CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF ELECTRICAL ENGINEERING

DIPLOMA THESIS

2018

MICHAL STANKE

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science

DIPLOMA THESIS AGREEMENT

Student: Stanke Michal

Study programme: Open Informatics
Specialisation: Software Engineering

Title of Diploma Thesis: Identifying similarities in malicious network behaviour

Guidelines:

1. Study and alanyze existing algorithms for identifying string and behaviour similarities.
2. In "real-world" data related to detected incidents, identify similarities between different types of malicious behaviour. Focus on Command and Control, adware and click fraud.
3. Implement at least two algorithms and in the automated detection framewok.
4. Determine and measure in % if and how much can be the data describing incidents reduced using the automated algorithms with preserving the information value.

Bibliography/Sources:

[1] J. Jusko, M. Rehak, J. Stiborek, J. Kohout and T.Pevny, „Using Behavioral Similarity for Botnet Command-and-Control Discovery," IEEE Inteligent Systems, vol. 31, issue 5, 29 Sep. 2016, pages 16-22.
[2] C. Wickramaarachchi, M. Frincu, P. Small and V. Prasanna, „Fast parallel algorithm for unfolding of communities in large graphs," IEEE High Performance Extreme Computing Conference (HPEC) 2014, Sep 2014.
[3] P. Chopade, J. Zhan, M. Bikdash, „Node attributes and edge structure for large-scale big data network analytics and community detection ," IEEE International Symposium on Technologies for Homeland Security (HST) 2015, April 2015.
[4] J. Shawe-Taylor and N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge Univ. Press, 2004.

Supervisor: Ing. Martin Rehák, Ph.D.

Valid until the end of the summer semester of academic year 2017/2018

prof. Dr. Michal Pěchouček, MSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, February 6, 2017

Declaration

I hereby declare I have written the submitted thesis myself and I quoted all used sources of information in accord with the Methodical instructions about ethical principles for writing academic theses.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 8. ledna 2018 ……

Acknowledgement

I would like to thank the supervisor of this thesis, Ing. Martin Rehák, Ph.D., for the opportunity to work on this project, for his motivation, guidance, feedback and support.

I would also like to thank my family, fellow students and friends, especially Radka Síbrová, for their continuous support and encouragement for the whole time of my studies.

Abstract

The goal of this thesis is to design and implement a framework to detect similarities in classified malicious network traffic, using and combining at least two similarity metrics. The results will be further used to reduce the data necessary for presenting individual security incidents without losing the information value.

Keywords: network traffic, web flow, similarity, malware, malicious software, security incident, cyber attack

Abstrakt

Cílem této práce je návrh a implementace frameworku pro identifikaci podobností v rozpoznaném škodlivém síťovém provozu, s použitím minimálně dvou metrik podobnosti. Výsledky budou dále využity pro snížení množství dat potřebných při prezentaci jednotlivých bezpečnostních incidentů bez ztráty informační hodnoty.

Klíčová slova: síťový provoz, webový tok, podobnost, malware, škodlivý software, bezpečnostní incident, kybernetický útok

Table of contents

Figures

1 Introduction

The internet is growing every day. We tend to use internet and cloud services not just for browsing the web and consuming content, but also for creating new content and experience, communication with people on the other side of our planet, but even as assistant in our households and everyday life. It's highly convenient and thanks to mobile devices also easily portable.

There are also new internet markets emerging in underdeveloped countries, which includes the biggest countries in the world like Brazil, China or India, whose users want to enjoy the same experience as others very soon after they get the internet connection.

The overall internet traffic has grown over 3 times in the last five years, and between years 2015 and 2016 it was by 30%. The growth was even higher for traffic generated by mobile devices, which almost doubled between these two years. [1]

But the popularity of the internet brings not just growth in the number of users and amount of data transferred, but also the variety and diversity of data is rapidly growing. Thanks to the omnipresent mobile devices, we collect data not just knowingly, but there are new types of data temporary or permanently recorded and stored in our devices. Mobile devices are getting smarter with many sensors for tracking our precise position, take pictures or record sounds and video. Thanks to them we are connecting our real lives to the internet.

The amount, diversity and sensitivity of data on the internet, and our connection to it, raises the cost of the data and creates new opportunities when these are lost, exfiltrated or just blocked from being accessed until ransom is payed. [2] The protection against these online threats is becoming highly important.

Because of our everyday connection to internet, many threats are distributed online too. Their identification is hard to achieve on itself, but even after the threats are found, we need to understand them – both those we have seen before and know the steps to mitigate their impact, and the new ones we need to analyze first.

This thesis aims to ease the analysis of detected threats by finding similarities in the detected malicious behavior in the network to better understand the threat behavior in time.

1.1 Motivation

The number and magnitude of online threats is growing with the internet traffic. [3] And so does the importance of their in-time identification, correct analysis and proper mitigation. The statistic from over 12,000 respondents in June 2015 showed that 1/3 of them paid for repairs. The percentage of respondents stating they have paid to mitigate some malware infection consequences is actually higher than people purchasing some antivirus software or tools to clean their computers. [4]

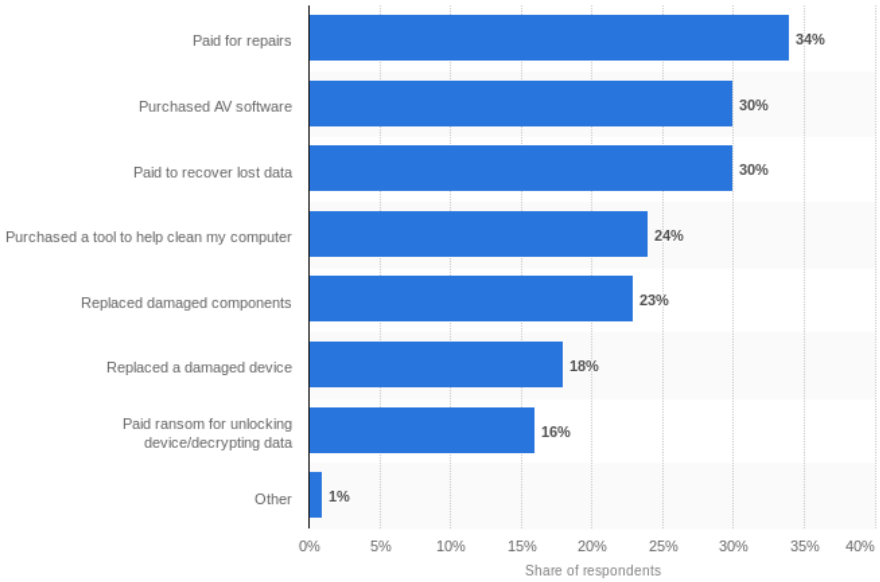

Figure 1: Financial costs incurred by malware incidents [4]

According to Honeywell, the cost of an industry cyber attack resolution goes over 20 000 USD per day, while the average time to resolve malicious attack is almost two months. Deploying a security solution system can save a company up to 3 millions USD every year. [5] The median financial costs of cyber crime for U. S. companies in 2014 and 2015 is estimated around 10 million USD, with the maximum over 60 million USD. [6]

According to the annual business report on csoonline.com, by 2021 the damage of cyber crime fight exceed 6 trillion USD. Global spending on the cyber security services and products is predicted to exceed 1 trillion USD for the years 2017 – 2021. [7]

These numbers only document how severe the problem of cyber crime is and what danger it means for both individuals personal data and business secrets.

On the field of cyber security, it’s important both to correctly identify malicious activities in the network, but also to allow analysts to analysis them and decide about the appropriate countermeasures. Having the most accurate detection without any false positives can be useless when the time to action is too long because of missing threat context and understanding, what is actually happening and how the threat evolves in time.

1.2 Assignment

The assignment of this thesis is to identify similarities in malicious network behavior to ease their further analysis and mitigation. The assignment will be now further analyzed to define what to include in this thesis, develop necessary software and gather and interpret data from designed experiments.

The assignment is divided into several tasks. From these tasks the structure of the thesis and its chapter will be deigned.

1.2.1 Study and analyze existing algorithms for identifying string and behaviour similarities.

This introduction part is to research the current approaches and algorithms for identification of similarities in network behaviour. It is important to briefly describe several solutions and identify present trends in this area.

1.2.2 In “real-world” data related to detected incidents, identify similarities between different types of malicious behaviour. Focus on Command and Control, adware and click fraud.

Based on the knowledge gained in the previous tasks, next step is to use some of the algorithms and apply them to find similarities in malicious behaviour. It’s important these algorithms need to work for malicious behaviour in general, not only specific types.

1.2.3 Implement at least two algorithms and in the automated detection framework.

In the next stage use multiple algorithms implemented in the previous step and combine them in an automated detection framework. Using multiple algorithms should greatly improve discovery of similarities across all types of malicious behaviour and allow to group data into relevant partitions based on the similarities.

1.2.4 Determine and measure in % if and how much can be the data describing incidents reduced using the automated algorithms with preserving the information value.

The final framework implementation should be benchmarked, how much the behaviour data can be reduced into partitions based on the uncovered similarities, without loosing too much information by putting unrelated behaviour into same partitions.

1.3 Important terms

1.3.1 Cyber threat

A cyber threat or cyber attack is an offensive and illegal activity by either individual or a group (e.g. company or nation) targeting a computer system or device. The usual goal

is to profit from stealing or altering data, hijacking data for ransom or device to use for further cyber attacks, or to cause loss or damage. [8]

There are many know kinds of attacks, ranging from installing various less or more sophisticated malware (malicious software) for different usage to techniques targeting users with social engineering.

1.3.2 Malicious software

Malicious software, shortly malware, is a general term denoting many form of malicious and intrusive software. It can refer to computer viruses and worms, trojans, spyware, adware or ransomware.

In the present times, most of the malware is designed to monitor and spy users or gain control over the infected devices. [3] Later it can be used to execute some specific code or perform other actions. The ability, when the attacker has control over the infected devices remotely is called “command and control”. [9]

Another widespread type of malicious software is adware. Adware is used by attackers to generate revenue by showing online advertisements to the users. It can open new windows containing ads, inject the ads into the operating system, internet browser or visited websites. The term adware is sometimes used to describe ad-supported software, which is free of charge and the authors gets revenue from showing ads in the application.

