BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

REAL-TIME DETECTION OF MALWARE CAMPAIGNS

DETEKCE KAMPANÍ ŠKODLIVÉHO SOFTWARU V REÁLNÉM ČASE

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. PATRIK HOLOP**
AUTOR PRÁCE

SUPERVISOR **Ing. LUKÁŠ ZOBAL**
VEDOUCÍ PRÁCE

BRNO 2021

Brno University of Technology
Faculty of Information Technology

Department of Information Systems (DIFS) Academic year 2020/2021

Master's Thesis Specification

Student: **Holop Patrik, Bc.**
Programme: Information Technology and Artificial Intelligence Specialization: Cybersecurity
Title: **Real-Time Detection of Malware Campaigns**
Category: Security

Assignment:

1. Familiarize yourself with the topics of malicious software (malware) and threat intelligence. Focus on types of malware, division into families, ways of distribution, and spreading in waves called campaigns.
2. Review internal Avast systems providing data and events concerning client detections. Concentrate on the use of available information for recognition of malware campaigns for threat intelligence purposes.
3. Design a system for real-time detection, visualization, monitoring, and reporting of malware campaigns. Real-time means that the system should focus on current data, not on being run retrospectively over old data.
4. Implement the system designed in the previous point.
5. Verify correctness of the implementation via a suite of unit, integration, and end-to-end tests.
6. Evaluate the implemented system with respect to its malware campaign detection abilities and use for threat intelligence inside Avast.
7. Assess your work and discuss possible future improvements.

Recommended literature:

- M. Sikorski and A. Honig: Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, No Starch Press (2012)
- Recorded Future: The Threat Intelligence Handbook, CyberEdge Group (2018)
- B. Ellis: Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data, Wiley (2014)
- Internal Avast documentation
- Additional literature as recommended by the supervisor or consultant

Requirements for the semestral defence:

- The first three items of the assignment and a part of the fourth item.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Zobal Lukáš, Ing.**
Consultant: Zemek Petr, Ing., Avast
Head of Department: Kolář Dušan, doc. Dr. Ing.
Beginning of work: November 1, 2020
Submission deadline: May 19, 2021
Approval date: October 22, 2020

Abstract

This thesis deals with a real-time detection of malware campaigns based on the available data of internal tools used in the Avast Software company. Its goal is to design and implement a system that obtains and processes messages representing incidents detected at clients. The system extracts and processes useful information and estimates if the threat data are related to an emerging or continuous malware campaign based on various criteria. The experimentation proves that campaign detection based on the carefully selected data and metrics is possible. The implemented system is integrated with other internal tools of the Avast Software company. This thesis also suggests steps for further improving the detection process.

Abstrakt

Táto práca sa zaoberá detekciou kampaní škodlivého softwaru v reálnom čase na základe dostupných dát z interných nástrojov spoločnosti Avast Software. Jej cieľom je navrhnúť a implementovať systém, ktorý dokáže automatizovane zachytávať a spracovávať správy o vzniknutých udalostiach a incidentoch u klientov, získať z nich potrebné informácie a vyhodnotiť, či sa jedná o prebiehajúcu kampaň škodlivého softwaru na základe rôznych kritérií. Experimentovanie potvrdzuje, že detegovanie kampaní na základe podrobne vybratých parametrov a metrík je možné. Implementovaný systém je integrovaný pre spoluprácu s internými nástrojmi spoločnosti Avast Software. Táto práca taktiež navrhuje možné vylepšenia detekčného procesu.

Keywords

malware, campaign, antivirus, detection, real-time event processing, Avast Software

Klíčová slova

škodlivý software, kampaň, antivírus, detekcia, spracovanie udalostí v reálnom čase, Avast Software

Reference

HOLOP, Patrik. *Real-Time Detection of Malware Campaigns.* Brno, 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Lukáš Zobal

Rozšířený abstrakt

Cieľom tejto práce je preskúmať možnosti detekcie kampaní škodlivého softwaru v reálnom čase na základe dostupných dát o incidentoch a detekciách rôznych hrozieb.

V úvode práce sa objasňuje dôvod, prečo je detekcia kampaní potrebná a na čo sa dajú využívať získané informácie. Naväzuje naň druhá kapitola, ktorá podrobne popisuje problematiku prezentovanú v úvode. Touto oblasťou sa zaoberá threat intelligence, oblasť kybernetickej bezpečnosti, ktorá umožňuje sledovať špecializovaným spoločnostiam a výskumníkom aktuálne hrozby, ich výskyt a pohyb na globálnej svetovej úrovni. Kampaňou sa rozumie dočasne zvýšený výskyt konkrétneho typu hrozby, súboru alebo jeho rodiny. Správna detekcia jednotlivých hrozieb, či už sa jedná o súbory alebo celé rodiny škodlivého softwaru, do ktorých je možné hrozby podobnej povahy a funkcionality klasifikovať, umožňuje nielen ich lepšiu detekciu a ochranu pred nimi, ale aj možnosť predpovede vývoja hrozieb.

V druhej kapitole sa rovnako objasňuje aj klasifikácia škodlivého softwaru a hrozieb do jednotlivých typov, ktoré zdieľajú všeobecné vlastnosti a vzory správania, prípadne účel, za ktorým bol škodlivý software vytvorený. Pre ich jednoduchšiu detekciu a konkrétnejší popis je nutné typy klasifikovať do menších podmnožín zvaných rodiny škodlivého softvéru. Vzorky v rámci jednej rodiny už majú veľmi podobné vzory správania a nízku variabilitu. Daná klasifikácia medzi typy a rodiny umožňuje hierarchické rozdelenie hrozieb, ale v praxi sa často vyskytujú hrozby, ktoré majú rôzne vzory správania a bolo by ich možné zaklasifikovať do rôznych typov zároveň, preto je nutné predpokladať i túto variantu.

Nasledovné kapitoly vysvetľujú spracovanie správ od klientov, ktoré nesú informácie o bezpečnostných incidentoch, detekciách a iných udalostiach. Dané správy môžu byť agregované a ich prenos zabezpečuje infraštruktúrny systém pre spracovanie udalostí Apache Kafka. Medzi typy správ, ktoré je pri detekcii kampaní v reálnom čase možné využiť, patria dáta z reputačných systémov pre jednotlivé súbory, informácie o detekciách súborových hrozieb, webové hrozby ako presmerovanie na nebezpečné internetové stránky atď. Pre každý typ alebo skupinu správ v rámci jedného systému je vyhradený samostatný kanál na prenos, kde je možné pomocou klientov správy odchytávať a spracovávať. Napriek tomu, že jednotlivé typy správ sú verzované a majú pevnú štruktúru, každý systém môže v správach ponúkať iné dáta ako napr. názvy súborov, ich heše a prípadne konkrétne informácie.

Za využitia informácii z predchodzích kapitol je možné navrhnúť službu, ktorá bude na základe daných správ vyhodnocovať možný výskyt kampaní pre jednotlivé súbory a prípadne rodiny či rôzne typy jednotlivých artefaktov reprezentujúcich vlastnosti s bližším popisom danej hrozby. Služba je nazvaná CaDet (Campaign Detector) a umožňuje nielen ukladanie informácií o detekovaných kampaniach, ale aj tvorbu upozornení na kanáloch pre komunikáciu v rámci spoločnosti, prípadne jej priebežné vyhodnocovanie voči voľne dostupným informáciám o kampaniach malwaru zo sociálnych sietí a internetu. Služba bola implementovaná v jazyku Python, verzovaná pomocou systému pre správu verzií Git, využíva databázové technológie NoSQL pre ukladanie informácií pre jednotlivé kampane a vďaka službe Docker poskytuje jednoduché možnosti nasadenia. Webové rozhranie dokáže prezentovať výskyt detekovaných kampaní vo forme interaktívnych máp, zobrazuje grafy aktuálnej aktivity sledovaných udalostí a poskytuje detailné informácie o danej hrozbe. Implementovaná služba taktiež poskytuje API pre interakciu s automatizovanými skriptami a inými analyzačnými službami. Celý proces je monitorovaný pomocou platformy Grafana a umožňuje generovať správy s upozornením o detekovanej kampani v rámci komunikačnej platformy Slack.

Služba CaDet bola otestovaná sadou jednotkových testov, ktoré majú za cieľ overiť funkcionalitu jednotlivých funkcií a modulov bez externých závislostí, a integračných testov, ktoré overujú spoluprácu jednotlivých modulov aj externých služieb. Pre správne overenie reálnych scenárov a udalostí, ktoré možu nastať, bola implementovaná aj sada end-to-end testov. Súčasťou vyhodnotenia funkcionality nepatrili len zmienené funkčné typy testov, ale aj overenie správnej detekcie kampaní škodlivého softwaru voči informačným správam zo sveta získaných zo sociálnych sietí i interných nástrojov. Celý proces experimentácie a jednotlivých detekcií bol vyhodnocovaný priebežne na aktuálnych hrozbách v danom období. Experimentácia potvrdila, že navrhnuté detekčné pravidlá sú schopné detekovať aj kampane, ktoré boli reportované externými zdrojmi.

V poslednej časti sú diskutované možné vylepšenia služby a prípadné aktuálne nedostatky, ktoré odhalil proces experimentácie. Záver práce vyhodnocuje aktuálny postup a zhŕňa získané poznatky.

Real-Time Detection of Malware Campaigns

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Ing. Lukáš Zobal. The supplementary information was provided by Ing. Petr Zemek, PhD as a consultant of the Avast Software company. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Patrik Holop
May 17, 2021

Acknowledgements

I would like to thank the above-mentioned Ing. Lukáš Zobal and Ing. Petr Zemek, PhD for their valuable advice and their time required for consultations of this thesis.

Contents

Chapter 1

Introduction

Digital threats have significantly evolved since their emergence together with their targets and focuses. People no longer need an expertise or higher education to use various digital technologies, computers or mobile phones on a daily basis. New vulnerabilities exploitable by criminals are discovered every day [1]. A substantial amount of the reported cybersecurity incidents is caused by malware (malicious software). In the past, computer programs such as malware were commonly spread by copying binary executable files from one digital system to another. For example, Brain – a program being considered to be the first computer virus affecting MS-DOS, traveled via floppy disks [21]. Nowadays, a more frequent way of spreading malware is by the Internet and the global web. Malware may not require a physical file and its presence can be traced only in memory of the targeted machine. There exist many various types of attacks that malware can exploit to achieve its goals, such as spying or blackmailing.

Threat intelligence has emerged as a new cybersecurity area to monitor the activity of malware and other digital threats in the world. Its purpose is to gather information on the current threats and vulnerabilities, to classify them into categories for easier identification, counter-defense and their detection and to predict new trends in cybersecurity [7]. It also helps companies to inform their consumers and the general public about the techniques used by criminals and what attacks and crimes customers might be vulnerable to. Examples of the institutions that are responsible for threat intelligence and threat monitoring are antivirus companies that try to detect the threats to protect their users and publish their findings via public reports and blogs.

Keeping track of the current status of malicious files that emerge in the world is a hard and complicated goal that requires cooperation among companies and the use of deployment automation. Antivirus companies deploy various systems and tools to achieve high-quality protection and minimize the false-positive decision rate (ordinary files incorrectly classified as harmful) [29]. End-user products installed on the protected systems inform the company about detected files and incidents that have occurred. They further provide enough information for the purposes of threat intelligence with strict respect to the privacy of the protected users. The company might use a specialized architecture and systems to collect all relevant data about such events and to process and evaluate them.

Incident reports provide much valuable information that can be used for threat intelligence like attributes of the related files, gathered artifacts from the analysis, patterns of behavior or their origin. These data can be analyzed to help the antivirus companies to enrich their detection capabilities and protect more users as fast as possible against the majority of the newly emerged threats. Real-time monitoring of such events can also give

malware analysts and security researchers a broad overview of how the threats spread. It also provides information about various types of threats that are currently prevalent in the world without the need for targeted information gathering from the past data.

The goal of this thesis is to analyze the above-mentioned events and their internal sources of the Avast Software company, design and implement an automated system that is able to gather and process such events. The designed system will be used to detect newly emerged or re-emerged malware campaigns with a minimal delay to alert researchers and employees of the antivirus company about the current threats. This helps to decrease the time necessary for a mitigation of the threat and lowers the researcher's need to look for specific information retrospectively when it might be too late and the malware has already infected more users than it was necessary.

The text is structured as follows. Chapter 2 further explains the term threat intelligence and its applications, how it differs from the threat data and its connection to malware campaigns. It also discusses the hierarchical classification of threats into severity, types and families and provides examples for each category. Threat data are required for campaign detection. Their sources consisting of internal analysis and reputation systems are described in Chapter 3. All mentioned systems use a shared message platform Apache Kafka to exchange produced messages. The used architecture with a closer explanation of the platform itself is provided in Chapter 4. A general design of the system as well as its goals, input data and outputs are described in Chapter 5. Chapter 6 sums up the details of the implementation and technologies used by the service designed in the previous chapter. The detection of campaigns is based on rules and conditions described in Chapter 7. It also provides an evaluation of the service and detection statistics gathered during the experimentation process. The process of automated testing using unit, integration and end-to-end tests is discussed in Chapter 8. It is followed by a list of suggestions regarding the future improvements of the service provided in Chapter 9. The thesis ends with a summarizing conclusion in Chapter 10.

