result, hosts receiving such information prevent brute-force attacks incoming from these addresses.

3.4 Summary

We introduced the state of the art related to the topic of this thesis. Brute-force attack detection (and prevention) is currently nearly always done at host level. Massive attacks can be detected by generic flow-based intrusion detection systems or statistical methods such as the analysis of time series. The most relevant to this thesis is SSHCure that analyses two flow statistics: packets-per-flow (PPF) and the number of flows per minute. A detailed identification of detected attackers is possible by NAT detection. Currently, there are several methods

16. The Polish national CERT, see `http://www.cert.pl`
17. The software is available under the Apache open source license on e-mail request to cert@cert.se, access to the official web page is restricted.
18. The national CERT of Sweden.
19. `https://code.google.com/p/collective-intelligence-framework/`
20. Research and Education Networking Information Sharing and Analysis Center associates mainly universities in the USA.
21. `http://www.abusehelper.be/`
22. The national CERTs of Finland and Estonia
23. `http://n6.cert.pl/`
24. `http://www.splunk.com`

available that inspect packets and analyses information from Layer 1 to 7. Concerning exploitation of external data sources, we recently have observed efforts to build systems for sharing and correlating various types of security-related data obtained in various networks (organizations).

Chapter 4

Contributions

The main contribution of this thesis is a design, implementation and long-term evaluation of flow-based detection of brute-force attacks in high-speed networks. We elaborate two novel approaches to the detection: signature-based and similarity-based. While the former is an analogy to pattern matching in deep packet inspection, the latter represents anomaly detection. We first use bidirectional flows as an input data and, in case of the similarity-based approach, we first apply clustering for the detection of brute-force attacks. Next, we propose the first taxonomy of brute-force attacks from the perspective of network flows which contributes to clear evaluation of detection methods and provides better understanding of these attacks within the research community. We also tackle the problem of lowering the false positives with respect to the attack mitigation. We propose two flow-based methods of more accurate identification of the attacker. Then we study behaviour of the attackers and design techniques of lowering false positives using other both internal and external data sources.

Flow-based detection of brute-force attacks As opposed to the host-based detection introduced in Chapter 3, both approaches benefit from the fact that the detection is performed at network level and thus can reveal even multiple and distributed attacks. Even though, the network-based detection has other advantages such as scalability, there is almost no such methods specialized to the brute-force attack detection. The most relevant work to this thesis is SSHCure that detects attacks using flow statistics that describe volume and speed of network traffic (see Subsection 3.1.2). Unfortunately, this system does not provide sufficient throughput in our campus network. There are also general flow-based intrusion detection methods and systems which are capable of brute-force attacks (see Subsection 3.1.2 too) but they rely on the fact that traffic generated during the attack affects monitored characteristics. For instance, it is possible to detect brute-force attacks by Holt-Winters forecasting of time series of observed flows if the attacks are massive and thus generate an increased number of flows with respect to the total number, i. e. hundreds of attempted login and passwords per minute. Other attacks, particularly stealthy ones, remain undetected. To overcome this limitation, our approaches searches for particular flows that belong to the attack traffic:

- The signature-based approach searches for flows matching exact values of features and statistics of an observed attack signature.
- The similarity-based approach finds attacks by clustering traffic since it is supposed that brute-force attacks consist of similar flows that differ by various attempted credentials.

Application of bidirectional flows We demonstrate usefulness of biflows in flow-based detection. Unlike state-of-the-art detection methods and systems, the proposed approaches

do not process unidirectional flows but biflows. This is feasible since online brute-force attacks consist of interactive communication between two parties[1]. What is more, the notion of biflow is natural and eases data analysis. If the flow acquisition layer is capable to export biflows directly or the data storage layer pair unidirectional flows to biflows, detection methods at the data analysis layer are then concentrated on the actual detection. That means that they do not need to pair unidirectional flows to biflows and thus save considerable computational resources that would be consumed by this operation at the data analysis layer.

Application of clustering in flow-based detection The similarity-based approach uses cluster analysis to group biflows that are similar in terms of their volume characteristics. Although this data mining technique was already applied in network-based intrusion detection [32, 16], this is one of the first applications in flow-based intrusion detection and in a different way. While other detection methods uses clustering for finding outliers, our approach discards the outliers and further processes only clustered biflows. This determines a type of used clustering algorithm. We apply DBSCAN (see Subsection 2.4.2), a density-based clustering algorithm that produces a partitional clustering.

Flow-based taxonomy of brute-force attacks For a clear evaluation of detection methods and better understanding of the brute-force attacks within the research community, we propose a basic flow-based definition of five types of attacks and probes:

- simple brute-force attack (SBA),
- multiple brute-force attack (MBA)
- distributed brute-force attack (DBA),
- network port scan (NS),
- application scan (AS).

Open-source implementations and technology transfers The prototypes that we develop are available under the 3-clause BSD licence, a permissive free software license[2] (see Appendix A). Some of them has been transferred to a university spin-off company.

Long-term evaluation in a real network The evaluation of proposed methods is done in a real campus network of Masaryk University, Brno, Czech Republic. The network is connected to the Czech academic network by 10 gigabit links and consists of about 15 000 networked hosts with public IP addresses. The network is heterogeneous: it is a mix of various hosts, services and security polices (from permissive to strict). We evaluate our methods in the state-of-the art network for long terms ranging from tens of days to two years. Unfortunately, this is not common. Still, there are many published methods that have been evaluated on obsolete and short-term data sets such as KDD Cup 1999 Data[3] or DARPA Intrusion Detection Data Sets[4] that were extensively criticized, e. g., in [81] or [53]. Although there are

1. In case of other types of attacks, such as TCP SYN flood attacks, the source address of acquired (bi)flows may be forged.
2. See `http://opensource.org/licenses/BSD-3-Clause`.
3. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`
4. `http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/`

first efforts to create data sets for flow-based intrusion detection [73], there is still a lack of them. That means that using data from a real network may be easier and more suitable.

Identification of attackers using extended flow record We propose two methods that process flow records extended by additional packet features and detect whether a given IP address is used only by a single host at the same time. We adapt the method which counts hosts behind NAT [7], and OS fingerprinting technique which uses the initial TCP SYN packet size to distinguish various OSs [31]. As opposed to the state-of-the-art methods described in Section 3.2 that process entire packets, our methods benefits from the extended flow records that provide sufficient data for the detection. Besides the standard 5-tuple, features and statistics, the extended flow record contains also the IP ID and TTL values, and furthermore, TCP packets with only the SYN flag set are not only aggregated to flows but also exported once more. As a result, in high-speed networks, the flow-based methods outperform the existing deep packet inspection methods. We also contributed to the development of other methods of the NAT detection that are described in the thesis of Vojtěch Krmíček [47].

Utility of data sources about attacks and attackers Beside the actual flow-based detection, we employ other data sources both internal and external to design a more efficient detection with respect to consecutive attack mitigation. We suppose that detection is a part of a NBA system, especially the data analysis layer, and there is another layer that processes results of the detection and reports and/or mitigates the detected attacks, for instance, by filtering unwanted traffic from the malicious source. Therefore, we study behaviour of attackers using both host-based and network-based analysis. Next, we study usability of information provided by honeypots and external data sources. Although several early warning systems emerged recently (see Section 3.3), we use rather simple external data sources introduced in Section 2.3.2 since the systems are not well documented. As a result, we proposed four techniques of making detection more reliable using:

- a time factor, i. e. sequential detection,
- other various detection methods, i. e. cumulative detection,
- honeypots,
- external data sources such as blacklists.

The cumulative detection is similar to the principle used in CAMNEP [66]. However, the cumulative detection is not limited to data from our network and/or flow-based methods only, e. g., we can employ external data sources or honeypots.

Visualization Last but not least, we show that suitable visualization of network (bi)flows enables to perceive a relatively large amount of data in one picture. We use parallel coordinate plots (see Section 2.5) for visualization of biflows clustered by the similarity-based approach (for instance, see Figure 5.5). In comparison to the standard plain text output, such visualization helps the user more quickly and easily identify various traffic patterns and characteristics.

Chapter 5

Flow-based Detection of Brute-force Attacks

This chapter answers a question whether it is *possible* to detected brute-force attacks at the flow level in real-time. At first, we propose a flow-based taxonomy of brute-force attacks as well as reconnaissance probes. This taxonomy is used not only in design and evaluation of detection methods but also in a study of behaviour of attackers. Next, we focus on detection of online brute-force attacks alone, particularly a source of the attacks and additional attack details. We describe two flow-based approaches to the detection, evaluate them in a real campus network and discuss their limitations.

5.1 Taxonomy

To the best of our knowledge, there is a lack of taxonomy of the brute-force attacks from the perspective of network flows. However, rigorous definitions of particular attack types are crucial for correct and clear evaluation of detection methods and systems. In this section, we propose a basic flow-level taxonomy of brute-force attacks and probes. We start with the *simple brute-force attack* that is a foundation for the following definitions of *multiple* and *distributed brute-force attacks*. We also distinguish and define two types of probes, the *network port scan* and the *application scan* because they are often mixed up by both researchers and practitioners. The proposed taxonomy can be further developed to reflect various aspects of attacks, e. g., similarly to [55].

The following definitions rely on accurate flow monitoring (use concrete numbers of packets and bytes) so we do not consider the use of packet sampling that would seriously distort the acquired flows.

Simple Brute-force Attack

The elementary type of the brute-force attack is a repetitive interactive communication between the single source and the single destination providing a particular service via the defined network protocol and destination port. The requests (incoming flows) carry attempted credentials and the replies (outgoing flows) information whether the login was successful or not. In contrast to other types of network attacks (e. g., DoS), attackers cannot hide their real address by spoofing the source address and generating malicious unidirectional traffic because they essentially need the reply of the attacked service. Because the attacker uses the same network protocol and only permutes login credentials in consecutive break-in attempts, the amounts of transferred bytes and packets are *similar* within the scope of these attempts. This applies for the service replies as well.

We use the notion of biflow to formally define this basic type of the attack. Let $b = (start\_fwd, start\_rvs, srcIP, dstIP, proto,$
$srcPrt, dstPrt, pkt\_fwd, pkt\_rvs, byt\_fwd, byt\_rvs)$ is a biflow, where

- $start\_fwd$ is a timestamp when the request starts,
- $start\_rvs$ a timestamp when the response starts,
- $srcIP$ is the source IP address,
- $dstIP$ the destination IP address,
- $proto$ used network protocol,
- $srcPrt$ the source TCP/UDP port,
- $dstPrt$ the destination TCP/UDP port,
- pkt_fwd and byt_fwd are amounts of transferred packets and bytes in the forward direction,
- pkt_rvs and byt_rvs are amounts of packets and bytes in the reverse direction.

Then *a simple brute-force attack* is an ordered set of biflows SBA:

$$
\begin{aligned}
&SBA = \{b_i | i \in [1, n]\} \\
&(\forall b_i, b_j \in SBA) : (b_i(proto) = b_j(proto) \wedge b_i(dstPrt) = b_j(dstPrt) \wedge \\
&b_i(srcIP) = b_j(srcIP) \wedge b_i(dstIP) = b_j(dstIP)) \\
&(\forall b_i, b_{i+1} \in SBA) : (b_i(start\_fwd) < b_{i+1}(start\_fwd)) \\
&(\forall b_i, b_j \in SBA, b_i \neq b_j) : (d(b_i, b_j) <= threshold)
\end{aligned}
$$

$srcIP$ is the attacker, $dstIP$ victim, n denotes the *cardinality* of the attack, $d(b_i, b_j)$ is a distance metric function [92] that measures the similarity of biflows, $T = b_n(start\_fwd) - b_1(start\_fwd)$ is a duration of the attack.

Multiple Brute-force Attack

This type of the attack is comprised of simple attacks from one source against two or more destinations at the same time. Formally, the *multiple brute-force attack* MBA is a set of simple attacks SBA:

$$
\begin{aligned}
&MBA = \{SBA_i | i \in [1, n]\} \\
&(\forall SBA_i, SBA_j \in MBA, SBA_i \neq SBA_j) : \\
&(SBA_i(srcIP) = SBA_j(srcIP) \wedge SBA_i(dstIP) \neq SBA_j(dstIP))
\end{aligned}
$$

Distributed Brute-force Attack

To make the attack more stealthy or more efficient, two or more sources send authentication requests to a single destination at the same time. We define the *distributed brute-force attack* DBA as follows:

$$
\begin{aligned}
&DBA = \{SBA_i | i \in [1, n]\} \\
&(\forall SBA_i, SBA_j \in DBA, SBA_i \neq SBA_j) : \\
&(SBA_i(dstIP) = SBA_j(dstIP) \wedge SBA_i(srcIP) \neq SBA_j(srcIP))
\end{aligned}
$$

Network Port Scan

Port scanning is a very popular technique for probing running network services. As opposed to the attacks defined above, such probes need not be answered by target hosts. As a result, we can observe only unidirectional traffic. We define the *network port scan* NS using an unidirectional flow $f = (start, srcIP, dstIP, proto, srcPrt, dstPrt, flags, pkt, byt)$:

$$
\begin{aligned}
&NS = \{f_i | i \in [1, n]\} \\
&(\forall f_i, f_j \in NS) : (f_i(dstIP) \neq f_j(dstIP) \wedge f_i(dstPrt) = f_j(dstPrt) \wedge \\
&f_i(protocol) \in \{TCP, UDP\} \wedge f_i(byt)/f_i(pkt) \in [min, max] \wedge \\
&f_i(flags) \in SCAN\_TYPES) \\
&(\forall f_i, f_{i+1} \in NS) : (f_i(start) < f_{i+1}(start) \wedge f_{i+1}(start) - f_i(start) <= t_{diff})
\end{aligned}
$$

$f(bytes)/f(packets)$ represents the value of *bytes per packet* that is known for particular scan techniques. $SCAN\_TYPES$ defines combinations of sent TCP flags used for scanning (it is an empty set for UDP scans) and t_{diff} the maximum time difference between two consecutive flows.

For example, TCP SYN scan is characterized by i) the length of the very first TCP packet that depends on the used operating system (typically from 48 to 64 B) and ii) settings of the single TCP flag (SYN). Note, we generally cannot distinguish between an unsuccessful TCP connection establishment and TCP SYN scanning in terms of the definition above.

Application Scan

Another type of probing is the *application scan*. It is an interactive communication at the application layer (as opposed to the network port scan at the transport layer). The main difference between the network port scan and the application one is that the former is not "visible" in the application log files whereas the latter is. Attackers may use this scan to confirm that an expected service is virtually up and running at the given network port. At flow level, it can be understood as a special type of the single brute-force attack, which is comprised of a small number of biflows. Opposed to the SBA, these application scan biflows are typically formed with distinct amounts of transferred bytes and packets:

$$
\begin{aligned}
&AS = \{SBA_i | i \in < 1, n >\} \\
&n < small\_number
\end{aligned}
$$

The requirement of the distinct amount of transferred traffic can by adjusted by the suitable distance metric function of SBA_i.

5.2 Finding an Attacker

In this section, we present two flow-based approaches to the detection of online brute-force attacks. While the first approach introduces *flow-based signatures* that are used for a description of traffic generated during brute-force attacks, the second one generally searches for *similar* traffic using clustering algorithms. The design and implementation of the approaches takes account of their deployment in a real large and high-speed network. In evaluation of both approaches, we focus on attacks on SSH and RDP, widely used and even

attacked services. What is more, we choose these secured services because they would examine not only whether it is possible to detect the brute-force attacks at the flow-level but also the detection capabilities with respect to encrypted traffic. A comparison of both approaches with each other and to the state of the art, and discussion of their limitations concludes this section.

5.2.1 Signature-based Approach

Similarly to pattern matching in deep packet inspection, signatures can be used in flow-based intrusion detection too. The flow-based signatures describe network traffic by specific values or ranges of values of flow features and computed statistics. The signatures are then searched in acquired flows and the source of this traffic is designated as an attacker. This detection is done in separate time windows, typically when exported flows are sent from collecting process to the metering process. In contrast to the time series analysis, this simple approach does not consider changes of the monitored traffic in time.

Selection of Signature Features and Statistics

Concerning brute-force attacks, we propose to use features and statistics describing both requests and replies thanks to the interactive nature of the attacks. Using both direction of communication in the signature we obtain just the interactive traffic and make this approach more precise. The proposed signature stems from the definitions of brute-force attacks presented in Section 5.1 and is generally based on the following six flow features and statistics:

1. common service port number,
2. client (ephemeral) port number range,
3. TCP flags[1],
4. flow duration,
5. number of transferred packets,
6. number of transferred bytes.

First, the port number defines which service is subject of interest, i. e. the detection is limited to the services running on a single server port. This is sufficient because the most popular attacked services such as SSH, Telnet, RDP or web applications using HTTP or HTTPS are run only on one and mostly well-known network ports such as TCP/22, TCP/23, TCP/3389 or TCP/80 or TCP/443, respectively. Second, the attacker is a client establishing a connection with the attacked server. The client port used by the attacker is picked by the TCP/IP stack of attacker's operating system from a pool of available ports. Although various implementations of the TCP/IP stack use different ephemeral ports ranges, the client port is not usually out of the range from 1024 to 65 535. The client port is present in the incoming flows (requests) as the source port and in the outgoing flows (replies) as the destination port. Third, the vast majority of attacked services uses the TCP protocol. Hence, the TCP flags can be employed in the signature. Only successfully established connections can rely the brute-force attack so we can use flags to distinguish attacks from probes such as port scans. Fourth, both

1. In case of other protocols than TCP, flags are unused.

login attempts and server replies form short-lived flows. To distinguish login attempts from other traffic, a flow duration can be specified. Finally, login attempts and server replies are of specific size, i. e. consist of the specific number of packets relaying the specific number of bytes. The use of these statistics in the signatures supposes that volume of traffic transferred during the login attempts and normal operation of a particular protocol differs.

Method

The process of creating signatures is laborious and tedious. Generally, we start with host-based analysis of brute-force attacks against a particular service by inspection of service log and network traffic. Then we derive the signature and validate it at the flow-level in the campus network comprising both production hosts and honeypots. We employ the production hosts to test various scenarios of the usage of legitimate service to reveal possible false positives. Honeypots are used in simulated brute-force attacks to reveal especially false negatives because all incoming traffic to a honeypot is assumed malicious by the definition.

Biflows that match all features and statistics of the signature are analyzed to assess type and originators of the attack in the final stage of the method. According to the taxonomy presented in Section 5.1, a number of unique sources and destinations determine different attack types. First of all, unique sources in biflows matching the signature are computed. Next, for each source address is determined whether the number or biflows reaches a predefined threshold for simple brute-force attack (SBA). The identified SBAs are then aggregated to distributed or multiple brute-force attacks (DBA and MBA, respectively) if it is possible. Finally, the found type of attacks, addresses of attacker(s) and victim(s) and other details about the attacks (such as its duration or number of attacked services) are produced as a result.

Examples of signatures

In the following text, we present examples of signatures for several popular services that we derived and validated in recent years. Some statistics used in these signatures depend on particular settings of the flow monitoring, especially the size of the monitoring time window and values of both active and passive timeouts. For example, the number of transferred packets of login attempts may significantly change if the active timeout value is less than the duration of these attempts.

Signature for attacks on SSH The SSH signature is crafted to reveal login attempts and replies of the server. During SSH password authentication, the user supplies a password to the SSH client, which the client transmits securely to the server over the encrypted connection. The server then checks that the given password is acceptable for the target account, and allows the connection if so [5].

To create the SSH signature, we set up a new SSH server with a public IP address in the campus network and monitored server authentication log and network flows for 30 consecutive days. We encountered 911 login attempts from 15 IP different addresses in total, and focused on this traffic in the flow-based analysis. As a result, we derived the following signature:

1. The well-known TCP port number of SSH servers is 22.

2. The ephemeral ports used by the attacker lies in the range of 1024 to 65 535.
3. Biflows carry the TCP SYN and TCP ACK flags (at least).
4. Flow duration is up to 5 seconds.
5. The number of transferred packets in both directions lies in the range of 10 to 30.
6. The number of transferred bytes in both directions lies in the range of 1400 to 5000.

Next, we analysed network traffic of various applications that utilize the SSH protocol to find out possible false positives. We performed the following scenarios in both Linux and Microsoft Windows operating systems:

- remote login via *ssh*[2] and consequent execution of two commands,
- remote login via *putty* 0.60 and consequent execution of two commands,
- remote copy of a large file (104.1 MB) via *scp*,
- remote copy of 835 files in 315 directories (5.3 MB) via *scp*,
- remote copy of 786 files in 192 directories (1.5 MB) via *WinSCP* 4.0.6[3] in SCP and SFTP mode,
- download a large file (104.1 MB) via *sftp*,
- backup 8000 files in 666 directories via *rsync* 2.6.9.

None of these scenarios generated traffic fully matching to the signature derived from real brute-force attacks. In a nutshell, these scenarios did not reveal any false positives.

Finally, we were validating the signature in a honeypot environment depicted in Figure 5.1 for 23 days. We deployed five high-interaction honeypots running on the VMware Server platform. The guests were running Ubuntu 8.10 Server Edition with a patched OpenSSH 5.1p1 server. Each guest machine provides ten user accounts that were reachable from the whole Internet. We chose user names and passwords that we spot during our host-based analysis and in other studies [65, 3]. The SSH daemon was listening to the TCP port 22. Any other services and daemons were disabled. At the host level, we totally observed 65 SBAs each comprising more than 20 attempts. All these SBAs were also detected by the flow-based signature (threshold set to 20). To conclude, this measurement did not reveal any false negatives too.

Signature for attacks on RDP

1. The well-known TCP port number of RDP servers is 3389.
2. The ephemeral ports used by the attacker lies in the range of 1024 to 65 535.
3. Biflows carry the TCP SYN and TCP ACK flags (at least).

2. Version OpenSSH_4.7p1 Debian-8ubuntu1.2, OpenSSL 0.9.8g 19 Oct 2007. Versions of *scp* and *sftp* were the same.
3. Build 358

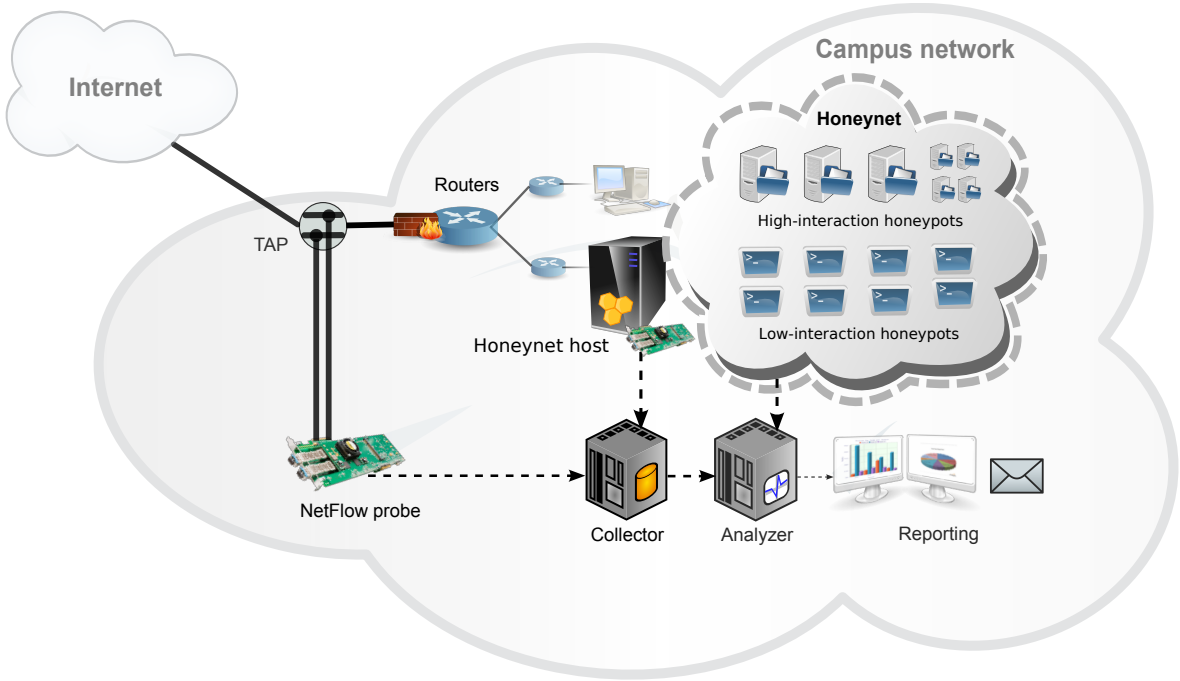

Figure 5.1: Honeynet in the campus network of Masaryk University

4. Flow duration is up to 5 seconds.

5. The number of incoming packets and outgoing packets lies in the ranges of 28 to 69 and 41 to 189, respectively.

6. The number of transferred bytes to and from the network lies in the ranges of 2700 to 8000 and 4000 to 17 999, respectively.