Closely related to adware is click fraud. A lot of advertisement systems use PPC[1] model. Click fraud is designed to click ads while imitating real users to generate revenue for the ad space holder.

1.3.3 Malicious behaviour

When a cyber attack is in progress we can observe various indications. The malware needs to be distributed, executed on the targeted machine or device and most likely it will send some data back. These activities can be observed from outside as unusual network traffic (volume, types of communication, connection destinations or other), and directly on the machine as unknown processes or unexpected memory or data changes.

This thesis focuses on the behaviour observed in the network from outside, without need of any knowledge from the device or user.

1 Pay per click

1.3.4 Behaviour similarity

The regular network traffic have its specifics. It can be time, when the particular user is active, the traffic volume he or she transfers to and from the internet, server the connections are established to etc. Malicious behaviour is likely to differ from the user activity, for example it can be distributed or communicate with unknown or previously unseen origins. Also the same threats will report similar characteristics. The assumption is that discovering similarities in malicious behaviour can ease their analysis.

2 Analysis

In this chapter I analyze in more details topics related to this thesis. In the first part, the current situation on the internet is introduced, how it evolves and changes in the recent years. This will be followed by how the "black hats"[2] do react on the internet trends, how the threats are evolving and changing to match the internet users. Then I will briefly mention some solutions for preventing and fighting against the cyber attacks. The last section is a deeper analysis of current trends in discovery of malicious behaviour and how the information can be used by threat analysts.

2.1 Growth of the internet

The vision of internet, a world wide network, first appeared in 1950s. Until 1980s there were several separated networks like ARPANET or BITNET, but none of them was global. Very important technologies and standards for the internet development as a global and decentralize communication resource were TCP/IP[3], specifying whole end-to-end communication across the internet network, DNS[4], translating domain names people understand (example.com) to IP addresses machines understand to identify and locate other computers in the network (127.0.0.1), or SMTP protocol for email transfer. [10]

Today the most known service of the internet is the WWW[5] invented in CERN in 1989 and opened to public in 1991. WWW is decentralized, similarly to the internet. It uses hyperlinks to navigate between web pages hosted on either the same or different servers. The pages are identified by URLs (uniform resource locators) and transferred via HTTP[6]. [10]

Since 1991 to present, WWW has become the most used service of the internet. The growth of the internet and WWW users population is enormous. In the 2001 it was estimated to be 500 million users on the planet, and the 1 billion milestone was reached only 4 years later in 2005. Between the years 2005 and 2010 the number of users doubled again to 2 billion. Today the number of internet users is estimated about 4 billion users, which in "only" half of the population of our planet. [11] That means there are still many people waiting to become internet users. The average from the last years is 600 000 new users connecting to the internet every day, mostly in underdeveloped countries like Brazil, China, India or Indonesia. But even in the Czech Republic there are still over 20% of population did not connected to the internet in the last year. [12]

2 Illegal hackers breaking computer security for gain
3 Transmission Control Protocol and Internet Protocol
4 Domain Name System
5 World Wide Web
6 Hypertext Transfer Protocol

	Africa	Asia	Europe	Latin America	Middle East	North America	Australia
Population	1 246	4 148	822	647	250	363	40
% of world	16.6	55.2	10.9	8.6	3.3	4.8	0.5
Internet Users	388	1 938	659	404	146	320	28
% of pop.	31.2	46.7	80.2	62.4	58.7	88.1	69.6
Growth (2000-17)	8 503 %	1 595 %	527 %	2 137 %	4 374 %	196 %	269 %

Figure 2: Number of internet users worldwide in millions [11]

Every year, there are not only new users accessing internet. There are also new services available and also new types of devices. Wi-Fi (wireless connection IEEE 802.11 standard) and next-generation cellular networks allowed the emerge of laptops and smaller devices to access the internet everywhere. Despite the first smartphone being sold around 2000, and the current platforms Android and iOS being first released in 2008 and 2007 respectively, smartphones had already reached over 10% of the global internet traffic. It's estimated it will surpass traffic of PCs[7] not later than 2021. [1]

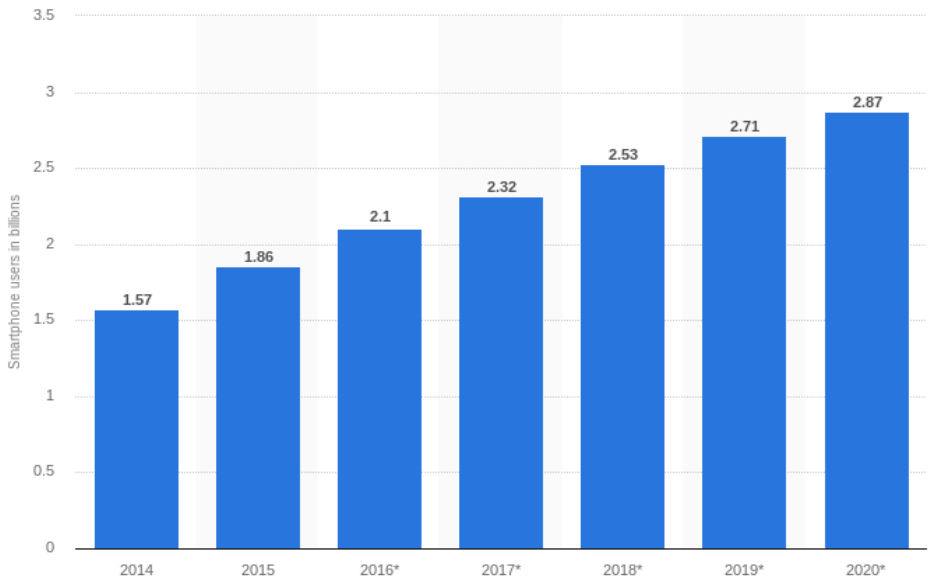

Figure 3: Number of smartphone users worldwide [13]

Another important technology worth mentioning is Bluetooth (wireless technology standardized as IEEE 802.15.1). Compared to Wi-Fi it has much shorter range and it is not suitable for connection to the internet, but it allows to inter-connect other devices to

7 Personal computers

smartphones or laptop and access the internet through them. Such devices can be so called wearables – smart watches, health and sport trackers, or any personal accessories or clothes which can be enhanced with electronics providing us with data about ourselves.

Figure 4: Apple Watch [14]

The internet now heads beyond what we know as WWW. With the expansion of small single-purpose electronic devices, the internet becomes internet of things[8] rather then internet of humans. [15] The ability of the new devices to produce massive amounts of data and upload them to cloud services establishes high requirements for data privacy and security. Data are becoming highly valuable asset for both users and companies.

2.2 Cyber threats

After the rise of the internet as a global and accessible resource, a wave of new threats has followed. New types of attacks and malware has emerged in the last year, most of them focused on users and companies data. [3]

After a malware gets into the victims device, it usually gains control over it, or allows the remote attacker to gain control over network, when it's needed, using command and control. These infected devices are called "zombie" and the whole collection controlled by the attacker are called "botnet"[9]. For some activities it's not even necessary the devices to be regular computers or laptops. Botnets can consist also from smartphones or small IoT devices too. The use of botnets differs. Some of the owners offer them for rent to provide specific action the customer wants, or can be used for own malicious activities. Usually botnets can be used to send e-mail spam, distribute

8 IoT

9 Combination of words "robot" and "network"

other malware or attack of DDoS[10] to either take some service down or as cover for some other activity, like stealing data. Examples of notable botnets in the last 10 years are Mariposa, Conficker, Zeus (or Zbot) or Mirai. The last one is especially interesting because it consistent of IoT devices like printers or IP cameras, which were used in 2016 for DDoS attack on web host company OVH and DNS provider Dyn, causing major internet services to be unavailable in North America and Europe. [16]

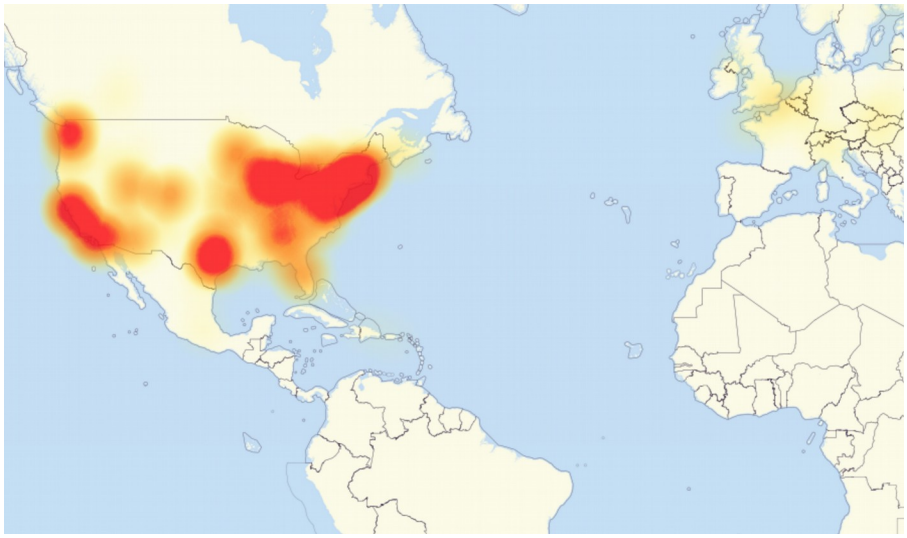

Figure 5: Dyn DDoS attack map [17]

Probably the best known and recently spreading type of threat is ransomware. This type of malicious software blocks access to data and asks the victim to pay ransom to allow the access again. The ways ransomware restricts access to files can differ, but the most effective one is to encrypt them and offer the decryption key to user in exchange for the payment. Thanks to the rise of cryptocurrencies, such as bitcoin, which are impossible to block and very hard to track down to real person, it's very hard to find the attackers unless they make some mistake. There is also no guarantee the victim will actually be able to recover the data back.

10 Distributed Denial of Service

Figure 6: WannaCry ransomware screenshot [18]

In 2013 CryptoLocker ransomware has been distributed via e-mail attachments and the existing Zeus botnet mentioned above. It encrypted specific types of data files on infected computers and asked victims for ransom in bitcoins in exchange for the private key to decrypt the files again. According to ZDNet who tracked Bitcoin payments to several of the bitcoin addresses between October 15 and December 18, the attackers could make over 25 millions USD in this timespan of two months. [19] After CryptoLocker more similar ransomware appeared. The notable ones are CryptoWall, Petya, Locky or WannaCry.