Chapter 2

Threat Intelligence

This chapter focuses on the topic of threat intelligence, its basic principles, goals and various focus areas that the cybersecurity topic includes. It also explains the aspects of malware campaigns, their possible interpretations and approaches used towards their detection. The necessity of sample classification, its hierarchical structure into malware types and families and various methods to achieve accurate results are also discussed and this chapter provides examples for each category. Accurate classifications and their ties to the detection of malware campaigns are discussed together with the importance of campaign detection [45]. Examples of the large-impact historical campaigns are provided at the end of this chapter.

2.1 Goals of Threat Intelligence

The main goal of threat intelligence is to provide closer information about the threats that have occurred or might occur in the future. The threats might be either active or mitigated. Information about their authors, origin of the threat and ways of the threat mitigation can be gathered by their analysis [7]. The field of threat intelligence does not represent a feed or a specific list of the current threats that can be encountered in a digital environment. Instead, it tries to connect the obtained threat data describing the individual threats and obtain deeper knowledge in order to find relations and non-trivial dependencies and achieve the above-mentioned goals to provide the required protection to both consumers and the general public.

2.2 Threat Analysis

One of the most important goals of antivirus companies is to analyze executable binary files, documents and other types of digital data to estimate the level of threat that they represent, as well as to gather necessary threat data that can be used for further evaluation. The analyzed files are commonly referred to as samples. Unique hashes of the samples, such as Secure Hash Algorithm, are used for easier sample identification and information exchange [29]. Samples can be analyzed either directly on the machines of end-users or sent to the cloud that deploys complex analysis systems. The latter approach decreases the processing load of the protected system. Its adoption highly increases mainly due to the expansion of the Internet of Things and might result in a safer environment because the sample is analyzed in an isolated environment [9]. There exist two main approaches towards the sample analysis described below.

2.2.1 Static Analysis

Static analysis involves information gathering from a file without its execution or simulation. Malware analysts or automated systems gather the properties of files by parsing the binary data using a decompiler (software used for reverse engineering to assemble higher-level information from the binary files) or disassembler that translates the binary code into assembly language, the symbolic machine code using standards and documentation of binary formats [22]. For other types of files, there exist specific parsers [15]. During the disassembling, file-format dependent data are shown. The data include file format headers, information about the used API and libraries or even identifiers named by the author. Once the data are parsed, analysts try to find malicious or shared patterns among the analyzed files [22].

2.2.2 Dynamic Analysis

Dynamic analysis involves running the tested sample in a sandbox environment. The sandbox environment is an isolated virtual machine, which, based on how the program behaves in the emulated environment, generates an analysis report containing, e.g., sandbox signatures [15]. Signatures generated during dynamic analysis are short labels generally describing the triggered behavior. They are mostly high-level tags describing a wide area of functionality, for example, *connected_to_internet*. Other signatures are more specific, such as *banker_behaviour*. Alternatives to sandbox environments are the emulators. While the software in the sandbox is running natively, the emulator simulates most of the machine instructions and runs the software step by step. Sandboxes are deployed as a part of malware research teams but also as a tool for behavioral analysis of samples directly at the end users [22]. Dynamic analysis might involve analysis of the process memory, network activity or file-system objects created in the environment during execution. Examples of the artifacts gathered during behavioral analysis using an online tool VirusTotal[1] are shown in Figure 2.1. It shows the screenshot taken during the analysis and obtained system paths.

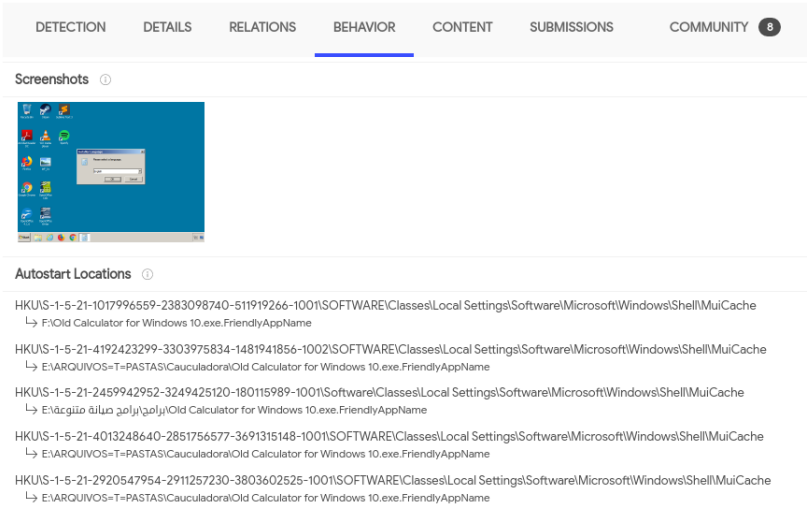

Figure 2.1: Example of artifacts gathered during behavioral analysis (VirusTotal)

[1] https://www.virustotal.com

2.3 Threat Classification

Based on the information and data obtained during both types of sample analysis described in the previous sections, an analyst or automated system can derive an abstract or precise classification of the potential threat. For various applications, different levels of classification abstraction are required. For example, to detect any sample and perform its removal from the infected system, only a knowledge of whether the file represents any general threat is required. A more precise description might be required in order to further investigate the incident and collect the related intelligence. For this reason, a hierarchical model of threat classification with various levels of abstraction is commonly used. The classification can be estimated either by malware analysts or automated expert, detection or machine-learning systems [22]. Because each classifier might use different data or consider various aspects, classification of the same sample might vary based on the classifier's metrics, such as accuracy [23]. Various levels of this hierarchical model are described below.

2.3.1 Severity

Severity represents the most abstract level of threat classification and provides information only about the general intention of the sample or whether the sample represents any threat at all. The classification is split into four main categories described below and sorted by level of threat they represent. However, various companies or classification models might use only a subset of the listed categories [15].

- **Clean** – Clean samples represent ordinary files, documents and other samples without any malicious activity or threat. It is the most prevalent sample category among the observed samples. This category includes commonly used web browsers, multimedia applications, text editors, etc.
- **Tool** – Tools are clean samples that are used to perform a specific type of activity that might be misclassified as suspicious or malicious under certain circumstances. Example of such a process is the mining of digital currencies, which is a legit activity that can be performed by malicious websites and samples without the user's consent.
- **PUP** – PUP (an abbreviation of *potentially unwanted programs*) fills a grey zone between clean and malicious samples. These programs do not represent any direct threat to the user, but they might cause inconvenience or rely on trickery to perform unwanted actions, such as installation or advertisement of third-party software.
- **Malware** – Malware (malicious software) represents samples that intent to harm, destroy or alter user's data without their consent, perform an illegal or highly suspicious activity and should be removed, isolated or handled otherwise to mitigate the threat they represent.

Malware often uses various sets of evasion techniques and methods, such as packing or encryption, to disguise itself as a clean sample to avoid any detection and neutralization by detection system or antivirus scanners [10]. Furthermore, any non-malicious sample might be a target of infection by certain types of malware that transforms it into malware itself, e.g., by injecting its malicious code into a clean executable file.

2.3.2 Malware Types

Severity is a satisfactory level of classification for detection of any threat, but it does not provide any closer information about the malicious activity of the malicious sample. This information might be used by security specialists in order to mitigate the threat, for purposes of threat intelligence or by malware analysts to locate and identify malicious sections of analyzed samples [39]. For these reasons, the malicious sample can be further classified into more specific malware types that represent the overall malicious activity performed by the sample, e.g., infection of clean samples, encryption of personal data to obtain a financial ransom or gathering information about the user [39]. Examples of the most common types of malware are described below [22].

- **Trojan** – Trojan (Trojan Horse) is named after an ancient Greek tale describing the statue of a wooden horse that was used as a gift handed to enemies of Greece to enter the city of Troy. In reality, the statue was a disguise for the warriors of an enemy faction. Once the statue was moved into the city, soldiers have started to plunder the city. Trojans use a similar technique and act as clean samples, but they perform malicious activities in the background without the consent of the user. This type of malware is also used as a general malware type if no precise classification is available [22].

- **Worm** – The main goal of the worm is to spread via the connected network, mail attachments, etc. They might carry a malicious payload or be used to set up a botnet. Worms do not need any physical file to spread and function in the memory of targeted systems, making them harder to detect. One of the first worms was called Morris worm, spreading via remote execution and exploiting weak passwords of the services [25].

- **Spyware** – Spyware is used to spy and gather information about its victims, targeted systems and to send the obtained data to its creator. This category includes programs that try to steal credentials or keyloggers that collect all pressed keystrokes.

- **File-infector** – File-infectors spread by infecting a legitimate software. This can be achieved by copying or injecting its own code or malicious sections into the targeted file. Code injection can also be performed by parsing and evaluating untrusted input of the application.

- **Dropper** – Droppers are used to spread, download, unpack, install or otherwise generate a different malicious sample on the targeted system. Their goal is to hide the carried malware and obfuscate the way of infecting the system.

- **Ransomware** – Malicious type with an increased prevalence in recent years that encrypts, removes or otherwise blocks access to the victim's data by blackmailing and requesting some form of ransom. Ransomware authors often request a payment in cryptocurrencies to anonymize their identity [41].

- **Bot** – Bots are automated programs to perform legit tasks, such as web scraping or data gathering. However, they might be used for malicious intentions to create a centralized or decentralized network of connected machines called a botnet. Bots are synchronized among themselves and wait for the execution commands from their command server or a certain event that triggers their action [17].

Classification of samples into malware types is not a standardized process across the world and the set of the detected types might differ in the used classification model. Moreover, the malicious sample might perform multiple malicious activities and as such is classifiable into multiple types and categories. Only a subset of detection systems supports estimation of more than one type and the category is chosen by the most important or obvious activity performed by the sample and detected by the classifier or analyst.

2.3.3 Malware Families

Malware types describe the general activity of the malicious software. However, each type can be split into variants or families representing a more specific classification of the sample. Families provide specific information about the performed activity. However, malware family might also be derived based on the malware creators, time period of its activity, relations with other families, etc. Malware families can be further divided into variants that represent specific aspects of the family, but they are typically not supported by automated classification systems due to lack of data required for training or a high number of possible classification classes. New names of the discovered variants and families are chosen by the companies and analysts that have identified them and their count is significantly higher than the general malware types [22]. This might cause inconsistencies, misclassifications and problems regarding information exchange because multiple antivirus companies might use different synonyms for the same malware families and variants [22]. Representative examples of malware families are described below [39].

- **Dealply** – It is an adware that installs the advertisement popups to a targeted web browser.
- **WannaCry** – Ransomware that has emerged in 2016 and has infected more than 300 000 users across the globe [6].
- **Zeus** – Type of trojan that tries to steal user credentials and confidential information from the victim. It spreads via phishing schemes and drive-by downloads.
- **Emotet** – This family has slightly changed its behavior from the past because it operates mainly as a banking malware organized in a cooperating botnet. Nowadays, the main observed purpose of this family is content delivery [4].
- **Wabot** – Wabot is a type of worm that spreads via IRC (Internet Relay Chat) and can change the system settings of affected clients.
- **Swisyn** – This type of trojan was first observed in 2009. It initiated connections to malicious websites and committed identifiable registry changes. It serves as a dropper for other types of malware.
- **GandCrab** – According to ZDNet, the name of this ransomware is derived from the online name of its creators, who represent themselves as *Grab* or *GandCrab*. GandCrab does not affect machines located in Russia and follows the Ransomware-as-a-Service (RaaS) marketing model [4].
- **Mirai** – Mirai is malware targeting Linux systems. The compromised systems are turned into a unified botnet for large-scale attacks. It also affects IoT devices, such as cameras and home routers.

- **XMRig Monero** – This malware is targeting vulnerable systems for the purposes of crypto mining Monero coin. It uses the resources of the targeted machines, also affecting Windows and Linux servers. This family was part of a large-scale campaign impacting over 15 million people across the globe [35].

2.4 Malware Campaigns

Malicious software of the same family tends to spread in waves called malware campaigns. This might happen soon after emergence of the malware, after a new vulnerability is discovered or due to other similar reasons. Detection systems report an increased rate of a specific type of detections tied to the given malware family [46]. WannaCry campaign in 2016 might serve as an example of a campaign in which more than 300 000 computers were infected by the same malware family in a short period of time [6]. One of the goals of threat intelligence is to detect, classify and inform about malware campaigns. There exist multiple interpretations of what a malware campaign is. Detection or even prediction of malware campaigns before they happen, e.g., interception of botnet communication, is crucial in a fast response for endpoint detection.

2.4.1 Types of Malware Campaings

Malware campaign is a general term and provides multiple approaches to the given problem. Campaigns might not be tied only to a certain family that has spread in a wave but also to property or a specific sample. Figure 2.2 shows changes of the malware family trends in the given time window obtained from reports of an online sandbox AnyRun[2]. However, data from sandbox might be altered by scans of artificial, test and theoretical samples not prevalent in the real environment. This thesis focuses on data from the end-user stations to avoid this problem. Types of malware campaigns that are analyzed as a part of this thesis are listed below.