Signature for attacks on LDAP

1. The well-known TCP port number of RDP servers is 389.

2. The ephemeral ports used by the attacker lies in the range of 1024 to 65 535.

3. Biflows carry the TCP SYN and TCP ACK flags (at least).

4. Flow duration is greater than 20 seconds.

5. No limit for the number of transferred packets.

6. The number of transferred bytes in both directions is less than 650.

Implementation

Although the core of the implementation is signature matching itself, there are also other factors that significantly influence detection efficiency.

Preprocessing The signatures use the well-known port so it is not necessary to process all traffic but only traffic originated and destined to this port. This pre-filtering considerably lower down the amount of traffic for signature searching. For example, the flow rate of the traffic in the campus network of Masaryk University was on average 5000 flows per second in peak hours including 15 flows per second for SSH traffic and 15 flows per second for RDP traffic.

Biflow aggregation Implementation of the signature-based detection also depends on the capability of the monitoring infrastructure. The signatures are based on biflows but network flows are traditionally considered as unidirectional. There are generally four possible approaches to obtain biflows:

1. aggregate the acquired NetFlow data by the detection application,
2. aggregate the acquired NetFlow data at the collector after data storage,
3. aggregate the acquired NetFlow data during their collection,
4. acquire biflows directly by the metering process.

Each of these approaches takes place in different layers of the NBA system and gradually put more and more requirements to the monitoring infrastructure. The first approach uses the traditional infrastructure based on NetFlow exporters and collectors storing unidirectional flows. Therefore, the biflow aggregation has to be done by the detection application before the actual detection. The advantage of this approach is no need to modify current monitoring infrastructure. However, this is the least efficient approach because all flows are again loaded from the storage to the memory and a flow pairing algorithm has to find corresponding flows in the whole batch of flows in the processed time window. In addition, every other application processing biflows has to obtain them independently again. We used this approach at the beginning of our research because there was no support for biflows in the probes or collectors. The second approach is very similar to the first one but the biflow aggregation is done by the collector application after the storing unidirectional flows. This approach uses the nfdump tool since November 2009 (version 1.6.1). The more advanced is to pair flows before their storage at the collector, especially during their collection. The unidirectional flows are thus efficiently processed in memory. The main drawback is the need of modified collector capable of this feature. More details can be found in the thesis of Vojtěch Krmíček [47]. To sum it up, the biflow aggregation is computationally expensive at the collector layer. The most efficient and accurate approach is to move this task to the earliest stage of the NBA, i. e. export biflows directly from the metering process at the time of their origin. From the operational point of view, a deployment of both exporters and collectors that support biflows is required. Although biflows export using IPFIX [83] is standardized since January 2008, there is still a lack of applications and systems that support it. Yet Another Flowmeter [4] is one of a few probes that processes packet data into bidirectional flow records that can be used as input to an IPFIX collecting process.

Signature matching Our implementation utilizes aggregation and filtering capabilities of the nfdump tool at the collector. The tool aggregates preprocessed unidirectional NetFlow

4. `http://tools.netsa.cert.org/yaf/`

data to biflows after storage and filter out biflows that do not match the signature. As opposed to the advanced matching using a decision tree [87], it is straightforward and easy to implement.

Evaluation

A long-term evaluation in a real environment shows steadiness and scalability of the developed signatures. Although each signature is tested during the process of its development in various operational scenarios including simulated attacks, this may not capture possible changes in behaviour of attackers, the protocol specification, its implementation or usage. For this reason, we have deployed our simple signatures for SSH and RDP attacks in our campus network. It consists of about 15 000 networked hosts with public IP addresses including hundreds of SSH and RDP servers. The network is open, i. e. with no strict firewall rules. As a result, it attracts attackers' attention.

SSH The signature for SSH attacks has been deployed in 2010. The signature itself matched 54 265 326 biflows of 35 980 different sources from November 13, 2010 to November 13, 2012. The number of SBA essentially depends of a predefined threshold. Its settings determines the sensitivity of the detection, i. e. the number of false positives and false negatives. The influence of the threshold to the number of SBAs is depicted in Table 5.1.

Threshold	Total SSH attacks	Outgoing attacks	Incoming attacks
1	2 044 971	727 725	1 317 246
10	925 471	349 816	575 655
20	550 296	155 153	395 143
30	478 105	144 720	333 385
40	416 002	135 711	280 291
50	362 672	130 790	231 882
100	114 783	41 600	73 183
200	39 483	23 382	16 101
300	28 309	16 631	11 678
400	4424	496	3928
500	2921	313	2608
600	2078	81	1997
700	1631	58	1573
800	185	27	158
900	22	20	2
1000	10	9	1

Table 5.1: Influence of the threshold to the number of simple brute-force attacks on SSH

We set the threshold to a certain value and estimate both false positives and true positives. We decided to set the value to 10 because we believe that this value should separate several unsuccessful attempts made by humans from machine-made attacks and even capture stealthy attacks that do not generate an anomalous amount of traffic. During the 2-year evaluation period, it is absolutely not possible to obtain ground truth due to the enormous amount of traffic, so we can only estimate the false and true positives by inspection of hosts from our network or their network traffic. This way we identified totally 307 sources of traffic from our network matched by the signature. Two hosts were independently reported by

other means so we believe that they actually conducted the detected attacks. Concerning the rest of these hosts, we have no evidence of their malicious behaviour from other sources therefore we consider them benign. These false positives were caused mainly by hundreds of hosts connected to a grid and then by few Nagios servers. Both type of hosts generated traffic that was matched by the signatures. Although we consider this false positive rate negligible, it is crucial for automated attacks mitigation (i. e. blocking the source) to eliminate all false positives. Hence, several such methods are proposed in Section 6.3.

We also evaluate performance of our implementation running at commercial off-the-shelf (COTS) hardware[5] during 26 days in a time period from December 7, 2012 to January 2, 2013. The average processing time of a 5-minute time window was 208 milliseconds with the standard deviation of 207 milliseconds. Table 5.2 shows top 10 longest processing times. These results indicate that our implementation greatly benefits from the data preprocessing and even the worst-case runtime is very short.

Total flows	SSH flows	Attackers	Biflows	Time [s]
958 368	2576	12	98	7.520
510 461	1180	6	39	5.634
1 223 142	3064	11	181	5.494
963 370	1192	8	39	4.857
369 721	1872	7	59	3.425
663 695	1930	9	53	3.413
271 311	7106	9	2810	3.347
724 389	83 152	10	5002	3.250
378 379	3798	10	86	3.012
342 699	8159	7	35	2.845

Table 5.2: Top 10 longest processing times of the detection in a 26-day period

RDP The signature for RDP attacks have been deployed in 2012. From our experience with false positives of SSH attack detection, we limit the matching to attacks coming to the campus network. The signature for RDP attacks matched 3 425 422 biflows of 22 241 different sources outside campus network from September 11, 2012 to December 31, 2012. The influence of the threshold to the number of SBAs is depicted in Table 5.3. We can see a trend similar to the SSH threshold with a slightly less prevalence of attacks comprising more than a few hundreds of biflows.

Discussion

In conclusion, the signature-based approach is very straightforward and simple, but for operational use it is necessary to employ other data sources supporting or contradicting its result. Methods of lowering false positives are described in more detail in Section 6.3. Performance of the signature-based detection is very good because it benefits from the data preprocessing at the collector. However, this limits detection to the traffic destined to a well-known network port. Brute-force attacks on a service running on a different port are thus missed. If the attackers know the signature and the used threshold for simple brute-force

5. Server Dell PowerEdge R300 with quad core CPU Intel Xeon X3353 at 2.66 GHz, 4 GB RAM, 130 GB RAID

Threshold	Incoming RDP attacks
1	209 378
10	131 414
20	61 985
30	22 943
40	10 197
50	5642
100	1253
200	136
300	23
400	2
500	0

Table 5.3: Influence of the threshold to the number of simple brute-force attacks on RDP

attacks (SBA), they can stay under the radar by changing the content (and thus the volume) and the rate of outgoing traffic.

5.2.2 Similarity-based Approach

Deriving signatures of break-in attempts to various services is a time-consuming process. Existing signatures needs maintenance as tools and systems generating monitored traffic are evolving and traffic patterns are changing. Additionally, zero-day attacks are not recognized. We attempt to address these issues by searching for *similar* flows instead of matching specific signatures. We believe that the similarity of traffic can point to machine-generated traffic, namely brute-force attacks.

Method

This approach is an extension of the previous one. It is based on clustering of captured biflows to separate groups of similar ones. The main idea is that biflows representing malicious traffic are grouped in clusters whereas biflows representing benign traffic are not similar, therefore they forms clusters with negligible number of members.

The whole detection consists of five phases:

1. extraction of points from biflows,
2. scaling of extracted points,
3. clustering of these points based on chosen distance function,
4. inspection of found clusters and corresponding original biflows,
5. determination of attack type and scale.

The detection is done for all biflows falling into a time window[6] of predefined size.

The method outputs types of the attacks based on the taxonomy presented in Section 5.1, number of biflows that form the attacks, IP addresses of the attacker(s) and victim(s) and timestamp of the first biflow of the attacks.

6. This implies that biflows are clustered by default by their timestamps.

Points extraction Before the actual clustering of biflows, we need to extract and normalize their features and statistics to form a points that will be passed at the input of a clustering algorithm. These points will represent original biflows in further phases of the detection.

In the beginning, we select appropriate features and statistics that describe a login attempt. We discuss all six features and statistics used in the signature-based approach (see Section 5.2.1). First, if the detection is performed on one selected protocol, there is no need to extract the destination port number[7] since all points would have the same value in one dimension that corresponds to the destination port. Second, client ports are mostly chosen randomly therefore this feature is not a good candidate with respect to the similarity. Third, for protocols using TCP, we can consider TCP flags if there is a chance that they are different for attack and benign traffic. Fourth, flow duration depends on load of both the attacker and the victim. If we use this feature, it can disturb clusters containing points that are similar in other dimensions. Finally, we have already showed that the numbers of transferred packets and bytes differ for login attempts and other traffic. Moreover, we believe that the numbers are similar in case of the login attempts. We intentionally do not discuss source and destination IP addresses because generally they can disturb clustering. For example, consider similarity of source addresses in the multiple brute-force attack and the single brute-force attack. The former exhibits more different addresses and the latter exactly one address. The IP addresses are thus used in the inspection phase. To sum it up, the most important biflow characteristics with respect to the similarity are the numbers of transferred packets and bytes.

We illustrate the use of the volume characteristics by an example. Let the point p_{id} represents a biflow in four-dimensional space as follows: $p_{id} = (pkt\_fwd, pkt\_rvs, byt\_fwd, byt\_rvs)$, where

- pkt_fwd and byt_fwd are amounts of transferred packets and bytes in the forward direction,
- pkt_rvs and byt_rvs amounts of packets and bytes transferred in the reverse direction,
- id is a flow identification used in further processing.

Figure 5.2 shows three set of points using parallel coordinate plots[8]. The first plot depicts points representing all 555 SSH biflows in a time window of one minute on December 17, 2012. The second one shows 466 points of an isolated multiple brute-force attack from one IP address. And the last plot 89 points of all biflows with the exception of the attack-related. We can see that the attack traffic is distinguishable from other traffic.

Points scaling Once we selected suitable features and statistics, we should scale them if their values are on different scales of magnitude. We assume that the similarity of the points is measured by distance metric function that is used by a clustering algorithm. But the distance function generally expects that all dimensions (i. e. features and statistics discussed above) have the same scale. For instance, if we use the numbers of incoming and outgoing packets and bytes as in the example above, we should take into the account that each IPv4 datagram contain more than 28 bytes by the definition. A disproportion between the numbers of packets and bytes is apparent in Figure 5.2 that is scaled by the numbers of bytes. As

7. We consider protocols using one destination port only.
8. This type of visualization is introduced in Section 2.5.

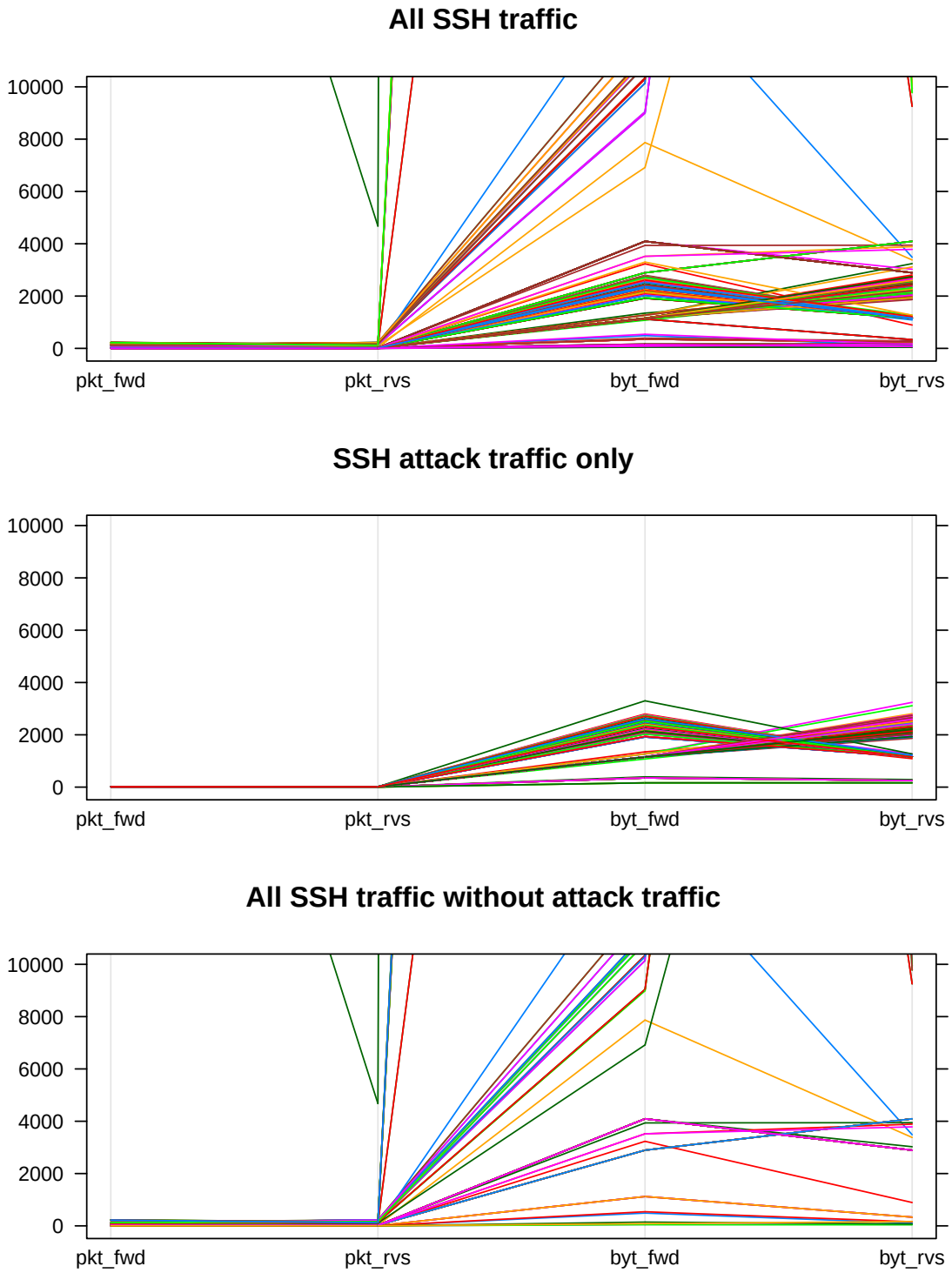

Figure 5.2: SSH traffic represented by points in four-dimensional space

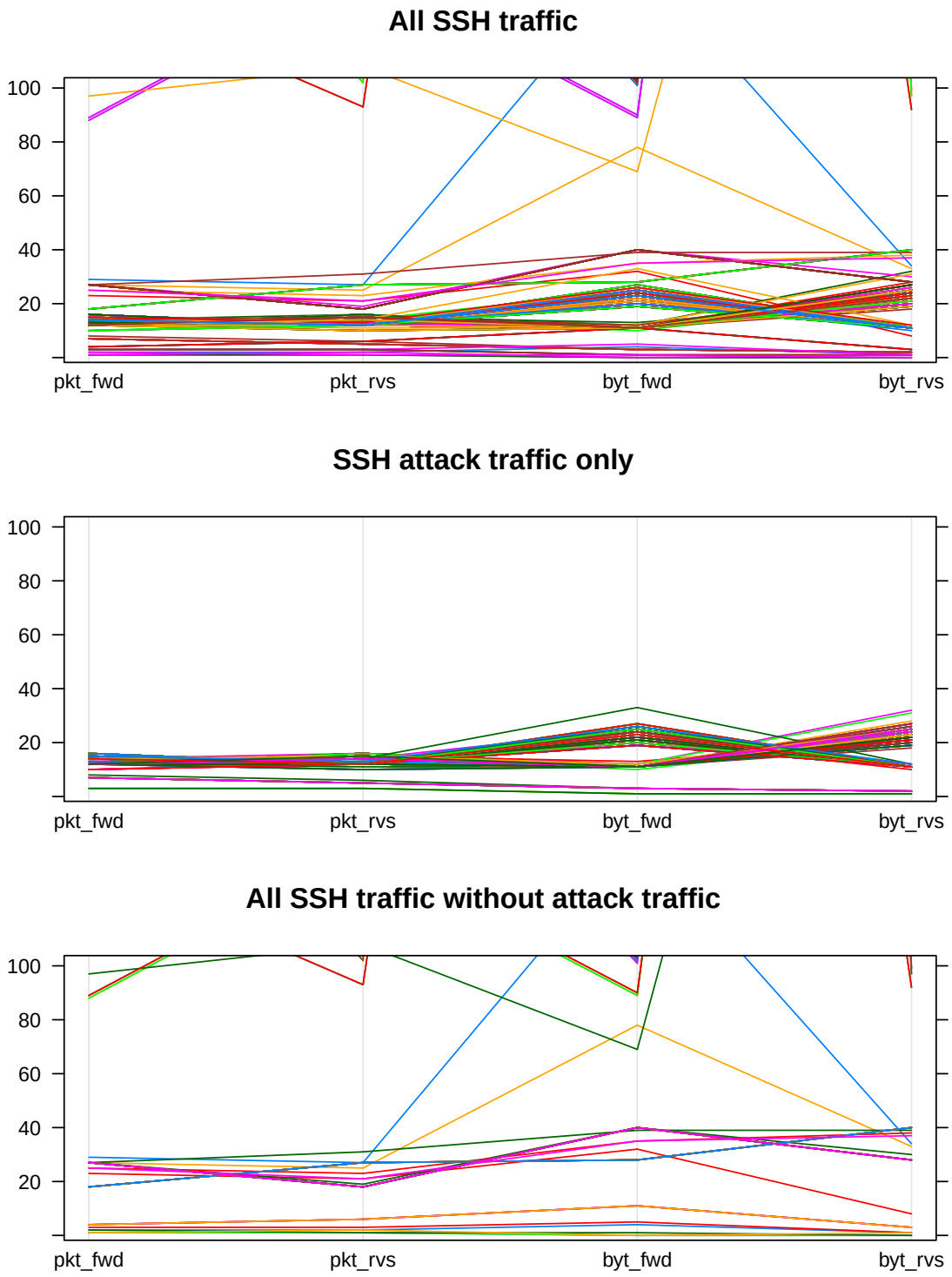

Figure 5.3: SSH traffic represented by *scaled* points in four-dimensional space (the same input data as in Figure 5.2)

a result, the difference between the numbers of packets in the both directions is not perceivable in these plots whereas the difference between the numbers of bytes is. However, if the numbers of bytes are scaled by a factor of 100, values of all dimensions range in the same magnitude. This scaling is shown in Figure 5.3.

To retain an easy interpretation of the points, we do not use standardization, i. e. subtraction of the mean and division by the standard deviation.

Clustering This key phase of the detection processes the points obtained in the previous phase and provides on the output a set of clusters of mutually similar points. The similarity is measured by distance of particular points in space defined by chosen flow features and statistics. Generally, there are four essential factors that affect the clustering:

- data representation (determined by the previous phases),
- the used algorithm itself,
- the used distance function,
- parameters of the algorithm.

As was mentioned in Subsection 2.4.2, there is an abundance of available clustering algorithms. To choose the suitable algorithm, we limit the number of usable algorithms by putting three basic requirements:

1. the number of clusters on the output is determined by the algorithm, not by the user in advance,
2. the algorithm must produce non-overlapping clusters, i. e. each point is in just one cluster,
3. the run time complexity is not greater than $O(n^2)$, where n is a number of processed points.

The first requirement stems from the fact that it is not possible to estimate the number of attacks in the future. The second one builds on the assumption that one biflow cannot carry more than one attack at the same time. Additionally, it helps with an interpretation of the results. Third requirement ensures that the detection method is efficient even on a high volume of the processed flows. Finally, note that we do not put any requirement concerning the completeness of the clustering because both partial and complete clustering can be used. In case of the former, we consider that the clusters consist of points corresponding to the attack attempts (biflows) and noise is the benign traffic. If we use the complete clustering, both attacks and benign traffic form clusters. That means we must distinguish them in the next phase.

As a result, we choose DBSCAN, a density-based partitioning clustering algorithm that can find arbitrarily shaped clusters with noise. The algorithm is described in detail in Section 2.4.2. We suppose that points representing the attack are similar and lie in the dense area of the space whereas sparse points which represent benign traffic form noise. The density-based algorithm thus enables to cluster points that intuitively belong together.

DBSCAN requires three input parameters and returns a set of clusters and a set of noise points. The choice of a type of a distance metric function and values of input parameters

Eps and $MinPts$ influence the result of the clustering and the detection. The metric function should consider the nature of the clustered points. We choose the Euclidean metric because it is a natural distance metric applicable for low-dimensional data. The distance parameter Eps determines the accuracy of the detection, i. e. the less Eps is, the more clusters may appear at the output and vice versa. This may lead to undesirable inclusion of some noise points to the clusters representing the attacks. The minimum number of points required to form a cluster $MinPts$ defines the ability to detect stealth attacks. If the attack spans more time windows and consists less than $MinPts$ biflows (points) in one time window, it is not captured because the corresponding points lies in the noise set. On the contrary, very low values of $MinPts$ may cause high false positive rate since very few points form clusters. For example, if the cluster consists of two or three points, we can hardly distinguish whether it is actually an attack or the small part of benign traffic that is coincidentally similar. Concrete values of the discussed parameters and their influence to the results of the clustering is presented further in this section.

Cluster inspection Once the biflows are separated to clusters, we may revise these cluster and the noise set to ensure the accuracy of the detection. It is supposed that each cluster contains traffic related to one or more attack classes defined in Section 5.1. This means that in case of one SBA, the inspected cluster should contain points representing only biflows with exactly one source IP address and exactly one destination IP address. For one MBA and one DBA holds that the cluster should contain points representing biflows with exactly one source address and one destination address, respectively. In case of the noise set, we suppose that it contains various points representing biflows with different source and destination addresses. If any of these assumptions do not hold, we may discard the corresponding clusters or its part for further processing. However, we may group clusters of points referring to one IP address or a group of IP addresses, i. e. the same attack class. Both cases are illustrated in the following example.

Figure 5.4 shows the result of the clustering by DBSCAN without the noise set and Table 5.4 numbers of IP addresses in all found clusters including the noise set. All four clusters are plotted in different colours in the figure. The noise set is not depicted for better clarity of the plot but it is present in the table. We analyzed in detail biflows corresponding to all points in each cluster and the noise set. Cluster 1 contains 22 points: 21 of them refer to a repetitive benign short connections of the same volume and different duration from one IP address to another and one point refers to another benign connection. This type of traffic is indistinguishable from the attack traffic, therefore it should be removed from the detection output by other means, e. g., by a whitelist maintained by the human. The cluster 2 contains 26 points: 16 points represent attack traffic from one IP address to 16 destination addresses and the rest is 10 benign conversations, each from one IP to another. In this case, we can apply a threshold for the minimal number of same source and/or destination addresses $minSame$ that must appear in the inspected cluster and a ratio of numbers of source and destination addresses with respect to the attack class. For instance, if the minimal number of same destination addresses is set to five and the ratio $ratio$ for MBA to $\frac{1}{10}$, only the attack traffic from one address to 16 addresses will be passed through. The clusters 3 and 4 contain totally 450 points that are solely related to the attack traffic. Although they actually belongs to a single MBA, they are separated to the two clusters because they are too distant from the perspective of DBSCAN and concrete settings of its parameters. This may be caused by the

distinct replies of two groups of attacked machines. We may merge these two clusters into one for further processing. Finally, there is no need to inspect the noise set since it contains only points that would form clusters with less than $MinPts$ members. The size of the set and the numbers of the unique addresses related to the points in this set confirm this assumption.