The family and versions of Petya (or NotPetya) were targeted mostly against Ukraine with one of the versions only pretending to be ransomware, but it actually destroyed all data stored on hard drives. [20]

Not just ransomware, but also individual data breaches document our dependence on electronic data. According to haveibeenpwned.com, a service collecting data from know breaches, the top 10 breaches contained data of over 3 billion of user accounts. The information usually contain username, email addresses and passwords. [21] And the number of data breaches in United States in the last 10 years is growing every year. [22]

2.3 Security solution systems

Computer security software and systems are used to prevent and react on unauthorized access, cyber treats and attacks. Malware or intrusion detection systems can be divided from different points of views.

2.3.1 Host based detection

Host based detection systems are using agent applications installed on the host devices, monitoring the activities like running processes, opened or accessed files. The host based detection systems were designed and in use in time when the interaction with other computers in the network were not frequent, like on mainframe computers. Examples of this approach are still seen in the design of the traditional end user antivirus solutions, which are installed on the user computer and monitor the running processes and watch for potentially malicious files stored on hard drives. The power of the host based deployment is the ability to see everything happening inside the system.

2.3.2 Network based detection

Network based malware and intrusion detection is an alternative in addition to the traditional host based detection. Instead of knowledge about activities and processes running on the endpoint devices, it detects malware based on the characteristics the malware exhibits via the network, when it's being distributed or communicates via its command and control channels. Because it does not have any applications installed on the users devices, it requires a standalone machine, usually on the network perimeter, at least to collect the network data. That makes it less suitable for personal use. However the centralization to a single place makes it ideal for enterprise use in company networks, as it scales well with the number of devices being connected to the network and protects the network rather then individual devices. This approach cover also any devices not secured by any other anti malware protection, like employees smartphones connected to the office wireless network.

2.3.3 Personal

End user solutions for personal use or SOHO[11] market are most frequently firewalls, general antivirus software or specific anti-malware software and tools. [23] There are also security techniques widely used, like sandboxing (isolation of the running program from the rest of the system) or using restricted access rights for running the programs.

Firewalls are network security programs monitoring the network traffic to and from the users computer based on defined security rules. The key for the traffic filtering is to understand the common protocols to inspect their traffic. For most of the users, the uncommon and unrecognized communication can be blocked, or enabled on demand.

Antivirus software, or anti-malware, is a computer software to prevent malicious software infection. Originally the first antivirus solutions were developed to remove computer viruses, later to protect from various computer threats. Present anti-malware can protects from ransomware, keyloggers, trojans, adware, spyware, or malicious JavaScript being executed by the browser. Anti-malware are often parts of complete

11 Small office or home office

security packages for personal use, accompanied by anti-spam and tools to protect from phishing, DDoS or social engineering frauds. Some of the packages also contain tools for private and secure web browsing, sandbox or ransomware detection based on the file changes, or even document backup tools. As example of personal anti-malware we can name products from McAfee, Symantec Norton, Kaspersky, Avast, Avira, AVG, ESET, F-Secure anti-virus and many more.

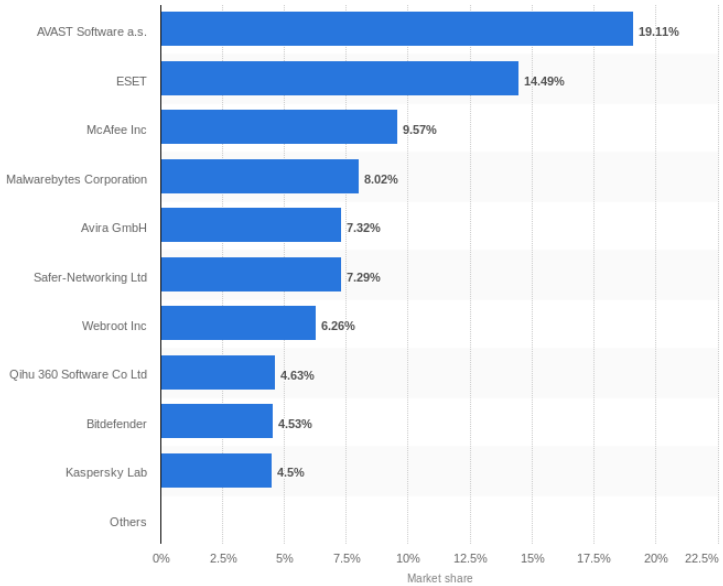

Figure 7: Personal antivirus market share [24]

2.3.4 Enterprise

The enterprise environment brings new challenges for cyber security. Devices in the company often contain business secrets and can connect into the company internal network with more confidential data available. Also the headcount of corporations or is hundreds or thousands times higher than the count of family members you protect by personal anti-virus. Also the threats and attacks are more sophisticated and frequent with the potential loss, respectively gain for the attacker.

Anti-malware makers offer special editions of their software for business customers with features specific for the environment, interoperable with enterprise software deployments and management. However larger organizations often use advanced company-wide solutions and systems to protect themselves too.

Enterprise firewalls builds a barrier on the perimeter between the internal network (LAN[12]), with access limited by access control, and the public internet network (WAN[13]), where the threats are expected to originate from. While personal firewalls are

12 Local area network

13 Wide area network

software, for enterprise firewall means dedicated hardware device or server monitoring all the traffic going through the perimeter it is supposed to guard. These firewalls and proxy servers have capabilities of packet inspection, looking at actual traffic content to remove malware or unwanted content from the communication, or block the communication entirely.

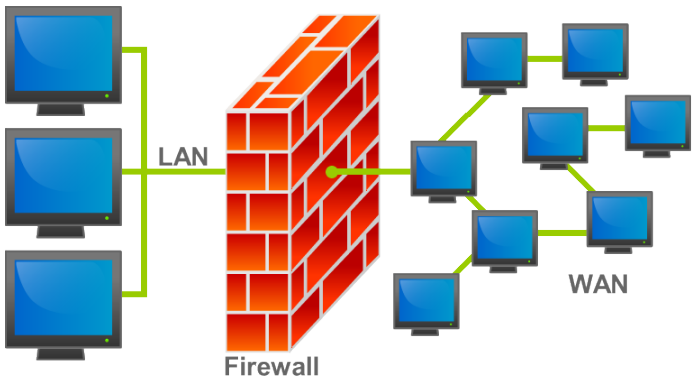

Figure 8: Firewall [25]

Slightly different from firewalls are network intrusion detection/prevention systems (NIDS). While firewalls look for intrusions to prevent or stop them at the very beginning, and limits access to the network, IDS aims to detect, describe and report suspected intrusions after they happen, e.g. based on other systems logs or reports. Often IDS monitors not just outer threats, but also those inside the company network. Open source examples of IDS can be Fail2Ban, Sagan, Snort or Suricata.

In 2014 Gartner recommended use of UBA[14] to detect insider threats, targeted attacks and financial fraud. [26] UBA looks for patterns of employees behaviour, analyzing those with statistical algorithms to detect anomalies, which can be reported and investigated. As the name suggests, the detection is not oriented on devices, but on actual users in the company systems. To analyze the big amounts of data, big data platforms, like Apache Hadoop, Cloudera, or cloud services are used in the background.

The more companies are successful, the more they value security to protect against competitors shady practices, industry espionage and sabotage. Dedicated security departments are not uncommon. As it's not possible to prevent all cyber threats completely, it's important to unfold them early and react quickly. [27] The number of various systems and solutions mentioned above brings the security analysts information, but it also maker harder to analyze them all. Software products and services used to combine, aggregate, analyze and prioritize are called SIEM[15]. As per Gartner definition, SIEMs are used for automated monitoring and regular vulnerability discovery. It is expected to provide real-time event management and analysis of security data from

14 User behavior analytics

15 Security information and event management

wide set of heterogeneous sources, filter incident information and reporting. The whole SIEM idea is driven by the need to support vulnerability and threats management and ease their analysis and mitigation. [28]

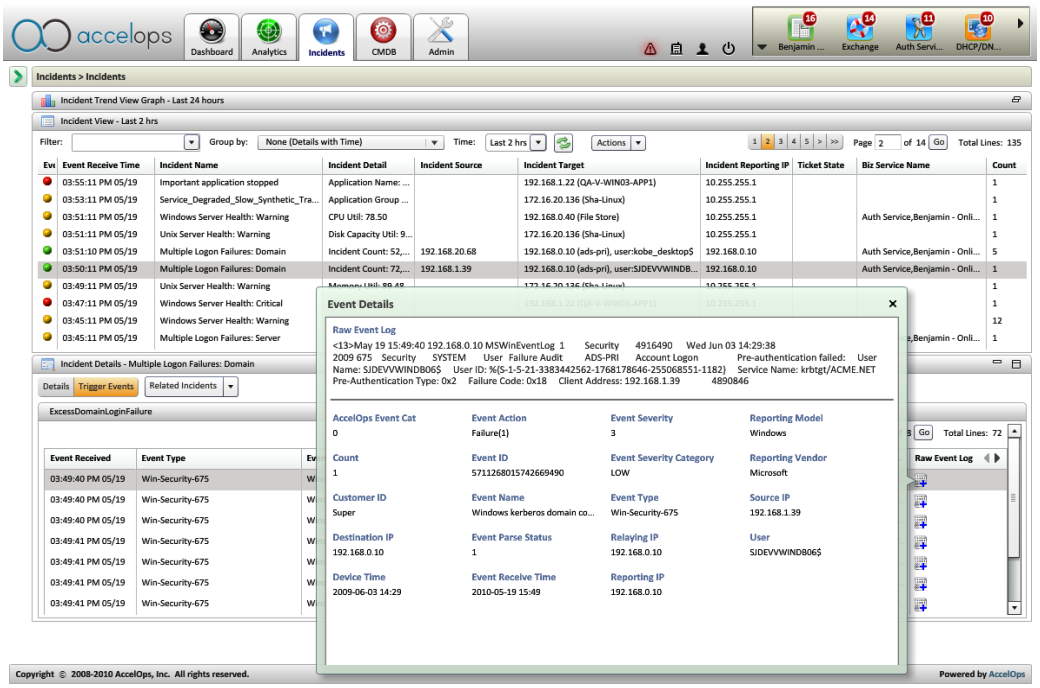

Figure 9: SIEM dashboard screenshot

According to Honeywell, the cost of an industry cyber attack resolution goes over 20 000 USD per day [5], which means time actually is money here. Any additional intelligence and knowledge during the incident analysis can help to react appropriately and make a different between successful cyber attack and successful resolution.