- **Sample campaigns** – This type of campaign represents an increased activity and prevalence of the specific sample that is commonly represented by its hashed contents, such as SHA-256.
- **Family campaigns** – Family campaign is tied to a certain malware family. It might represent an aggregation of all sample campaigns classified as the same family or as a separate wave not tied to any sample campaign. The latter approach is more general because samples that are not themselves part of any sample campaign can be recognized as a part of the campaign of their corresponding family.
- **Artifact and fileless campaigns** – Each sample has a set of static or dynamic properties that can be scanned, analyzed or detected. These properties are either imports, sections and other values obtained during static analysis of the sample, or named objects and web domains that the sample tried to access when it was executed in a monitored environment. Such properties are called artifacts and their activity can be monitored in order to be evaluated, e.g., an increase of requests towards a specific web domain used to distribute malware. Some values can also be obtained from monitoring user activity, such as domains.

[2] https://any.run/malware-trends/

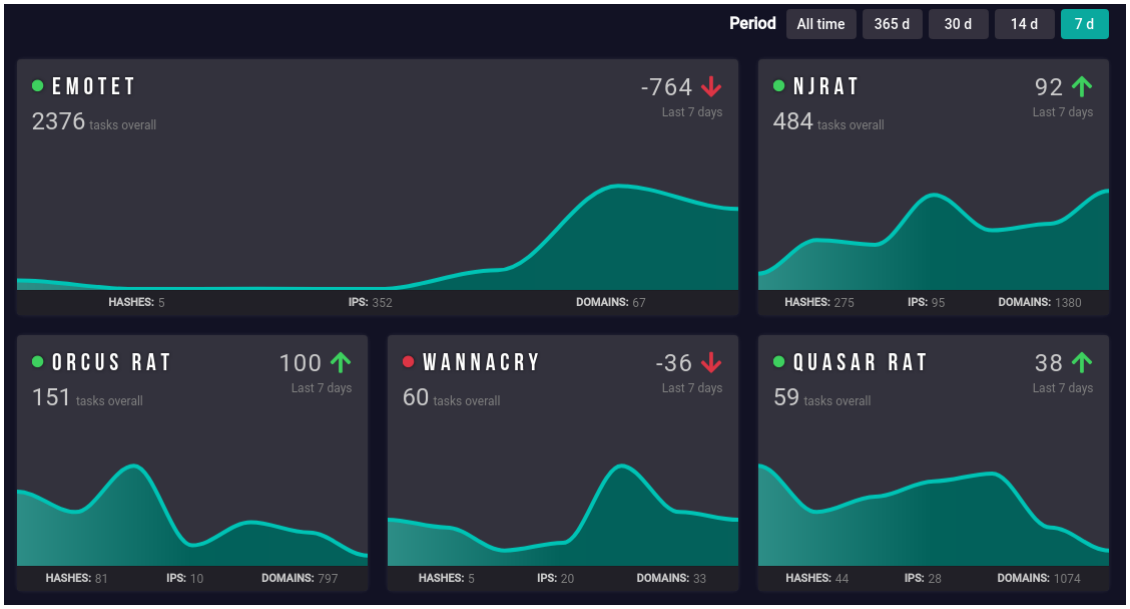

Figure 2.2: Changes of malware family trends in a public sandbox AnyRun

2.4.2 Historical Malware Campaigns

This section provides an overview of the selected malware campaigns that occurred in the past with a large-scale impact on the affected systems. The main events are described in chronological order for each campaign. The counter-defense reactions of the cybersecurity researchers are also discussed.

Wannacry Ransomware Campaign (2017)

The WannaCry ransomware campaign was a worldwide cyberattack. This cryptoworm malware started its activity in May 2017. The main targets of this campaign were machines with Windows OS. The infected users were requested to pay a financial ransom in order to decrypt their data. This campaign was estimated to affect approximately 300 000 devices across 150 countries. The initial attack is believed to happen on May 12, 2017 [50]. The first observed samples have originated in Asia. During the initial day of this campaign were reported over 230 000 infected computers. The majority of the infected devices were machines with the operating system Windows 7 [6]. The official support of this operating system ended in 2020 despite its popularity among users.

The substantial spread was partially mitigated after the discovery of a hardcoded switch directly within the malware [50]. In the initial version of WannaCry malware, the user data encryption was conditioned. The files were encrypted only when the ransomware was unable to connect to a specific domain. The security researchers used a mechanism called DNS sinkhole to monitor the queries generated by the malware [26]. DNS resolver obtains false results from the server that has blacklisted the given domain. The server is able to monitor the number of queries and the provided result is a substitute pretending the be the original domain. This allowed the analysts to observe the sample count and to stop the malware from spreading. The second switch was discovered on May 14 in a new variant of this

family, but the discovery happened the same day the new variant was observed. Other similar attempts were observed during the campaign.

In a counter-response, hacker organizations tried to attack the substituted domain to slow down the progress of the investigation. A large botnet was organized by Mirai; a malware family described below [42]. This purpose of the formed botnet was a DDoS attack, but the domain was consequently protected. The servers were to handle much more traffic and the attempts to destabilize the domain have failed. Cybersecurity researchers also attempted to recover the encryption keys. It was discovered that the keys might be recovered under specific conditions. Examples of the conditions were that the system was not rebooted and the ransomware process was not terminated otherwise. The encryption API did not completely wipe the memory, and prime numbers used for encryption could be obtained from the memory dump. The main response ended four days after the start of the campaign, but the WannaCry samples are still being observed by threat-intelligence researchers [3].

Mirai Botnet Campaign (2017)

A new campaign of Mirai Botnet has emerged on March 25, 2017 [43]. This was not an emergence of a new threat because Mirai botnets were already used for DDoS attacks in 2016 [42]. A new rising trend of command injection techniques used by this variant of malware was spotted by cybersecurity researchers. The formation of a new botnet was confirmed by the analysis of spread samples and collected events.

The malware name *Mirai* was formed based on a tool that used to scan various IoT devices with weak or default login credentials [37]. The credentials were used to connect to the device, and this led to post-exploitation and organization of the affected devices into a large botnet. Botnets created by malware families such as Mirai use Command and Control server (C&C) for distribution of information and commands. The server listens for responses from individual bots that scan the network for a potential new device to be exploited. If such device is found, the bot reports the IP address of the device to its command server. The affected devices continue normal activity and operations until the bot receives a specific command.

The countries of North America and Japan were among the affected destinations by this campaign [43]. The samples that were part of this campaign had cryptomining modules built in, which allowed the exploitation process to use the resources of the infected devices for mining. Furthermore, they also included DDoS components that allowed the botnet to cooperate in a synchronized attack. The vulnerable service used to spread this family was Telnet with a default port 23.

Ryuk Ransomware Campaign (2020)

In September 2020, security researchers observed a group of related phishing campaigns [18]. This required the involvement of incident response teams to mitigate the emerged threats. Ryuk is a ransomware that is able to bypass multiple anti-malware measurements. If necessary, it is also able to disable a computer network. An analysis of the techniques and tools used by Ryuk's authors was provided by Sophos Managed Threat Response team [18].

Ryuk did not pose a newly discovered threat. It has been repeatedly detected since 2018 [4]. The British security software and hardware company Sophos reported a low number of detections in the first half of 2020. However, the new campaign, which took place in September 2020, represented the next evolution of Ryuk. It used new techniques

to target a vulnerable network and deploy its ransomware modules. The spread was caused by a malicious document that was able to execute another malicious executable that served as a loader. The *Buer Loader* used was different from the one used by Ryuk in the past, relying on Emotet. Ryuk dropped other malicious files that served as a connection to the attacker's hosted C&C server, allowing the attackers to continue the exploitation of the company's internal network. It also used various techniques to distribute its files to discovered servers, such as Remote Desktop Protocol (RDP) clipboard transfer [18].

Chapter 3

Threat-Data Sources

This chapter describes the sources of threat data that are obtained during various types of analyses discussed in the previous chapter. Threat data are shared via a common message platform and are the main source of information regarding malware campaigns. The data needs to be processed and evaluated in order to detect and visualize malware campaigns in this thesis. A summary of internal detection systems and specific parts of antivirus products that are used for analysis, scanning, incident handling and processing of the obtained data are also presented in this chapter. The mentioned systems, as well as the process of analysis, are described in general and abstract terms because most of them are the intellectual property of the Avast Software company. Their description is based on the internal documentation of the company [30, 32, 45, 46].

3.1 Reputation Systems

The first source of data described in this chapter is multiple reputation systems. Each system is focused on a different type of sample. A reputation system is an application or algorithm that allows rating all evaluated entities based on various rules in order to build trust and reliability in the rated entity. Reputation systems that evaluate and rate individual samples, as well as distribute the obtained information about them, are used for the purposes of threat intelligence and managing malware detections.

Among the basic information that each reputation system in the Avast Software company processes, there are the date and the time of emergence of the samples and a total amount of their detected and confirmed occurrences in the end-user clients, referred to as prevalence. Prevalence of the occurrence detected in other analysis systems, such as the sandbox deployed in the client application, is stored and handled separately from the prevalence of sample detected by common scanners on the target system. A user can request analysis of suspicious files in the sandbox or such analysis can be triggered by an execution of an unknown sample, about which the client has no information. The current classification of the sample is also distributed via the reputation system.

The reputation system handles two different classification metrics. The first type of data are the results of complex analysis in the company, and the second is a first-line classifier deciding if the sample requires any further analysis. The latter classifier is less precise the former. However, its results are useful if the results from the complex analysis are not available.

The prevalence of the data affects the way of how the analysis systems might make their classification regarding classified threat-level. Samples with a very high prevalence are distributed among many users across the world and often represent clean severity. When a scanner encounters a unique file that has a very low prevalence, the file might be sent to the cloud or take a longer time for execution of deeper analysis than the time required for analysis of more prevalent samples, in which the system puts more trust. Such samples with a very low prevalence are referred to as *loners*. If the sample is prevalent, there is also a higher chance that a valid classification already exists for the given sample.

Because the amount and variety of evaluated samples processed among all client applications of the company is significantly high, the reputation systems are designed as distributed platforms with multiple end nodes, to which the client can be connected in order to share information about the newly observed sample or executed analysis. Each reputation system is tied to a certain file format, about which the system collects necessary information.

In this thesis, the data from two independent reputation systems are used, one for PE file format commonly used on Windows operating system and the second one for APK files supported by Android on mobile devices [8, 24].

3.2 Anti-Rootkit Systems

Anti-rootkit systems are designed to mitigate the risk and fight the malicious software classified as a *rootkit*, the purpose of which is to prepare the infected environment for further infection by another type of malware or to hide the existing deployed malware, acting as an environmental management program. Rootkits operate on low levels of the operating system and try to hide the running processes, files of the filesystem or alter registries to prevent the detection. Anti-rootkits use drivers to access the lower-level layers of the operating system similarly to the rootkits as well as other techniques to counter the rootkit activity.

Each incident and the generated detection by the anti-rootkit system of the Avast Software company contains an identifier of the sample that has caused the incident, commonly its SHA-256 hash. Among the other provided data, there is the reason for the detection incident, e.g., suspicious process behavior, meta-data of the sample and the process context, such as its parent process path and the related registry key.

3.3 Web-Threat Systems

The number of Internet users is constantly rising and the number of web threats as well. The topic of web threats has become even more important with the rise of the Internet of Things, because such systems are often targets of attempts to create botnet networks that might be used for DDoS attacks or are vulnerable to such attacks themselves [28]. Attackers use more and more sophisticated techniques to distribute or execute all types of malware over the Internet, web pages and redirects. Files and executable samples can access the domains similarly as the users do via web browsers and use it for malicious purposes or malware distribution.

The end-user client software of the antivirus products tries to monitor web threats and report the encountered incidents. A user does not need to access a malicious website directly because even an infected web or malicious link might, through multiple redirect attempts, connect to the malicious websites that are parts of the redirection chain. Analysis

of the whole redirection chain is complicated and requires much detected data which can be used for purposes of threat intelligence. Among such data, there are attempts of cross-site redirection and cross-site request forgery, detections of suspicious HTTPS requests, suspicious cookies and verification by a list of blacklisted websites that have been marked as phishing or malicious in the past. Analysis of target IP addresses is also required because an attacker can use redirection for man-in-the-middle (MITM) attacks. This can be achieved by monitoring the network activity and intervention in case of suspicious traffic.

Domain Monitoring

Similarly to the activity of reputation systems described in Section 3.1, web-protection systems process domain reputation to prevent access to any of the blacklisted or malicious sites. The rating of the analyzed website is connected with multiple reputation ratings of other properties, such as the rating of binary files downloaded from the domain. If the website distributes malware, it can be marked as dangerous as well. The system needs to maintain the list of forbidden sites that is constantly updated and allow analysts to remove the detection or change the classification in case it was evaluated incorrectly or it is a false-positive result.

Network Protocol Monitoring

Network monitoring systems and web shields often monitor various network protocols in order to ensure that the requests are not targeting domains with a bad reputation. This does not affect only older protocols like HTTP, but newer ones as well, e.g., QUIC developed by Google to provide less connection overhead [34]. When the application tries to connect through a certain protocol, the monitoring system can collect the same information as the process monitor, such as a parent process or the hash of the sample that has initiated the connection. Requests of network protocols are also evaluated based on the related certificates that might be signed, invalid, or revoked, on headers of the requests, etc.