Cluster	Cluster size	Unique source IPs	Unique destination IPs
1	22	2	2
2	26	8	23
3	193	1	38
4	257	1	40
N	57	6	24

Table 5.4: Numbers of IP addresses in the inspected clusters and the noise set (denoted as N)

Attack classification The points in the clusters that passed through the inspection phase are classified in the final phase of the detection. The type of the attack (SBA, MBA or DBA) captured in each cluster is determined according to the taxonomy presented in Section 5.1, particularly by the ratio of distinct source and destination addresses.

Implementation

Similarly to the signature-based approach, the efficiency of this approach is influenced by preprocessing and biflow aggregation too. If we focus on the most popular services using the well-known network ports, we can significantly reduce the amount of flows that are paired to biflows and then processed by this method. Concerning biflow acquisition, we cannot obtain biflows by nfdump aggregation since it outputs biflow aggregates instead of particular biflows that we need for further processing. These biflow aggregates generally contain statistics describing more than exactly one biflow. This means that the volume characteristics used in clustering would be distorted by this aggregation. As a result, we implemented a simple biflow pairing algorithm that outputs a set of individual biflows and suggested to the author of the nfdump tool to add this feature to the future version.

As opposed to the signature-based approach, the clustering, the core of the detection, is a more sophisticated process. Its implementation is thus key with respect to the efficiency of the whole method. We searched for a well-known and freely available implementation of the DBSCAN algorithm because we suppose that such implementation is tested by many of its users. Finally, we choose the DBSCAN implementation from the ELKI framework introduced in Section 2.4.3 for three reasons:

1. the best performance – according to [69], the DBSCAN implementation in ELKI outperforms other available implementations,

2. active development – ELKI is continually improved and maintained (the last stable version was released in December 2012),

3. extensibility – there is a possibility to substitute DBSCAN for another clustering algorithm implemented in the framework, e. g., for evaluation purposes.

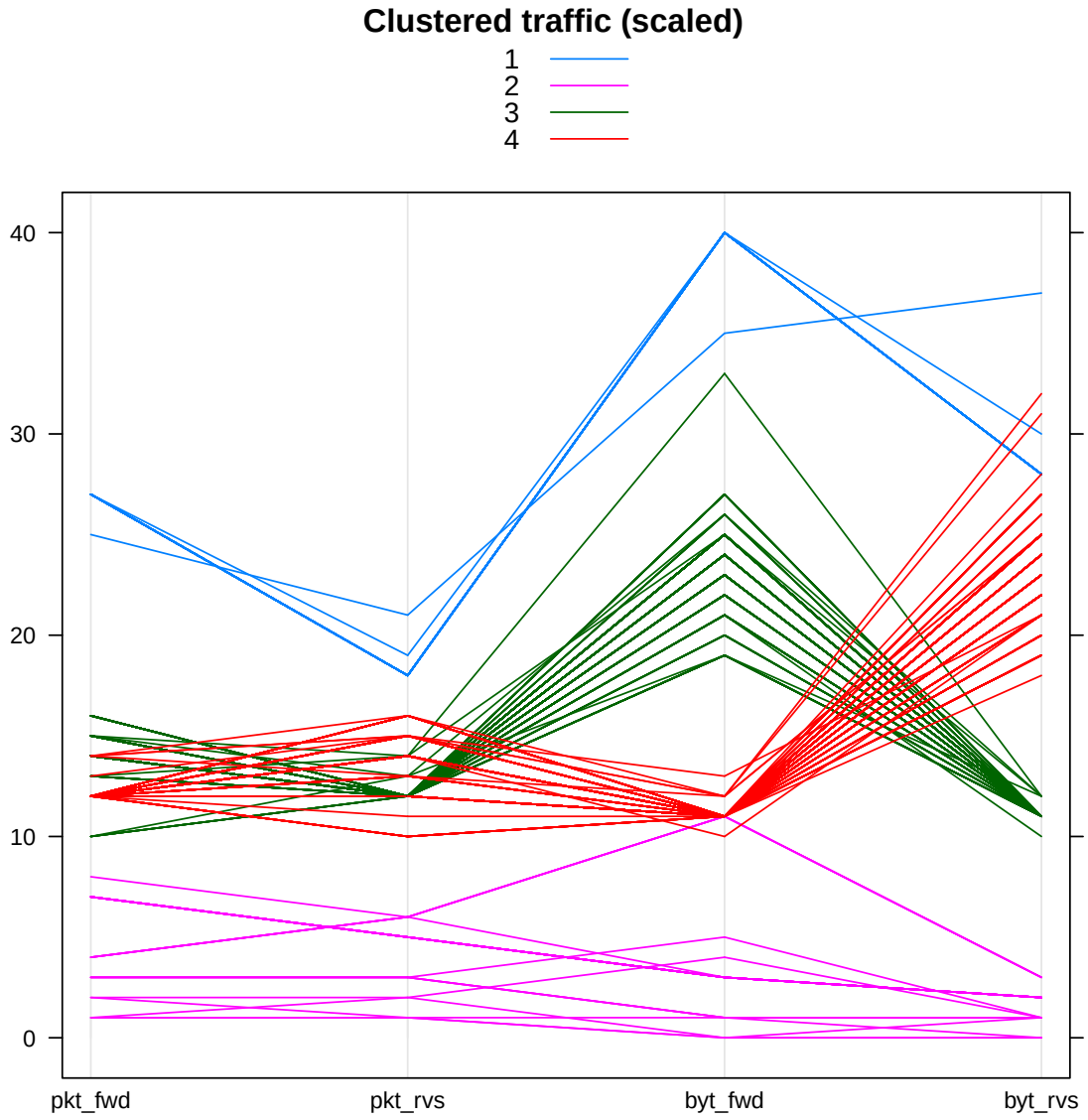

Figure 5.4: Result of clustering points by DBSCAN without the noise set. DBSCAN grouped scaled points to four clusters that are plotted in different colours.

For points scaling and presenting results of the clustering by parallel coordinate plots, we use R project [64], a system for statistical computation and graphics, with the Lattice package [67].

Evaluation

To evaluate the accuracy of the method, we chose again the popular services SSH and RDP. In the following text, we discuss results of the detection runs on ten 5-minute time windows from the Masaryk university network. The time windows were chosen randomly from each month from the period from March 1, 2012 to December 31, 2012. The ground truth was obtained by the manual analysis of the time windows.

We fixed two parameters: $MinPts$ of DBSCAN to 10 and threshold for the minimal number of same IP addresses $minSame$ used in the cluster inspection also to 10. The Eps parameter of DBSCAN was varied to find out how the results are influenced by this parameter.

Concerning the $MinPts$ value, we believe that ten is a reasonable trade off that allows to separate unsuccessful attempts made by humans from machine-made and possibly stealthy attacks. For example, if we set $MinPts$ to 1 and run DBSCAN on the data set depicted in Figure 5.3, it outputs 29 clusters and an empty noise set. DBSCAN with $MinPts$ set to 10 outputs 4 clusters and the non-empty noise set. The analysis of these results shows that 25 clusters from the first run are very small (about 2 members) and are included in the noise set of the second run. The settings of the first run is an extreme case that does not benefit from the chosen clustering algorithm. In contrast, high values of $MinPts$ such as 300 for our data set would cause no clusters at output.

While $MinPts$ actually determines the minimal cardinality of detected SBAs, the threshold $minSame$ applies mainly to a MBA and a DBA that are captured in a cluster. It should separate low-volume residual traffic (e. g., application scans) originating from one source to many destinations from traffic of MBA, and similarly, in case of DBA (traffic coming from many sources to one destination). The choice of the value of $minSame$ is thus the same as for $MinPts$.

Table 5.5 and Table 5.6 show numbers of detected attacks of each attack class with respect to the different settings of DBSCAN for SSH and RDP, respectively. The $ratio$ parameter used in the cluster inspection was set to 10 for SSH attacks and to 2 for RDP attacks, i. e. the RDP attack detection is more sensitive to MBAs and DBAs than the SSH attack detection. In our operational experience, SSH attacks are often conducted to tens of victims whereas RDP attacks even to two hosts.

SSH While we identified exactly one SBA in all time windows by the manual analysis, the method outputted a considerably higher number of SBAs. Although all reported SBAs comprise similar biflows with respect to the volume characteristics, all of them were false positives such as short SSH interactive connections, transfers of small files or communication in a grid. This corresponds to the fact that almost three quarters (39 out of 53) of all detected SBAs comprises 20 or less biflows. Some reported SBAs are present in several time windows. This implies that they steadily appear during several months. The high number of SBAs in November and December is caused by non-merged clusters with same source and destination IP addresses. The SBA that was revealed by the manual analysis is a stealthy attack that consists of only 16 biflows. It was not detected by the method with settings $Eps = 10$. We

Time window	Date	Time	SBA			MBA			DBA		
			Eps			*Eps*			*Eps*		
			10	20	30	10	20	30	10	20	30
1	March 6	23:00	5	6	6	0	0	0	0	0	0
2	April 30	03:00	3	5	5	0	0	0	0	0	0
3	May 24	02:35	3	3	4	0	0	0	0	0	0
4	June 25	06:40	2	2	2	0	0	0	0	0	0
5	July 8	10:30	2	2	2	0	0	0	0	0	0
6	August 14	19:50	2	2	2	0	0	0	0	0	0
7	September 5	03:55	2	3	3	1	1	1	0	0	0
8	October 6	01:20	3	3	3	0	0	0	0	0	0
9	November 27	17:25	21	16	14	0	0	0	0	0	0
10	December 31	09:30	10	9	10	0	0	0	0	0	0

Table 5.5: Numbers and types of detected attacks on SSH with respect to Eps in ten time windows randomly selected from each month from March to December 2012

found biflows belonging to this attack in the noise set. Other runs with greater values of Eps were successful.

Next, we identified one MBA and no DBA in all time windows. This is consistent with the result of the method. The single MBA was aimed at 13 hosts while the method outputted 10 host. The three other hosts were omitted in the cluster inspection because they received less than ten biflows ($minSame$ was set to 10).

Various values of Eps influence results in 6 time windows. The most distinctive difference occurred in the November time window. Results of the clustering are depicted on Figures 5.5, 5.6 and 5.7. On one hand, some SBAs were found in more than one cluster and thus reported several times in case of $Eps = 10$ and $Eps = 20$. On the other hand, although all traffic ends up in a single cluster for $Eps = 30$, this settings insufficiently takes the advantage of the clustering algorithm. A side-effect of the higher values of Eps is slightly higher numbers of biflows within detected SBAs that is caused by a bigger toleration of the similarity by the algorithm.

RDP As opposed to SSH, RDP attacks are more prevalent in the selected time windows. We manually identified 40 SBAs, 2 MBAs and 2 DBAs in total while the method with $Eps = 10$ detected 50 SBAs, 2 MBAs and no DBA. The detection is more precise than for SSH: the number of false negatives is significantly lower and there are no false positives. We believe that it is caused by smaller versatility of RDP in comparison to SSH. The false negatives are mainly stealth attacks comprising from 10 to 20 biflows, then a low-rate MBA against two hosts and a DBA coming from two host. These attacks were dissolved in several clusters and thus do no meet the $minSame$ threshold. In addition, some SBAs were found in more clusters of the same time window. One DBA coming from two hosts was detected as two SBAs.

Various values of Eps influence results in 7 time windows. The difference between numbers of detected attacks of each type is rather marginal. Again, the higher Eps is, the more biflows are considered similar, i. e. more biflows form a cluster and possibly an attack.

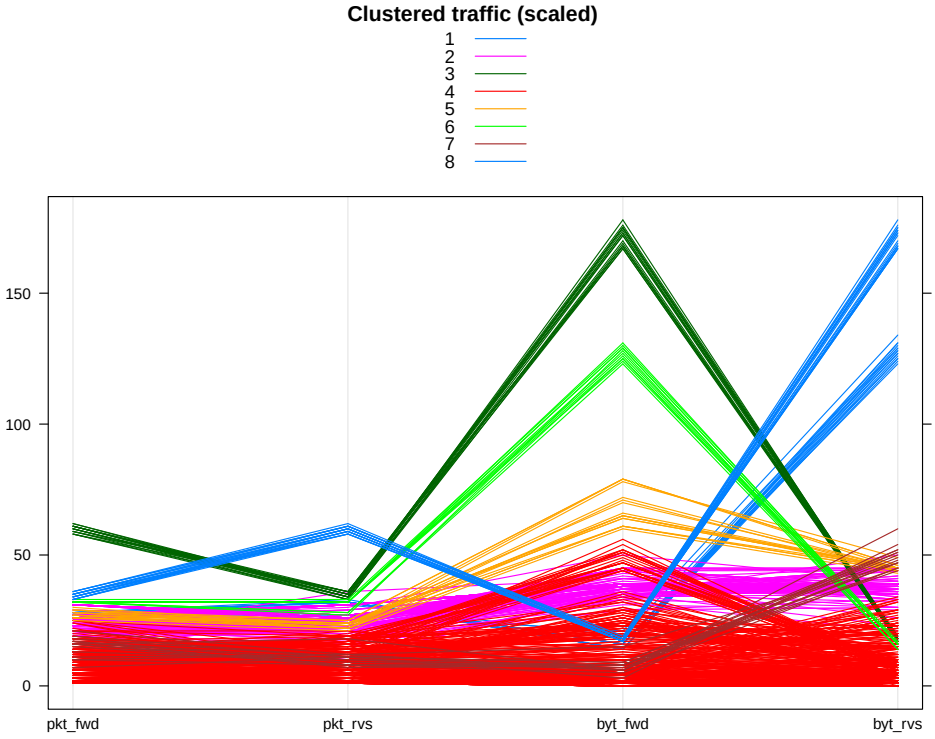

Figure 5.5: Results of the clustering of the November SSH data set with $Eps = 10$ (without the noise set)

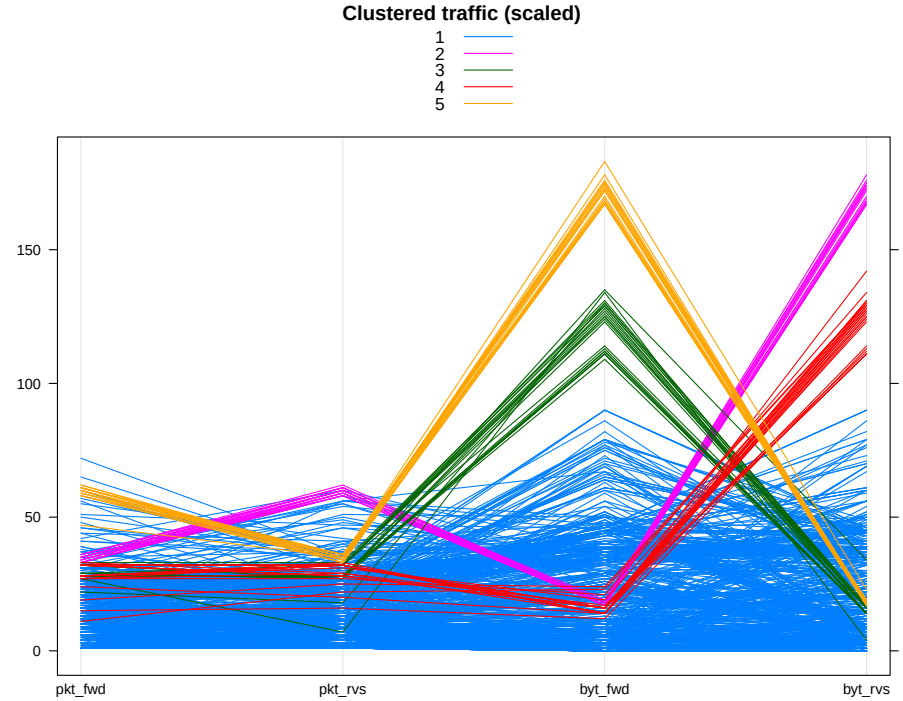

Figure 5.6: Results of the clustering of the November SSH data set with $Eps = 20$ (without the noise set)

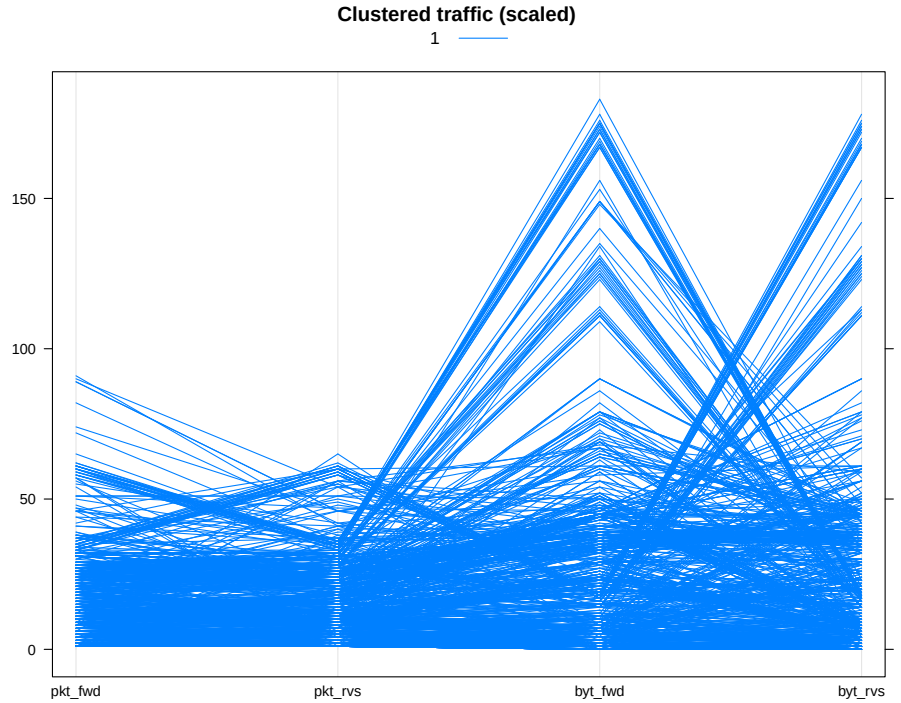

Figure 5.7: Results of the clustering of the November SSH data set with $Eps = 30$ (without the noise set)

Time window	Date	Time	SBA			MBA			DBA		
			Eps			*Eps*			*Eps*		
			10	20	30	10	20	30	10	20	30
1	March 6	23:00	6	6	3	0	0	0	0	0	0
2	April 30	03:00	2	2	2	0	0	0	0	0	0
3	May 24	02:35	0	0	1	0	0	0	0	0	0
4	June 25	06:40	9	10	10	0	0	0	0	0	0
5	July 8	10:30	5	5	4	0	0	0	0	0	0
6	August 14	19:50	2	2	2	0	0	0	0	0	0
7	September 5	03:55	6	6	8	0	0	0	0	0	0
8	October 6	01:20	2	3	3	2	2	3	0	0	0
9	November 27	17:25	9	9	9	0	0	0	0	0	0
10	December 31	09:30	9	9	8	0	0	0	0	0	1

Table 5.6: Numbers and types of detected attacks on RDP with respect to Eps in ten time windows randomly selected from each month from March to December 2012. Different numbers of attacks detected by different settings of Eps in the same data set are marked by red colour.

Performance The most critical phase of the method with respect to performance is the clustering. Hence, we measured processing time of the DBSCAN algorithm implemented in the ELKI framework that ran on COTS hardware[9]. The 20 randomly selected time windows of SSH and RDP traffic (see Tables 5.5 and 5.6) were reused and the runtime was measured

9. Notebook Lenovo Thinkpad X301 with dual core CPU Intel Core2 Duo U9400 at 1.40 GHz, 4 GB, SSD HDD

for three different values of Eps. The average processing time of 60 DBSCAN runs was 374 milliseconds with the standard deviation of 340 milliseconds. Table 5.7 shows all values of runtimes ordered by the number of processed biflows. To sum it up, the runtime of this implementation of DBSCAN grows linearly with the number of biflows regardless of different settings of Eps parameter.

Time window	Biflows	DBSCAN runtime [ms]		
		Eps		
		10	20	30
RDP 3	28	19	18	18
RDP 6	94	36	44	77
RDP 5	125	50	63	75
RDP 1	136	80	113	219
RDP 7	181	74	87	96
SSH 4	185	75	87	144
SSH 5	191	69	96	104
RDP 2	234	167	180	181
RDP 4	281	142	173	206
RDP 10	296	167	214	334
RDP 8	376	125	166	246
SSH 8	392	173	239	268
SSH 1	467	230	292	491
SSH 6	509	556	566	500
SSH 3	567	839	742	817
SSH 2	590	227	264	398
SSH 7	835	630	819	733
SSH 10	1184	676	729	759
RDP 9	1242	1229	945	1105
SSH 9	1251	831	845	931

Table 5.7: DBSCAN runtimes ordered by the number of processed biflows (*Time window* refers to time windows in Tables 5.5 and 5.6, respectively)

Discussion

This is a generic approach and its detection capability essentially depends on a chosen clustering algorithm and its parameters. We chose the DBSCAN clustering algorithm but it is possible to substitute this algorithm for another. Suitable candidates would be other algorithms available in the ELKI framework or X-means [56], an extension of K-means with estimation of the number of clusters.

The number of parameters and their intuitiveness determines the feasibility of the chosen clustering algorithm. DBSCAN requires three input parameters: $MinPts$, Eps and a distance function. The first one is straightforward: it controls the minimal size of the cluster and thus the minimal cardinality of the attack(s). The value of Eps is not so intuitive because it depends on characteristics of processed data (points) and the distance function. As a result, we evaluate the method with three different values of Eps. Analogously to the choice of the clustering algorithm, we may also substitute the Euclidean distance function. We experimentally ran DBSCAN with the same parameters as in the evaluation described above but

with Manhattan distance function. The numbers of detected attacks in the November time window were equal but the both result set differs in two of 21 attacks.

We believe that there are also other factors that considerably affects the detection: i) the choice of features and statistics in the points extraction phase, ii) factor in the point scaling phase, and iii) settings of the cluster inspection phase. A thorough study of their relevance concluded by recommendations how to estimate their values is left for future work.

From the operational point of view, we recommend to limit the detection to one service and process only traffic destined to its well-known port. Although the processing of the whole traffic might reveal other attacks that would be otherwise missed, the computational costs are very high.

To conclude, the similarity-based approach clusters together similar biflows, namely with respect to volume of transferred packets and bytes. The detection method is controlled by several various parameters. On one hand, the parameters enables flexible settings of the detection, on the other hand, they allow to degrade the usability of the method. Next, the method is inherently vulnerable to false positives because the similar traffic might be generated even by legitimate services or users. To cope with this fact, we need to employ other data sources that support or contradict the detection result. Methods of lowering false positives are described in more detail in Section 6.3.

5.2.3 Discussion and Future Work

Comparison of the two detection approaches

The similarity-based approach is a generalization of the signature-based one. The former finds attacks by clustering similar traffic whereas the latter searches for traffic matching the signature that is strictly defined by the exact values of features and statistics. Although the core of the detection is different, the approaches share four properties.

First, they both process biflows acquired by the monitoring infrastructure in separate time windows of specified size. The standard size of time windows is 5 minutes but, in principle, it is possible to shorten the size to obtain earlier detection. The influence of shorter time windows on biflow aggregation should be properly investigated to ensure that it does not distort the acquired biflows. In addition, the settings of the underlying infrastructure essentially influences acquired data and thus the results of the detection. The influence of settings of both active and passive timeouts would be investigated too. Next, NetFlow v5 and v9 are currently widely used for exporting the classic 5-tuple with volume statistics but there are first attempts to use IPFIX to export other features and statistics, namely from the application layer. These "extended biflows" carry more information that would be utilized in both approaches, e. g., an identification of the application protocol or its payload length.

Second, although both approaches can process all acquired data, this is not feasible in production deployment in high-speed networks. Both approaches benefit from preprocessing of input data by filtering out all traffic that is not destined to the well known port(s) of the monitored service. To detect attacks against more than one type of service at the same time (such as SSH and RDP), we must deploy more instances of the method (one for SSH and another for RDP).

Third, both signature-based and similarity-based approaches require some user-defined parameters. The former requires only the threshold of the number biflows matching the signature and the latter, among others, $MinPts$ and $minSame$ parameters that are analogies

of such threshold. Almost each phase of the similarity-based approach may be parametrized so the detection may be more flexible than the signature matching. However, the simplicity of the signature matching allows its easier deployment and settings.

Finally, regardless of the principle of the detection, both approaches suffers from false positives. Although the false positive rate is low, it is not zero. To eliminate the false positives, we may observe behaviour of the suspected attackers before the detected attack (even using honeypots), utilize both internal an external data sources of untrustworthy IP addresses, networks, autonomous systems or even whole countries. We describe these methods in more detail in Section 6.3. An application of time series analysis on results of both detection approaches to lower false positive rate is left for future work. We found out that the vast majority of false positives are caused by a negligible number of hosts that periodically run jobs such as file backups or network monitoring checks. Therefore, attacks repeatedly detected in a particular daytime may be excluded from the results. Nevertheless, this method could be vulnerable to hiding actual attacks conducted at the same time.