2.4 Behaviour similarity

One way to reduce a network incident analysis work is to identify patterns in the detected malicious network behaviour. This section introduces several algorithms to identify similarities in network behaviour in particular in web traffic.

2.4.1 IP blacklist

IP blacklist is a very simple instrument to protect the network against known “bad” remote servers. The list contains known malicious IP addresses obtained from public resources or historical data. However adding any IP address the malware communicates to may block more than one wants, especially when the number of IPv4 addresses is low and multiple services, both legitimate and malicious, can be provided from the same IP.

Example can be shared web hosting or cloud hosting. Also the IP blacklist won't protect from advanced malware using previously undetected servers.

IP blacklists are mostly used for spam protection and the IP addresses are being enlisted or delisted based on the reputation of email being sent from the address. [29]

2.4.2 Connection success ratio

In a behaviour based mobile malware detection model proposed by Tri-Hai Nguyen and Myungsik Yoo from the Soongsil University, connection rate and connection success ratio is used to detect compromised mobile devices. [30] The assumption behind is that the probability of unsuccessful connection for a normal well-functioning device is very low. Unsuccessful connection would mean a bug in installed application or the requested service being down. On the other hand infected devices are more likely to make unsuccessful connections or even connections to blacklisted IP addresses.

If the number of pending requests without response exceed some threshold, the device exhibiting the bahaviour is considered infected and its access into the network can be restricted. For illustration, the thresholds is set to 30 requests in the proposal.

2.4.3 Connection rate

According to the same paper from the Soongsil University, the connection rate itself is also likely to differ between normal and infected devices. [30] It assumes a normal device will make new connections less frequently and to a limited number of services in a period of time, while infected devices will try to connect many times to many services or other devices.

If the number of connections in a specified time exceeds some threshold, the device is again considered infected and treated accordingly. For illustrations, the threshold is set again to 30 requests and expected connection rate is 1 request per second.

2.4.4 N-gram URL similarity

N-gram is a sub-sequence of n items (e.g. words) in a given text. N-grams are usually applied in probabilistic language processing and for predictions using Markov models. The following figure shows an example how N-grams are constructed.

This is an example sentence.

N=1 (unigrams):

this, is, an, example, sentence

N=2 (bigrams):

this is, is an, an example, example sentence

N=3 (trigrams):

this is an, is an example, an example sentence

Figure 10: N-grams examples

Neetu Singh from the Central University of Himachal Pradesh, India, and Narendra S. Chaudhari from the Indian Institute of Technology proposed to calculate similarity between URL using N-grams, to classify the web page topic without the actual content being known, only from the text contained in the URL. [31]

In the proposed similarity measure, the URLs addresses are converted to lowercase and divided into words. A presence of a word is indicated as “1” or “0” in the feature vector. The classification algorithm then runs in two phases.

The first step is to divide the training dataset is divided in two disjoint subsets T containing positive classifications and T' negative.

The algorithm starts with forming distinct combinations of k features from the tested URL α. Then the algorithm computes the count of the URLs in T where the each combination of the the k features occurs, and the same for the set T'. The tested URL α is re-classified positive when the count is higher for the set T, or negative when the number is higher for T'.

$$S_T(\alpha, T \cup T') = \sum_{S \subseteq [n], |S|=k} \sum_{y \in T} \left| (y|S = \alpha|S) \right|$$

$$S_{T'}(\alpha, T \cup T') = \sum_{S \subseteq [n], |S|=k} \sum_{y \in T'} \left| (y|S = \alpha|S) \right|$$

$$f(\alpha) = \begin{cases} 1 \; if \; S_T(\alpha, T \cup T') > S_{T'}(\alpha, T \cup T') \\ 0 \; if \; S_{T'}(\alpha, T \cup T') < S_T(\alpha, T \cup T') \end{cases}$$

The proposed algorithm has been compared to another one, which computed sum of distances between α and all URLs β from both T and T'. α is then classified based on the sum of distances. If the sum of distances with all β from T is less, α is classified positive, or negative when the sum of distances from T' is less.

2 Analysis

$$S_T = \sum_{\beta \in T} d(\beta, \alpha)$$

$$S_{T'} = \sum_{\beta \in T'} d(\beta, \alpha)$$

$$f(\alpha) = \begin{cases} 1 \; if \; S_T < S_{T'} \\ 0 \; if \; S_{T'} > S_T \end{cases}$$

The parameter *k* has been tested with values *2*, *3* and *4*, where *2* showed the best results. When *k=4*, the proposed algorithm brings almost no improvement in the classification accuracy, compared to the second one.

2.4.5 Query string similarity

Query string is an optional part of URL containing information sent from the client to the server resource. It has a form of key-value pairs without any order or hierarchy.

http://example.com/some/resource/?foo=John&bar=Doe

{foo: John, bar: Doe}

Figure 11: Query string illustration

For discovering new servers connected into malware command and control botnets, a team of researchers at Cisco made an assumption that the same application will receive the same parameters or keys in case of URL query strings. [32] The values of particular keys are not considered, as they are likely to differ across different clients or even requests.

The researchers have defined a similarity function between two query strings and the contained key-value pairs using model called “bag of words”. Bag of words means we create an unordered set from all the observed words, noting the number of every word occurrence alongside. For the query string similarity, the vocabulary is the list of all query string keys in the particular URL. The query string values are ignored.

http://example.com/some/resource/?foo=John&bar=Doe
http://example.com/another/?foo=Jane&baz=Roe

{foo: 2, bar: 1, baz: 1}

Figure 12: Sparse vector constructed from URLs query string

Every set of URLs *s* having a query string can be then represented as a sparse vector q_s, where each k_i is equal to the number of occurrences of *i*-th key in all query strings.

$$q_s = (k_1, k_2, k_3, \ldots, k_v)$$

To avoid address the potential problem of very common key, which won't discriminate between two sets of URLs, TF-IDF scaling is used.

Finally to determine the similarity between two vectors containing at least one non-zero value, cosine similarity is used.

2.4.6 Path similarity

Path in URL in HTTP requests define the location of resource being requested. Unlike query string, path elements are ordered. In the example URL *http://example.com/some/other/resource/* the path root is "some" with child "other" and "resource" on the deeper level.

In the same paper Cisco researchers made a similar assumption for both query strings and paths, that the same application will provide same services and resource son the same paths. [32]

For this similarity, every set of URLs is represented by a tree, where the root is "/" and every path is decomposed into child nodes and the tree is build top down. The deeper the directory is in the path, the deeper is in the node. This respects the importance of directory order in the path.

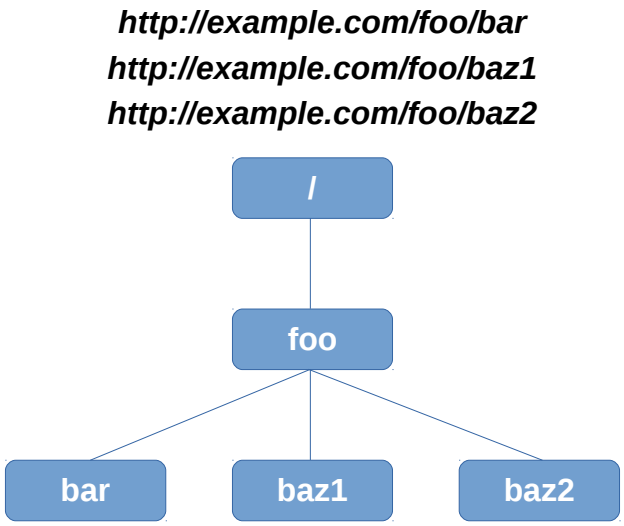

Figure 13: Tree constructed from example URLs paths

The paper proposes its own tree similarity metric specifically designed to honor the ordering of directories (depth of nodes).

$$K(m,n)=I_{m=n}\cdot\left(1+C\cdot\sum_{\substack{u\in child(m)\\ v\in child(n)}}K(u,v)\right)$$

Where *n*, *m* are two nodes being compared, $I_{(n=m)}$ becomes 1, when the nodes are equal, 0 otherwise. *child(n)* return all children of the node *n*. Parameter C determines the speed of increase or decrease for deeper levels. The similarity itself it the defined as follows with *n* and *m* being the root nodes of each tree:

$$s(n,m)=\frac{2K(m,n)}{K(n,n)+K(m,m)}$$

Again the similarity is not defined for trees with no non-root node, which represent set of URLs without any path known.

2.4.7 Similarity of transferred bytes and connection timing

Visited URLs are the most obvious and visible information about web requests, but for the encrypted HTTPS connection the URLs are not available. According to the latest latest Let's Encrypt report, over 60% websites are visited over HTTPS today. [33] Luckily there are more information to be measured for HTTPS requests, which are the size of request/response, the time the request and response loop takes and the interval between two consecutive requests.

In the paper [32], each request and the corresponding response can be then described as a vector of for numbers:

$$r=\left(\log\left(1+r_{bs}\right),\log\left(1+r_{br}\right),\log\left(1+r_{d}\right),\log\left(1+r_{i}\right)\right)$$

where

- r_{bs} is the number of bytes send
- r_{br} is the number of bytes received
- r_{d} is duration of the request and response loop
- r_{i} is the interval between the consecutive request

From the values of vectors r for the requests in a set, histogram is created to show the distribution of the values and arranged in a vector form. Cosine similarity of two vector forms is the similarity of two corresponding set of requests.

The only limitation of this metric is it requires some minimal number of requests to build the histograms.

2.5 Partitioning

Identifying the similarity itself is not expected final result to present to the security analyst. But with the similarities obtained, the incident data can be partitioned into groups of same or very similar flows which can be shown and available for detailed inspection.