DNS (Domain name system) is also vulnerable to malicious intentions through data tunneling and it is dangerous to access the DNS servers blacklisted by any resolver [54]. Through DNS monitoring, the analysis systems are able to quickly evaluate the accessed server and its reputation to provide their feedback, terminate the connection or update the blacklist if necessary.

3.4 Mobile-Threat Systems

Mobile devices are vulnerable to general web threats described in Section 3.3, but also to platform-specific exploits. Installation of the third-party software obtained from an unknown or unverified source can lead to the execution of malicious code on the device as well as accessing a malicious website. With the rising number of mobile users, major antivirus companies often provide a mobile client to extend the range of their protection and obtain reports about the incidents that has happened on the systems of mobile phones. This thesis focuses on data provided by an Android client.

When an incident happens on a mobile device, monitoring client can provide multiple data about the malicious APK that has caused the incident, such as the name of the package, its version and meta-data, system flags for the given application, its granted permissions or information about the operating system as well as the current reputation of the sample

in the time of the incident. Associated certificates of the application can also be obtained from the incident report. However, Android devices do not tend to verify a whole chain of certificates with a root certification authority as other file formats like PE designed by Microsoft do [8, 24].

3.5 Tagging System

As opposed to the systems described in the previous sections, the internal tagging tool of the Avast Software company called *Tagger* is an analytical tool not directly communicating with the end-user client applications installed by the users of the antivirus company. It was designed as a high-performance platform for malware research purposes. Its main goal is to store and provide tags for samples, detection strings and other objects [44]. The tag represents a tuple consisting of severity, type, family, a strain of the family and the target platform. Tagger continuously evaluates and updates the tags based on partial classifications and detection statistics provided by other internal systems and can serve as a main source of classification for other systems. It is also able to determine the confidence of the classification, based on which was the tag generated. Low-confidence classifications are less reliable than tags with high confidence. This system supports multiple file formats and acts as a general, robust and trusty classifier as well as the source or retrospective evaluations based on accessing past data to generate various types of reports. As the main identification for a gathering of classification tags, SHA-256 hashes of the samples are used [44].

Detections and Detection Definitions

One of the objects that Tagger is able to evaluate and create tags for are detections and detection definitions. Detection definitions represent a summary of the defined conditions and rules that the given sample needs to have in order to be detected by them [45]. It is a common way of endpoint malware detection [44]. Detection definitions have a unique name and also contain information about its origin, e.g., an algorithm of their creation. For example, detection definitions can be obtained by hashing multiple sections of the binary sample. Definitions can be created for non-malicious files as well, so they require a tag providing closer information on the subject of the detection. Once the conditions provided by definition were met, the result is called detection.

Detection, also represented by a structured string, contains specific values obtained or calculated by the detection algorithms that can be parsed to gather more information about the incident and sample [44]. Figure 3.1 shows example of the detection string that includes information about the platform `Win32` (Windows OS with a 32 bit architecture). It was generated as a generic malware detection (`Malware-gen`), it shows the internal type of detection (`PE3`) and a more precise classification estimated to be the Trojan malware type (`troj`).

```
Win32:Malware-gen|PE3-F4A5CB560011BB85FE9A584D916D51A6|troj
```

Figure 3.1: Example of Avast Software detection string

3.6 Clustering System

Clusty is the internal clustering system used at Avast Software to cluster files based on their shared properties obtained during static and dynamic analysis. This service collects up to 1 000 000 new samples from a variety of internal sources on a daily basis. It groups the similar analyzed samples into clusters. Clusty uses its own internal classifier or other external sources like the parsed detections of multiple antivirus products to estimate the final classification of the whole cluster [32]. The analyzed sample may be sent to other internal analysis tools, such as sandboxes, for deeper analysis if no report is currently available. Analysis tools used during the sample processing are determined by various factors like the file format of the sample. Figure 3.2 shows an example of the generated cluster, the list of shared properties of the individual samples and the final classification by antivirus detections.

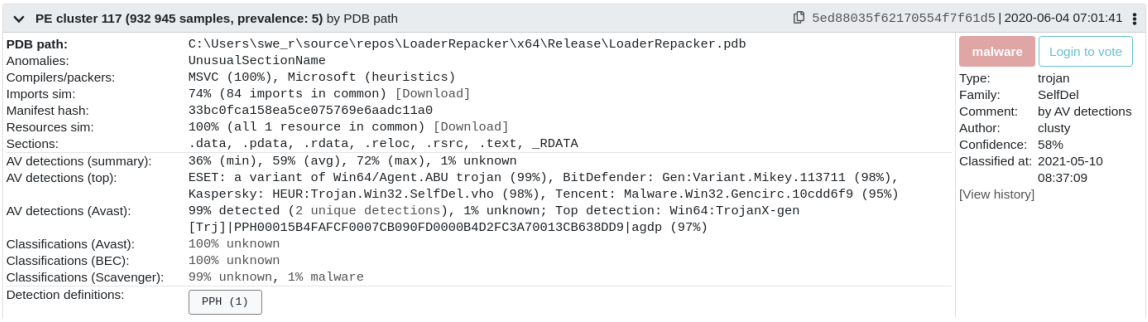

Figure 3.2: Example of the generated cluster

Once the most important data about the file are gathered, the clustering process is initiated and the most suitable cluster is found for the given sample. Specific conditions need to be met in order for the sample to be clustered. Each sample can be assigned to a single cluster. The properties of the cluster need to be updated once the suitable cluster was found and the sample was assigned. The sample might introduce new properties or not use the ones that previously represented the whole cluster, such as sections and names of the imported libraries. Each property has predefined importance that is taken into account during the clustering process. The sample is more likely to be clustered based on the property with higher importance.

Priorities on the individual properties are determined by experimentation and expert feedback of the malware analysts. More generic properties are less important than the prioritized ones. The list containing the four most important properties taken into account of the PE files, except the validity of the file and matching rules used for detection and classification, are:

1. A verified digital signature
2. Mutexes (100% match)
3. Schedules tasks (100% match)
4. Named sections (100% match)

A verified digital signature has a higher priority than the named sections. As is shown in the case of mutexes, scheduled tasks and named sections, clusters created by these properties require all samples to have the same values and achieve a 100% match in order to be in the same cluster.

3.7 Graph Database

If the modeled system abstracts a set of interconnected nodes, it can be formally represented as a graph G, where $G = (E, V)$. The graph is defined by the set of edges E that interconnect individual vertices and V represents the set of vertices as individual nodes of the graph [52]. This formal representation allows the storage of data into a graph database operating over the designed graph topology. This type of topology can help to efficiently store the interconnected data and decrease the time and resources required to query the related data. Graph databases are a subcategory of NoSQL databases that follow the graph topology scheme [52]. Furthermore, both the vertices and edges can be represented by their weighted values as degrees and edge weights, respectively.

Artifacts and properties of the analyzed samples can be represented by a corresponding graph topology. Because all data are stored as partial nodes of the graph database and the relationships between objects as edges, the topology provides additional information about the activity of each node. The node represents a particular property or artifact of the sample that is stored in the database. Two nodes are connected if their values were observed during a single analysis or an event. For example, a relationship between the reported sample from the clients can be represented by a weighted edge between the unique user identificator (GUID) and the hash of the given sample. The weight of such an edge represents the number of reports for the given sample by the specific user [30].

Example of such graph is shown in Figure 3.3. Nodes `A` and `B` represent unique users. Nodes `C` and `D` are samples reported by these users. The degree of node `D` is equal to three, because it has a connection with three other nodes (two input and a single output edge). However, the weighted value of input connections is fourteen, because user `8abc2c` reported six incidents involving the sample `0DFF12A` and user `12bf5a` eight incidents.

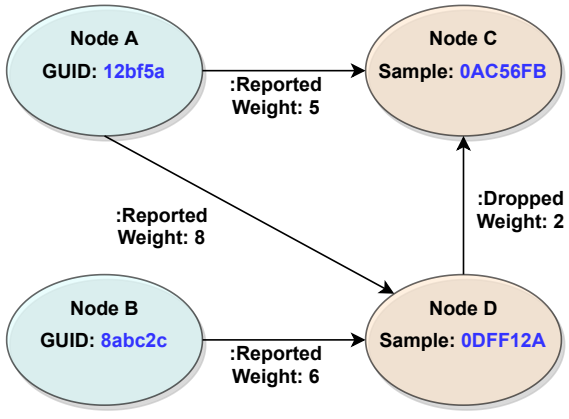

Figure 3.3: Example of the graph stored in graph database

Relationships between various artifacts and samples are stored in an internal graph database of Avast Software continuously updated by an internal service that provides an

interface for obtaining information about connections between the properties and samples [30]. The internal Avast service provides an interface to query the database via an API or directly by the composition of database queries.

Cypher Query Language

Cypher is a declarative query language the allows to query graph databases designed by a company Neo4j [53]. It allows to expand the individual stored nodes, obtain any non-trivial relation or find connections between the related nodes. This can be used to obtain information about the users that have reported an incident related to the given sample, obtain the list of samples that have accessed a certain domain, etc. Example of a Cypher query is shown in Figure 3.4. A keyword `MATCH` represents a searching statement for an existing node, label or a relation. `RETURN` specifies the properties that should be obtained and returned by the query. The shown query can be used to obtain up to ten domains that have been accessed by the specified sample represented by the hash `0AC56FB`.

```
MATCH (:Sample name: '0AC56FB')
-[:ACCESSED]->(domain:DOMAIN)
RETURN domain LIMIT 10
```

Figure 3.4: Example of Cypher selection query

Chapter 4

Message Platform Apache Kafka

Data transfers and information sharing between independent services can be realized by multiple approaches, such as API endpoints. Each approach has its advantages and disadvantages for specific situations and design patterns. In this chapter, the architecture of a streaming platform Apache Kafka is discussed. Kafka serves as a backbone of the internal communication between various services, reputation and detection systems that are described in the previous Chapter 3.

4.1 Architecture of the Message Platform

This section describes the basic architecture of the Apache Kafka streaming platform, its principles and the advantages of its usage. Kafka is an open-sourced project[1] providing high-performance data pipelines and streaming analytics initially developed at LinkedIn [40]. The main idea of a message streaming platform is to replace a high amount of databases that serve to store data with an architecture that is able to analyze and process continuous data *flow* that allows easier distribution of the data among the flow-processing *streaming* application while keeping its scaling capabilities [40].

Advantages of Using Message Streaming Platform

The rise of micro-services has led to new ways on how to transmit messages among interconnected applications that have dependencies among each other. Usage of the old APIs can lead to additional demands on the service that is less scalable than a separate platform to distribute messages [40]. The second problem is the durability of each individual message. When processing of certain messages is more resource-demanding and the processing application is under higher load for a certain time, message durability provided by Kafka allows the application to reduce the difference of processes messages once the resources are freed. This platform is also designed to be highly performant, highly available and redundant, supporting a high throughput of messages [16].

Transmission System

The communication is established mainly between producers and consumers that share the same medium provided by Kafka. Kafka implements a system for transmission via topics [16]. Topic represents a queue of messages that allows producers authorized to the

[1] https://kafka.apache.org/

platform to publish messages into the queue, which is identified by its unique name. Each Kafka consumer has an option to subscribe to the given queue and start consuming messages that were published into it. Each published message is referred to as a record and each record contains meta-data and key-value pairs [40]. Each topic is stored in the form of a log structure file. To store the messages and serve all requests created by subscribed consumers, each topic is located at a server called a broker. Each broker must receive a message from the producers, assign them corresponding offsets and commit the messages to storage on the disk [40]. Kafka brokers provide retention capabilities to keep the messages for a certain period of time before they are deleted from the queue. To keep the possible downtime as low as possible, Kafka supports data replication of topics into multiple brokers, where the broker contains cloned messages from the topics. If one of the brokers stops responding, requests are forwarded to the other brokers. This requires a central coordinating system that is called Zookeeper Cluster, which coordinates all brokers in the cluster and allows them to exchange information [16]. This is realized by storing meta-data for each broker. A general overview of the architecture from the point of client applications can be seen in Figure 4.1. Kafka can be connected to external data stores (connectors). Producers and consumers interact with the Kafka cluster and commit their progress after atomic operations. Streaming applications leave event processing to the Kafka Stream API, which produces the results as messages to other topics.