Evasion

If the attackers discover that the network is monitored for attacks, they may use three techniques for evading the detection.

The first one called *black box probing* restricts the number of attempts in a given time window under the detection threshold. This technique is more efficient if the attackers control more than one hosts and can distribute attack traffic among them. In case of blocking one host or group of hosts caused by the successful detection, the attackers can lower down the attacking rate and, what is important, they can find out whether they stay under the threshold or still not.

The second technique inserts random delays between attack attempts to distort the packet aggregation and mimic legitimate traffic. The biflows are acquired according to the fixed active and passive timeouts, among others. The artificially introduced delays may cause that the exported biflows differ in numbers of transferred packets and bytes. As a result, they are not matched by the specific signature, nor form a cluster.

Flow stretching, the third technique, exploits features of used protocols to create the illusion of legitimate traffic. Some protocols including SSH, RDP and HTTP(S) allow by design to exchange arbitrary data during the process of authentication. This protocol feature can be abused to inject random data to inflate both the flow volume size and the flow duration. Table 5.8 illustrates NetFlow data of SSH authentication attempts before and after application of flow stretching. Again, any detection based on the specific signature or the similarity of volume characteristics is infeasible.

Although all these techniques are applicable to both detection approaches, we have only little evidence [34] of their use by the attackers. However, they are inexpensive to implement and potentially very effective.

Comparison to the state of the art

An analysis of time series of volume characteristics is generally capable to detect brute-force attacks too. The difference between this approach and the two described ones lies in the nature of both the input and the output, and in efficiency of detection of stealthy attacks. As opposed to the signature-based and similarity-based approaches that process traffic in the

Duration [s]	Protocol	Src IP:Src Port	Dst IP:Port	Packets	Bytes
1.310	TCP	147.251.AA.BB:49297	147.251.CC.DD:22	12	1197
0.269	TCP	147.251.AA.BB:49320	147.251.CC.DD:22	11	1157
0.436	TCP	147.251.AA.BB:49329	147.251.CC.DD:22	11	1157
0.196	TCP	147.251.AA.BB:49358	147.251.CC.DD:22	11	1173
0.155	TCP	147.251.AA.BB:49308	147.251.CC.DD:22	11	1157
0.273	TCP	147.251.AA.BB:49318	147.251.CC.DD:22	11	1157
0.270	TCP	147.251.AA.BB:49343	147.251.CC.DD:22	11	1157
0.259	TCP	147.251.AA.BB:49344	147.251.CC.DD:22	11	1157
0.206	TCP	147.251.AA.BB:49355	147.251.CC.DD:22	11	1173
0.190	TCP	147.251.AA.BB:49362	147.251.CC.DD:22	11	1157

Duration [s]	Protocol	Src IP:Src Port	Dst IP:Port	Packets	Bytes
8.157	TCP	147.251.AA.BB:49368	147.251.CC.DD:22	142	44441
5.501	TCP	147.251.AA.BB:49379	147.251.CC.DD:22	99	30389
14.227	TCP	147.251.AA.BB:49367	147.251.CC.DD:22	239	76837
6.722	TCP	147.251.AA.BB:49369	147.251.CC.DD:22	119	36981
5.429	TCP	147.251.AA.BB:49372	147.251.CC.DD:22	98	29865
18.184	TCP	147.251.AA.BB:49375	147.251.CC.DD:22	302	97593
2.239	TCP	147.251.AA.BB:49387	147.251.CC.DD:22	47	13125
1.304	TCP	147.251.AA.BB:49380	147.251.CC.DD:22	32	8033
23.320	TCP	147.251.AA.BB:49374	147.251.CC.DD:22	384	124865
1.798	TCP	147.251.AA.BB:49386	147.251.CC.DD:22	40	10737

Table 5.8: Network flows of SSH authentication attempts before (top) and after (bottom) application of flow stretching

single time window, the time series analysis initially needs to learn the pattern of observed values of the time series. That means the detection cannot start before the learning phase is finished, i. e. immediately after the deployment. There is also a possibility to poison the learning phase with the attack traffic. This could be done intentionally by the attacker who is aware of the detection or unintentionally by the administrator who is not aware of the presence of the attacks in the learning period. Figure 5.8 shows time series of the number of flows of RDP traffic in 5-minute time windows in December 2012 and 1-day portion from the beginning of the time series. If the learning phase starts at the beginning of the depicted time series, the method learns that peaks are parts of the normal traffic. However, the most distinctive peaks represent massive network scans. Brute-force attacks are also present in the detailed time series, but they did not generate anomalous volume of traffic so they are indistinguishable from other (benign) traffic. This example also demonstrate that times series analysis can hardly detect low rate (stealthy) attacks. Next, some advanced methods of time series analysis allows to take into account the nature of the time series, e. g., trend and seasonal components, but they are not generally applicable to all types of time series. For instance, Holt-Winters forecasting method (see Subsection 3.1.2) performs well on time series with seasonal variations and trend but fails on noisy time series with random peaks and valleys. A suitable time series for the Holt-Winters method is, for instance, the total number of flows in the campus network. This time series is depicted including its decomposition to the particular components in Figure 3.1. The other example is the discussed time series shown in Figure 5.8. Analogous to similarity-based approach, advanced methods of time se-

ries analysis require settings of several parameters that may be difficult for an inexperienced user. Finally, in contrast to the two approaches, the time series analysis typically outputs the difference between the measured and the expected values. Any other details including the source(s) and the destination(s) of the possible attack must be find out by other means such as an analysis of top talkers in the given time window.

SSHCure (see Subsection 3.1.2) is similar to the signature-based approach. All phases of SSHCure actually uses a signature based on concrete ranges of the packet per flow (PPF) statistics. Hence, this detection has similar properties and capabilities as the signature-based approach and may be evaded in the same way that we discussed above. The main difference between two approaches is in the used features and statistics. We believe that in networks of different types and speed values of PPF used in SSHCure may vary more than number of transferred packets and bytes.

5.3 Summary

First of all, we proposed the first flow-level taxonomy of brute-force attacks (SBA, MBA and DBA) and probes (NS and AS) based on both biflows and unidirectional flows. The attack definitions contain several parameters so they can be utilized in various types of evaluations of brute-force attack detection. Next, we described two novel flow-based detection approaches: signature-based and similarity-based. The former searches for network traffic matching the flow-based signature that is strictly defined by the exact values of features and statistics whereas the latter finds attacks by clustering similar traffic. Using the proposed taxonomy, we not only evaluated both approaches on testing dataset but also deploy them in the real campus network for a long-term evaluation. Results show that both approaches are capable of detection of brute-force attacks even on services that encrypt communication, namely SSH or RDP. In addition, performance of both approaches allows their deployment in large and high-speed networks. To sum it up, the proposed flow-based approaches complements host-based methods and enables the detection of brute-force attacks at the network level.

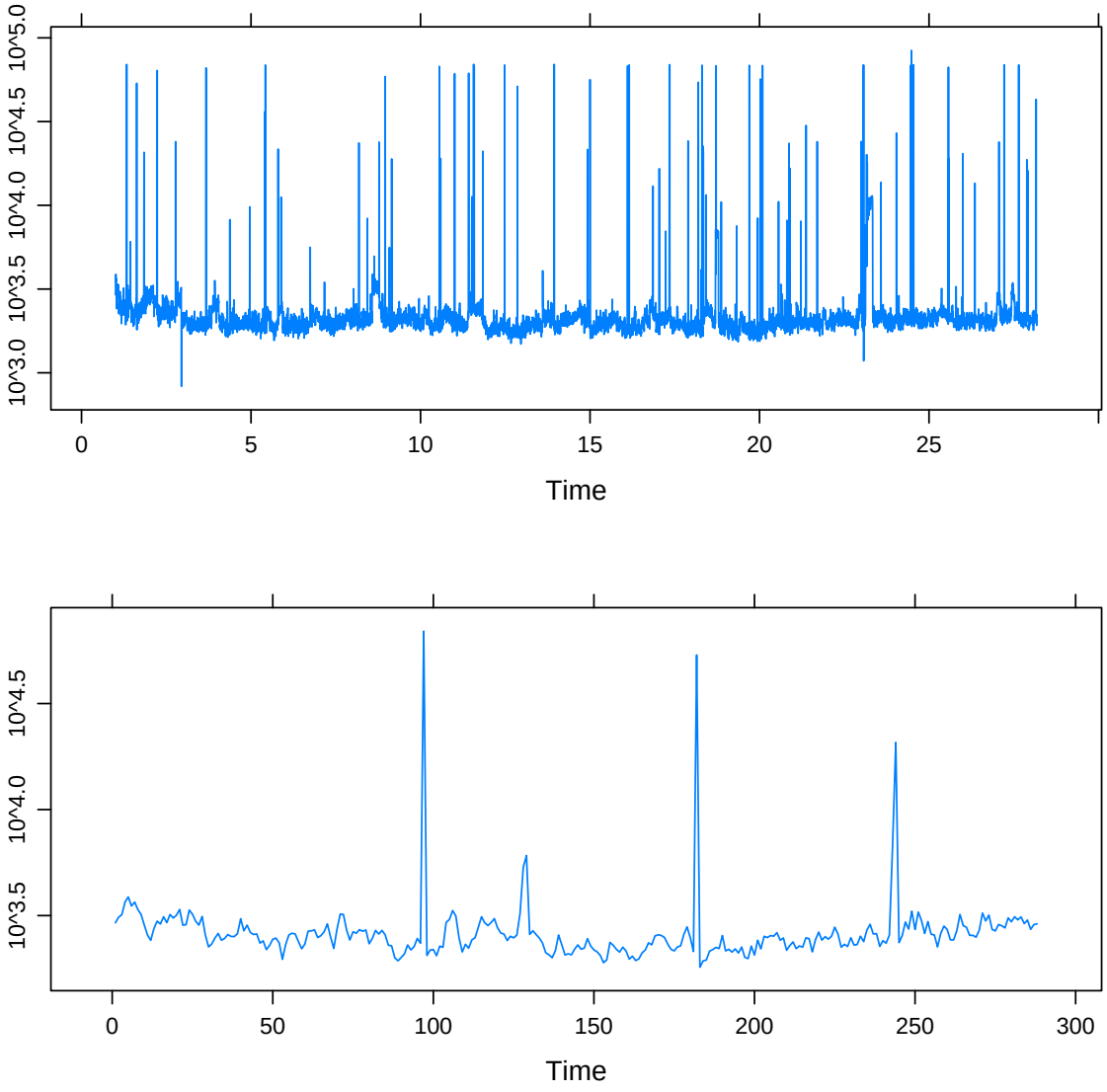

Figure 5.8: Time series of numbers of flows of RDP traffic in 5-minute time windows in December 2012 (top) and its portion for December 1, 2012 (bottom)

Chapter 6

Lowering False Positives Using Analysis of Attackers' Behaviour

A fundamental requirement for operational use of intrusion detection is zero false positive rate even with the risk of false negatives. We suppose that a detected intrusion is passed to a security team or to a system that is capable of filtering network traffic related to the intrusion. In either case any false positive leads to wasting of resources and time: the security team has to check the reported attack, which may involve workload of other staff; the system may filter out traffic of benign users. Hence, we focus on lowering false positives from various points of view. First, we propose two flow-based methods of identification of an attacker with respect to traffic filtering. Next, we study behaviour of attackers from the perspective of the attacked network to verify the generally accepted assumption that attackers probe the network before the actual attack. Finally, we use findings of the study to design techniques for combination of results of various intrusion detection methods deployed in-house and even by third-party organizations. The use of external data sources is supported by our experience that the majority of attacks is still non-targeted and thus hit more than one network.

6.1 Identification of an Attacker

Attack mitigation or prevention is often limited to filtering the source or the destination IP addresses. The most widely used identifiers of attackers are thus IP addresses, although Network Address Translation is commonly deployed and used for various operational reasons (e. g., mitigation of IPv4 address exhaustion). Consequently, more than one host or person may share one identifier (IP address). In addition, with the rise of the use of network access points providing the same IP address to distinct users over time, the identification based solely on the IP address is not sufficient.

We propose two methods that attempt to identify the attacker more precisely using information from packet headers. In fact, we perform NAT detection or passive fingerprinting of the source of the traffic. The output of these method could be taken into account in a decision whether to filter or not the source identified by IP address. An injudicious IP filtering may have disastrous impact on innocent users behind the same IP address and considerably degrades the quality of service.

6.1.1 Extended NetFlow

To develop the new methods, we extended the flow record by the IP ID and TTL fields and modified the export of TCP packets with the SYN flag set. We extract both the 16-bit IP ID value and the 8-bit TTL value from the first packet belonging to a new flow. After that, the flow is created and added to the flow cache. The extraction of both values is done in one step. The flows are exported from the flow cache after the expiration of the active and the inactive

timeouts or if there is no free space to add a new flow record. In case of the TCP packets with only the SYN flag set, extended flow records are immediately created and exported to collector, i. e. they are not inserted in the flow cache. Such mechanism allows to obtain the size of TCP SYN packet that is used in passive fingerprinting. However, this modification increases the number of flows generated due to the TCP three-way handshake. The rest of the metering process remains unchanged.

6.1.2 NAT Detection Using the IP ID field

This detection algorithm is based on observation of the ID field in IP datagram. We adapt the method described in [7] for high-speed networks. The original method inspects each observed IP datagram and tries to identify sequences of datagrams with a linear IP ID increase. The number of sequences is equal to the number of distinct communicating hosts. The proposed method does not processes headers of IP datagram but flow records that contains the IP ID value of the first packet in the given flow. Due to the fact that current versions of operating systems except the Microsoft Windows operating system family increment the IP ID value only within the scope of one flow, the detection is limited to the Windows systems.

A pseudocode is shown in Algorithm 1. We consider only flows with TTL greater than 64 because current Windows systems typically set the initial value to 128 and there is very low probability that packets transmitted by the host running Windows could traverse 64 hops. What is more, we can thus filter out traffic of Linux machines because they commonly set the initial value of the TTL to 64.

At first, the algorithm queries the data storage layer for all unique communicating source IP addresses with TTL greater than 64 in the given time window. Only IP addresses that transmit more than $IPLIM$ flows are processed. This should limit hosts with very little Internet traffic. Their intranet traffic "steals" IP IDs from searched sequences because it is invisible for the flow acquisition layer. Next, we query the data storage layer for all flow records of each source IP. That means we are counting sequences (hosts behind NAT) for each IP separately. If the currently processed flow and the following one occurred in more than $TIMELIM$ seconds we skip the processed one (similarly to [7]).

Then we have to cope with the fact that time resolution of timestamps in NetFlow records is only one millisecond. That means that there is a possibility that two or more flows occur in the same time in high-speed networks. We thus do not work only with consecutive flows. As a result, the algorithm tries to coalesce neighbouring IP IDs of the flows that occur at the same time. We suppose that these flows are produced by one host. However, if the difference between two IP IDs is greater than user-defined $DISTANCE$, they are considered to be two hosts. Next, we go through all elements of coalesced IP IDs sequences that occurred at the same time and find the nearest fit in the existing sequences of IP IDs. If the difference between the last IP ID in sequence and a current candidate is greater than $DISTANCE$, we build new sequence. Otherwise the last IP ID is replaced. All comparisons are done modulo 2^{16} because IP ID is a 16-bit field. Finally, the algorithm outputs an array of sequences (last IP ID values of the sequences). Each sequence in this array represents one communicating host.

Implementation As was mentioned in Subsection 6.1.1, this method requires a metering process that is capable of exporting the packets features (IP ID and TTL) that are not present in the standard NetFlow record. That means that the collecting process should also support

Algorithm 1 IP ID

```
IPS ← get all unique source IPs with TTL > 64
for all IP in IPS do
  if number of flows of IP > IPLIM then
    FLOWS ← get all its flows ordered by (FLOWSTART, IPID)
    for all FLOW in FLOWS do
      FLOWSTART_NEXT ← get start timestamp of the next flow
      if FLOWSTART_NEXT − FLOWSTART > TIMELIM then
        break;
      end if
      for all flows with the same start timestamp do
        coalesce "near" IPIDs
        add to SAMETIME_ARRAY
      end for
      if SEQS_ARRAY is empty then
        add the last element of SAMETIME_ARRAY into SEQS_ARRAY
      end if
      for all IPIDs in SAMETIME_ARRAY do
        if the nearest fit is found then
          replace SEQS_ARRAY[current] by IPID + PACKETS
        else
          add IPID + PACKETS into SEQS_ARRAY
        end if
      end for
      skip all already processed flows
    end for
  end if
  if |SEQ_ARRAY| >= 2 then
    print IP is NAT
  end if
end for
```

the extended flow but currently available collectors support only a basic set of flow elements defined by the NetFlow or IPFIX protocols and adding new elements is almost impossible. To overcome this limitation, we overwrite some unused default elements with the TTL and IP ID values, see Table 6.1. Such approach allows us to work with most of the collectors until there will be available full-featured IPFIX collector. Our implementation uses the NfSen collector and the nfdump tools.

Flow features	NetFlow v5	NetFlow v9	IPFIX
TTL	ToS	ToS	min/max TTL
IP ID	Src AS	Src AS	fragment Id

Table 6.1: NetFlow/IPFIX allocation of the TTL and IP ID features. Unused NetFlow features were overwritten by the new features of the extended flow.

6.1.3 ΔTCP SYN

Similarly to the initial values of TTL field in the IP header defined by a particular OS, we can observe such behavior in the case of sizes of TCP SYN packets. When a relevant component of the OS creates such TCP SYN packet, it has various size depending on TCP header options as illustrated in Table 6.2. This feature can be used in a passive fingerprinting. For example, Linux usually produces 60-byte TCP SYN packets while Windows XP creates 48-byte and Windows Vista 52-byte TCP SYN packets[1].

There is a possibility to alter these default packet lengths, nevertheless ordinary users usually do not have appropriate knowledge and/or motivation for that obfuscation (but we should not underestimate them, see [41]). So there is a good chance that the packet length was originally set to the default value of a particular operating system.

ΔTCP SYN method identifies various OS behind a particular IP address by inspecting size of the TCP SYN packets generated by each host in observed network. A pseudocode for this method is shown in Algorithm 2. At the beginning, the algorithm queries the data storage layer for all communicating source IP addresses (*IPS*) with just TCP SYN flag set and consequently all corresponding flows (*FLOWS*) for each IP address from *IPS* and then starts to examine them. Next, all TCP SYN packet sizes that have appeared for particular IP address are stored to the array *SYN_ARRAY*. Finally, if this *SYN_ARRAY* contains more than one value after evaluation of the particular IP, it indicates that behind such IP address are more OSes (hosts).

Algorithm 2 TCP SYN

```
IPS ← get all unique IP addresses ordered by (IP address, SYN length) with the TCP SYN flag set only
for all IP in IPS do
    FLOWS ← get all flows according to IP from IPS
    for all FLOW in FLOWS do
        SYN_LENGTH ← get length of TCP SYN packet from FLOW
        if SYN_ARRAY does not contain SYN_LENGTH then
            add SYN_LENGTH into SYN_ARRAY;
        end if
    end for
    if |SYN_ARRAY| >= 2 then
        print IP is NAT
    end if
end for
```

6.1.4 Evaluation

The both methods were evaluated in two laboratory testbeds and in a subnet of the campus network. In the following subsections, we describe three different setups and then present results for each method in the described environments.

1. In comparison to Windows XP, the Vista TCP SYN packet extended length is caused by the presence of the Window Scale TCP Option plus one NOP (No-Operation)

Operating system	TCP Header Options					Default SYN packet size
	Max. Segment Size	Window Scale	Selective ACK	Timestamp	32 bit Padding	
	4 Bytes	3 Bytes	2 Bytes	10 Bytes	1 Byte	
Windows Vista Bus.	yes	yes	yes	no	3 × nop	52
Windows XP Prof.	yes	no	yes	no	2 × nop	48
Linux 2.6.*	yes	yes	yes	yes	1 × nop	60
FreeBSD 7.1	yes	yes	yes	yes	1 × nop	60
NetBSD 5.0	yes	yes	yes	yes	5 × nop	64
MacOS X 10.5.7	yes	yes	yes	yes	3 × nop, 2 × eol	64

Table 6.2: TCP header default options used in current operating systems

Testbed

Laboratory The two testbeds consist of a NAT device, one or four PCs with various operating systems behind the NAT and the exporter of extended NetFlow. The NAT device used in the first testbed was implemented by well-known Linux *iptables*. In the second case, we used a SOHO router with wireless access point: *Edimax BR-6204WG*.

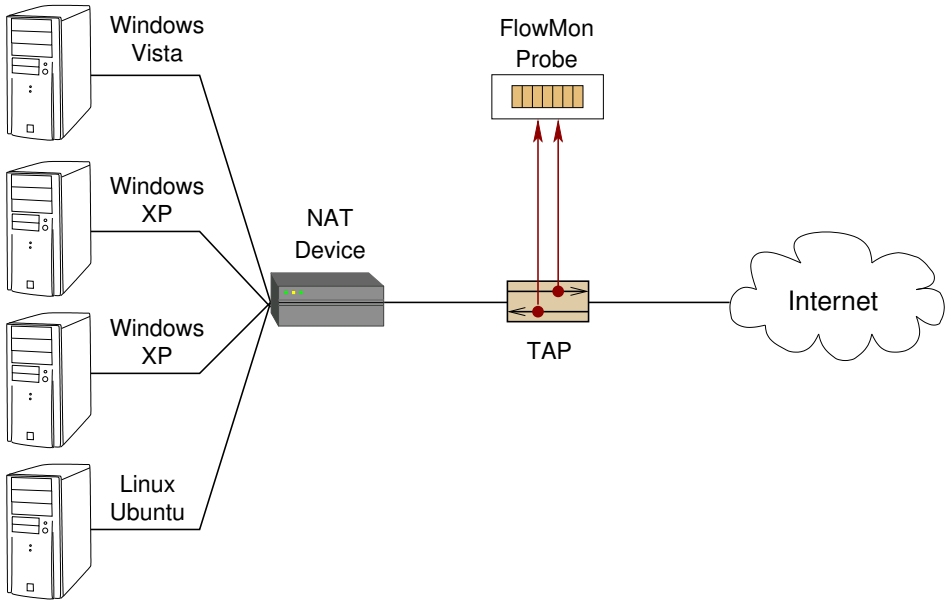

Figure 6.1: NAT detection and evaluation testbed

The evaluation of both methods was performed on testing data obtained from the following testbed configuration:

- a single Windows host behind the NAT (iptables/router),
- a single Linux host behind the NAT (iptables/router),

- two Windows XP (Home and Professional), one Windows Vista and Ubuntu Linux host behind the NAT (iptables/router),

We generated 10 different traffic samples and observed accuracy of the methods. The samples comprise ICMP Echo traffic by the ping tool, UDP traffic by the Voice over IP application Skype and TCP traffic generated by browsing web pages. All OSes were in the default configuration, no NAT hiding was enabled. We also studied an influence of background traffic generated by the NAT device.

The production network The evaluation in the subnet of the campus network lasted one month and differs both in characteristics of the traffic and the number of monitored hosts. There were tens of production hosts[2] and four known production NATs. In contrast to the laboratory testbed, the subnet was passively monitored, i. e. we did not generate any testing traffic.

IP ID

Table 6.3 sums up the accuracy of the method based on the IP ID observations in the laboratory testbed. We performed the same test with the Edimax SOHO router and obtained very similar results. The router specification confirms our hypothesis that its firmware is based on Linux source code. Next, only one of the four known NATs was detected in the production network.

OS	**NAT by Linux iptables**	
	with background traffic	no background traffic
Win XP	60%	0%
XP/Vista	80%	80%
Linux	0%	0%
XP/Linux	60%	60%

Table 6.3: Accuracy of the IP ID detection method for various OSes behind NAT based on Linux *iptables*

Hence, the method is efficient on traffic composed by a considerable number of flows originated from Windows machines and destined to the Internet. We illustrate such conditions by Figure 6.2. It depicts values of IP ID in time for two hosts running different operating systems. While the red points corresponding to a Windows machine form more or less a line, the blue points representing flows from a Linux machine are spread randomly.

The presence of the intranet traffic and exporting IP ID only from the first packet of the flow lessen the efficiency of the method. Concerning the intranet traffic, this method is limited in the same way as the original one described in [7]. To sum it up, the missing points of the red "line" in Figure 6.2 are caused by the intranet traffic and/or flows that contain more than one packet.