When using multiple similarity functions in parallel, multiple partitionings are created, which need to be later combine to a final one. This problem of finding an optimal solution based on individual partitioning consensus is equivalent to ILP[16] [34], which is NP-hard in general. For this reason the problem of combining multiple partitionings is solved by heuristics.

$$min\sum_{i=1}^{n}\sum_{j=i+1}^{n}M(i,j)d(i,j)$$

s.t.:

$$M(i,k)+M(j,k)-M(i,j)\leq 1,\forall i,j,k$$

$$M(i,j)\in\{0,1\}$$

$$i,j,k\in\mathbb{N};i,j,k\leq n$$

16 Integer linear programming

2.5.1 K-means

K-means algorithm is a method for cluster analysis. K-means clustering is used for clustering n observations into k clusters such that the sum of squares between observations distances in all clusters is minimal. The algorithm itself consists of two steps.

First k cluster centroids $\mu_1, \ldots, \mu_k$ are chosen randomly.

Then until convergence this is repeated.

$$\forall i : label_i := \arg\min \|x_i - \mu_j\|^2$$

$$\forall j : \mu_j := \frac{\sum_{i=1}^{|C|} 1\{c_i = j\}\, x_i}{\sum_{i=1}^{|C|} 1\{c_i = j\}}$$

A drawback of the k-means algorithm, which may not be intuitive for every use, is the necessity to specify k – the number of clusters that will be created in the final partitioning.

2.5.2 Hierarchical clustering

Hierarchical clustering is a method for building a hierarchy of clusters from the data using the (dis)similarity between them. [35] Generally there are two approaches for hierarchical clustering:

- Agglomerative clustering, where each data is considered as a standalone cluster at the beginning, and these clusters are pairwise merged based on their similarity and build the nested hierarchy.
- Divise clustering is an exact opposite. At the beginning all observations form a single cluster which is being divided.

The first formula the clustering need to use is a distance metric to computer pairwise distances between each two observations. For the incident flows it can be a similarity metric from the previous part 2.4.

d(a,b) = 1	*d(b,c)= 2*
d(a,c) = 3	*d(b,d) = 2*
d(a,d)= 3	*d(c,d) = 4*

Figure 14: Example hierarchical clustering - distances

Let's start with four observations forming four clusters *a*, *b*, *c* and *d*. When the similarity *s* between all pairs is computed, the agglomerative method finds the two most similar clusters (step 1). In the example above it's *a* and *b*. The two closest clusters are merged, creating a single cluster *ab* (step 2). For the cluster *ab*, no similarities to *c* or *d* are known. There are various strategies commonly used:

- Complete-linkage clustering takes the maximum distance (minimum similarity) for the merged clusters.

$$d(ab,c)=max\{d(a,c),d(b,c)\}$$

- Single-linkage clustering takes the minimum distance (maximum similarity).

$$d(ab,c)=min\{d(a,c),d(b,c)\}$$

- Average-linkage clustering takes the mean values of the distances *d(a,c)* and *d(b,c)*.

$$d(ab,c)=\frac{d(a,c)+d(b,c)}{2}$$

- Weighted-linkage strategy takes the weighted mean of values of the distances *d(a,c)* and *d(b,c)*. The weight *w* can be an importance of the observations in *a* and *b*, or their number.

$$d(ab,c)=\frac{d(a,c)*w(a)+d(b,c)*w(b)}{w(a)+w(b)}$$

After the distances between the new cluster *ab* and every other cluster *c* and *d* is computed (step 3), the clustering can start a next iteration by finding the closes pair from clusters *ab*, *c* and *d*.

The visualization of the final complete-linkage clustering of the observations in our example would like like this.

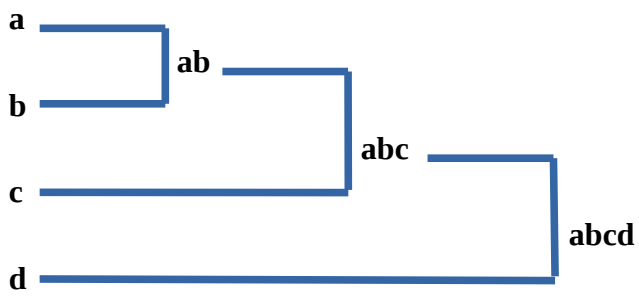

Figure 15: Hierarchical clustering visualization

The final results of the hierarchical clustering is a hierarchy of possible clusters, but without any bound where the process is stopped, it will eventually end up with the whole data set in a single cluster. The bound can be either a number of clusters, which is not intuitive for all use cases, or a distance threshold. When there is not pair of clusters

with a distance less than the threshold T, the process of clustering is stopped and the current partitioning is the final one.

The proposed solution to find a good threshold (but necessary not optimal) is by semi-supervised learning. [36] First labeled data with target clusters C_1, …, C_n are used to computer the an F-measure for different threshold values and computed clusters HC_1, … Hc_n.

$$P(i,j)=\frac{|C_i \cap HC_j|}{|HC_j|}$$

$$R(i,j)=\frac{|C_i \cap HC_j|}{|C_i|}$$

$$F(i,j)=\frac{2.P(i,j).R(i,j)}{P(i,j)+R(i,j)}$$

$$F=\sum_{i=1}^{n}\frac{|C_i|}{N}F(i)$$

At the end the threshold values with the highest F-measure value is chosen. According to the experiments in the paper, a set of 50 labeled data examples is usually enough to find a reasonable threshold value close to the optimum.

2.5.3 Consensus matrix

To combine multiple clustering, consensus matrix is a very simple measure. In a consensus matrix M, the element M_{ij} is defined as a fraction of the number of partitionings P where i and j are in the same cluster and the total number of partitionings. [34]

$$M_{ij}=\sum_{k=1}^{|N|}P_k(i,j)$$

The matrix M then represents a single similarity measure ($s(i,j) = M_{ij}$), for which e.g. hierarchical clustering can be applied to construct the final clusters.

2.5.4 Iterative construction

The paper on discovering botnet command and control networks [32] offers also an algorithm for combining multiple partitionings. In the proposed multiagent system, the opinion about relation between every pair of servers *i* and *j* from all agents is combined and a matrix *M* is constructed from the aggregated values, where the values M_{ij} is the number of partitionings, where *i* and *j* are in the same cluster. This matrix is equivalent to the consensus matrix 2.5.3, only the elements are not divided by the number of partitionings.

The final partitions are built in iterations. In each iteration a matrix *T* is defined using a threshold *t*, which is being decremented in every iteration. The elements of *T* are defined as follows.

$$T_{ij} = \begin{cases} 0 \, if \, M_{ij} < t \\ 1 \, if \, M_{ij} \geq t \end{cases}$$

The matrix *T* is used as a graph adjacency matrix and every connected component is added to the final partitioning as a new cluster and corresponding elements removed from *M*. The threshold t is then decremented, new matrix *T* created and the process goes over again until *t=1* or there are no rows in *M* left. If there are any rows left at the end of the process, they are added as singleton[17] clusters into the final partitioning.

This algorithm is tailored to the requirement of a low number of false positives and creates only clear clusters.

17 Containing only single element

3 Proposed solution design

This chapter is dedicated to the description of the input data, framework design, and detailed description of algorithms in use.

3.1 Requirements

The framework will take information about incident flows as input, and produce their partitioning based on the extracted behaviour similarities. To achieve that in a reliable way, it needs to fulfill the following requirements.

- Simple to use – the framework should be easy to set up and feed with input data.
- Stateless – the execution should be stateless across multiple incidents. The incidents are disjunct and no information is meant to be carried between any two of them.
- Deterministic – two runs with the same configuration and the same input data have to produce the same results.
- The input has to be machine readable for further processing and later presentation to the user.

3.2 Input data

Input for the framework will be already identified malicious network behaviour in a form of individual incidents. Every incident consists of flows with the following information:

- timestamp
- URL address (for HTTP flows)
- destination IP address
- bytes sent/received
- duration of the request and response loop

Besides these information, there is also information available what classifier has identified the specific flow as malicious, which provides some insight if the flow was part of the malware communication (and what type of malware) or some suspicious file download or some other kind of activity.

These information also splits the flows into observed malicious events with the same classification, destination and proximate time. However the events may not be complete because the classification of some of the flows was not specific enough or because the same activity was more spread in time. That can be caused for example by missing data or because the device was disconnected during the night.

The task is to group the flows into partitions based on their similarity to provide the analyst with a complete overview of how was the malicious behaviour of each incident changing in time.

3.3 General framework design

The input data will be preprocessed first from the input format to internal data object representation, forming the collection of flows corresponding to the identified incidents and events classified.

The data will be further processed to determine the similarity/distance between the malicious events using the behaviour similarities from 2.4.5-2.4.7. The results of the similarity functions need to be further aggregated to form the final partitioning, which is the expected result.

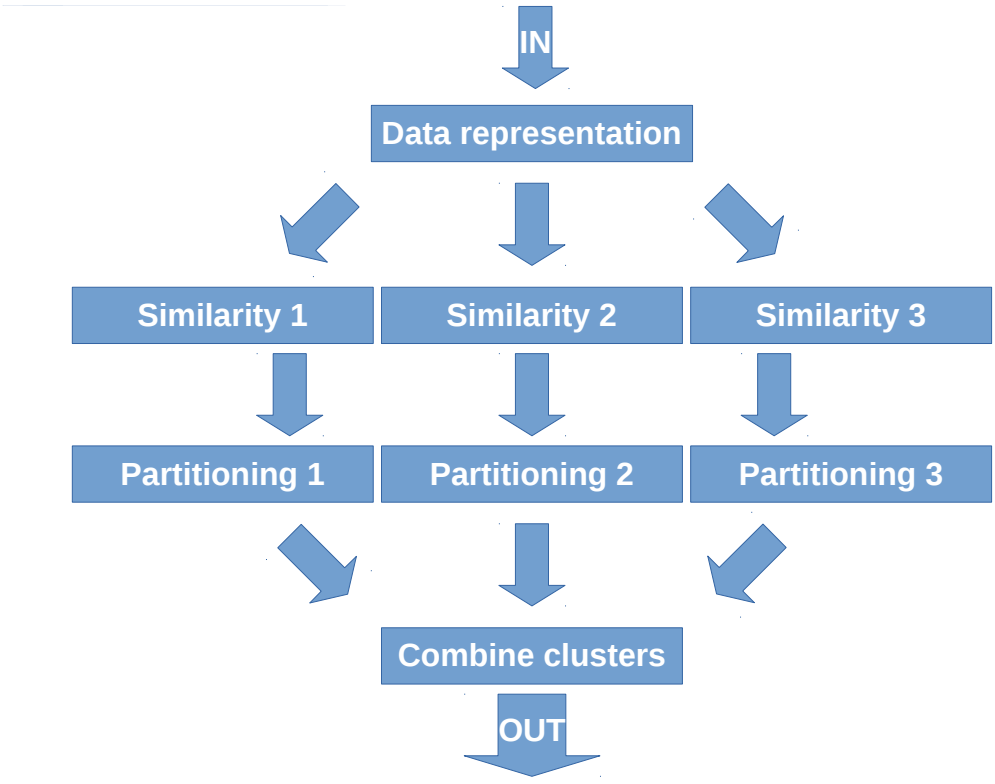

Figure 16: Proposed framework structure

3.4 Algorithms

From the algorithms mentioned in 2.4 I have decided to choose 2.4.5-2.4.7 (which are based on the research of Ing. Martin Rehák, Ph.D., the supervisor of this thesis). The three similarity functions are consistent in their output and can be easily run along each other and their results further combined into a single partitioning.