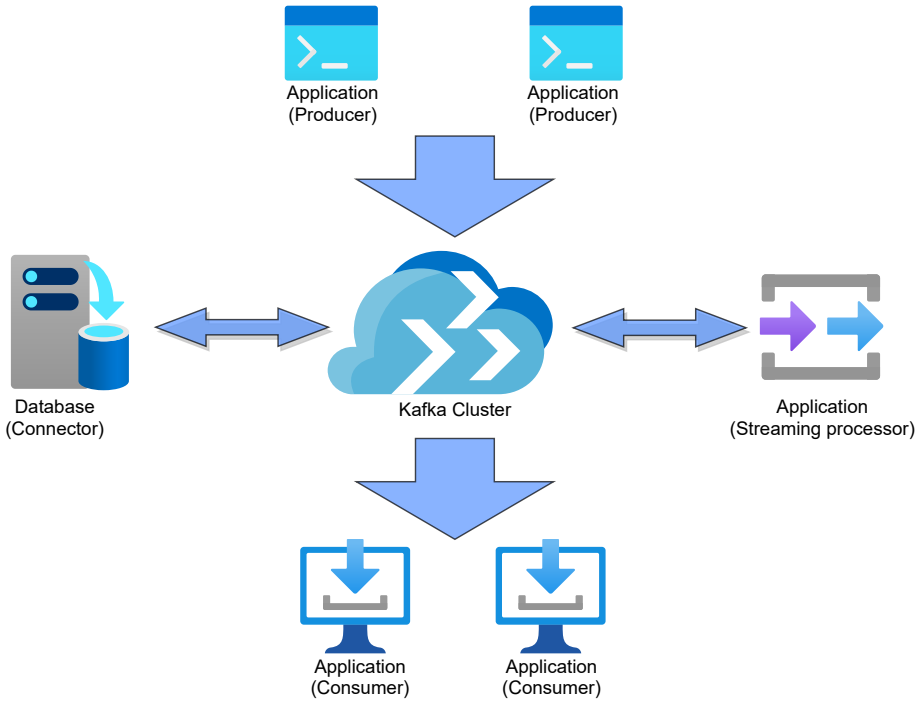

Figure 4.1: An overview of Kafka architecture

To ensure additional data replication and scalability, each topic may consist of several partitions into which the topic is broken down. Partitions represent a subset of all messages that were produced and stored in the given topic. Each message will be assigned a partition, part of which the message will become. Kafka ensures that all messages in one partition keep the causality by storing the original order in which the messages were received [40].

Each partition can be replicated among brokers based on its replication factor. If the replication factor is lower than the number of brokers, each broker contains only a subset of all partitions [16].

4.2 Protocol Buffers

To increase the efficiency of the streaming application as well as to provide a certain level of validation, produced messages can have a strictly typed structure based on the description by the overlaying language. Examples of the technologies for this purpose are protocol buffers (protobufs) designed by Google [2]. The motivation behind the protobufs is to describe the structure of binary data and easily generate parser in any of the supported programming languages, which provides rigorous integration with the production systems [2]. Among the supported languages are C++, Python, Java or Go. This system of data serialization is also platform-neutral [2]. For generating valid parsers in the targeted language, a protobuffer compiler named *protoc* is provided. A disadvantage of using automatically generated parsers is a disarranged code style that might not meet the common criteria set for the source code of a project.

Messages can have multiple attributes and each attribute has the corresponding data type. Protocol buffers also support lists in the form of repeated attributes and optional fields that might not be present in the message [33]. Each field has a unique number used to identify the field in the binary format. Messages support nested design, such that any message can have a field defined by another message type. The syntax also supports enumerators as a pre-defined list of values [2].

If the current message format needs to be updated, protocol buffers provide multiple guidelines on how the format change can be achieved. For example, to ensure compatibility with the older type of messages, new fields should be set as optional or repeated. Many field data types are compatible with respect to each other. Example of a protocol buffer used to specify a format of the message is shown in Figure 4.2.

```
message Search {
  required string query_text = 1;
  optional int32 page_number = 2;
  optional string client_name = 3;
  enum SearchType {
    SIMPLE = 0;
    EXTENSIVE = 1;
    OVERALL = 2;
  }
  SearchType search_type = 4 [default = SIMPLE];
}
```

Figure 4.2: Example of a protocol buffer message description

4.3 Architecture of Threat-Data System Communication

Because the number of analyzed samples is huge, all advantages described in Section 4.1 are leveraged to ensure perpetual protection of the users and continuous sample analysis by internal systems that are deployed either in the cloud or internal servers. Each threat-data analysis, monitoring and reputation system described in Chapter 3 produces messages that are handled by Kafka and received at the back-end servers of the antivirus company. Each message contains meta-data like geo-location, time and a record containing data produced by the given system. Each system also uses one or more topics for specific types of messages, e.g., reporting an incident or a newly discovered sample with zero prevalence. To lower the overhead caused by the number of produced messages, related messages are aggregated by the gateway servers or directly at client applications and sent as just one common message. When an incident occurs, there is a high chance that multiple messages will be produced by the client because the antivirus company tries to collect much possible information required to classify and evaluate the threat and for threat-intelligence purposes respecting the privacy of users. There are exceptions, such as requests for the reputation systems that represent short messages with a few fields and are not aggregated as the other types of messages [46].

Chapter 5

Cadet - A Real-Time Campaign Detector

This chapter presents a design of the real-time campaign detection service CaDet (**Ca**mpaign **Det**ector). It is designed as a web service that is able to analyze and visualize the processed threat data. The chapter provides more information on the input data and consumed messages that are used for campaign detection, an internal data flow and rule-based decision making for campaign detection. Furthermore, as a part of the system design, a way of campaign-detection validation was proposed and described in this chapter. The chapter concludes with a description of service monitoring and alerting required for validation of the correct deployment and data processing.

5.1 Input Data and Messages

The main source of input information required for real-time detection of malware campaigns are data feeds from threat-data services described in Chapter 3. Every threat-data reputation, analytical or detection system deployed on the end-user system sends the incident or detection-related reports back to the company for closer analysis by publishing its messages resulting in continual Kafka feed. The real-time detection service is designed to consume messages from eight unique topics. Each topic consists of multiple partitions. The average number of partitions for the supported topics is above fifty. Each topic represents a specific area of the monitored domain described below [32]:

- **Main feed of file reputation system** – This feed consists of common client requests and system responses to determine the prevalence of the analyzed sample. Responses might also contain a low-priority, quickly determined and abstract classification from a supporting classifier.
- **Additional feed of the file reputation system** – Messages regarding unknown or low-prevalent samples are separated from the common feed because they are also analyzed by additional services due to reasons described in Chapter 4. The structure of the messages is similar to the ones from the main feed.
- **Anti-rootkit feed reports** – All incidents detected thanks to the anti-rootkit module of the client scanner are published to this topic. It also contains meta-data about the drivers and environment of the system at the time of incident occurrence.

- **General incident reports** – Messages published into this topic represent aggregated data from various modules and a general model of the incident that might occur on the monitored station, e.g., protected PC of the user. Some of these messages are duplicated reports from other systems that can be consumed from their specific topics. However, this feed might provide additional or reduced information that differs from the mentioned alternatives.

- **Reports of networking-related incidents** – Some incidents are not tied to a specific observed sample, e.g., user action executed by a mouse click, or a networking shield monitoring the ingress and egress network traffic. Incidents like these are reported into a separate topic.

- **Domain reputation system feed** – Similar to the file reputation system, a domain is also evaluated and rated based on its basic properties described in the previous chapters. Requests to obtain a reputation of the domain are handled by this feed.

- **DNS reputation system feed** – When a domain needs to be translated into an IP address, the generated DNS request (A, AAAA, etc.) is captured and reported for analysis purposes. If the translated domain is malicious, the connection is terminated to protect the client.

- **Android (mobile) incident reports** – The last topic stores messages from mobile clients as well as their related reputation statistics, detections and incident meta-data.

The above-mentioned topics store messages from all reported incidents. However, it is necessary to filter input data that are invalid. The designed system must be able to detect and discard occasional corrupted messages or messages with an invalid format. Each message has a strictly typed structure described by protocol buffers which provide an interface that allows to use them for parsing the consumed messages into language-specific structures or detect an unsupported or corrupted message. The overall rate of message production is very high, so the system must allow assigning multiple consumers for each subscribed topic. This leads to faster message processing and lower delay between the time of message report and campaign detection. Each consumer has an assigned unique thread of the application process, so the application data should be thread-safe when possible and the service needs to avoid common conflicts caused by parallelism, e.g., race conditions when multiple consumers try to access data of the same campaign.

The parsed structures of protocol buffers contain all data from each message, many of which are not relevant nor important for the detection of malware campaigns. For this reason, the structures need to be transformed in an internal representation that contains only the relevant data. Such objects must allow data serialization and can be stored into an internal database to reduce the amount of stored data and increase the processing speed.

Incident reports need to be further filtered by multiple metrics. The first metric is an identifier of the client installation. Each product installation has a unique identifier to distinguish it from other clients and products. A client might also report multiple similar incidents in a row, e.g., repeated detection of the same sample detected by a scanner. This might cause flaws in counts of detections and such incidents should be processed as one. The second filtration is necessary due to replicated incident reports across multiple topics. For example, when an isolated incident occurs, the anti-rootkit system reports the incident, but since no additional data were collected, the same incident can be reported as a general

incident stored in a separate topic as well. Incidents like these should be detected and aggregated by the designed system.

Detection Parameters

All consumed events provide many different data that can be used to compile campaign detection rules. Therefore the selection of the relevant data is required. The following attributes are selected based on the value they provide as the primary source for campaign detection information and are used during campaign detection experimentation described in Chapter 7 [30, 45, 46]:

- **Client detections** – Client detections are described in Section 3.5. Detections can be obtained separately for the top layer (the analyzed sample by the scanner) and subsequent detections for the files generated or dropped by the analyzed sample, its unpacks, etc. Client detections are used not only as a source of embedded classifications using a detection parser that can parse their format but also as a source of campaign detection rules.
- **File paths** – This category includes file paths and file names that are recognized as individual fileless campaign categories. The obtained paths underwent a normalization process to unify their format.
- **Domains** – This category includes domains and URLs, which are also recognized as individual fileless campaign categories. They are also presented as a property of sample campaigns, e.g., when the given sample accessed the domain.
- **Classification** – Classification is obtained by parsing client detections, using embedded classifications present in the parsed events or by requesting the classification from Tagger that serves as the main tagging service. The presence of embedded classification and the individual layers of the classification tag can be used as parameters for experimentation.
- **Application protocol types** – The application protocol represents an additional layer bound to the detected event. Some types of messages are protocol-specific, such as IRC communication mentioned in Chapter 2. This value can be used to improve the detection of certain types of campaigns.
- **Certificate** – The Authenticode signature follows the X.509 format of certificates [19]. The certificate contains, among other properties, the subject of the certificate, the certification authority, etc. It is also possible to obtain information about the current status of the certificate. The state of the certificate may be valid, expired, revoked, damaged, etc. These parameters can be used for experimentation.
- **Sandbox prevalence** – This value represents the prevalence of the sample in the internal sandbox environment, which is deployed directly as a part of an antivirus product. More suspicious or new unknown samples can be monitored for a limited time in an isolated environment directly on the protected system and the sandbox report is registered. The files can also be analyzed by the sandbox at the user's request.

- **Shield triggers** – Antivirus products consist of many different scanners and protection mechanisms called *shields*. This property provides additional information about the service that registered the incident. It can be used to identify the source of detections.
- **Avast triggers** – Avast triggers provide more detailed information about the incident itself. The trigger is usually a specific monitored event, such as the execution of a binary file.
- **Smart Home data** – Additional data about the source in the case of smart homes, such as device OS, were used for experimentation focused on detections of IoT and Linux campaigns.
- **General activity** of the graph database – It represents a degree activity of individual nodes and edges. The experimentation was focused on the client submission nodes. It provides activity data for multiple time slices, which can be aggregated into the hour, daily and monthly statistics. The activity was described in Chapter 3.
- **Weighted activity** of the graph database – Opposed to the general activity described above, this type of activity represents weighted relations between the queries nodes, such as samples and properties.
- **Cache prevalence** – Consumed events are stored in the database for a limited time. The event database is also referred to as the cache window. The experimentation involves different time window durations for the stored events. The event cache is further described in Chapter 6.

5.2 Classification and Campaign Detection

The service CaDet is designed to support all campaign types described in Chapter 2. This list includes sample campaigns related to a certain file represented by its SHA-256 hash, malware family campaigns providing information about the activities of certain malware families and artifact campaigns. The campaigns might contain information about additional samples which are tied either to a file or artifacts and properties, e.g., the visited domains or file-system paths obtained during static or dynamic analysis. To determine if the observed sample or property is malicious and to estimate its precise classification, the designed service uses a hierarchical classification approach involving multiple systems [32]:

1. **Tagger sample/detection classification** – This is the most precise classification because it represents a result of weighted comparison from multiple classification sources and internal systems. The best classification was picked based on the calculated confidence, which can be obtained as a part of the classification result. Each incident report contains a set of detection strings based on which the incident was detected. If the sample has no classification, at least a subset of detection strings reported with an incident might be classified and used as a temporary classification.
2. **Embedded classifications** – Some messages contain embedded data about classification results estimated by internal systems that are not focused on precise classification but at least the severity necessary for neutralizing an unknown threat. Classification like these are embedded into the reputation, incident or network feeds. Using

their data helps to decrease the request load created by the application towards the main tagging system.

3. **Indirect classification** – This type of classification is used in the case of domains. Currently, Tagger does not support a direct classification of domains. However, a list of samples that have accessed the given domain can be obtained from the internal graph database. Once the samples are classified by Tagger, a majority vote is be used to determine the classification of the given domain.

To get the final classification of objects that are not supported by Tagger like domains or specific artifacts, an internal graph database must be used. It follows the specification described in Chapter 3. Relationships between artifacts and samples are stored in an external graph database continuously updated by a third-party internal service which provides an interface to obtain information about connections between various properties and samples. Because all data are stored as partial nodes of the graph database and the relationships between objects as graph connections, it provides additional information about the activity of each node. Each node represents a particular property or artifact of the sample. The designed service estimates the final classification based on the classifications of samples sharing the same artifact, a list of which is obtained from the internal graph database. The activity of all nodes can be used for retrospective supportive inputs of sample prevalences in a specific time window. This helps to cover the cases when the service has no previous information about the sample or artifact.