2. We do not provide the exact number of hosts due to fact that they dynamically appear and disappear during the day.

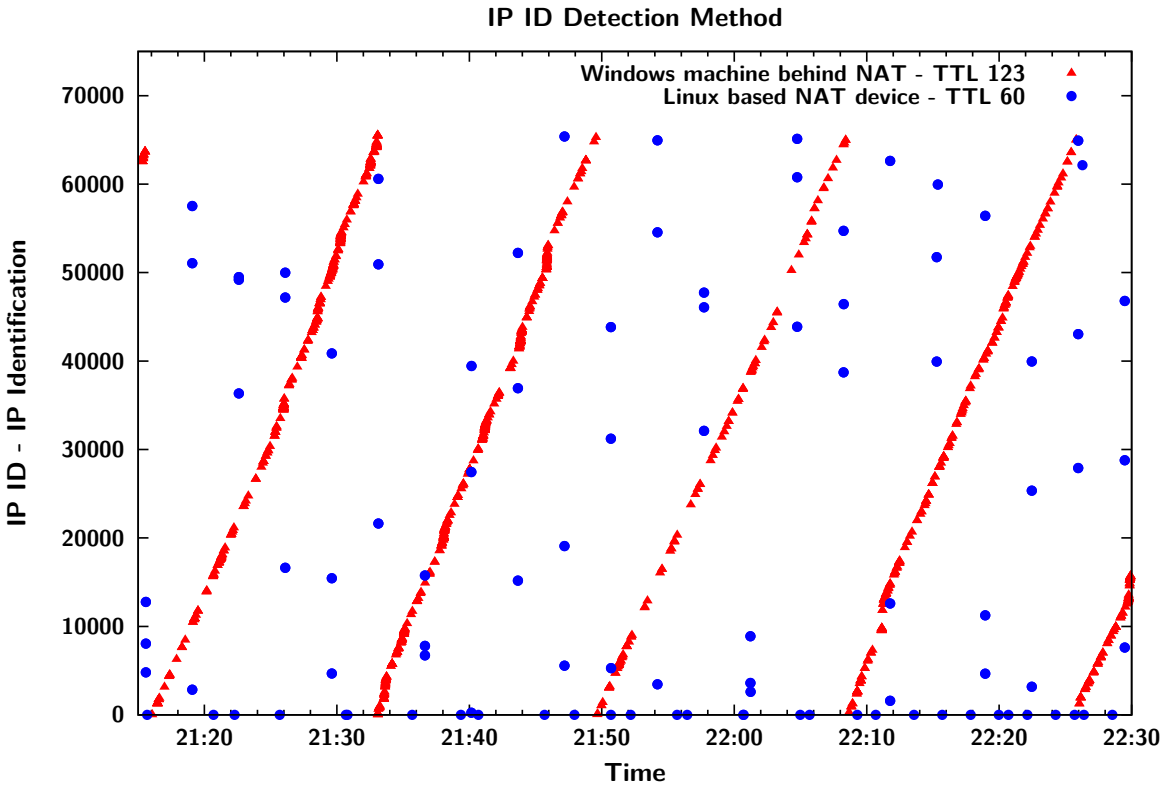

Figure 6.2: IP ID values observed by the modified NetFlow probe for NAT device based on Linux *iptables*

In general, the detection of NAT with Windows based host behind is hard to evade since the modification (i. e. randomization) of the IP ID value assignment in the Windows operating systems is close to impossible.

ΔTCP SYN

Table 6.4 sums up the accuracy of the method based on the TCP SYN packets size observations in the laboratory testbed. Again, we obtain the same results with the Edimax router. Concerning the evaluation in the production network, two of the four NATs were detected.

OS	**NAT by Linux iptables**	
	with background traffic	no background traffic
Win XP	100%	0%
XP/Vista	100%	100%
Linux	0%	0%
XP/Linux	100%	100%

Table 6.4: Accuracy of the Δ**TCP SYN** method for various OSes behind NAT based on Linux *iptables*

Consequently, this method is reliable in heterogeneous networks because the SYN packet size differs in various operating systems. We performed measurements that showed that it is possible to distinguish between Windows Vista, Windows XP/2000, Linux/FreeBSD and NetBSD/MacOS. However, NATs with homogeneous network behind (e. g., Windows XP based) stay unrevealed. Figure 6.3 illustrates TCP SYN packet sizes observed in our experimental setup with one Windows machine hidden behind the Linux based NAT device. To evade this method, an advanced settings of TCP stack is required. Hence, the method is relatively resistant against NAT hiding techniques.

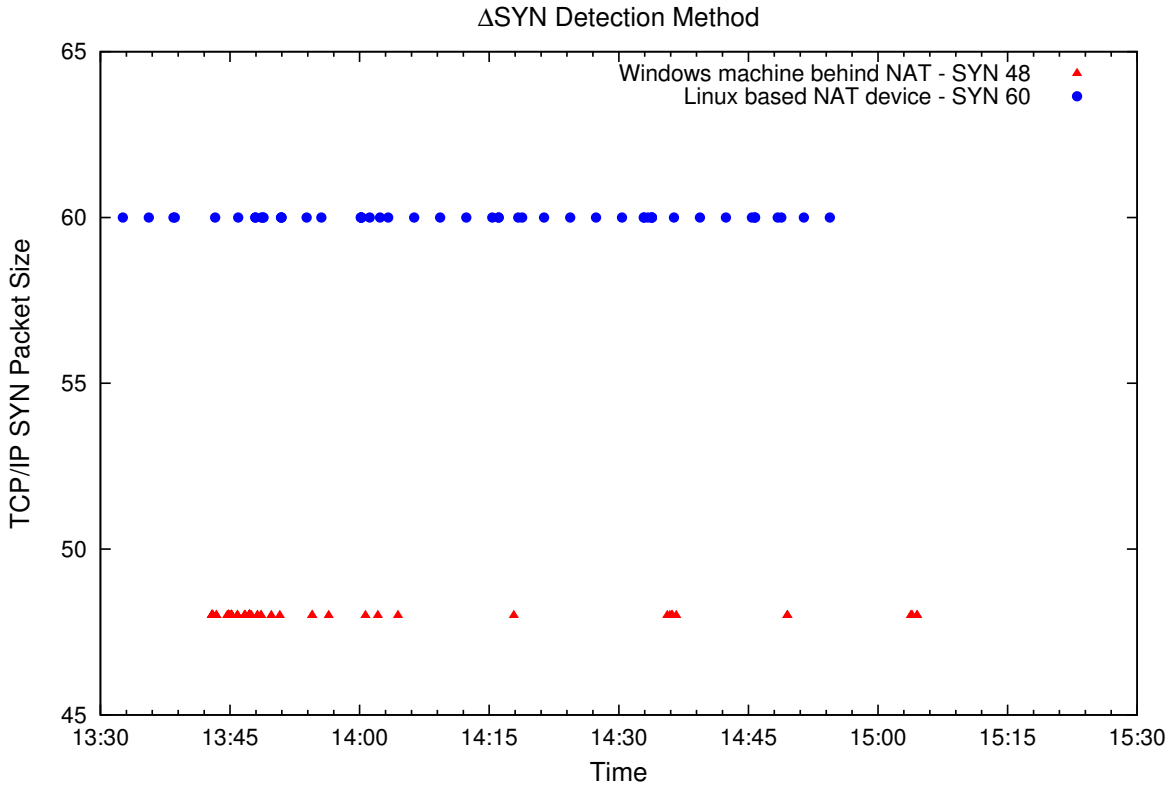

Figure 6.3: TCP SYN size values observed by the modified NetFlow probe

6.1.5 Discussion and Future Work

The evaluation showed that the two proposed methods are prone to false negatives. To improve the detection accuracy, we employ other methods of the identification and aggregate their results. This technique is discussed in more detail in Section 6.3.

On one hand, the methods benefit from the flow-based approach and have higher throughput than methods inspecting the whole packet payload. On the other hand, their deployment is very tightly connected to the deployment of the modified metering process that is capable of exporting the extended NetFlow. The lack of collectors that fully support flexible flows records (e. g., IPFIX collectors) caused that the TTL and IP ID values are exported as unused elements of the standard NetFlow.

The export of flow records that contain also IP ID of the last packet in the flow is a topic for future work. This extension should help with *overlapping flows* shown in Figure 6.4. Our implementation determines IP ID of the last packet in a given flow simply as a sum of IP ID of the first packet and the number of packets in the flow. This value is then used for the nearest fit test. This does not work in case of the flow Z that can be omitted in the processing. Next, we could consider also timestamps of the last packet of the flow and then distinguish whether the flow X lasted longer then the flow Y. The issue of insufficient time resolution of timestamps was discussed in Subsection 6.1.2.

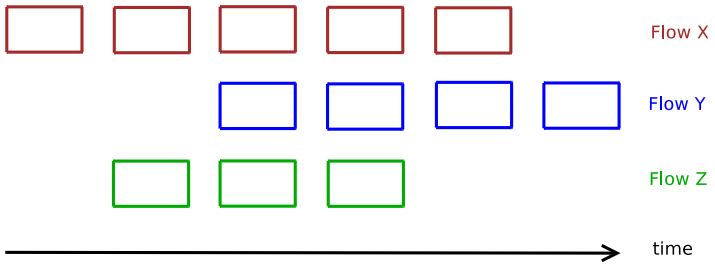

Figure 6.4: Overlapping flows

6.1.6 Summary

We proposed two methods for more precise identification of the source of the attack: one flow-based extension of the existing method of counting NATted hosts and one original method of passive fingerprinting using the size of TCP SYN packets. We do not attempt to find out whether the source of the attack is a host at the attacker's desk but decide whether the filtering the source IP address(es) is reasonable or not. We suppose that attackers commonly use (numerous) stepping stones to hide their activities and preclude their identification.

6.2 Attackers' Behaviour

To design techniques for lowering false positives, we investigate behaviour of the attackers. We start with a flow-based analysis of attack traffic of the whole campus network. This large-scale analysis is complemented by a host-based analysis of attacks captured on honeypots and a testing server. While the flow-based analysis allows to monitor a large number of hosts, the host-based analysis, especially an analysis of honeypots, offers deeper insight to the attackers' behaviour.

6.2.1 Flow-based Analysis

To enlighten real attack scenarios, we analyzed network traffic in real 10 gigabit network according to the taxonomy described in Section 5.1. We focused on SSH flows collected at two 10 gigabit uplinks of the campus network We chose SSH because it is constantly very popular service among attackers (as outlined even in [20]).

The main analysis covers the period of 39 days in two sets: the first one lasted from May 10 to June 10, 2010 and the second one from October 8 to 14, 2010. We chose data from May and October because they are the most busiest in comparison to other months. The purpose

of these two sets is to show if (and eventually how) the behavior and attacks evolves in time. Additionally, we also analyzed the entire time period between May 10 to October 10 with respect to port scanning to answer a question if the actual attacks are preceded by scanning and probing.

Measurement Setup

The campus 10 gigabit uplinks were monitored by two hardware-accelerated probes that monitor the uplinks without any packet loss and export non-sampled NetFlow v9 data to the NfSen collector. The timeouts used for NetFlow export were set as follows: the active timeout to 300 seconds and the inactive one to 30 seconds. Note that timeout settings essentially influences exported flows. For instance, if these values are too small, the flows may be exported prematurely and thus some attacks may not fit the definitions.

The acquired NetFlow data were processed in 5-minute time windows in automated way to identify the defined attack classes. We searched for *SBAs* against SSH services using the signature-based detection approach. The used signature is based on the SSH signature described in Subsection 5.2.1. The differences between the signatures lies in a greater range of transferred bytes and the between consecutive biflows. The range of bytes was set from 1000 to 5000 and the time between biflows to 3 seconds. In case of MBAs and DBAs, we used the same parameters except the time criteria. We consider the time difference between *all* biflows of the SBAs from the attack source (in case of MBAs) or to the attack destination (in case of DBAs). Concerning NS, we searched for TCP SYN probes: $f(bytes)/f(packets) \in [40, 64]$ B, $SCAN\_TYPES = \{SYN\}$. In our experience, the vast majority of scanning probes satisfies this *bytes per packet* rate. An analysis of the application scans is left for future work.

Then we manually correlated detected attacks to the entries in log files of SSH daemons to obtain the ground truth. We searched for messages describing a break-in attempt: e. g., `Invalid user webmaster from attacker's_IP` or `Failed password for invalid user root from attacker's_IP port number ssh2`. Unfortunately, we could not directly access all SSH logs in our campus network. Therefore it is sufficient for validation if the attacker of MBAs is found in a log file of one victim.

Results

We observed in total 20 885 simple brute-force attacks (SBA): 16 819 in the first data set and 4443 in the second one. The vast majority of SBAs, 20 793 (99.55 %), forms 107 MBAs and 73 SBAs (0.35 %) form a single DBA. Only 21 SBAs (0.1 %) cannot be aggregated to any MBA or DBA. The prevalence of the attack classes is similar in both data set even they are 4 months distant; as shown in Figure 6.5. To support the flow-level analysis, we inspected log files at all SSH servers that we could access and found 99 attacks[3], i. e. 76.7 % of attacks observed at the flow level. The rest of the attacks (23.3 %) we could not confirm neither deny. We do not consider the DBA in further text because a single appearance is not statistically relevant and it was not found in logs.

To detect and mitigate attacks effectively, we are interested in attackers' behaviour as well as their origin, i. e. geographical location of the attacking host. Table 6.5 shows unique attackers of particular attack classes. The table also says that there is a recidivism of some attackers. Next, attack duration varied, Figure 6.6 depicts number of attacks lasting from 5

3. MBAs, a DBA and SBAs that do not form any other attack type.

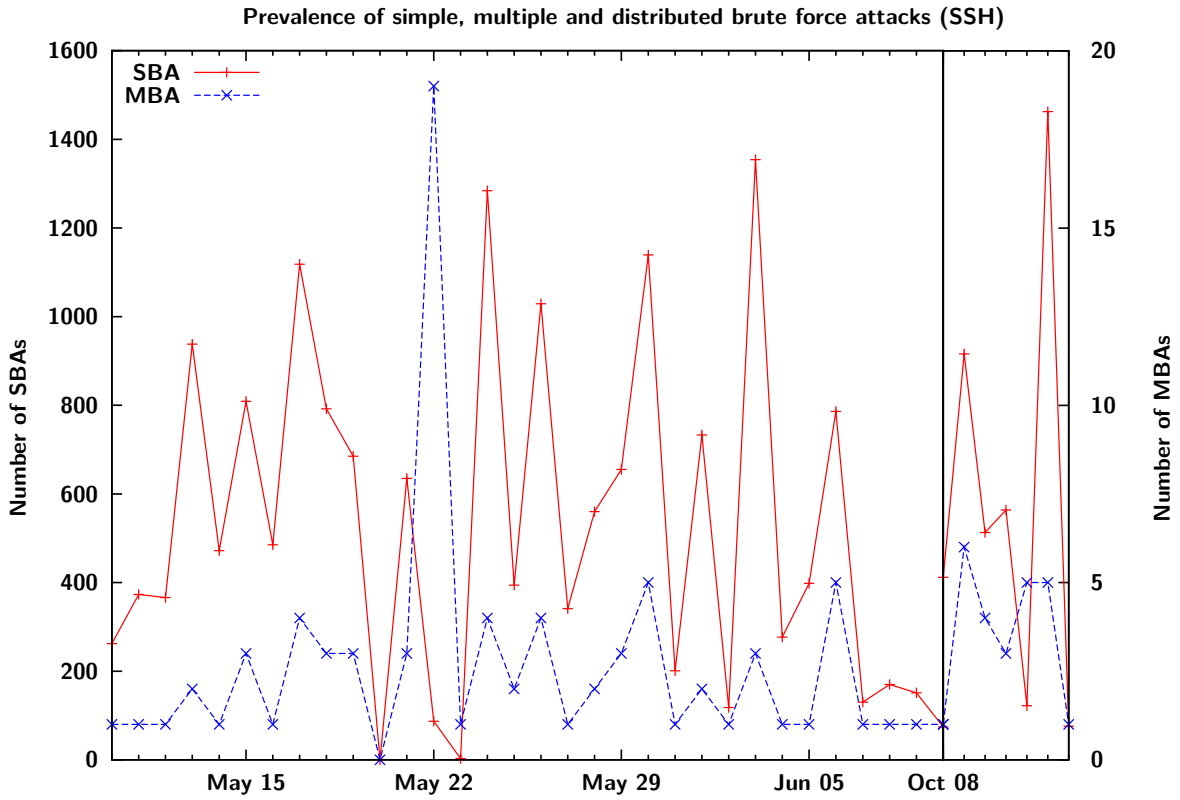

Figure 6.5: Daily sums of particular classes of attacks on SSH in the /16 campus network in two periods in 2010

minutes to 3 hours; the longest attack lasts 14 hours and 50 minutes. The distribution of the durations seems to be long-tailed.

Another important question concerning attackers' behavior is *Are brute-force attacks preceded by network port scanning?* We consider attacks from the second data set (7 days in October 2010) and search for the closest preceding TCP SYN port scans (NS, as defined above) from the attacker's IP address in time period from May 10, 2010 to October 2010. All 25 MBAs attacks were preceded by NS that occurred from 0 to 1360 minutes (22 hrs 40 mins) before the MBAs; 11 NSs were spot up to 5 minutes before the attacks. The results are depicted in Figure 6.7.

Next, we queried RIR databases by the WHOIS protocol [21] to obtain attackers' origin, i. e. autonomous system and country code, and resolved reverse DNS name for attackers' IP addresses. Attackers came from 24 different countries and 58 autonomous systems. Table 6.6 shows top 10 countries of origin and Table 6.7 top 10 autonomous systems. The vast majority of attackers conducting MBAs (63 attackers, 79.7 %) did not have set any DNS reverse record, but in case of attackers of non-aggregable SBAs we found only two empty records out of ten.

The high occurrence of MBAs in both data sets opens other questions relevant to the network- and flow-based detection: *How many hosts are attacked? What is the cardinality of attacks?* We observed 1 850 617 biflows that form 107 MBAs against 4271 hosts. The average

Attack class	Total attacks	Unique attackers	Unique attackers [%]
SBA	21	10	47.6
MBA	107	79	73.8
DBA	1	73	100

Table 6.5: Numbers of unique attackers and attacks for each attack class. "SBA" stand for non-aggregable SBAs.

number attacked hosts in one MBA was 219 using 107 882 biflows, while the median was 151 hosts using 7780 biflows. Histograms of these counts are shown in Figure 6.8 and 6.9.

6.2.2 Host-based Analysis

Attacks on honeypots

For this study, we used the same testbed as in the SSH signature validation described in Subsection 5.2.1 (see Figure 5.1). We observed totally 65 SSH brute-force attacks during a 23-day period. Despite the fact that the fifty user accounts on five machines were secured by weak passwords, only 3 attacks were successful (4.61 %). Next, we also observed 16 application scans, i. e. attacks with less than 20 repetitive login attempts. While no traffic (including TCP and UDP scans) originating in the campus network was not observed, there were logged totally 938 TCP and 501 UDP scans originating outside the network.

Scans Table 6.8 summarizes numbers and types of scans destined to particular honeypots. The most popular TCP port was 1433 (MS SQL) with 197 scans followed by 80 (HTTP) with 79 scans and 4899 (radmin) with 67 scans. Considering UDP ports, the majority of scans were aimed at ports 1026 and 1027 (106 and 83 scans, respectively).

In addition, we observed that 21 of 34 scans of the standard SSH port (TCP port 22) were followed by SSH brute-force attacks originating from the same IP address. The time between the scan and the attack varied from 6 minutes to 2 hours. In case of the successful attacks, only one of three attacks was preceded by the network scan. The other two attacks was preceded by application scans about 1 and 9 hours before the attack. A log file analysis shows that the attackers established TCP connections to the port 22 but did not enter any password. The log file contained the following record: `sshd did not receive identification string from <attacker's IP address>`.

Multiple attacks Next, we aggregated both successful and unsuccessful SBAs to three MBAs and studied attack scenarios. We analyzed all attacks conducted by intruders that were successful at least one time. The first MBA was preceded by a SSH network scan of all five honeypots. The attacker was eventually logged on the only one honeypot as "guest" by password "guest123". He or she was successful in 1 minute and 4 seconds, after 44 login attempts. The vast majority of attempts were tried with password as same as username. After successful break-in, the attacker continued with the attack until the honeypot was shut down. The total number of login attempts was 2191. No other attacker's activity was observed (e. g., modification of filesystem or file downloading). The second MBA was preceded by TCP SYN/RST scan of all five honeypots. Similarly to the first MBA, although the attacker conducted brute-force attacks against all honeypots, he or she was successful on

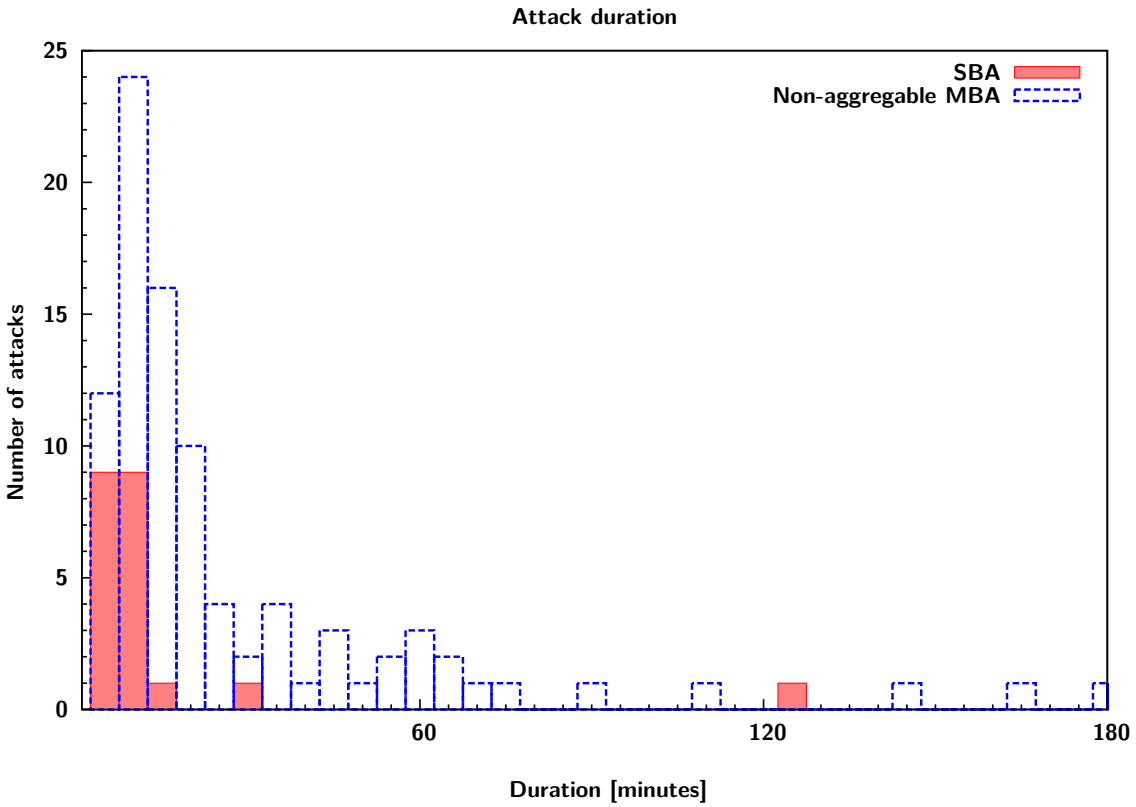

Figure 6.6: Distribution of attack durations – detailed view from 5 minutes to 3 hours

the only one honeypot. After 56 attempts tried in 3 minutes and 6 seconds, the attacker was logged as "guest" by password "12345". Again, no other attacker's activity was observed. After 4.5 hours, the attacker performed another TCP SYN/RST scan of the same honeypot and then, after 38 minutes, tried to log in as other users than "guest" for 9 times. The third MBA was preceded by application scan of four honeypots. Similarly to the first two MBAs, the attacker conducted attacks against four honeypots, but succeeded on the only one host after 401 attempts in 21 minutes and 48 seconds. He or she was logged as "test" by password "qwerty". Again, the majority of login attempts were tried with passwords as same as usernames and no other attacker's activity was observed. Further, the attacker continued with attacks against other three honeypots.