For each function output I have decided to apply hierarchical grouping algorithm described in 2.5.2. To produce the final groups of flows from their hierarchical groupings, consensus matrix is used as described in 2.5.3.

3.4.1 Query string similarity function

As noted above, I have chosen three similarity functions using the URL query string, URL path and bytes and timing of the connection.

The query string similarity function operates with query keys. Two query key sets are assumed to be similar based on the occurrence of the same keys in both of them. The query strings are available for unencrypted HTTP communication only. When the query string is not available, or none is used, the similarity is not defined. Otherwise it's the cosine similarity between two sparse vectors of the key values occurrences observed.

$$s(q_1, q_2) = \frac{q_1 \cdot q_2}{|q_1||q_2|}$$

The similarity needs to be computed for every 2-combination of vectors in an incident, that contain any non-zero value. Vectors with no non-zero elements represent URLs without query string being known, which may be cause just by missing the information for HTTPS traffic.

3.4.2 Path similarity function

The path similarity function operates with directories from each path. The availability of the information is limited the same way as in the previous case of the query string, for unecrypted HTTP communication only. If there are no directories in the path, the similarity is not defined. Otherwise the directories are organized into a tree hierarchy similar to Unix file system with "/" as root. Two trees are compared using the following similarity function.

$$s(n, m) = \frac{2K(m, n)}{K(n, n) + K(m, m)}$$

where:

$$K(m, n) = I_{m=n} \cdot \left(1 + C \cdot \sum_{\substack{u \in child(m) \\ v \in child(n)}} K(u, v) \right)$$

The similarity needs to be computed for every 2-combination of the trees. The time complexity to compute depends on the tree structure, in the worst case of all nodes

being the children of the root the complexity is a product of the numbers of children or the root, plus one for the single comparison the roots.

$$s(T_1, T_2) = O((|T_1| - 1).(|T_2| - 1) + 1)$$

3.4.3 Transferred bytes and connection timing similarity function

This last function computes the similarity from histograms of transferred bytes number and the timing of consecutive requests. This information is available for both unecrypted and encrypted HTTP(S) communication, however a minimal number or requests is necessary.

The more similar the number of bytes transferred both ways and the request timing is, the more similar the two sets of flows are. If the requirement of minimal requests is met, the similarity is computed as a cosine similarity between the two vector representations of the histograms $r = (log(1 + r_{bs}), log(1 + r_{br}), log(1 + r_d), log(1 + r_i))$, where the r_x are the number of bytes transferred and requests timing respectively.

3.4.4 Heuristic using classification information

As mentioned in 3.2, for some malicious events a specific classification is known too. For two different highly specific classification we may assume the classified events are different too and putting them together will result in a loss of information from one of them. For a pair where only one or even none of the classifications is that specific, these are allowed to be put into the same cluster unless another different highly specific classification would end up in the same cluster too.

3.4.5 Hierarchical clustering

For creating partitionings based on the similarity function individually, hierarchical clustering will be used.

In the basic version, only a threshold value t is used to stop merging the clusters when there is no pair of clusters with a distance $<t$.

However in this stage we may use the further knowledge of event classifications. If the merged pair of clusters would contain two different specific classifications, the pair of clusters may not be merged and the whole clustering process should stop, even if the threshold t has not been reached yet. The goal of this heuristic is to avoid any potential loss of information, that would happen by merging two clusters with different classification, albeit they seems similar according to the metric.

3.4.6 Final clustering

After individual partitionings using similarity functions 3.4.1-3.4.3 and 3.4.5 are produced, a final clustering need to be produced. In the chapter 2.5 I have introduced several heuristics how to aggregate multiple partitionings into a single one, but I have decided to use a modified combination of both.

Because none of the similarity functions is defined for any pair of flows, a consensus matrix *M* 2.5.3 is computed first, with the modification where for every pair *i* and *j*, only partitionings for defined similarities are taken into account. E.g. for events consisting of HTTPS flows only (with both path and query missing from the URL), only the last metric using transferred bytes and requests timing is considered, and the denominator in the formula is 1. On the other hand for HTTP events, paths and query strings both are known, but the number of flows may be too small to compute the last metric, and the denominator in the formula is 2. For pair of events where for both all the functions are defined, all of them are taken into account and the denominator is 3. If there are no metrics available for the pair, the value M_{ij} is *0*, as we have no knowledge for the particular pair.

Once this modified consensus matrix *M* is computed, a single iteration of the second algorithm 2.5.4 is made, with the threshold set *t=1/2*. The intuitive meaning is that at least two of the applicable metrics need to agree on putting the two observations into the same cluster. For a pair where all metrics are applicable, the majority of them need to agree. Where only a single metric is defined, it’s fully trusted.

4 Implementation

This chapter describes implementation details and shows several code snippets as examples.

4.1 Overview

The general implementation of the framework is written in Kotlin[18] and Java[19], and using Apache Maven[20] for building. Java SE (standard edition) is a widely used platform portable language used for both desktop and server applications, even for developing mobile apps for Android. Kotlin is about 6 years old programming language which runs on the Java virtual machine and offers great interoperability with Java code and dependencies both ways. Java code can be used from Kotlin and vice versa.

As noted in chapter 3.4 this thesis is based on the previous research of Ing. Martin Rehák, Ph.D., the supervisor of this thesis, and uses parts of the implementation of the similarity functions written in Java.

For some simple data pre- and postprocessing, that's not an essential part of the framework and was specific to my measurements and for verifying the implementation, I have created several standolne bash scripts, most commonly to run the framework with specific input data and arguments and semi-automatically verify the produced results afterwards.

4.2 Input data

To reduce the requirements I have chosen CSV text files as the format for input data. It is human readable (one can see by eye the actual content of the input data), however the size of the text files and the time needed for parsing is not great.

For that reason I have encapsulated the mapping from input format to internal data object representation into a single object behind an interface so it's easy to replace for a different format.

```
interface InputFileReaderInterface<FLOW> {
    fun getFlowsFromFile(file: String): List<FLOW>
}
```

18 https://kotlinlang.org/
19 https://www.java.com/
20 https://maven.apache.org/

4.3 Features and similarity computation

Given the fact there are more algorithms for computing URL and network traffic similarity, there is a potential that some similarity functions will be added or replaced in the future. To make this potential change smooth as possible, I have defined two function interfaces, one for the "feature" extraction from the set of flows and one for the similarity function itself. The similarity function does not even have to be normalized.

```
fun <FLOW, M> computeFeature(flows: Array<List<FLOW>>, feat:
 (List<FLOW> -> M)): List<M>
```

To extract a feature from a list of flows the `feat` function takes the list and returns some generic type `M`.

```
fun <M> computeSimilarity(values: List<M>, sim: (M,M) ->
 Double): Array<Array<Double>>
```

To later computer pairwise similarities, the list of values of the same generic type `M` is needed and a function, that takes two arguments of type `M` and computes their similarity.

```
// The two function definition
fun exampleFeat(flows: List<FLOW>): Array<Int> { … }
fun exampleSim(a: Array<Int>, b: Array<Int>): Double { … }

// Usage somewhere else in the code
val values = computeFeature(flows, ::exampleFeat)
val similarity = computeSimilarity(values, ::exampleSim)
```

Using generics and static references, the two functions can be put together very easily, requiring no specification of generic types in the code using these functions. The only requirement is that the actual type (`Array<Int>` in the example) returned by `exampleFeat` and accepted by `exampleSim` matches.

4.4 Hierarchical clustering

For the hierarchical clustering I have reused the open source Java implementation of the algorithm available under Apache License 2.0 [37] from the central Maven repository with some small adjustments mentioned in 3.4.5.

4.5 Output format

For each incident the result of clustering is a set of clusters, each represented by a set of flow sets identifiers of the type `String`. I have again chosen CSV text files as the format for output, but again the conversion and writing to output files is behind an interface and easy to replace with console output, logger or completely different output serialization if required.

```
interface ResultsWriterInterface {
    fun writeResult(result: List<Set<String>>, inputFileName:
 String)
}
```

5 Evaluation

This chapter will describe the experiments used to verify the solution is applicable to our case and to verify the hypotheses from the chapter 3.

5.1 Input datasets

To verify the framework meets the requirements and to see its accuracy, I have prepared several experiments with real traffic data. I have created three datasets:

1. complete data from 100 different incidents, consisting of ~420,000 flows in total
2. manually labeled data from 15 incidents
3. complete data from 500 incidents, consisting of 2.3 million of flows in total

The first dataset I have used for creating input data with synthetic TP[21] examples by splitting already known groups of malicious events (3.2) in two. For one of them I have also manipulated input for some similarity function result, with preserving the expected output of the two splits being combined in the output, by shifting the flows in time, changing the remote destination, removing path or query string from the URL. This synthetic data I have been using during development to see if the framework works as expected.

The second dataset was first manually labeled and I have used this dataset for the F-measure (2.5.2), later to verify that the framework setup is not overfitted for the synthetic examples in the first dataset and to control the results for potential too many FP[22].

The last and largest dataset was used for quantitative measurements of the framework accuracy and data reduction.