Because Tagger provides information about classifications via REST API, fallback classifications are used when the classification was either not found or the rate of API requests was higher than the current architecture supports.

Once the classification is determined, the campaign-related information can be derived. The emergence of a new campaign can be detected by comparing the current re-occurring prevalence of the sample and its overall spread in the past (e.g. last month). Data like the name of the originating topic and its activity stored in the graph database can help to determine its prevalence. A reason based on which the campaign was detected should be stored and properly visualized together with campaign data. The classification is used to determine the priority of certain detected campaigns so that the service is more focused on malware campaigns rather than a campaign of clean samples. The presence of the sample in an already active campaign or the end of the spreading period and finalizing/archivation of the campaign is determined in a similar way. All related data obtained from the events, such as sample properties or classifications, are recalculated based on each consumed and relevant event. Figure 5.1 shows the overall abstract design of the service.

Multiple campaigns can be interconnected. Common connections binding two samples are unpacking or installation, e.g., a certain installer can be used to install a different malware, or the sample is dynamically unpacked to avoid the detections. Because the incident reports can contain detections of the unpacked samples as well as the parent files, similar connections (in case that both files are parts of the active campaigns) can be connected and visualized.

Properties of campaigns of each category need to be visualized appropriately. Because the service is designed as a web application, its dashboard is rendered in real-time. The dashboard visualizes a summary of the most prevalent campaigns with the highest priority. Because each campaign stores data about the countries in which the threat was detected, a geographical map is rendered to easily navigate in the rendered state of the latest detected

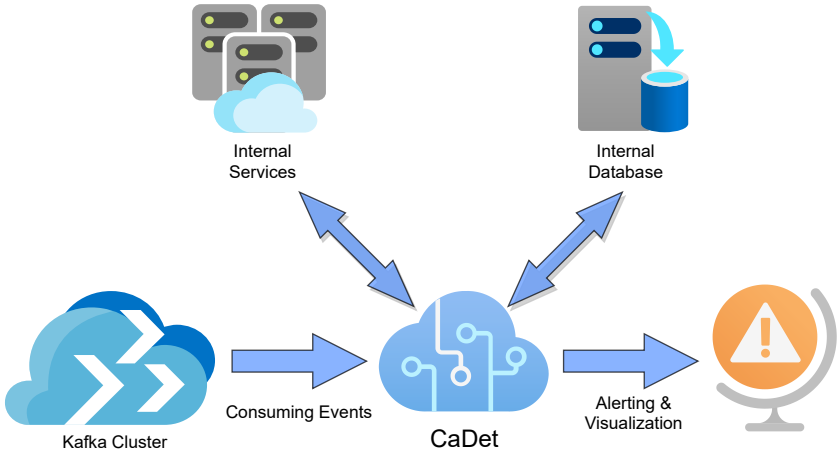

Figure 5.1: Abstract design of the detection service CaDet

threats. The designed service should display a unique map for each detected campaign to monitor the activity of the detected threat.

The service should also provide an API interface for communication with other services. The API provides information about detected campaigns based on their IDs. The user should be able to get data about a single campaign or a list of results based on the key attributes, such as the hash of the sample or family name. The data returned must be identical to the data visualized in the web interface.

Campaign Score and Detection Rules

Detection rules are the main proposed solution for the detection of active malware campaigns. The rules represent either a comparison of one of the attributes described in the previous section with a specific value or a combination thereof. Detection rules are an abstraction over the different detection conditions of malware campaigns and help to present the basic explanation for the detection of each campaign. The rules were initially only the individual conditions that operated with the values of parameters described in Section 5.1. Based on the experimentation described in Chapter 7, their representation was adjusted to use the following attributes:

1. Weighted activity of the graph database
2. Cache prevalence
3. Campaign score

The weighted activity and cache prevalence were described in the previous section. *Campaign score* is introduced as a new numerical metric, which is calculated on the basis of partial embedded feed classifications and individual properties of the consumed event. Its values range in the interval $\langle 0, 1 \rangle$. It effectively detects related events leading to campaign detection. The more important an event is in terms of campaign detection, the higher its value. This score also affects the probability of obtaining additional data from external sources about the sample due to the stochastic approach. The score is influenced by many parameters and all parameters described in the previous section are covered by the

experimentation. All properties that affect the malware score in the final implementation are described in Section 7.2.2. Example of the campaign detection rule based on different conditions is shown in Figure 5.2.

```
Detect campaign if:
      # Newly detected samples, low prevalent ones
      -- Weighted graph activity during the last day
         has achieved the score 550
      and
      -- the last month average is 0.
      # Significant activity changes, short term campaigns
      -- General activity of the last hour
         has achieved the score 500
      and
      -- the activity during the last day is 0.
      otherwise use general rule:
      -- Hourly activity must be at least 8x higher than the last day.
```

Figure 5.2: Example of a simple campaign detection rule based on activity

5.3 Validation of Results

The results provided by the application need to be validated and if the current rules for campaign detection are not sufficient, the service needs an external notification about the missed campaigns. For this purpose, data from social media and blog posts will be used for detection of missed malware campaigns as well as the detection of relevant campaigns in case the rules need to be updated or to be more strict for campaign detection. Social media are a common source of data interchange between security researchers and companies based on the detected IoCs (indicators of compromise) or the increased activity of certain malware families or samples. Such tweets are spread using specific hashtags or the messages have a common structure that is easily parsed by an automated service. There exist multiple services that aggregate such social media messages and provide an API to easily access the aggregate statistics as well as current trends of malware families and campaigns for threat-intelligence purposes. Data from such services can be used for automated validation of the detected campaigns. The other source is an internal communication channel of the company that exchanges information about the newly detected threats. A more detailed description of the used sources is provided in Section 7.1.2.

5.4 Monitoring and Alerting

The activity of the proposed detection service should be monitored by collecting information about various metrics. Example of the collected metric is the number of processed events or requests related to the external services. The number of processed events needs to be monitored due to possible failure of data source or consumer. The number of requests for external services, such as the internal graph database or Tagger, should be monitored, as

the services may have a temporary outage and they need to be disabled in the configuration file. This metric also helps to identify the service load generated by CaDet. Monitoring should be performed by one of the available platforms focused on monitoring such data. This approach does not require additional processing resources of the service itself. The visualization is also left to the selected platform.

An alert is needed to inform cybersecurity researchers about the current situation and the emergence of a new malware campaign. There are many other reporting systems that need to be monitored and alerts simplify the interaction of users with the service. For this purpose, one of the company's internal communication channels should be selected and integrated into the service. This step should be consulted with the potential users in order to find the most suitable communication channel for alerting. Each alert should include a hash of the sample, family name or value of the artifact associated with the detected campaign. It should also contain the reason for detecting the campaign and the name of the source, as analysts can decide based on these values whether the alert requires their attention. The alert must include a link to the CaDet web interface to show details about the detected campaign.

Chapter 6

Implementation

This chapter provides information about the implementation of the real-time malware campaign detection service CaDet designed in the previous Chapter 5. It discusses the used technologies and provides detailed information about the process of event consumption. The web interface of the service and its public application interface are also presented. The implemented service is used for experimentation described in Chapter 7.

6.1 Used Technologies

This section discusses the technologies used for the implementation. It also presents the reasons for choosing selected technologies. The technologies used are divided into several categories based on their main functionality.

Web Technologies

The web service was implemented using the programming language Python 3. It simplifies the rapid prototyping process and its packages provide access to many web services for the presentation of detection results, the database technologies interface and the Kafka infrastructure. Python is a language often used in Avast Software, which simplifies the maintenance of the service [46]. As an underlying web framework, a micro-framework Flask was chosen. Flask provides more flexibility than a high-level framework Django. The framework also provides methods for API implementation and JSON processing. CaDet is not a service focused on a large throughput of API requests, so the usage of request-optimized FastAPI is not necessary. Flask also provides a way of running the web application using the default deployment server that is a part of the framework. However, the Flask default development server is not suitable for production deployment and the provided guidelines recommend the usage of a different tool. Since the application will be deployed in a UNIX environment, Gunicorn was chosen as the used UNIX-compatible WSGI HTTP Server [22]. WSGI (Web server gateway interface) is a specification describing how a web server communicates with web applications. Gunicorn supports multiple web frameworks, including Flask. It supports an instantiation of multiple workers responsible for handling requests and decides how to communicate the incoming requests to the framework process. This approach supports the scalability of the service regarding API and the web interface.

The basic structure of the web interface was designed using HTML and its style using CSS and Bootstrap. A templating language for Python called Jinja2 was used for the creation of dynamic web pages. Flask provides support for an extension Jinja2, which al-

lows the insertion of pseudo-Python commands directly into a predefined HTML template. Based on the arguments passed to generating function handled by Flask, template commands can be executed and the final HTML page will be dynamically generated [22]. This is useful for generating HTML documents with dynamic content, for example, visualization of campaigns. Jinja2 also supports defining macros that serve as methods for generating HTML content and help to keep the DRY principle[1] and high code readability.

Virtualization Technologies

Because the application was designed such that it is easily containerized, Docker was chosen as the base virtualization platform. The main commands for assembling a container image are written in the configuration file called *Dockerfile*. The containerization ensures that the application execution is not dependent on the underlying hardware of the execution environment, which generalizes the options for deployment on different servers [38]. Each image is built from the base image `Python-3.8`. The implemented service is designed as a multi-container application. This means that the web instance, database, consumers and other modules are executed in separate containers. A tool *Docker compose* supports this approach and is used to define and deploy the application. Individual containers can be linked based on their dependencies. The linkage is used for the interconnection of the database with other containers (consumers for event insertion, the web application for obtaining data about campaigns, etc.). Each predefined container also supports an option of an automatic restart in the case of an unexpected failure, which promotes a high availability of the application in case of an error.

Database Technologies

For a suitable database technology, a general-purpose document-based NoSQL database MongoDB was chosen. It splits the database into multiple collections that contain the stored entries called *documents* [14]. Documents may not have a predefined scheme like SQL database and provide a more flexible approach towards storing variable data. MongoDB also supports indexing for high-demand documents, which was used to decrease the time necessary to find documents based on the most used properties [22]. Another type of index supported by MongoDB is the Time-to-live index (TTL). The TTL index is tied to a specific date-time property. Once the predefined interval expires, the document will be automatically removed from the collection. This is useful for defining time windows/ranges and is used in the case of event cache described in the previous chapter. It also supports scaling options via sharding and data replication that might be used to further increase the stability of the implemented application. MongoDB and Apache Cassandra were compared in order to choose the best suitable database technology during the initial experimentation and implementation of the service. Apache Cassandra is also a NoSQL database used in real-time processing applications [13]. It is a highly scalable database promising high performance. It requires a column-oriented schema of the stored records decreased the flexibility provided by MongoDB. The more flexible model used by MongoDB has proven to be more suitable for the implemented service and the usage of Cassandra did not result in a significant time reduction of database operations. For these reasons, MongoDB was chosen because it is already used by other internal systems of the company, which simplifies the future monitoring and deployment of the service by responsible people.

[1] Abbreviation of „Don't repeat yourself".

Event Processing Technologies

Confluent Kafka for Python[2] was chosen as the underlying interface for communication with Kafka architecture. This interface is reliable, performant and Confluent Kafka was founded by the creators of the original Apache Kafka. It provides a thread-safe implementation of consumers and producers, which meets the demands of the designed service described in Section 6.2.

6.2 Event Consumption

Confluent Kafka mentioned in the previous section is a framework that supports simple prototyping of Kafka consumers and producers. In the comparison study of multiple Kafka interfaces and popular Python libraries, such as Pykafka and RdKafka, it was proven that it provides the highest throughput of messages and has the least message-processing overhead [5]. This makes it suitable for event consumption from multiple sources, such as incident report feed, and gives the application more time to process individual events.

However, the interface is simplistic and does not provide some of the methods of other concurrent libraries that make it easier to manage individual consumers, monitor the offset of the consumed messages, etc. These features were implemented manually using a hierarchical structure of consumers in the module `event_consumers`. The class `KafkaConsumer` represents an instance of a consumer that is able to subscribe to a single topic using the credentials and data from the configuration file and forward the consumed messages into an event handling module. This type of consumer can be run as a non-blocking operation in a separate daemon threat of the application. The class `KafkaConsumerGroup` implements the construction and management of a group of dependent consumers that are related to a single specified topic. It provides an interface to set a number of requested concurrent consumers that subscribe to the given topic and executes them in their assigned threads. The class `CaDetEventConsumer` is the highest abstraction of the hierarchical consumer tree and defines the consumer groups for individual topics consumed by the service and manages those groups. A provided setup method instantiates the main consumer based on the selected event type. This is used in a script `kafka-events-consumer.py`, which allows consuming events from a single topic based on the provided argument.

Each consumer runs in a separate docker container defined by the Docker compose tool. This allows the operator to easily manage individual feeds, view logs related to individual consumed feeds, restart a failing consumer without affecting other consumers, or view the updates of each consumer. Figure 6.1 demonstrates the architecture of the consumers and the relations between different containers.