Discussion Our results can be compared to results of [65] and [3], two studies that utilize honeypots to create an attacker profile. In contrast to these studies, we did not observe any activities including downloading, installing, running malicious software or password change in case of three successful attacks. But we can confirm very low percentage of successful attacks. Generally, attempted username and password patterns are very similar to those reported by both studies. Next, we observed the majority of attacks were preceded by TCP scans that is different to the findings in [3]. On the contrary, we confirm other findings

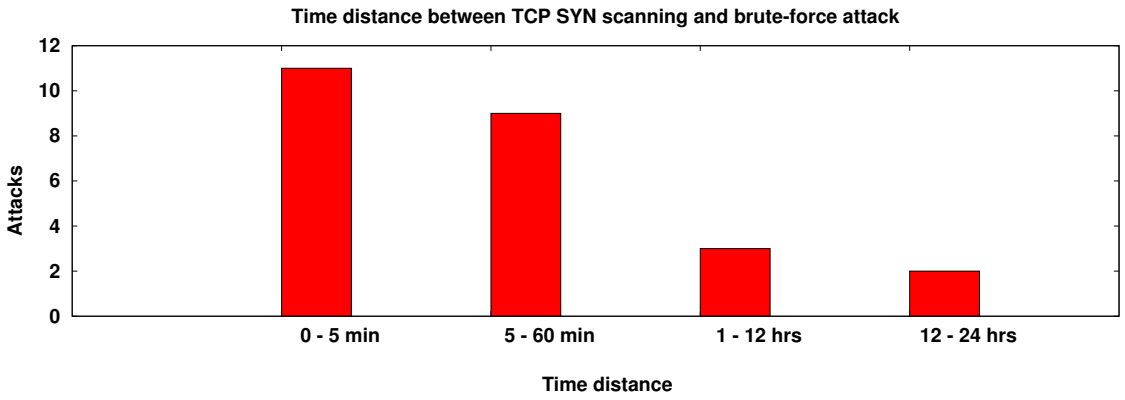

Figure 6.7: Number of MBAs that were preceded by NS grouped by the time distance between them

in [3] that the attacks follow very simple and repetitive patterns such as the attacks continued although the attacker has already guessed the correct password.

Discovery of a new server

We deployed a new SSH server on one Linux desktop with a public IP address to study if attackers discover and attack the server. All user accounts were secured by strong passwords and the access log was monitored for 30 consecutive days. In total, we encountered 911 attempts from 15 IP addresses. Table 6.9 shows the top 12 user account names and the number of failed attempts for each account. The first four places are occupied by the same account names as in Table 1 in [71] and some other names are present in both tables too. However, the last four places in our table are occupied by first names common in Czech. We inspected the access log and found many other Czech names. Hence, we suppose attackers observed a domain name of our server and consequently chose the Czech dictionary.

6.2.3 Summary

We performed the flow-based and the network-based analysis of attackers' behaviour. In the flow-based analysis, we inspected non-sampled NetFlow data of SSH traffic in the time period covering 39 days. The results show two interesting phenomenons: i) the overwhelming majority of biflows formed multiple brute-force attacks and ii) network port scanning always preceded the actual attacks. These findings can influence design of detection techniques, particularly flow-based detection of brute-force attacks. In contrast to the host-based detection, the flow-based detection can capture more precisely multiple attacks. Future work should validate the generality of these findings by undertaking similar extensive measurement for other network protocols such as RDP, FTP or HTTP. The host-based analysis using honeypots fully confirms the first phenomenon and partially the second one. In addition, it shows an incidence of application scans. Last, but not least, the experiment with the deployment of a new server suggests that the attackers do use some techniques of network reconnaissance. Otherwise, they would not attack the new server.

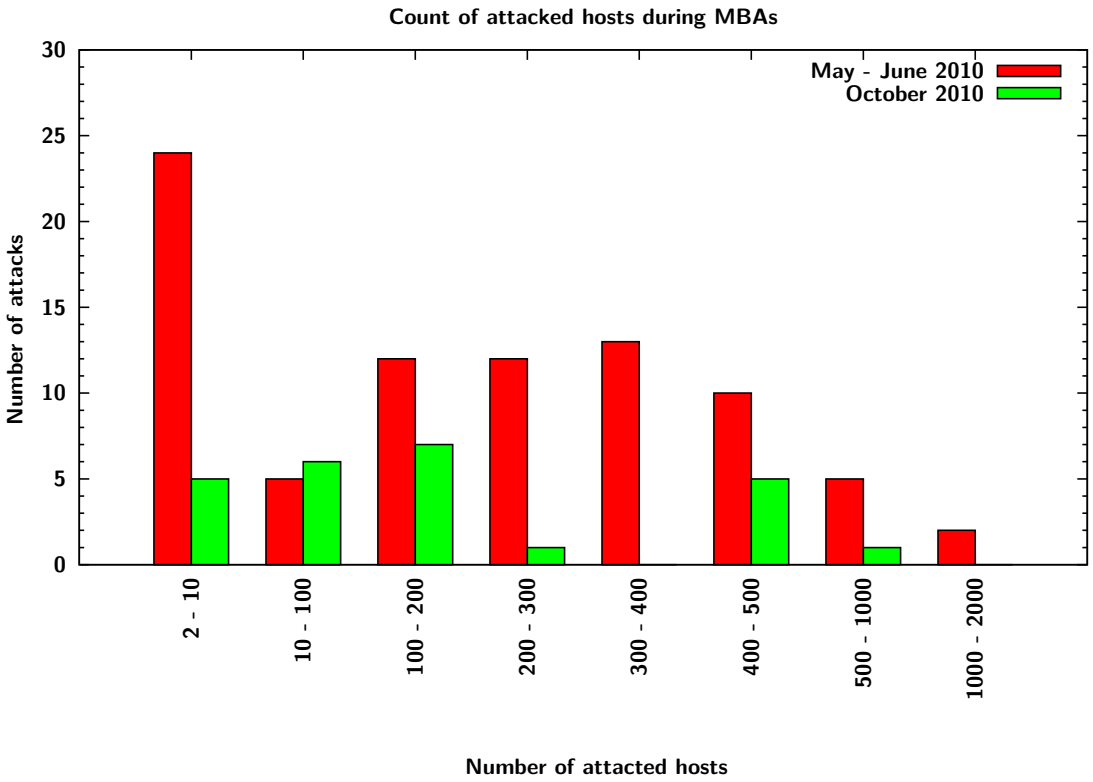

Figure 6.8: Numbers of hosts attacked in one MBA

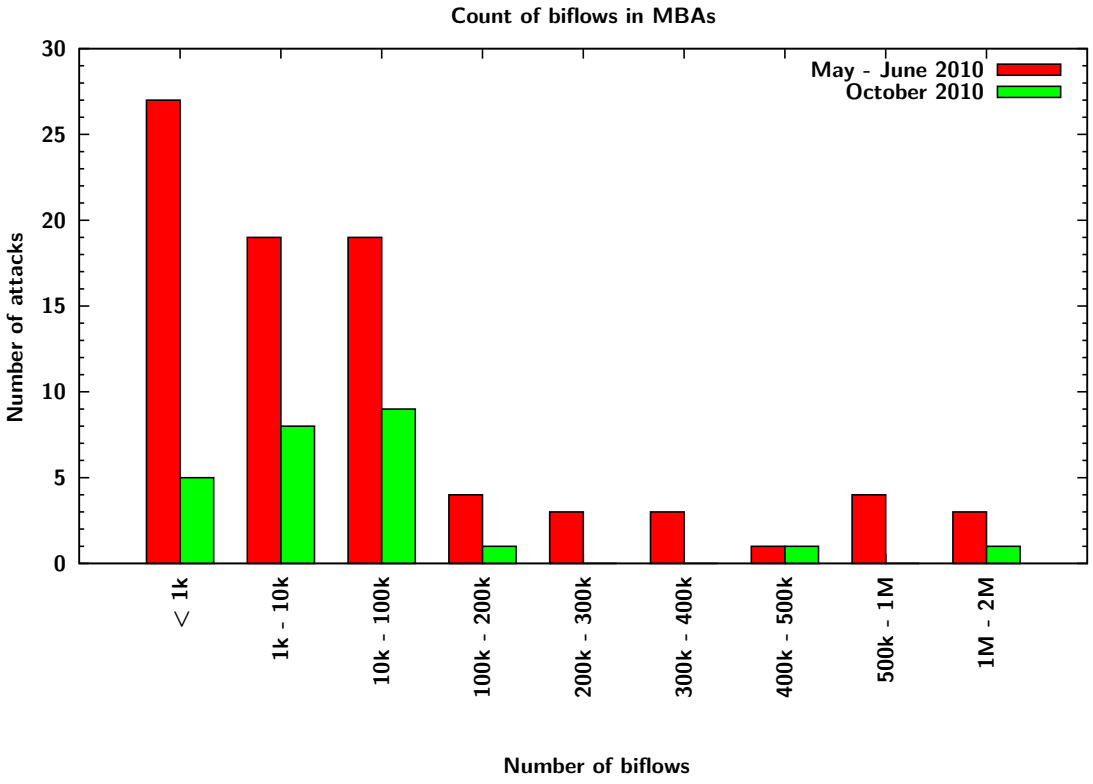

Figure 6.9: Numbers of biflows forming one MBA

6.3 Lowering False Positives

In this section, we propose four techniques of making detection more reliable, particularly with respect to false positives. These techniques stem from the findings presented in the previous subsection. While the *sequential* detection addresses the time factor of malicious events that form an attack, the *cumulative* detection aggregates results produced by various detection methods at the same time. The use of methods based on honeypots and external data sources such as blacklists is orthogonal to these techniques: they can be employed by both techniques as well as used autonomously. Last but not least, all techniques can generally benefit from both in-house detection and the external data sources.

6.3.1 Sequential Detection

We believe that finding sequences of various events that consecutively occur in time can distinguish between false positives and true positives. We suppose that a decision made on grounds of a detection of one isolated event more likely leads to the false positive than a detection based on a chain of events. Such sequential detection is demonstrated on a two-step detection of brute-force attacks that we have deployed in our campus network.

Method

This method is suitable for all services running on a specific port (or more ports). It supposes that there are two isolated detection methods: one for network scanning and the other for brute-force attacks. We focus on SSH and RDP protocols and employ the fact that the majority of attacks are MBAs and these attacks are preceded by scanning the service port. First, one-to-many communication of the popular services such as SSH or RDP is anomalous whereas one-to-one communication is common. Therefore, the limitation on attacks from one source to more destinations considerably eliminates false positives caused by incorrect detection of a benign one-to-one connection. Second, we suppose that regular users should not scan hosts providing the service because they learnt about the service location by other means. Any port scanning that occurred before possible attack attempts is thus considered to be a reconnaissance traffic attributed to attackers.

On one hand, the more detection methods are employed in the sequential detection and the more extensive detection results are provided, the more accurate the whole detection is. On the other hand, the time and space complexity of the sequential detection grow with the number of employed methods, the size and the amount of their results, and with the maximum duration of the sequence. For instance, we can detect network port scanning and report *whether* the scanning occurred sometimes in the past or *exactly when* it occurred and *how many attempts* it consisted of. The former saves the space and time required to store and find the detection results since it occupies only one bit instead of several bytes of a more complex structure. The latter allows to take into account only recent and/or massive scanning that should eliminate any doubts about its relevance to the consecutive attack. Another example is whether to detect scanning in last few *hours* or in last few *days*. Again, the former has lower requirements on the storage because it processes a smaller amount of data. The former also eliminate false positives caused by the recent change of the source IP address. However, scanning that occurred a week before a consecutive attack is not considered and the consecutive attack is not outputted by the sequential detection as in the latter case.

We have deployed this method for sequential detection of both SSH and RDP brute-force attacks using the signature-based approach presented in Subsection 5.2.1 and TCP SYN scanning. In case of SSH, we consider only MBAs against more than 10 hosts and scanning containing more than 10 flows. Once such MBA is detected, its source address is searched whether it scanned the campus network in the last 7 days or not. In case of RDP, we also consider only MBAs but against more than one host. Concerning scanning, the required number of flows is also ten but we search for scanning only in the last 24 hours. The different parameters of the method for these two services are set due to the different incidence of the attacks and limited capability of the network infrastructure that we use for attack mitigation. We set shorter time for searching scanning that preceded RDP attacks since they are more prevalent than the SSH attacks. As a result, all reported attackers can be blocked by the current network infrastructure. What is more, the results of the analysis presented in Subsection 6.2.1 showed that even only the 1-day interval would be sufficient.

Regarding implementation, we periodically run both parts of the sequential detection and store their results (including timestamp) in two separate tables in a relational database. We rely on PostreSQL[4] that supports table partitioning, among others. The use of PostreSQL allows efficient running of queries in the table storing details of detected scanning. Both SSH and RDP sequential detections have been implemented as a plugin for the NfSen collectors and released under the BSD licence (see Appendix A).

Results

In total, we detected 953 SSH MBAs preceded by TCP SYN scanning in a time period from June 9, 2011 to February 1, 2013 (603 days) and 8486 RDP MBAs preceded by TCP scanning too in a period of from October 8, 2012 to February 25, 2013 (140 days). Figures 6.10 and 6.11 show the daily numbers of SSH and RDP attacks, respectively. Although we intentionally omit RDP attacks preceded by scanning that occurred earlier than before one day, the number of RDP attacks is still considerably higher than the number of SSH attacks.

The number of reported MBAs by the sequential detection is significantly lower than the number of SBA regardless preceding scanning in the same time period. The difference is presented in Table 6.10. These numbers are consistent with findings in Subsection 6.2.1 that the vast majority of SBAs form MBAs.

Although the sequential detection efficiently eliminates false positives, in case of SSH, we identified two benign hosts that generate both successful and unsuccessful TCP connections. The latter is indistinguishable from TCP SYN scanning therefore such hosts are outputted by the method. To lower false positives to zero, the use of whitelist is still necessary.

6.3.2 Cumulative Detection

As opposed to the sequential detection, this approach combines the very recent results of several independent detection methods to provide a more accurate detection. On one hand, if the majority or even all methods come to the same results (e. g., the same source IP address of an attacker), we suppose that it is more likely *not* a false positive. On the other hand, a result reported only by one method may point to the false positive. Next, each method of the cumulative detection might report results with certainty, i. e. an estimation by the method itself of how much the result can be the true positives. The certainty of each employed method

4. `http://www.postgresql.org/`

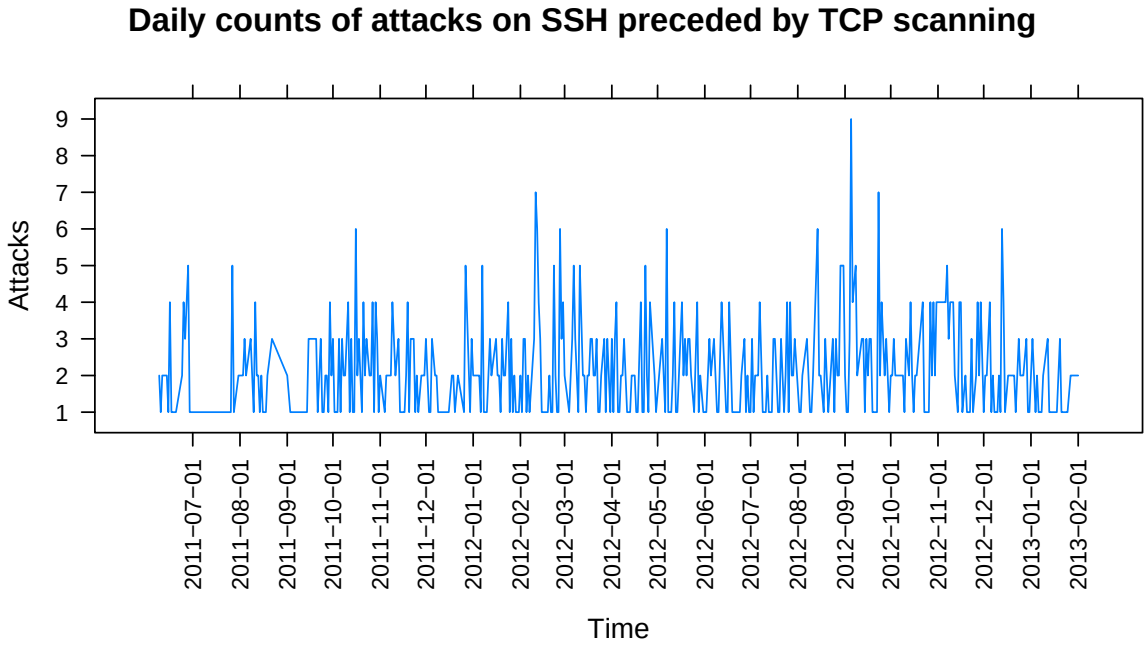

Figure 6.10: Daily sums of attacks on SSH from June 9, 2011 to February 1, 2013 (603 days)

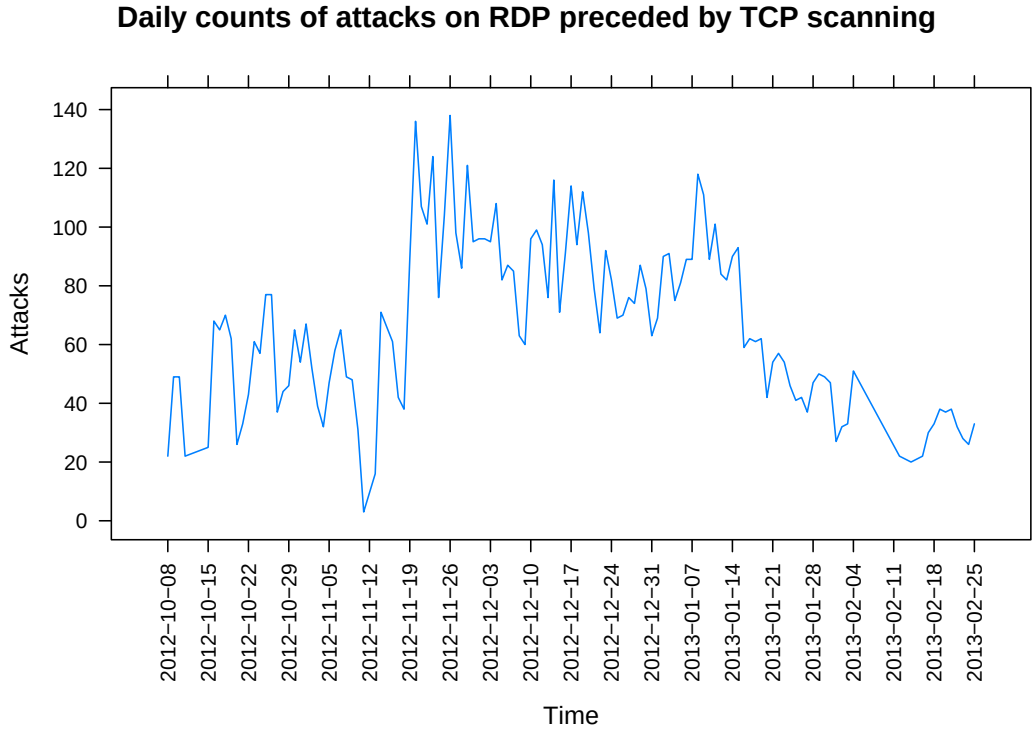

Figure 6.11: Daily sums of attacks on RDP from October 8, 2012 to February 25, 2013 (140 days)

is then used by the cumulative detection to compute the total certainty. We show the cumulative detection on an example of NAT detection by 5 various methods.

Method

The cumulative detection of NAT employs **IP ID** and **ΔTCP SYN** methods described in the Section 6.1 and three methods described in the thesis of Vojtěch Krmíček [47]:

- **ΔTTL** detects NAT devices by identification of different TTL values introduced by various OSes behind the NAT device; based on the approach described in [57].
- **ΔSubnet TTL** performs analysis of a distribution of TTL values over the inspected subnet; different TTL values than expected in the observed subnet may be a NAT device.
- **Port sequences** analyses various port sequences in the communication originating from a particular host which can indicate more hosts behind a NAT device.

While the detection capability of the individual detection methods is limited, their combination is much more powerful. Therefore, we synthesize results from individual methods into final decision about NAT presence for each IP address. Each detection method outputs an IP address and a certainty c_i of a NAT presence with the following interpretation:

- **0** – the method i cannot decide if it is NAT or not – no signs of NAT was detected in available data,
- $\mathbf{0 < c_i < 1}$ – the method i supposes NAT presence – several signs of NAT presence was detected in available data,
- **1** – the method i detected NAT – signs of NAT in available data are unambiguous.

Since each positive result of detection method increases the total certainty that the given IP address performs NAT, the total certainty C_{IP} of N methods for the address IP is computed as follows:

$$C_{IP} = 1 - \prod_{i=1}^{N}(1 - c_i)$$

Note that zero certainty of some detection methods does not affect the total certainty if at least one method outputs the non-zero certainty. However, if the single method report some address with 100 % certainty, i. e. $c = 1$, the total certainty C is equal to one too.

Generally, an implementation of the cumulative detection is straightforward. We implemented and released a NAT detection system as a plugin for the NfSen collector (see Appendix A).

Results

We illustrate the contribution of the method on the data used in the evaluation of the IP ID and **ΔTCP SYN** methods in Subsection 6.1.4. Tables 6.11, 6.12 and 6.13 show certainties of all five employed methods and the computed total certainty. The third row of Table 6.11 and the first row of Table 6.12 show that the total certainties of the detected IPs are high

although the IPs were not detected by first two methods. The cumulative detection benefits from the other three methods which detected the IPs with high values of the certainty. In contrast, the output of the first two methods essentially contributed to the total certainty when the TTL-based methods failed, see the second and the fourth row of Table 6.13. The attacker tries to hide the presence of NAT by rewriting the TTL value but this does not affect the detection capabilities of other methods. In conclusion, the cumulative detection benefits from its application on different detection methods that give the same type of results. Not only it lowers false positives but also it is more resistant to evasion of detection than the particular methods in themselves.

6.3.3 Using Honeypots

Honeypots are widely used and understood as research tools for tracking the attackers. However, they can be utilized also in a production environment since they should not produce any false positives. What is more, they are capable to capture zero-day attacks. The detection using honeypots can be used as an independent method or one method employed in the sequential or/and cumulative detection. In the following text, we describe two detection methods based on a monitoring both low- and high-interaction honeypots.

Method

In fact, the first method is another example of the cumulative detection since it combines the flow-based and host-based approaches. It monitors network flows destined to the honeynet and passwords attempted during brute-force attacks against services running on the honeypots. The method simultaneously searches the acquired flows and authentication logs on honeypots in separate time windows. All source addresses that are found in both data sources are clearly attackers because benign users should not visit a honeypot *and even* guess more than $pass\_limit$ passwords, where $pass\_limit$ is a parameter set by user. The second method is purely flow-based: it searches for flows destined not only to the honeynet but also to the production network. Such traffic points to attackers that (accidentally) hit not only the honeynet but also the production hosts. While both methods are simple, the most challenging and crucial for their efficiency is to attract the attackers and hide the fact that they are interacting with honeypots.

The honeynet including the monitoring infrastructure is depicted in Figure 5.1. It occupies a /24 campus subnet and consists of two types of high-interaction honeypots running on the VirtualBox platform, tens of low-interaction honeypots created by honeyd and a NetFlow probe that monitors the network interface of the VirtualBox host. The use of virtualization allows to run more than ten guest machines on a single host and decreases the maintenance costs. The latter is particularly important because it allows very quickly to deploy the same machines in the whole honeynet. The first type of high-interaction honeypots are full-fledged hosts with hundreds of megabytes of RAM running Ubuntu 8.10 Server Edition. We intentionally chose the old version of a widely used Linux distribution and installed web servers with fake content to appear more attractive for attackers. To run more guest machines on the single host, we use the second type of high-interaction honeypots: single-purpose hosts with a few megabytes of RAM running Micro Core Linux, a minimal Linux operating system[5]. These "micro" honeypots are dedicated for gathering attempted SSH passwords.

5. http://distro.ibiblio.org/tinycorelinux/

The password are gathered on both types of honeypots by a special pluggable authentication module (PAM) of a patched SSH server[6]. Unfortunately, the high-interaction honeypots consume a lot of system resources so we also deploy less demanding low-interaction honeypots to fill the remaining unused address space dedicated for the honeynet. Although they only simulate real hosts, they are still more valuable than a darknet. As opposed to the high-interaction honeypots focused on SSH, honeyd running the low-interaction honeypots simulates other network services namely Telnet, FTP and e-mail (POP3 and IMAP). Passwords attempted for login to these services are gathered and stored too. To sum it up, both types of honeypots run variety of commonly used services to attract attackers and gather attempted passwords. We leave for future work to examine if it is preferable to simplify the described infrastructure, for instance, use only darknet with flow monitoring.

Results

The first method reveals totally 117 connections to the honeynet with password guessing in a time period from November 21, 2012 to March 3, 2013. The $pass\_limit$ parameter was set to 4 since we believe that users that accidentally attempt to login to the given honeypot enter the password maximally three times. Table 6.14 shows the attacked services and numbers of attempts. The method clearly identifies the attackers but the number of results is very low, about one attacker per day in the chosen period.

In contrast, the second method detected much more traffic: totally 99 118 simultaneous accesses to both the honeynet and the production network in a time period from February 3, 2013 to March 4, 2013. A hexbin scatter plot[7] of the numbers of accesses to both the production network and the honeynet is depicted in Figure 6.12. The most numerous are accesses to the single honeypot and the single production host. This traffic may be attributed to host misconfiguration rather than to reconnaissance or attack so the conservative approach is to take into account more contacted hosts in the sequential or cumulative detection.