5.1.1 Data statistics

	Incidents	Events per incident				Flows per event			
	total	**min**	**max**	**avg**	**med**	**min**	**max**	**avg**	**med**
1.	100	2	166	45.60	49	1	56660	92.19	2
2.	15	3	72	14.73	17	1	889	47.13	3
3.	500	2	1286	54.78	50	1	106810	83.67	2

Figure 17: Input data statistics

21 True positive

22 False positive

5.2 Single similarity function experiments

To understand the expected results I have started with using each similarity function alone first. For all the three similarity function, the range of values is *<0;1>*. The range of values for cosine similarity itself, used for the query string similarity and bytes and timing similarity, is actually *<-1;1>*. However the vectors always consist of zero or positive values[23], thus the cosine of their angle can never be a negative number.

I have run the metrics one by one alone on the data from the first dataset with ~4560 events in total. The following graphs are cumulative histograms of pairwise similarities among all the events that were suitable for their computations.

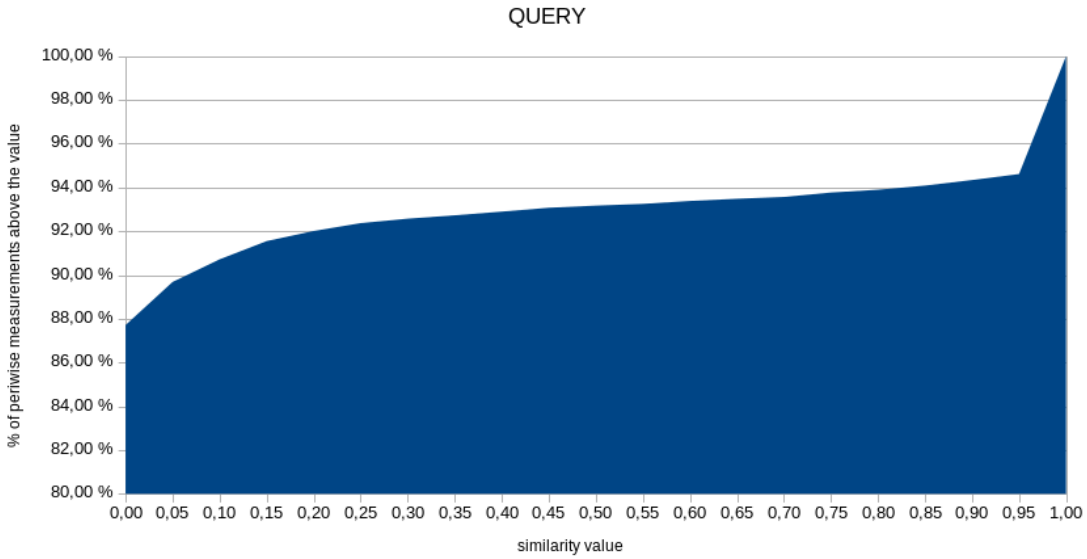

Figure 18: Cumulative histogram of query similarity

1300 (28 %) of events had some query string in any of the corresponding flow URL for the similarity function to be computed. The distribution has very smooth course over the the interval with the exception of very beginning, where over 87% of pairs are not similar to each other at all. 5% of the highest values are between 0.95 and 1.0.

23 The number of occurrences of a key in query string can never be a negative number, the same for the number of transferred bytes or time duration

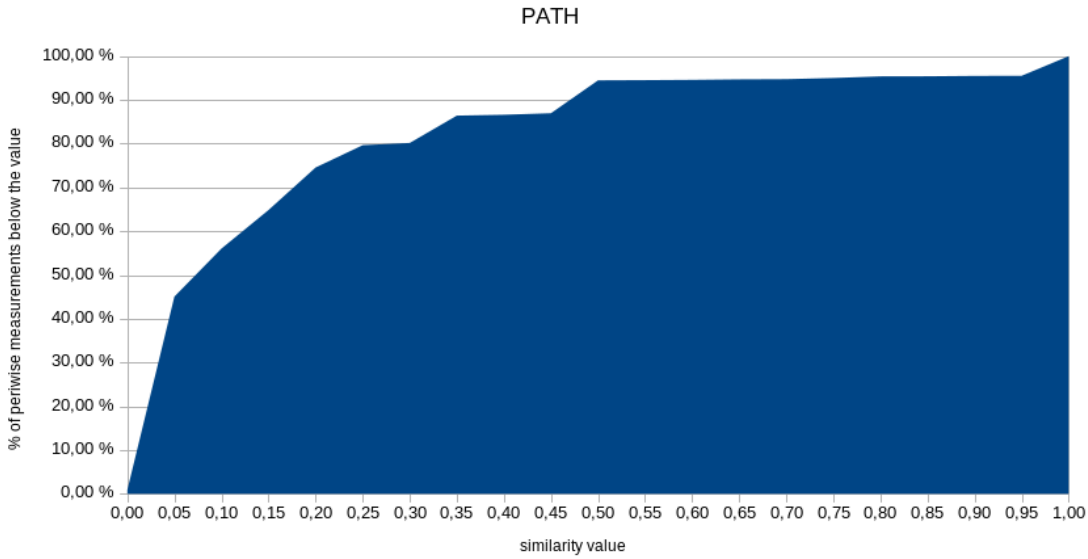

Figure 19: Cumulative histogram of path similarity

2500 (54 %) of events had a path in any of the corresponding flow URL for the similarity function to be computed. Around 80% of the pairwise similarity values are below 0.3, and 94.5 % is below 0.5. However almost 4.5% of all applicable pair similarities is above 0.95. The course of the values shows most of the pairwise similarities is very low, but only 0.9% of them is pure zero.

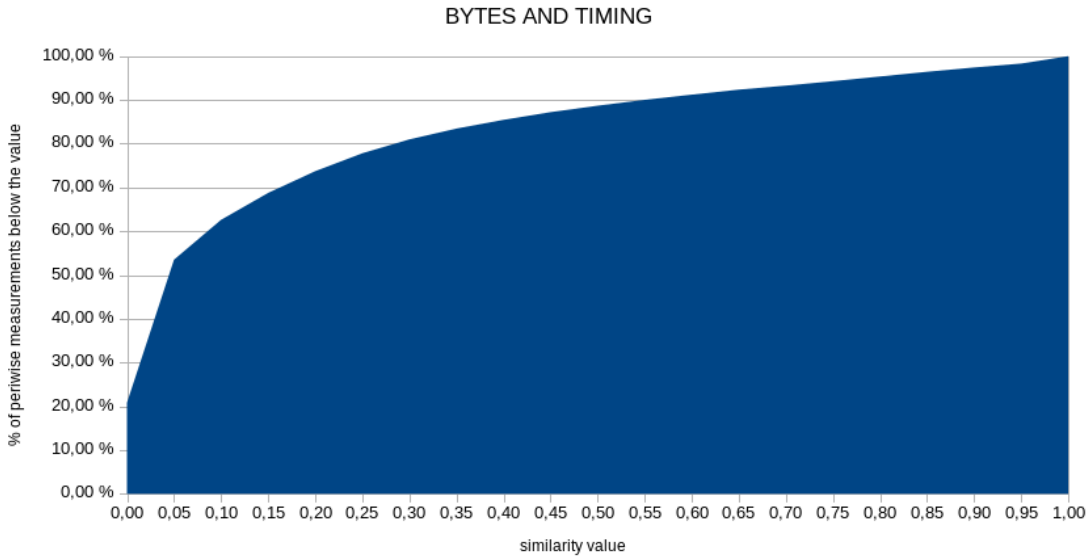

Figure 20: Cumulative histogram of bytes and request timing similarity

730 (16 %) of events had enough flows (at least 20) for the similarity function to be computed. At first the low percentage of events seemed odd to me, but it actually correspond to the low median of flows per event. To make sure most of the data are not

ignored by this metric I have measure also the percentage of flows contained in the 16% of events, which turned to be 95% of all flows.

21% of the pairwise bytes and timing similarities are 0. Approximately the same percentage as for path similarity (81%) is below 0.3 and almost 89% is below 0.5. The overall distribution in the range is smoother with only 1.5% of highest values between 0.95 an 1.0.

In total at least two similarity function were applicable to 31%, but all of them to only 7.5%. For 40% of events, no metric could be used, but these contained only <2% of flows.

If we compare the histograms of all three similarity functions, we see that for all of them most of the values are below 0.3, in case of query strings similarity it's actually below 0.05 already. For path and query similarity there is roughly 5% between 0.95 and 1.0, which corresponds to the assumption that the same resources or services are provided on the same paths and accept the same arguments in form of query string.

The complete tables with exact percentages of the histograms can be found in 8.1.

To determine a good value of hierarchical clustering thresholds for all the three similarity functions, I have run the hierarchical clustering algorithm with weighted mean recalculation for the clustered events and with threshold step by *0.05*. For the second data set consisting of 10415 flows in 221 events and 15 incidents the best values determined by the F-measure method were the following:

- query similarity threshold: 0.5
- path similarity threshold: 0.45
- bytes and request timing similarity threshold: 0.5

However it's necessary to note, that these values may differ based on the labeled data. Also the F-measure formulas (2.5.2) can be made more complex to give different weights for FP[24] or FN[25] results.

5.3 Combined similarity function experiments

In the previous section I have shown the behaviour of each similarity function standalone, but in the final framework all of them are used to create the final clusters. The final consensus matrix algorithm (3.4.6) does not have any parameters, only the individual partitioning as input. Below you can see an overview table of measurement with the third data set consisting of 500 security incidents with ~27,000 events and 2.3 millions of flows in total.

24 False positive
25 False negative

		without classification information used	**with classification information used**
clusters	total	19 126	19 171
	bigger than 1	3 193	3 188
events in clusters bigger than 1	total	10 410	10 389
	avg	3.26	3.26
	med	2	2
flows in cluster bigger than 1	total	1 604 297	1 603 994
	avg	502.44	503.13
	med	12	12
clusters with mixed classifications		32	0

Figure 21: Combined similarity overview

For the first measurement I have ignored the event classification information. All the ~27,000 events were clustered into ~19,000 clusters, which is about 30% reduction in the total number. In clusters containing multiple events, over 10,000 events were contained, with average number of 3.26 event per cluster. In terms of flows, 70% of them were contained in those clustered events. Only 32 clusters contained events with multiple different specific classifications, what we can consider as a loss of information.

In the second run I have enabled the mechanism mentioned in 3.4.4 and 3.4.5 to avoid this type of false positive clusters by taking the classifications into account. As the clusters with different classifications got split, the total number of clusters has increased by 45, which seems like a reasonable trade off for eliminating all 32 false positive clusters.