6.3 API

Because the data representing detected campaigns might be used by other automated systems or scripts as well, CaDet provides a public application interface for other services that allows them to easily access the processed data, such as campaign information or malware family trends. The implemented API tries to follow principles defined by RESTful API (Representational state transfer) guidelines [47]. The communication is based on server-client architecture. It is stateless, which means that each API request is handled inde-

[2] https://www.confluent.io/

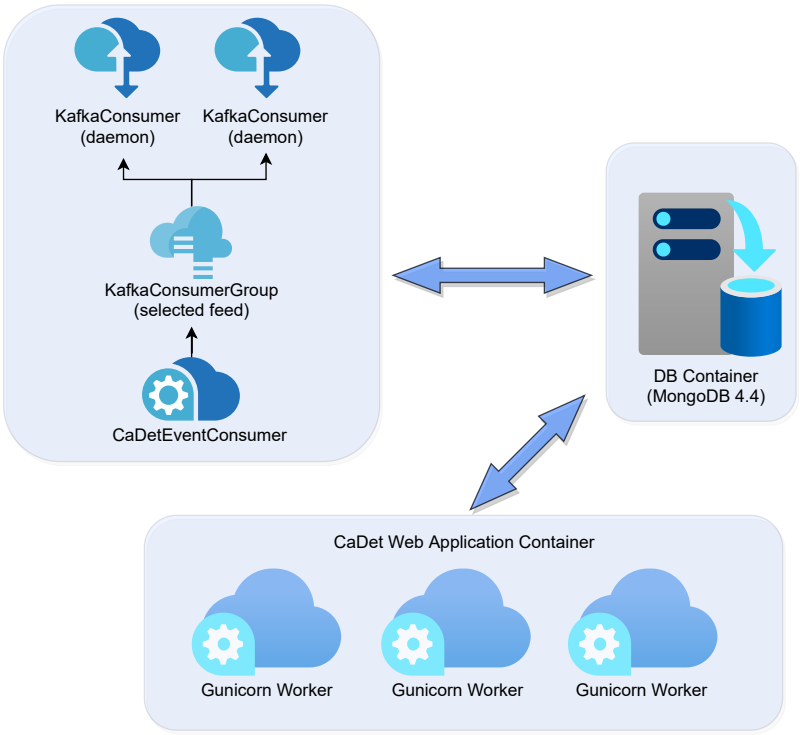

Figure 6.1: Container relationships and consumer tree

pendently because there is no relation between different requests and users. All requested resources are identifiable. Response caching is automatically handled by the underlying frameworks and web server.

The implemented system provides multiple endpoints, responses of which contain data serialized using JavaScript Object Notation (JSON). Because the underlying database stores data as binary JSON, which allows it to support data structures like date, such objects need a valid JSON representation. For the dates, a format based on ISO 8601[3] was chosen. ISO 8601 is an international standard describing the exchange of date and time-related data. All implemented endpoints are described below.

- `/api` – An overview and simple description of all API endpoints (JSON).
- `/api_human` – A human-readable visualized overview and a simple description of all API endpoints generated using the Swagger UI interface.
- `/api/sample_campaigns/id/<campaign_id>` – Get information about sample campaign identified by the given ID.
- `/api/sample_campaigns/sha256/<sha256>` – Get information about all sample campaigns for the given SHA-256.
- `/api/family_campaigns/id/<campaign_id>` – Get information about malware family campaign identified by the given ID.

[3] https://www.iso.org/obp/ui#iso:std:iso:8601:-1:ed-1:v1:en

- `/api/family_campaigns/name/<family_name>` – Get information about all campaigns for the given malware family.
- `/api/fileless_campaigns/<property_name>/id/<campaign_id>` – Get information about fileless campaign identified by the given ID and its type.
- `/api/fileless_campaigns/<property_name>/value/<property_value>` – Get information about all campaigns of the given artifact or property based on its type.

CaDet provides two ways to describe the API interface - a machine-parsable JSON for other systems that may detect accidental changes or a new version of the API. The human-readable form is automatically generated using a Flask extension Swagger UI[4], which allows the application to generate the description based on the predefined JSON file. The human-readable interface also recognizes various parameters that are required and passed to the endpoint so that the user can directly test the interface using a generated form near the endpoint description. Swagger UI automatically visualizes and highlights the obtained JSON results. Example of generated documentation for family campaigns is shown in Figure 6.2.

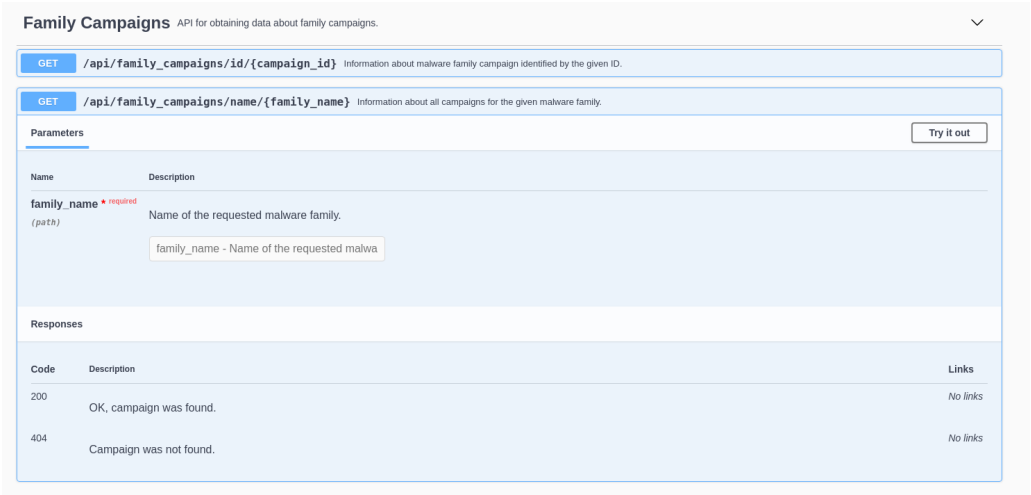

Figure 6.2: API documentation generated using Swagger UI

6.4 Dashboard

Dashboard of the service summarizes and visualizes important data that the service provides. One of the visualized elements is a world map that is shown in Figure 6.3. It represents the current state of malware campaigns related to specific countries currently detected by the service. The map shows numbers of the detected campaigns that are still active and were observed in the given state. The maximum number of the detected active campaigns in a single state is used to calculate the density of the color of any given country by calculation the ratio between the maximum and observed number in the given country. The more dense the country color is, the higher amount of the detected campaigns was

[4] https://swagger.io/tools/swagger-ui/

found in the country. Initially, popup bubbles of different diameters were displayed over the countries to visualize this metric. However, this feature was replaced by the color density based on user feedback. The bubbles make it difficult to focus on certain smaller countries. This approach makes it easy for users to see which countries are currently the most targeted by malware campaigns. The world map is rendered using Datamaps framework for Javascript[5].

Below the world map, there is a sorted list of currently active and the most prevalent family and sample campaigns. It visualizes the basic statistics, such as their classification. Furthermore, a number of samples belonging to the corresponding campaigns is shown as an attribute and was used for sorting the results. Hash of the sample, name of the malware family or ID of the campaign can be used to easily navigate and access the more descriptive campaign data via hyperlinks that redirect the users to lists of campaigns represented by the given properties.

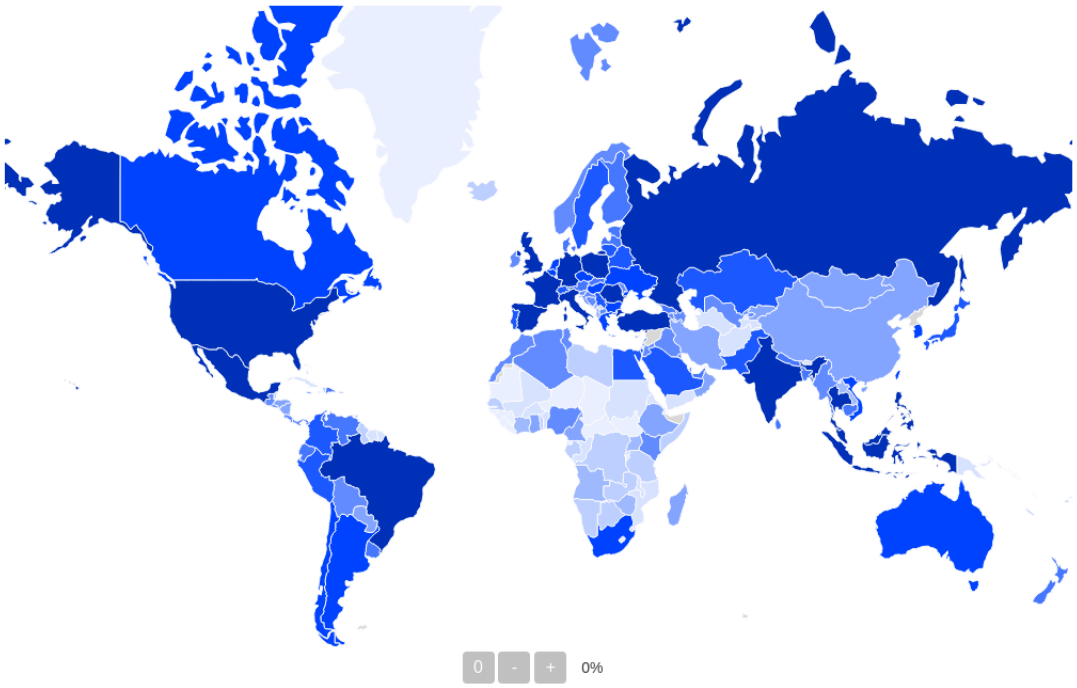

Figure 6.3: A map visualizing ratios of detected campaigns per country

Interactive World Map

The Datamaps framework does not currently support direct interaction options. The predefined event handlers of the map have to be re-implemented using Javascript in order to achieve this functionality. This allows users to interact with the map and use the countries as clickable references that redirect them to detailed statistics about the given country. The re-implementation has made it possible to implement a zooming feature controlled by the mouse scroll wheel. The default map constructor was replaced to use the implemented handling function `_handleMapReady` once the map is generated. This function replaces all

[5] https://datamaps.github.io/

`mouse-over` event handlers of the countries. It also establishes their references to country details and initializes the custom zoom handler. The custom zoom handler disables mouse-click events when the mouse is dragged. This allows the user to easily navigate and translate the zoomed map without unexpected redirections. The predefined zoom scales are calculated based on Equation 6.1 for ten various levels of scaling. The scaling definitions were inspired by the various framework methods that use the positional operations and the generated maps.

$$scale_i = e^{0.1 \cdot i \cdot \log(max\_scale)} \tag{6.1}$$

The new scale is recalculated when the user has scrolled with the button or clicked on the zoom control buttons. The same applies to the translation of the map when a mouse is dragged. The animation is ensured by consequent interpolation between the recalculated values in the set time interval.

The set of the shown statistics about the given countries is determined by the current active campaigns. The most active sample, family, domain and file path campaigns are shown in separate tables sorted by the sample count to prioritize larger campaigns confirmed by many events. The selected country is highlighted on the map to represent the current selection. Example of the interface for the detected campaigns is shown in Figure 6.4.

Family campaigns with the highest observed sample count in Brazil (Top 20):

#	Family name	Campaign ID	Initial reason	Sample count ↓	Status
1	miscx	608dbd998f4920cf091e2c2d	High prevalence in the event cache (122).	1573247	Active
2	segurazo	608dbe824cf69dd763edb14d	High prevalence in the event cache (121).	1052334	Active
3	ramnit	6093d9cc545113e3c4d50512	High prevalence in the event cache (232).	701690	Active

Sample campaigns with the highest observed count in Brazil (Top 20):

#	Sample	Campaign ID	Classification	Initial reason	Count ↓	Status
1	1e16a01ef44e4c56e87abfbe03b2989b0391b172c3ec162783ad640be65ab961	608eb0b72478176e0f64be28	clean	High prevalence in the event cache (151).	328188	Active
2	cb9902c9d4745389d2246821e4d4179b2fe2a9cf8277c891d9c213b3368295ef	608dbfe7eb35ecbb7412300f		High prevalence in the event cache (91).	323397	Active
3	0ed4f3cb3ab2613773364b784abe81421066df9ef60d780e01b3e4b39fe471f9	608ec7a31367cabf6ee367f2	clean	High prevalence in the event cache (127).	177312	Active

Domain campaigns with the highest observed sample count in Brazil (Top 20):

#	Value	Campaign ID	Initial reason	Sample count ↓	Status
1	datarouter.ol.epicgames.com	608ec870896a411bd9d09700	High prevalence in the event cache (61).	411587	Active
2	sb.scorecardresearch.com	608eaff86434590acdb8152b	High prevalence in the event cache (81).	370790	Active

Filepath campaigns with the highest observed sample count in Brazil (Top 20):

#	Value	Campaign ID	Initial reason	Sample count ↓	Status
1	C:\Users\User\AppData\Roaming\uTorrent\uTorrent.exe	608ed1021367cabf6ee3c743	High prevalence in the event cache (101).	332420	Active
2	C:\ProgramData\Milvus\Helpdesk\bin\OpenHardwareMonitorLib.sys	609515628c240b7975fed766	High prevalence in the event cache (101).	127736	Active

Figure 6.4: Visualized statistics about the given country

6.5 Campaign Data

The user interface of individual campaigns is represented by collapsible Bootstrap panels[6] for each campaign. Along with the corresponding campaign name (malware family, SHA-256 hash or an artifact value), representative badges with the most important data, such as classification, state of the campaigns or the related threat-data sources, are displayed. Example of provided data about the detected campaign can be seen in Figure 6.5. The shown campaign of the sample with the hash prefix `36a1ddc2b87` was detected based on the reputation and domain feed. It was initially detected on May 1st, 2021 and archived on the same day. The list of visualized properties is tied to the type of campaign and its attributes. Each campaign bears an initial reason for its detection, e.g., a sudden increase in the observed prevalence of the observed family. Campaigns also provide Alpha-3 codes of the countries[7], in which the campaign-related samples were observed. Each campaign can be connected to other campaigns based on mutual relations. For example, both unpacked sample and its parent are detected with their corresponding campaigns and the user can navigate to the related connected campaigns.