6.3.4 Using External Data Sources

Beside the intrusion detection based on monitoring network traffic or hosts in the administered network, there are external data sources available that may be employed in the sequential or cumulative detection. Most often, they provide information about a given IP address in two different ways: i) additional information such as a domain name or a location, ii) its presence in the results of intrusion detection systems maintained by external organizations. Both data sources may generally improve the in-house intrusion detection, regardless it is network-based or host-based. In the following text, we introduce several external sources and examine their usability, especially if IP addresses of attackers that were detected in our campus network are present in external blacklists.

Domain Name

The absence of a domain name may point to an insecure host controlled by the attackers since every Internet-reachable host should have a name [4]. What is more, the results of the analysis presented in Subsection 6.2.1 indicated that the majority attackers did not have set

6. The standard OpenSSH server does not pass the attempted password to the PAM if the password was used for login as a non-existing user.
7. Hexagon binning is a form of bivariate histogram useful for visualizing the structure in large datasets. [15]

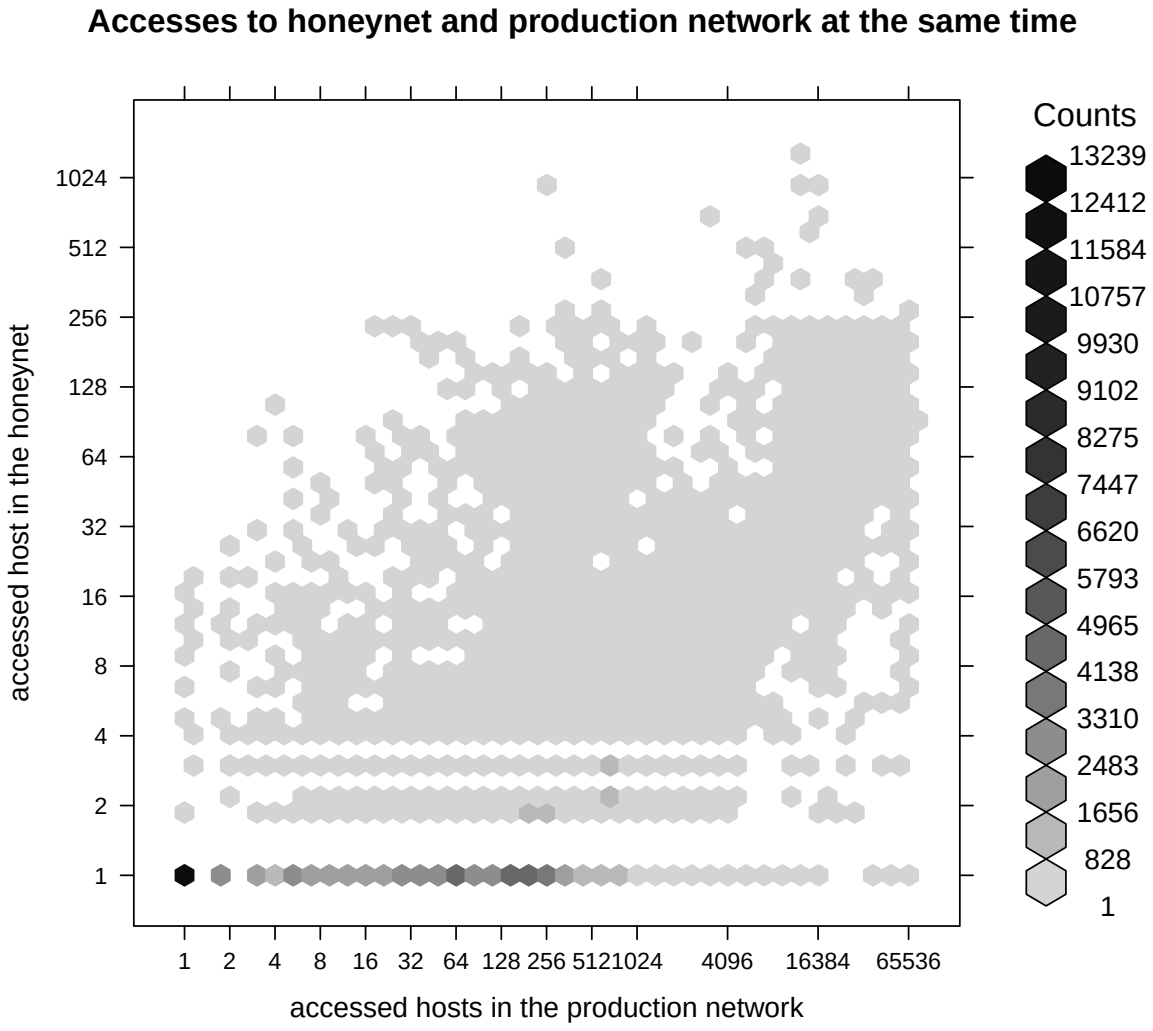

Figure 6.12: Hexbin scatter plot of numbers of accesses to both the production network and the honeynet

any DNS reverse record. This also confirms the results of the sequential detection of SSH attacks described in Subsection 6.3.1: 572 out of 953 attackers' IP addresses have no domain name.

Location

A restrictive detection may take into account information about geographical and/or administrative location of the attacking IP address. These information are typically available in RIR databases that stores autonomous system number or country code, among others. The analysis in Subsection 6.2.1 showed that attackers came from a relatively small number of countries (24) and autonomous systems (58). That implies that manual maintenance of a list of suspicious countries or networks would be feasible.

Blacklists

External blacklists allows to confirm detection results or even prevent the non-targeted incoming attacks. It is supposed that non-target attacks consecutively hit more than one network. We demonstrate this hypothesis by a comparison of IP addresses of attackers on SSH detected by the sequential detection (see Subsection 6.3.1) and by the OpenBL.org project (see Subsection 2.3.2). We thus compare results obtained in the campus network by the flow-based detection and results from a geographically distributed host-based detection. In a time period from December 8, 2012 to March 8, 2013 (90 days), 80 of 113 attackers' IP addresses detected by the sequential detection were reported by OpenBL.org too.

Reputation Systems

For more extensive use of blacklists, there is possible to employ systems that aggregate information from more than one blacklist. Again, we investigate how many IP addresses detected by the SSH sequential detection or a corresponding autonomous system is seen by other reputation systems.

We start with BGP Ranking that computes reputation for autonomous systems (see Subsection 2.3.2). Out of 113 attackers detected in the 90-day period, 60 unique corresponding ASes were found. Figure 6.13 depicts ranking of ASes containing the reported IPs. We can see that the attackers came from the first quarter of the most malicious ASes; the total number of ASes was 67 865. The AS containing our campus network was on the 7868th place (marked by grey line in the plot), i. e. the majority of ASe containing the reported addresses were considered more malicious than "our" AS.

Next, we survey the occurrences of the detected IPs in the Warden system (see Subsection 2.3.2) in a time period of June 27, 2012 to November 22, 2012 (148 days). Out of 251 addresses detected by the SSH sequential detection, 229 were also reported by other Czech campus networks to Warden too. Table 6.15 summarizes how many sensors outside our campus network detected the same IP addresses. Figure 6.14 shows time differences between the detection in our network (plotted at zero) and in other campus networks for 25 randomly chosen addresses[8]. We can see that some IP addresses were detected in other networks sooner, some others not and the difference of the vast majority of addresses is up to four hours. This observation is the same for the whole set (not only for the sample of 25 addresses). Both the count of detection in other networks and the time difference may be taken

8. The plot of all addresses lacks details, points are very dense, we therefore sample the whole set.

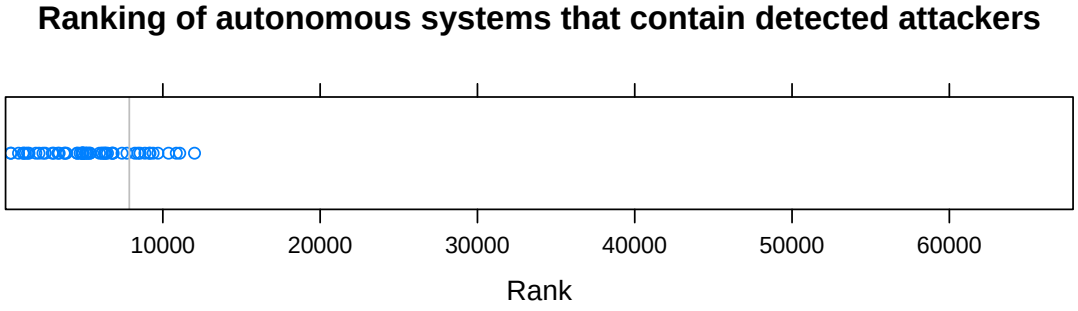

Figure 6.13: Ranking of 60 autonomous systems that contain 113 reported IP addresses of attackers aiming at SSH

into account in the cumulative or sequential detection. The more (different) sensors detected the same address in various networks at around the same time, the more probability of being true positive.

6.3.5 Discussion

Although the presented techniques helps to lower false positives, they may be tricked by attackers to produce false negatives. First, the sequential detection is prone to stealth attacks that span a wider time period than is covered by the detection or have different properties than expected. For instance, in case of the sequential detection of SSH attacks (see Subsection 6.3.1), attackers stay undetected if they scan the network more than 7 days before the actual brute-force attack and/or focus on exactly one host (i. e. they do not conduct a MBA). Second, the cumulative detection is inefficient if it is composed by detection methods based on the same principle or/and type of input data. For example, the NAT detection (see Subsection 6.3.2) using only the TTL value inspection would be unsuccessful if the NAT device was set to rewrite the TTL value in packets passing through it. Third, although honeypots produce no false positives, this advantage is balanced by the fact that there is a low probability that attackers hit the honeypot network segment (see Subsection 6.3.3). For instance, the method monitoring accesses to honeypots could substitute the TCP SYN scanning detection in the sequential detection to eliminate possible false positives caused by the scanning detection. But this bears a high risk of false negatives induced by the use of honeypots. Finally, the main problem of the use of external data sources is the trustworthiness of data providers. If a data source provides biased information, regardless caused by poisoning by attackers or unsound detection, results of the sequential or cumulative detection using this information may be considerably distorted. The false positives provided by the data source may lead to false positives of the detection that use the source such as benign connection is considered to be a brute-force attack. Similarly, the false negatives or outage of the data source preclude the detection, e. g., unreported unauthorized access to other networks cannot help to detect an attack in the administered network. In addition, the timely cumulative detection essentially relies on the recency of a data source. For instance, a blacklist updated with a delay of several hours is useless.

We illustrate both the sequential and cumulative detection approaches using flow-based methods only, but both approaches are not limited to the flow-based or network-based de-

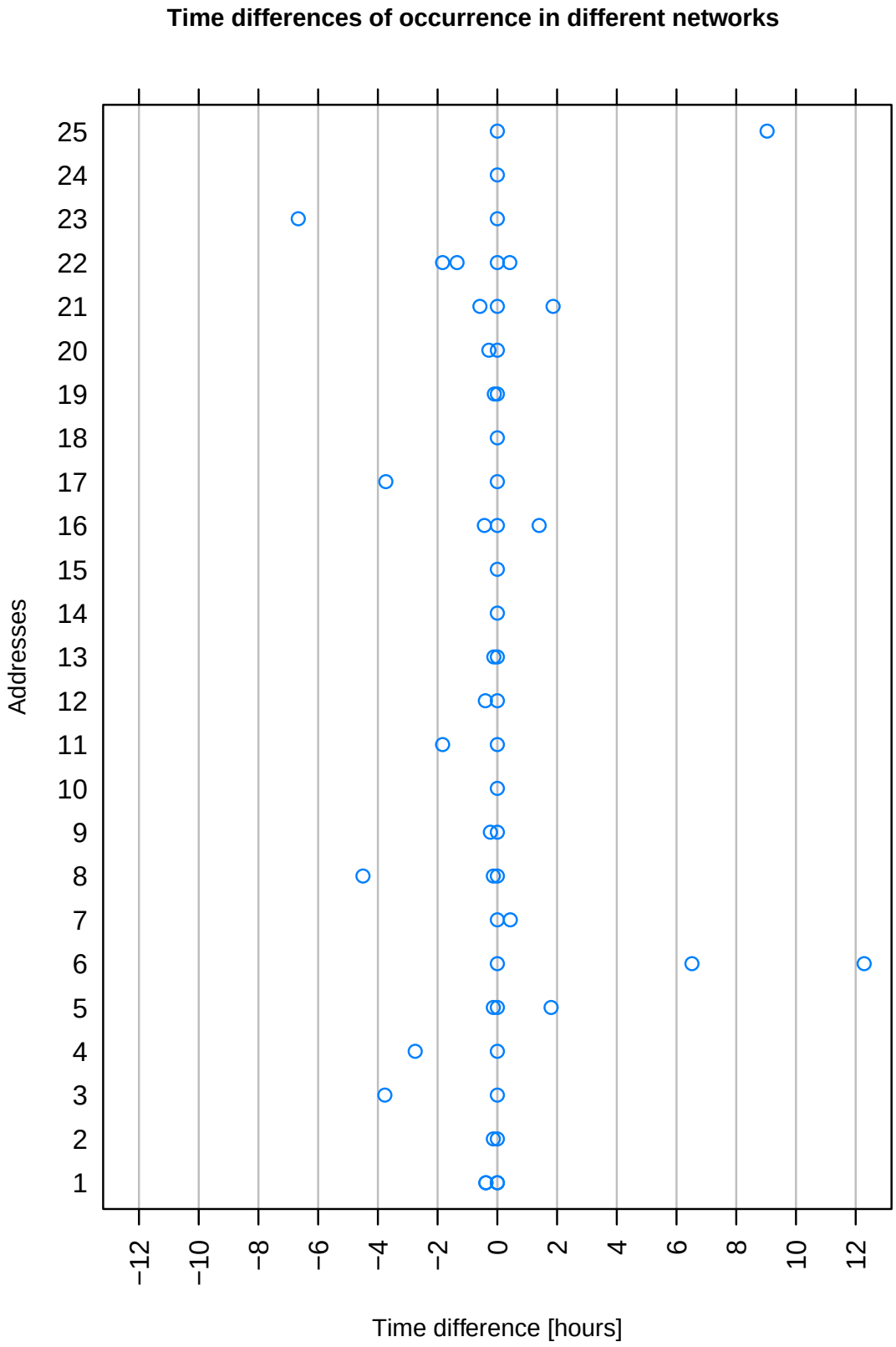

Figure 6.14: Time differences between the detection in our network (plotted at zero) and in other campus networks for 25 randomly chosen addresses

tection. An example of employing another data source is our tool that detects whether users responded to phishing e-mails[9]. Once a URL is found in a reported phishing e-mails, all communication from the administered network to the URL is searched in NetFlow data.

The drawbacks of the use of honeypots are a high cost of the building the infrastructure and a relatively low incidence of attackers visiting them in comparison to the production network. The detection capability may be improved by deploying more types of services to more honeypots but this requires more system and human resources. In future work, we focus on attracting more attackers to honeypots using a "network **f**unnel" that would route traffic destined to various unused parts of the administered network to honeypots. Consequently, the funnel should concentrate suspicious traffic to the honeypot segment that is usually smaller than the unused network space.

Information about the origin of attackers, particularly their country, should be used carefully since there may be errors in the geolocation databases. On one hand, its injudicious use may be dangerous in case of countries with the highest internet population such as the USA or countries that usually host our users. On the other hand, it is easily possible to make a decision about traffic coming from "exotic" countries with respect to the expected location of users. For example, in case of our campus network located in the Czech Republic, we do not expect any incoming SSH traffic that originated in a remote developing country. As opposed to the country codes, AS numbers allow a finer localization of an IP address. Hence, the use of AS numbers should be more reliable. The application of methods for analysis of time-series of both types of localization information is left for future work.

To conclude, the use and parametrization of all techniques of lowering false positives represents a trade-off between false positives and false negatives, i. e. (in)sensitivity of the intrusion detection and its practical usability.

6.4 Summary

Since we suppose that every detection method is vulnerable to false positives, we studied behaviour of attackers and proposed various methods that attempt to eliminating the false positives. At first, we focused on methods for more precise identification of the source of the attack because injudicious traffic filtering may negatively affect benign users. We proposed one flow-based extension of the existing method of counting NATted hosts and one original method of passive fingerprinting using the size of TCP SYN packets. Then we performed the flow-based and the network-based analysis of attackers' behaviour. The results showed that multiple brute-force attacks are prevalent and the actual attacks are preceded by network reconnaissance. As a result, we proposed four various techniques of making detection more reliable: the sequential and the cumulative detection, the use of methods based on honeypots and external data sources such as blacklists. Using real data from the campus network, we demonstrated that these techniques lower the false positives and allow to deploy the detection methods in a production network.

9. `http://www.muni.cz/ics/services/csirt/tools/phigaro`

Rank	Country	Unique attackers
1	China	29
2	South Korea	10
3	USA	8
4	Russia	5
5	Taiwan	4
6	Poland	4
7	Italy	3
8	Spain	3
9	Romania	2
10	Netherlands	2

Table 6.6: Top 10 countries of attackers' origin (geographical location)

Rank	ASN	Unique attackers
1	4134	12
2	4837	7
3	9318	4
4	4766	4
5	4808	3
6	5617	2
7	4847	2
8	4538	2
9	42116	2
10	31334	2

Table 6.7: Top 10 autonomous system numbers of attackers' origin

Honeypot	TCP scans	TCP SYN scans	TCP SYN/RST scans	UDP scans
H1	164	157	7	81
H2	203	191	12	108
H3	195	183	12	107
H4	202	190	12	109
H5	174	154	20	96

Table 6.8: Numbers and types of network scans destined to five honeypots

Account name	Login attempts
root	104
test	24
admin	13
guest	9
user	7
temp	7
student	6
dino	5
michael	4
martin	4
josefa	4
josef	4

Table 6.9: Top failed login attempts

Service	Time period	Signature-based approach		Sequential detection	
		total	per day	total	per day
SSH	Jun 6, 2011 – Nov 13, 2012	211 837	351.3	196	0.3
RDP	Oct 8, 2012 – Dec 31, 2012	109 997	785.7	5780	41.3

Table 6.10: Numbers of detected attacks on SSH and RDP in 5-minute time windows by the signature-based approach and the sequential detection in total and in average per day

OS	**IP ID**	**ΔTCP SYN**	**ΔTTL**	**ΔSubnet TTL**	**Port Seq.**	**C**
Win XP	60%	100%	100%	80%	60%	100%
XP/Vista	80%	100%	100%	80%	60%	100%
Linux	0%	0%	60%	50%	40%	88%
XP/Linux	60%	100%	100%	80%	60%	100%

Table 6.11: Results of the cumulative NAT detection for various OSes behind NAT by Linux *iptables*. All OSes in default configuration, no NAT hiding enabled, some background traffic from NAT device.

OS	**IP ID**	**ΔTCP SYN**	**ΔTTL**	**ΔSubnet TTL**	**Port Seq.**	**C**
Win XP	0%	0%	0%	80%	40%	88%
XP/Vista	80%	100%	0%	80%	60%	100%
Linux	0%	0%	0%	50%	35%	67.5%
XP/Linux	60%	100%	100%	80%	60%	100%

Table 6.12: Results of the cumulative NAT detection for various OSes behind NAT by Linux *iptables* (same testbed setup as for Table 6.11). No background traffic from the NAT device.

OS	**IP ID**	**ΔTCP SYN**	**ΔTTL**	**ΔSubnet TTL**	**Port Seq.**	**C**
Win XP	0%	0%	0%	0%	40%	40%
XP/Vista	80%	100%	0%	0%	60%	100%
Linux	0%	0%	0%	0%	35%	35%
XP/Linux	60%	100%	0%	0%	60%	100%

Table 6.13: Results of the cumulative NAT detection for various OSes behind NAT by Linux *iptables* (same testbed setup as for Table 6.11). No background traffic from the NAT device. TTL rewriting was enabled.

Service	Attempts
SSH	101
POP	12
FTP	4

Table 6.14: Numbers of login attempts to services provided by the honeynet

Counts	Addresses
1	105
2	51
3	48
4	18
5	7
Total	229

Table 6.15: Numbers of Warden events with an IP address from other networks that was also detected by the SSH sequential detection

Chapter 7

Conclusions

Currently, detection of online brute-force attacks on authentication is nearly always done at host level. In spite of the fact that network-based detection offers advantages such as scalability or better capability of detecting distributed attacks, we were not aware of any network-based or even flow-based detection method. We therefore asked a question whether it is possible to detect online brute-force attacks at flow level in real-time. Although massive attacks can be detected by some existing flow-based intrusion detection systems which use statistical methods, we are also interested in attacks of lower intensity. As a result, we proposed two novel flow-based approaches: *signature-based*, an analogy to pattern matching in deep packet inspection, and *similarity-based* that applies cluster analysis to separate similar flows carrying authentication attempts from common traffic. Both approaches benefit from analyzing bidirectional flows since brute-force attacks consist of interactive communication between two parties. For clear evaluation of methods of brute-force attack detection, we propose the first taxonomy of these attacks from the perspective of network flows. Using this taxonomy, we have evaluated both approaches on SSH and RDP services in the Masaryk University network, a 10 gigabit campus network connecting about 15 000 hosts. We showed that both approaches are computationally feasible if we use simple preprocessing. We thus answered the main question: the flow-based detection is capable of detecting brute-force attacks in real-time and, what is more, is capable to detect the attacks even on services that encrypt communication.

We dealt not only with the attack detection itself but also with eliminating false positives since this is crucial for consecutive attack mitigation. Since a decision based on incorrect results of intrusion detection negatively affects benign users, a fundamental requirement for operational use of the detection is zero false positive rate even with the risk of false negatives. We therefore proposed six methods of lowering false positives. First two methods focus on more precise identification of the source of a detected attack. Both methods analyze extended flow records enriched by non-standard packet features, namely the IP ID and TTL values and the size of TCP packets with only the SYN flag set. The other four methods are based on results of our study of attackers' behaviour. We found out that multiple brute-force attacks are prevalent, confirmed the general assumptions that actual brute-force attacks are preceded by network reconnaissance, and the majority of attacks is still non-targeted and thus hit more than one network. Consequently, we proposed methods that take into account i) a time factor of the whole attack, ii) correlation of results of other methods, iii) accesses of the attacker to honeypots, and iv) external data sources related to attacks and attackers. Using input data from the campus network, we then demonstrated that these methods enables lowering the false positives.

The signature-based approach to detection of brute-force attacks on SSH and RDP with methods of lowering false positives had been already deployed in routine operation by CSIRT-MU, Computer Security Incident Response Team of Masaryk University, in 2010 and

2012, respectively. Since the time, thousands of attacks with almost zero false positive rate were mitigated and reported to Warden, a Czech academic early warning system. In addition, the prototype was transferred to a university spin-off company and became a part of a successfully sold product.

7.1 Future Work

We identified future work in the following five various areas.

Export and storage of flow information using IPFIX Although the IPFIX protocol was standardized by the IETF in 2008, there is still a lack of exporters and collectors with full IPFIX support. However, IPFIX allows to natively export bidirectional flows and set timestamps of the flow start and end with a nanosecond resolution. The former offloads biflow acquisition from the analysis layer to the flow acquisition and data storage layer, and enables various flow-based detection methods to utilize the natural bidirectional view of network traffic. The latter is very important for detection methods which rely on accurate flow timestamps. We have already encountered this limitation of the NetFlow format that provides only millisecond resolution.

Application of clustering in flow-based detection For the similarity-based approach, we use available volume characteristics of biflows, chose DBSCAN and the Euclidean distance metric from a large variety of options (see Subsection 5.2.2). As future work, we left the investigation of the impact of changing these components (parameters) on the detection results and performance. Though, a possibility of using packet features extracted from the application layer inherently depends on development of both the IPFIX exporters and collectors.

Ground truth data and infrastructure for evaluation Although there are first efforts to create ground truth data sets for flow-based intrusion detection [73], there is still a lack of them. However, contemporary data sets consistently obtained in various types and sizes of networks are essential for evaluation and benchmarking of flow-based detection methods. Another option for repeatable evaluation is to simulate attacks in a laboratory testbed and capture generated network traffic. Since such simulation is laborious task, we started to develop Cybernetic Proving Ground, a virtualized environment that enables researchers to rapidly perform repeatable attack scenarios.

Exploitation of other data sources Besides improving detection based on analysis of network traffic, improving detection using both internal and external data sources is another challenge. Concerning honeypots, we currently focus on attracting more attackers to (a small number of) honeypots using a "network funnel" that would route traffic destined to various unused parts of the administered network to honeypots. We also suggest to deploy other honeypots recommended in [28] to extend a portfolio of efficient traps. Concerning external data sources, we fully agree with the authors of [27] that there is a lack of common formats for exchange information about detected attacks and security incidents, and lack of automation and correlation (partly even due to legal and organisational issues).