6 Conclusion

In the last chapter of this thesis the assignments will be analyzed with references to relevant sections and chapters. After the analysis, a section summarizes the general work and results. The very last section of this chapter and thesis is dedicated to future plans.

6.1 Assignment completion

6.1.1 Study and analyze existing algorithms for identifying string and behaviour similarities.

Some of the recently published algorithms and similarity function for strings, URL and network traffic similarities are mentioned in 2.4. As the framework uses multiple metrics in combination, in 2.5 I have included some options for combination of their results.

6.1.2 In “real-world” data related to detected incidents, identify similarities between different types of malicious behaviour. Focus on Command and Control, adware and click fraud.

The common similarities for malicious HTTP and HTTPS traffic noted in 2.4 are the connection destination, the characteristics of the data transferred and connections timing, the requested resource of the remote servers and the parameters sent in the requests.

6.1.3 Implement at least two algorithms and in the automated detection framework.

In chapter 3 I have proposed design of a framework using multiple metrics. In chapter 4 I have described some patterns necessary for each logical component of the framework to hold to ensure the framework is scalable in the number of similarity functions and also in terms of input and output data formats.

6.1.4 Determine and measure in % if and how much can be the data describing incidents reduced using the automated algorithms with preserving the information value.

The chapter 5 contains statistics from the input data and measurements of data reduction of malicious network behaviour into similar clusters, and their interpretation.

6.2 Summary

The goal of this thesis was to study existing algorithms identifying similarities in network traffic and behaviour, and implement multiple of them in a framework.

My first contact with this area was in my bachelor thesis, where I have tested and compared different approaches for full-text search in URL addresses itself. Later during my part time contract I worked under the supervisor of this thesis, Ing. Martin Rehák Ph.D., on processing malicious traffic data, where I realized the problem of an effective presentation of the identified security incidents.

Althought the measured numbers show clear reduction in the data in terms of possible redundancy, the main benefit of the framework is it allowed to measure these with the potential to add and benchmark more metrics in the future.

During the work on the thesis, I have learned some new skills. Fro the technical ones most notable the Kotlin language I have used. Compared to Java, I was used on, Kotlin had broader support and use of the functional programming paradigm and I better understand its strengths now. Secondly, I have better understood the security incidents information value and the related network traffic specifics, despite maybe not in all details.

To sum up I need to notice it was a great experience working on the thesis, broadening my knowledge and experience of the field.

6.3 Future work

After the three similarity functions have been implemented in the framework, there is some work needed before an actual deployment. First the supported input data format is limiting for high volumes and only files are supported as input now. To use the framework as part of a bigger application, it needs to be integrated depending on the specific data model and storage.

The framework itself does not allow any sort of pluggable modules, but is designed to allow changes or additions in the set of the similarity functions. The claimed accuracy of N-gram similarity may be worth to compare with the existing framework metrics with the option of replacing or supplementing the path similarity in the future. The selection of metrics can be also affected by how many data from the input data set it can actually process.

7 References

1: Cisco VNI. The Zettabyte Era: Trends and Analysis. 2017. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html
2: Statista.com. Number of ransomware attacks worldwide from 2014 to 2016. 2017. https://www.statista.com/statistics/494947/ransomware-attacks-per-year-worldwide/
3: Dyfed Loesche. Ransomware Makes up Small Share of Growing Malware Threat. 2017. https://www.statista.com/chart/10045/new-malware-specimen-and-share-of-windows-based-malware/
4: Kaspersky Lab. Financial costs incurred by malware incidents according to internet users worldwide as of June 2015. 2015. https://www.statista.com/statistics/463714/financial-costs-incurred-by-malware-incidents/
5: Honeywell. Industrial-Level Intelligence: Stand Up to Industrial Cyber Risks. 2014, https://www.honeywellprocess.com/library/marketing/brochures/Honeywell-RiskManager-Infographic.pdf
6: Statista.com. Total annualized cost of cyber crime targeting U.S. companies in 2014 and 2015. 2015. https://www.statista.com/statistics/193444/financial-damage-caused-by-cyber-attacks-in-the-us/
7: Steve Morgan. Top 5 cybersecurity facts, figures and statistics for 2017. 2017. https://www.csoonline.com/article/3153707/security/top-5-cybersecurity-facts-figures-and-statistics-for-2017.html
8: Techopedia. What is a Cyberattack? - Definition from Techopedia. 2017. https://www.techopedia.com/definition/24748/cyberattack
9: Techopedia. What is a Botnet? - Definition from Techopedia. 2017. https://www.techopedia.com/definition/384/botnet
10: Wikipedia contributors. History of the Internet - Wikipedia. 2017. https://en.wikipedia.org/wiki/History_of_the_Internet
11: internetworldstats.com. Number of internet users worldwide from 2009 to 2017, by region. 2017. https://www.statista.com/statistics/265147/number-of-worldwide-internet-users-by-region/
12: Český statistický úřad. Internet v mobilu má 41 % dospělých Čechů. 2017. https://www.czso.cz/csu/czso/internet-v-mobilu-ma-41-dospelych-cechu
13: eMarketer. Number of smartphone users worldwide from 2014 to 2020 (in billions). 2016. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/
14: Wikipedia contributors. Apple Watch - Wikipedia. 2017. https://en.wikipedia.org/wiki/Apple_Watch

15: Louis Columbus. Roundup Of Internet Of Things Forecasts And Market Estimates. 2016. https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/#2c7cde76292d
16: Kyle York. Dyn Statement on 10/21/2016 DDoS Attack. 2016. https://dyn.com/blog/dyn-statement-on-10212016-ddos-attack/
17: DownDetector, CC BY-SA 4.0. A map of internet outages in Europe and North America caused by the Dyn cyberattack (as of 21 October 2016 1:45pm Pacific Time).. 2016. https://commons.wikimedia.org/wiki/File:Level3_Outage_Map_(US)_-_21_October_2016.png
18: Public domain. WannaCry Decryptor. 2017. https://en.wikipedia.org/wiki/File:Wana_Decrypt0r_screenshot.png
19: Violet Blue. CryptoLocker's crimewave: A trail of millions in laundered Bitcoin. 2013. http://www.zdnet.com/article/cryptolockers-crimewave-a-trail-of-millions-in-laundered-bitcoin/
20: Matt Suiche. Petya.2017 is a wiper not a ransomware. 2017. https://blog.comae.io/petya-2017-is-a-wiper-not-a-ransomware-9ea1d8961d3b
21: Troy Hunt. Have I been pwned? Pwned websites. 2017. https://haveibeenpwned.com/PwnedWebsites
22: Identity Theft Resource Center. Annual number of data breaches and exposed records in the United States from 2005 to 2016. 2017. https://www.statista.com/statistics/273550/data-breaches-recorded-in-the-united-states-by-number-of-breaches-and-records-exposed/
23: Bitkom, Bundeskriminalamt. Which of the following types of security software do you use on your private computer?. 2014. https://www.statista.com/statistics/429652/internet-users-security-software-usage-germany/
24: OPSWAT (Metadefender.com). Market share held by the leading Windows anti-malware application vendors worldwide, as of August 2017. 2017. https://metadefender.opswat.com/reports/anti-malware-market-share#!/
25: Bruno Pedrozo, CC BY-SA 3.0. Schéma d'un pare-feu entre un LAN et un WAN. 2017. https://commons.wikimedia.org/wiki/File:Firewall.png
26: Avivah Litan, Mark Nicolett. Market Guide for User Behavior Analytics. 2014. https://www.gartner.com/doc/2831117/market-guide-user-behavior-analytics
27: Trustwave. Which IT security tasks are you facing the most pressure to address?. 2017. https://www.statista.com/statistics/709789/most-pressing-global-cyber-security-issues/
28: Amrit T. Williams, Mark Nicolett . Improve IT Security With Vulnerability Management. 2005. https://www.gartner.com/doc/480703/improve-it-security-vulnerability-management
29: CGP Holdings, Inc.. DNSBL Information - Spam Database and Blacklist Check. 2017. https://www.dnsbl.info/
30: Tri-Hai Nguyen, Myungsik Yoo. A behavior-based mobile malware detection model in software-defined networking . IEEE. 2017. 978-1-5386-2168-4

31: Neetu Singh, Narendra S. Chaudhari. N-gram approach for a URL similarity measure. 2016 1st India International Conference on Information Processing (IICIP). 2016. 978-1-4673-6984-8
32: J. Jusko, M. Rehak, J. Stiborek, J. Kohout and T. Pevny. Using Behavioral Similarity for Botnet Command-and-Control Discovery. IEEE Intelligent System. 2016.
33: Josh Aas. Looking Forward to 2018. 2017. https://letsencrypt.org/2017/12/07/looking-forward-to-2018.html
34: Tao Li, Mitsunori Ogihara, Sheng Ma. On combining multiple clusterings. CIKM. 2004.
35: Ana L.N. Fred, Anil K. Jain. Combining multiple clusterings using evidence accumulation. 2005. http://ieeexplore.ieee.org/document/1432715/
36: Kristine Daniels, Christophe Giraud-Carrier. Learning the Threshold in Hierarchical Agglomerative Clustering. 5th International Conference on Machine Learning and Applications, 2006. ICMLA '06. 2006.
37: Lars Behnke. Implementation of an agglomerative hierarchical clustering algorithm in Java.. 2017. https://github.com/lbehnke/hierarchical-clustering-java

8 Attachments

8.1 Cumulative histograms values

8.1.1 Query similarity

	0.0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5
%	87.71	89.68	90.72	91.55	92.01	92.37	92.57	92.73	92.90	93.07	93.17
	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1.0	
%	93.25	93.38	93.48	93.57	93.77	93.89	94.08	93.34	94.61	100	

8.1.2 Path similarity

	0.0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5
%	0.90	45.06	56.04	64.85	74.57	79.66	80.17	86.41	86.60	86.96	94.46
	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1.0	
%	94.52	94.58	94.70	94.74	95.00	95.37	95.40	95.48	95.51	100	

8.1.3 Bytes and request timing similarity

	0.0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5
%	20.81	53.50	62.61	68.76	73.72	77.81	80.98	83.46	85.44	87.21	88.68
	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1.0	
%	90.02	91.21	92.35	93.26	94.28	95.35	96.43	97.41	98.31	100	