Visualization of network traffic and attacks Effective visualization of a large amount of network data considerably speeds up not only a design of detection methods but also the operational use of the designed methods. This hypothesis is also supported by [27].

Bibliography

[1] Elke Achtert, Hans-Peter Kriegel, and Arthur Zimek. ELKI: A Software System for Evaluation of Subspace Clustering Algorithms. In Bertram Ludäscher and Nikos Mamoulis, editors, *Scientific and Statistical Database Management*, volume 5069 of *Lecture Notes in Computer Science*, pages 580–585. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-69476-2. URL http://dx.doi.org/10.1007/978-3-540-69497-7_41.

[2] Elke Achtert, Sascha Goldhofer, Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. Evaluation of Clusterings – Metrics and Visual Support. *Data Engineering, International Conference on*, 0:1285–1288, 2012. ISSN 1084-4627. URL http://doi.ieeecomputersociety.org/10.1109/ICDE.2012.128.

[3] E. Alata, V. Nicomette, M. Kaaniche, M. Dacier, and M. Herrb. Lessons learned from the deployment of a high-interaction honeypot. In *EDCC '06: Proceedings of the Sixth European Dependable Computing Conference*, pages 39–46, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2648-9. URL http://dx.doi.org/10.1109/EDCC.2006.17.

[4] D. Barr. Common DNS Operational and Configuration Errors. RFC 1912 (Informational), February 1996. URL http://www.ietf.org/rfc/rfc1912.txt.

[5] Daniel J. Barrett and Richard E. Silverman. *SSH, The Secure Shell: The Definitive Guide*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001. ISBN 0596000111.

[6] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, SIGMOD '90, pages 322–331, New York, NY, USA, 1990. ACM. ISBN 0-89791-365-5. doi: 10.1145/93597.98741. URL http://doi.acm.org/10.1145/93597.98741.

[7] Steven M. Bellovin. A technique for counting NATted hosts. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 267–272, New York, NY, USA, 2002. ACM. ISBN 1-58113-603-X. URL http://doi.acm.org/10.1145/637201.637243.

[8] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975. ISSN 0001-0782. URL http://doi.acm.org/10.1145/361002.361007.

[9] E. Boschi, L. Mark, J. Quittek, M. Stiemerling, and P. Aitken. IP Flow Information Export (IPFIX) Implementation Guidelines. RFC 5153 (Informational), April 2008. URL http://www.ietf.org/rfc/rfc5153.txt.

[10] Daniela Brauckhoff, Bernhard Tellenbach, Arno Wagner, Martin May, and Anukool Lakhina. Impact of packet sampling on anomaly detection metrics. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 159–164, New York, NY, USA, 2006. ACM. ISBN 1-59593-561-4. URL http://doi.acm.org/10.1145/1177080.1177101.

[11] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: Identifying Density-Based Local Outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 93–104, 2000.

[12] Jake D. Brutlag. Aberrant Behavior Detection in Time Series for Network Monitoring. In *LISA '00: Proceedings of the 14th USENIX conference on System administration*, pages 139–146, Berkeley, CA, USA, 2000. USENIX Association.

[13] Elie Bursztein. Time has something to tell us about network address translation. In Úlfar Erlingsson and Andrei Sabelfeld, editors, *Proceedings of the 12th Nordic Workshop on Secure IT Systems (NordSec'07)*, Reykjavik, Iceland, October 2007. URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/Bur-nordsec07.pdf.

[14] Marco Canini, Damien Fay, David J. Miller, Andrew W. Moore, and Raffaele Bolla. Per Flow Packet Sampling for High-Speed Network Monitoring. In *Proceedings of the First International Conference on Communication Systems and Networks (COMSNETS'09)*, January 2009.

[15] Dan Carr, ported by Nicholas Lewin-Koh, and Martin Maechler. *hexbin: Hexagonal Binning Routines*, 2013. URL http://CRAN.R-project.org/package=hexbin. R package version 1.26.1.

[16] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009. ISSN 0360-0300. doi: 10.1145/1541880.1541882. URL http://doi.acm.org/10.1145/1541880.1541882.

[17] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), October 2004. URL http://www.ietf.org/rfc/rfc3954.txt.

[18] B. Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard), January 2008. URL http://www.ietf.org/rfc/rfc5101.txt.

[19] M. I. Cohen. Source attribution for network address translated forensic captures. *Digital Investigation*, 5(3-4):138–145, 2009. URL http://dblp.uni-trier.de/db/journals/di/di5.html#Cohen09.

[20] Hewlett-Packard Development Company. Top Cyber Security Risks Threat Report for 2010. http://dvlabs.tippingpoint.com/toprisks2010, 2010.

[21] L. Daigle. WHOIS Protocol Specification. RFC 3912 (Draft Standard), September 2004. URL http://www.ietf.org/rfc/rfc3912.txt.

[22] R. Danyliw, J. Meijer, and Y. Demchenko. The Incident Object Description Exchange Format. RFC 5070 (Proposed Standard), December 2007. URL http://www.ietf.org/rfc/rfc5070.txt. Updated by RFC 6685.

[23] H. Debar, D. Curry, and B. Feinstein. The Intrusion Detection Message Exchange Format (IDMEF). RFC 4765 (Experimental), March 2007. URL http://www.ietf.org/rfc/rfc4765.txt.

[24] Loh Chin Choong Desmond, Cho Chia Yuan, Tan Chung Pheng, and Ri Seng Lee. Identifying unique devices through wireless fingerprinting. In *WiSec '08: Proceedings of the first ACM conference on Wireless network security*, pages 46–55, New York, NY, USA, 2008. ACM. ISBN 978-1-59593-814-5. URL http://doi.acm.org/10.1145/1352533.1352542.

[25] Alexandre Dulaunoy. BGP Ranking: Security Ranking of Internet Service Providers. February 2011. URL http://www.terena.org/activities/tf-csirt/meeting32/dulaunoy-bgpranking.pdf.

[26] Alexandre Dulaunoy. Fast and Scalable Ranking Scheme for Internet Resources – Current Usage and What's the Future? September 2011. URL http://www.terena.org/activities/tf-csirt/meeting34/dulaunoy-ranking.pdf.

[27] ENISA. Proactive Detection of Security Incidents. Technical report, December 2011. URL http://www.enisa.europa.eu/activities/cert/support/proactive-detection/proactive-detection-report.

[28] ENISA. Proactive Detection of Security Incidents: Honeypots. Technical report, November 2012. URL http://www.enisa.europa.eu/activities/cert/support/proactive-detection/proactive-detection-of-security-incidents-II-honeypots.

[29] L. Ertöz, E. Eilertson, A. Lazarevic, P. Tan, V. Kumar, J. Srivastava, and P. Dokas. *Next Generation Data Mining*, chapter The MINDS — Minnesota Intrusion Detection System. MIT Press, Boston, USA, 2004.

[30] Martin Ester, Hans-Peter Kriegel, Jörg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of The Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231. AAAI Press, 1996.

[31] Fyodor Dostoevsky. Remote OS Detection via TCP/IP Fingerprinting (2nd Generation). URL http://insecure.org/nmap/osdetect/.

[32] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1–2):18 – 28, 2009. ISSN 0167-4048. URL http://www.sciencedirect.com/science/article/pii/S0167404808000692.

[33] Sharon Goldberg and Jennifer Rexford. Security vulnerabilities and solutions for packet sampling. In *Proceedings of the IEEE Sarnoff Symposium*, 2007. Invited paper.

[34] Peter N. M. Hansteen. If We Go One Attempt Every Ten Seconds, We're Under The Radar, 2012. URL http://bsdly.blogspot.cz/2012/04/if-we-go-one-attempt-every-ten-seconds.html. Retrieved online April 7, 2012.

[35] Shane Harris. The Cyberwar Plan. *National Journal*, 2009. Retrieved online December 26, 2009 at http://www.nationaljournal.com/njmagazine/cs_20091114_3145.php.

[36] J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an Autonomous System (AS). RFC 1930 (Best Current Practice), March 1996. URL http://www.ietf.org/rfc/rfc1930.txt.

[37] Laurens Hellemons, Luuk Hendriks, Rick Hofstede, Anna Sperotto, Ramin Sadre, and Aiko Pras. SSHCure: A Flow-Based SSH Intrusion Detection System. In *Dependable Networks and Services*, volume 7279 of *Lecture Notes in Computer Science*, pages 86–97. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-30632-7. URL http://dx.doi.org/10.1007/978-3-642-30633-4_11.

[38] Christian Hennig. *fpc: Flexible procedures for clustering*, 2013. URL http://CRAN.R-project.org/package=fpc. R package version 2.1-5.

[39] R. Hofstede, I. Drago, R. Sperotto, A. Sadre, and A. Pras. Measurement Artifacts in NetFlow Data. In *Proceedings of the Passive and Active Measurement conference, PAM'13*, Hong Kong, China, 2013.

[40] Alfred Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1:69–91, 1985. ISSN 0178-2789. URL http://dx.doi.org/10.1007/BF01898350.

[41] IronGeek. Osfuscate: Change your windows os tcp/ip fingerprint to confuse p0f, networkminer, ettercap, nmap and other os detection tools, 2008. URL http://www.irongeek.com/i.php?page=security/osfuscate-change-your-windows-os-tcp-ip-fingerprint-to-confuse-\p0f-networkminer-ettercap-nmap-and-other-os-detection-tools.

[42] Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recogn. Lett.*, 31(8):651–666, June 2010. ISSN 0167-8655. URL http://dx.doi.org/10.1016/j.patrec.2009.09.011.

[43] I.T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer, 2002. ISBN 9780387954424.

[44] Piotr Kijewski and Paweł Pawliński. Proactive Detection and Automated Exchange of Network Security Incidents. 2012. URL http://www.cert.pl/PDF/MP-IST-111-18.pdf.

[45] Tadayoshi Kohno, Andre Broido, and K. C. Claffy. Remote Physical Device Fingerprinting. *IEEE Trans. Dependable Secur. Comput.*, 2(2):93–108, 2005. ISSN 1545-5971. URL http://dx.doi.org/10.1109/TDSC.2005.26.

[46] Vojtech Krmicek, Jan Vykopal, and Radek Krejci. Netflow Based System for NAT Detection. In *Co-Next Student Workshop '09: Proceedings of the 5th international student workshop on Emerging networking experiments and technologies*, pages 23–24, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-751-6. URL http://doi.acm.org/10.1145/1658997.1659010.

[47] Vojtěch Krmíček. *Hardware-Accelerated Anomaly Detection in High-Speed Networks*. Ph.D. Thesis, 2011. URL http://is.muni.cz/th/51640/fi_d/.

[48] W. Kumari and D. McPherson. Remote Triggered Black Hole Filtering with Unicast Reverse Path Forwarding (uRPF). RFC 5635 (Informational), August 2009. URL http://www.ietf.org/rfc/rfc5635.txt.

[49] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 219–230, New York, NY, USA, 2004. ACM. ISBN 1-58113-862-8. URL http://doi.acm.org/10.1145/1015467.1015492.

[50] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 217–228, New York, NY, USA, 2005. ACM. ISBN 1-59593-009-4. URL http://doi.acm.org/10.1145/1080091.1080118.

[51] S. Leinen. Evaluation of Candidate Protocols for IP Flow Information Export (IPFIX). RFC 3955 (Informational), October 2004. URL http://www.ietf.org/rfc/rfc3955.txt.

[52] Jianning Mai, Ashwin Sridharan, Chen nee Chuah, Hui Zang, and Tao Ye. Impact of Packet Sampling on Portscan Detection. *IEEE Journal on Selected Areas in Communications*, 24:2285–2298, December 2006. ISSN 0733-8716.

[53] John McHugh. Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, 2000. URL http://doi.acm.org/10.1145/382912.382923.

[54] Alfred J. Menezes, Paul C. Van Oorschot, Scott A. Vanstone, and R. L. Rivest. *Handbook of Applied Cryptography*, chapter Identification and Entity Authentication. CRC Press, 1997.

[55] Jelena Mirkovic and Peter Reiher. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, 2004. ISSN 0146-4833. URL http://doi.acm.org/10.1145/997150.997156.

[56] Dan Pelleg and Andrew Moore. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 727–734, San Francisco, 2000. Morgan Kaufmann.

[57] Peter Phaal. Detecting NAT Devices using sFlow, 2003. URL http://www.sflow.org/detectNAT/.

[58] P. Phaal, S. Panchen, and N. McKee. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. RFC 3176 (Informational), September 2001. URL http://www.ietf.org/rfc/rfc3176.txt.

[59] J. Postel. Internet Protocol. RFC 791 (INTERNET STANDARD), September 1981. URL http://www.ietf.org/rfc/rfc791.txt. Updated by RFCs 1349, 2474.

[60] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. URL http://www.ietf.org/rfc/rfc793.txt. Updated by RFCs 1122, 3168, 6093, 6528.

[61] Aiko Pras, Anna Sperotto, Giovane C.M. Moura, Idilio Drago, Rafael Barbosa, Ramin Sadre, Ricardo Schmidt, and Rick Hofstede. Attacks by "Anonymous" WikiLeaks Proponents not Anonymous, December 2010. URL http://doc.utwente.nl/75331/.

[62] J. Quittek, T. Zseby, B. Claise, and S. Zander. Requirements for IP Flow Information Export (IPFIX). RFC 3917 (Informational), October 2004. URL http://www.ietf.org/rfc/rfc3917.txt.

[63] J. Quittek, S. Bryant, B. Claise, P. Aitken, and J. Meyer. Information Model for IP Flow Information Export. RFC 5102 (Proposed Standard), January 2008. URL http://www.ietf.org/rfc/rfc5102.txt. Updated by RFC 6313.

[64] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. URL http://www.R-project.org/. ISBN 3-900051-07-0.

[65] D. Ramsbrock, R. Berthier, and Michel Cukier. Profiling Attacker Behavior Following SSH Compromises. In *Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP International Conference on*, pages 119–124, 2007. doi: 10.1109/DSN.2007.76.

[66] Martin Rehak, Michal Pechoucek, Karel Bartos, Martin Grill, Pavel Celeda, and Vojtech Krmicek. CAMNEP: An intrusion detection system for high-speed networks. *Progress in Informatics*, (5):65–74, March 2008. ISSN 1349-8614. URL http://www.nii.ac.jp/pi/n5/5_65.pdf.

[67] Deepayan Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer, New York, 2008. URL http://lmdvr.r-forge.r-project.org. ISBN 978-0-387-75968-5.

[68] Karen Scarfone and Peter Mell. Guide to Intrusion Detection and Prevention Systems (IDPS). NIST, 2007. Recommendations of the National Institute of Standards and Technology. Retrieved online December 26, 2009 at http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf.

[69] Erich Schubert. Benchmarking with ELKI, January 2013. URL http://elki.dbs.ifi.lmu.de/wiki/Benchmarking.

[70] Matthias Schubert, Zhanna Melnikova-Albrecht, and Rainer Holzmann. optics_dbScan: The OPTICS and DBScan clustering algorithms, January 2013. URL http://weka.sourceforge.net/packageMetaData/optics_dbScan/1.0.3.html.

[71] C. Seifert. Analyzing Malicious SSH Login Attempts, 2006. Retrieved online January 3, 2010 at http://www.securityfocus.com/infocus/1876.

[72] Y. Shafranovich, J. Levine, and M. Kucherawy. An Extensible Format for Email Feedback Reports. RFC 5965 (Proposed Standard), August 2010. URL http://www.ietf.org/rfc/rfc5965.txt. Updated by RFC 6650.

[73] A. Sperotto, R. Sadre, D. F. van Vliet, and A. Pras. A Labeled Data Set For Flow-based Intrusion Detection. In *Proceedings of the 9th IEEE International Workshop on IP Operations and Management, IPOM 2009, Venice, Italy*, volume 5843 of *Lecture Notes in Computer Science*, pages 39–50. Springer Verlag, October 2009.

[74] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. An Overview of IP Flow-Based Intrusion Detection. *Communications Surveys Tutorials, IEEE*, 12(3):343 –356, 2010. ISSN 1553-877X. doi: 10.1109/SURV.2010.032210.00054.

[75] Anna Sperotto, Ramin Sadre, and Aiko Pras. Anomaly Characterization in Flow-Based Traffic Time Series. In *Proceedings of the 8th IEEE international workshop on IP Operations and Management*, IPOM '08, pages 15–27, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-87356-3. URL http://dx.doi.org/10.1007/978-3-540-87357-0_2.

[76] Anna Sperotto, Ramin Sadre, Pieter-Tjerk Boer de, and Aiko Pras. Hidden Markov Model modeling of SSH brute-force attacks. In *Integrated Management of Systems, Services, Processes and People in IT*, volume 5841/2 of *Lecture Notes in Computer Science 5841*, pages 164–176, Berlin, October 2009. Springer Verlag. URL http://doc.utwente.nl/68309/.

[77] Lance Spitzner. Honeypots: Definitions and value of honeypots. May 2003. URL http://www.tracking-hackers.com/papers/honeypots.html.

[78] P. Srisuresh and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). RFC 3022 (Informational), January 2001. URL http://www.ietf.org/rfc/rfc3022.txt.

[79] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007. URL http://www.ietf.org/rfc/rfc4960.txt. Updated by RFCs 6096, 6335.

[80] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. ISBN 0321321367.

[81] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *Proceedings of the Second IEEE international conference on Computational intelligence for security and defense applications*, CISDA'09, pages 53–58, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-3763-4. URL http://dl.acm.org/citation.cfm?id=1736481.1736489.

[82] J. L. Thames, R. Abler, and D. Keeling. A Distributed Active Response Architecture for Preventing SSH Dictionary Attacks. In *IEEE Southeastcon 2008*, pages 84–89, 2008.

[83] B. Trammell and E. Boschi. Bidirectional Flow Export Using IP Flow Information Export (IPFIX). RFC 5103 (Proposed Standard), January 2008. URL http://www.ietf.org/rfc/rfc5103.txt.

[84] Jan Vykopal. A flow-level taxonomy and prevalence of brute force attacks. In *Advances in Computing and Communications*, pages 666–675, Berlin, 2011. ISBN 978-3-642-22714-1. URL http://dx.doi.org/10.1007/978-3-642-22714-1_69.

[85] Jan Vykopal and Tomáš Mrázek. Packet Capture Benchmark on 1 GE, 2008. CESNET technical report 22/2008. Retrieved online December 26, 2009 at http://www.cesnet.cz/doc/techzpravy/2008/packet-capture-benchmark/.

[86] Jan Vykopal and Martin Vizváry. Flow-based detection of RDP brute-force attacks. In *Security and Protection of Information 2013, Proceeding of the Conference*, Brno, Czech Republic, 2013. University of Defence. To appear.

[87] Jan Vykopal, Tomas Plesnik, and Pavel Minarik. Network-Based Dictionary Attack Detection. In *International Conference on Future Networks*, pages 23–27, Los Alamitos, CA, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3567-8. URL http://doi.ieeecomputersociety.org/10.1109/ICFN.2009.36.

[88] Jan Vykopal, Tomáš Plesník, and Pavel Minařík. Validation of the Network-based Dictionary Attack Detection. In *Security and Protection of Information 2009, Proceeding of the Conference*, pages 128–136, Brno, Czech Republic, 2009. University of Defence. ISBN 978-80-7231-641-0.

[89] Jan Vykopal, Martin Drašar, and Philipp Winter. *Flow-based Brute-force Attack Detection*, pages 41–51. Fraunhofer Research Institution AISEC, Garching near Muenchen, 2013. ISBN 978-3-8396-0474-8.

[90] Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. *SIGCOMM Comput. Commun. Rev.*, 35(4):169–180, 2005. ISSN 0146-4833. URL http://doi.acm.org/10.1145/1090191.1080112.

[91] Kim Zetter. Weak Password Brings 'Happiness' to Twitter Hacker, 2009. URL http://www.wired.com/threatlevel/2009/01/professed-twitt/. Retrieved online January 3, 2010.

[92] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search – The Metric Space Approach*, volume 32. Springer, 2006.

[93] Jian Zhang and Andrew W. Moore. Traffic Trace Artifacts due to Monitoring Via Port Mirroring. In *Proceedings of the Fifth IEEE/IFIP E2EMON*, pages 1–8, 2007. ISBN 1-4244-1289-7. URL http://www.cl.cam.ac.uk/~awm22/publications/zhang2007traffic.pdf.

[94] H. Zimmermann. OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1094702.

[95] Pavel Čeleda, Radek Krejčí, Jan Vykopal, and Martin Drašar. Embedded Malware – An Analysis of the Chuck Norris Botnet. In *European Conference on Computer Network Defense*, pages 3–10, Los Alamitos, CA, 2010. ISBN 978-1-4244-9377-7. URL `http://2010.ec2nd.org/`.

Appendix A

List of Author's Publications

Book Chapters

1. Vykopal, Jan – Drašar, Martin – Winter, Philipp. Flow-based Brute-force Attack Detection. In *Advances in IT Early Warning*. Garching near Muenchen : Fraunhofer Research Institution AISEC, 2013. ISBN 978-3-8396-0474-8, pages 41–51. [Author's contribution: 33 %]

Conference Papers

1. Vykopal, Jan. A Flow-Level Taxonomy and Prevalence of Brute Force Attacks. In *Advances in Computing and Communications*. Berlin : Springer Berlin Heidelberg, 2011. ISBN 978-3-642-22714-1, pages 666–675. [Author's contribution: 100 %]

2. Vykopal, Jan – Plesník, Tomáš – Minařík, Pavel. Network-based Dictionary Attack Detection. In *Proceedings of International Conference on Future Networks (ICFN 2009)*. Los Alamitos, CA, USA: IEEE Computer Society, 2009. ISBN 978-0-7695-3567-8, pages 23–27. [Author's contribution: 33 %]

3. Vykopal, Jan – Plesník, Tomáš – Minařík, Pavel. Validation of the Network-based Dictionary Attack Detection. In *Security and Protection of Information 2009, Proceeding of the Conference*. Brno: University of Defence, 2009. ISBN 978-80-7231-641-0, pages 128–136. [Author's contribution: 50 %]

4. Vizváry, Martin – Vykopal, Jan. Flow-based detection of RDP brute-force attacks. To appear in *Security and Protection of Information 2013, Proceeding of the Conference*. Brno: University of Defence, 2013. [Author's contribution: 30 %]

5. Krmíček, Vojtěch – Vykopal, Jan – Krejčí, Radek. Netflow Based System for NAT Detection. In *Co-Next Student Workshop '09: Proceedings of the 5th international student workshop on Emerging networking experiments and technologies*. New York, NY, USA : ACM, 2009. ISBN 978-1-60558-751-6, pages 23–24. Rome, Italy. [Author's contribution: 40 %]

6. Minařík, Pavel – Vykopal, Jan – Krmíček, Vojtěch. Improving Host Profiling with Bidirectional Flows. In *Proceedings of International Symposium on Secure Computing (SecureCom09)*. Vancouver, Canada: IEEE Computer Society, 2009. ISBN 978-0-7695-3823-5, pages 231–237. [Author's contribution: 15 %]

7. Čeleda, Pavel – Krejčí, Radek – Vykopal, Jan – Drašar, Martin. Embedded Malware – An Analysis of the Chuck Norris Botnet. In *European Conference on Computer Network Defense*. Vyd. 1. Los Alamitos, CA : IEEE Computer Society, 2010. ISBN 978-1-4244-9377-7, pages 3–10. 2010, Berlin, Germany. [Author's contribution: 15 %]

Workshop Papers

1. Drašar, M. – Vykopal, Jan. Bruteforcing in the Shadows – Evading Automated Detection. FloCon 2012, Austin, Texas, USA. [Author's contribution: 50 %]

Software

1. Vykopal, Jan. SimFlow – a similarity-based detection of brute-force attacks. 2013. `http://www.ics.muni.cz/~vykopal/simflow/` [Author's contribution: 100 %]
2. Vykopal, Jan – Vizváry, Martin. SSH brute-force attack detection plugin. 2012. `http://www.muni.cz/ics/research/projects/4622/web/ssh_brute_force` [Author's contribution: 90 %]
3. Vizváry, Martin – Vykopal, Jan. RdpMonitor – RDP authentication attack detection plugin. 2012. `http://www.muni.cz/ics/services/csirt/tools/rdpmonitor` [Author's contribution: 10 %]
4. Husák, Martin – Vykopal, Jan. Honeynet monitoring plugin. 2012. `http://www.muni.cz/ics/services/csirt/tools/honeyscan` [Author's contribution: 30 %]
5. Krmíček, Vojtěch – Vykopal, Jan – Plesník, Tomáš – Ružička, Andrej – Čeleda, Pavel – Trunečka, Michal. NetFlow-based NAT detection module. 2009. `http://www.muni.cz/ics/research/projects/4622/web/natdet` [Author's contribution: 20 %]