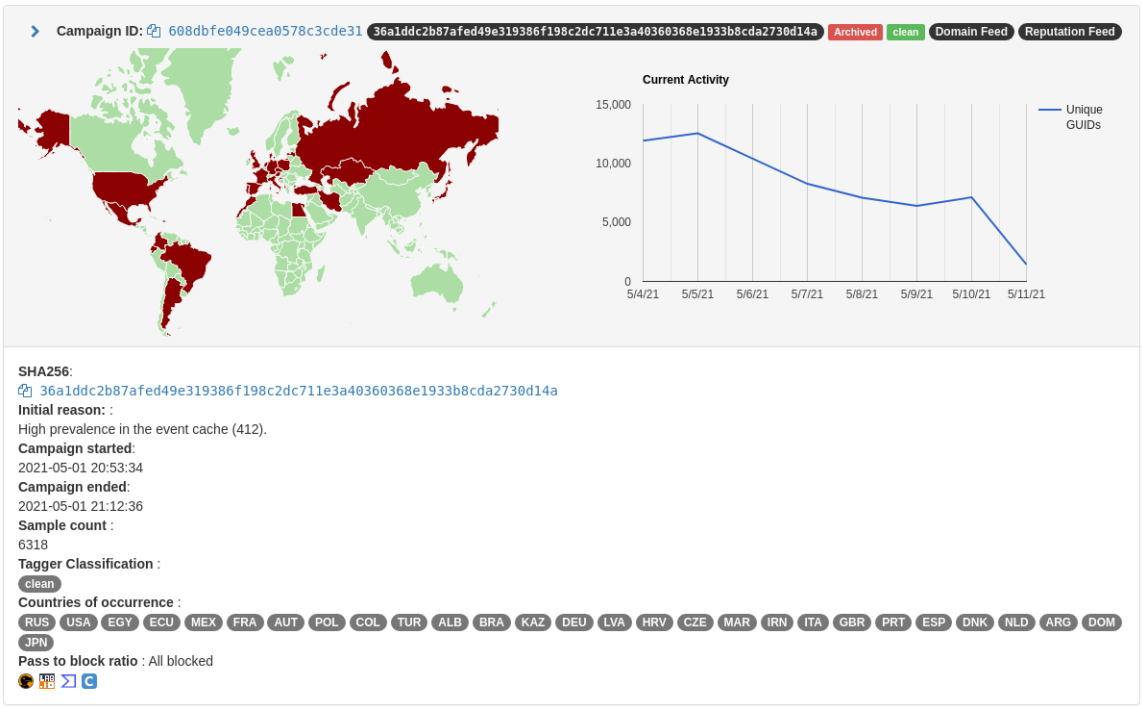

Figure 6.5: Example of the detected sample campaign

Family campaigns, as opposed to sample campaigns related to one specific sample, show a list of hashes of the samples belonging to the given campaign. This list is currently limited up to a thousand unique samples. To visualize the main differences of the samples, an internal clustering system Clusty is used to obtain information about the cluster into which the individual samples belong. The system tries to group samples by their mutual relations, properties and classification. This list of hashes is visually distinguished by an

[6] https://getbootstrap.com/docs/3.3/
[7] https://www.iban.com/country-codes

assignment of a unique color for each hash based on its cluster. Different color means a different cluster. If the sample does not belong to any cluster, its style remains unchanged. All related operations are handled directly by the client.

The header of the campaign details visualizes a map of countries, in which the sample, family or property was observed. CaDet does not provide further information about the current changes in the campaign since the initial detection and a time of its last update. To resolve this problem and support the detected campaign with data from external tools, the unique number of users that has seen the given sample or property is obtained from the graph database and visualized per each day of the last week. These data do not only help users to confirm the campaign, but it also provides further information about the campaign activity. In the case of fileless properties, the value can be interpreted and stored in the graph database under multiple possible categories. For example, a file name can be interpreted as a file name or a file path. Two charts, each for a different interpretation, are displayed due to these reasons.

Each campaign type supports searching for campaign information based on basic and advanced properties. The basic properties include campaign ID and the most specific search property. It is the hash of the sample for sample campaigns, the name of malware family for family campaigns and the type of the property or its value for fileless campaigns. If the user requires to narrow the search results by more specific criteria, those can be displayed by clicking on the specified button. The advanced properties include sample classification, time range of the campaigns, option for sorting the results (e.g. by the highest sample count). It provides an option to specify the sources of detection and other specific attributes based on the type of campaign, such as a specific hash that should belong to a certain family campaign. Visualized advanced search form is shown in Figure 6.6.

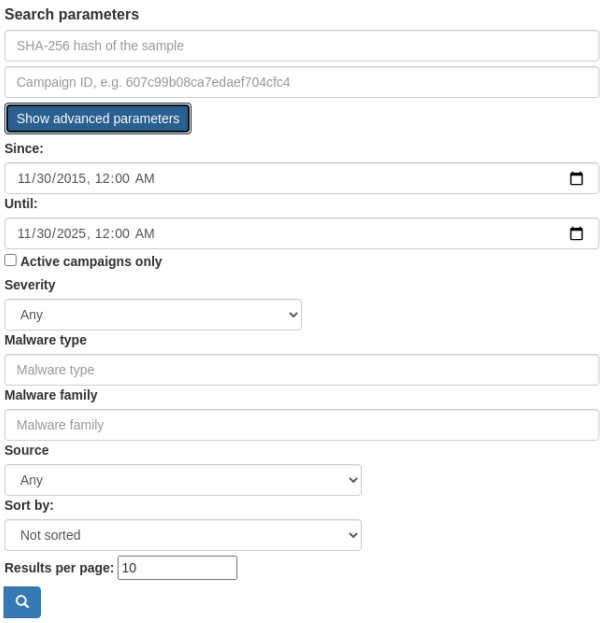

Figure 6.6: Basic and advanced search properties of sample campaigns

6.6 Alerting and Monitoring

The implemented service is able to inform the security researchers, analysts and all involved people by generating an alerting message about a newly detected malware campaign into a shared communication channel in the company. Such alerts must contain identifiers of the detected campaigns and their main representative data, such as related samples or sources. An interface for alerting using the Slack platform was implemented for this purpose. Slack is the main communication channel of the company. The interface is able to generate alerts by posting messages into the shared channel. Each channel is identified by its incoming webhook tied to a specific application. The webhook provides a unique URL that supports the posted JSON data representing the structure of the message. CaDet implements a custom interface that allows to easily construct different types of messages and post them to the selected webhook. Example of the constructed messages is shown in Figure 6.7.

Figure 6.7: Slack message alerting about a detection of the new campaign

Activity of the designed detection service is monitored via a visualisation platform Grafana[8] which supports creation of custom dashboards and charts that show real-time or aggregated data using various metrics. Each event, such as connection to the service and accessing its dashboard, successful API response or request, error detection, etc. can be tracked by sending an UDP protocol message into a StatsD daemon accepting incoming messages and further pushing such data to Grafana for processing and visualisation [46]. Grafana also supports alerts in case the used metrics change, e.g., the number of processed Kafka messages is lower than a certain number which helps for continues service failure detection and maintenance [20]. Interface for UDP communication was implemented as a part of utility methods. Example of the graph monitoring the number of consumed events is shown in Figure 6.8.

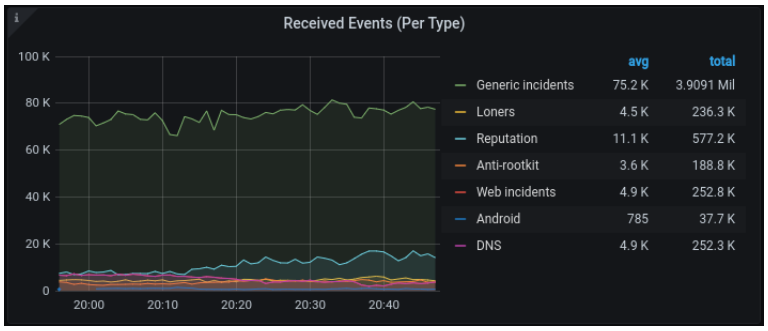

Figure 6.8: Monitoring of the processed events using Grafana platform

[8] https://grafana.com/

Chapter 7

Experimentation

This chapter describes the process of experimentation based on the principles presented in the previous chapters. It focuses on the scope of experimentation and the selected parameters that are used for campaign detection. This chapter also discusses the individual contributions of proposed detection rules described in Chapter 5 and event sources discussed in Chapter 3. The evaluation of the detection capabilities of the implemented service is also provided. The campaign detection was further verified based on the data reported by external sources to determine how the service is generalizing outside of the scope of internal systems.

7.1 Experimentation Scope

Experimentation and evaluation of the campaign detection are based on a comparison of two individual types of campaign reporting sources. Both set the boundaries of experimentation regarding campaign evaluation. This section discusses their mutual comparison. It also provides information on categorizing campaign-reporting sources into two groups based on their credibility.

7.1.1 Internal Sources of Campaign Reports

The first types of source are the internal systems of Avast Software, which already collect event information and report the obtained knowledge. These include systems described in Chapter 3, such as the graph database or an internal large-scale database of the retrospective data. The main disadvantage of these systems is that they are either affected by all data sources (without focusing on the client) or provide only a way to search for retrospective data lacking real-time detection capabilities. If the campaign is detected by CaDet, the campaign is evaluated retrospectively using data collected from these sources. If the properties of the detected campaigns are not appropriate and the campaign should not to be detected, the detection rules described in Section 5.2 are changed to prevent such detection.

7.1.2 External Sources of Campaign Reports

The second type of campaign reporting source is based on external detections. External data sources are used to verify that the service is generalized enough to model the real world to the extent of the events consumed. Data about ongoing or emerging campaigns are obtained from independent external reports. The data represent public reports created

by cybersecurity researchers and various antivirus companies and are for comparison with internal data.

The first type of data selected is a blog post from research labs and other antivirus companies. They are considered reliable and do not require further validation other than detection of the family aliasing. Such institutions published mainly two types of reports:

- Newly discovered threats, unknown malware families, new malware strains, etc.
- Reported campaigns of already known malware families and IoCs.

CaDet is tested against both types of the above-mentioned situations. Although data exchange agreements already exist between cybersecurity companies to increase their detection capabilities, this allows for a cross-comparison of their own detection approaches with the proposed solution. Each antivirus company has its own usually disjointed group of customers, so the use of this information is crucial for general threat reporting of large-scale campaigns.

The collected reports often provide not only key detection information, such as the name of the family currently spreading, but also IoC examples that can be verified using internal analysis systems. The internal systems provide additional information about the threats and help adjust the current version of detection rules. Among the selected antivirus companies whose reports were monitored were the following companies: ESET, Kaspersky, McAfee and Symantec. They issue reports on a regular basis, but each company has its own format of the reports provided. Each campaign report is manually evaluated and confirmed. The data collected need further analysis to obtain all possible information. For example, IoCs could be obtained by additional research because Symantec reports do not include IoCs. The samples reported by ESET are typically represented by SHA-1 hashes, which are not supported by all internal systems, etc. Reports from other research platforms or threat-intelligence research groups, such as Cisco Talos Intelligence Group, are also considered reliable.

The reported and published campaigns usually represent a prevalent campaign with a large-scope impact. These campaigns are also verified by antivirus companies, so they are considered trustworthy. However, the number of ongoing campaigns is larger, and there are also smaller campaigns that are not reported on the official blogs. They may not be considered to be a campaign by the definition of other companies or their scope of impact is too narrow. For these reasons, other external services are used to monitor current events. A list of other external services is described below.

- Reports from the public sandbox Any.Run[1]. The data of this service can be used to monitor the activity of different malware families in the exposed environment. This sandbox follows the analysis principles described in Chapter 2.
- Reports from *Malware bazaar* provided by the service Abuse.ch[2]. It provides a summary of researchers on malicious software, similar to Any.Run, supports automated information gathering of the followed security researchers presenting their reports on Twitter (a social media platform), provides up-to-date threat-intelligence data on threats such as the most seen malware families, analyzed file types, etc. Example report of the top submitted malware families directly to Abuse.ch service is shown in Figure 7.1.

[1] https://any.run/malware-trends/
[2] https://abuse.ch/

- Malware URLs[3] is a service to detect active phishing or malicious domains. Top reports can be obtained publicly, but reports are updated after a longer period of time.

Data from the above-mentioned sources are not considered reliable and are used to confirm blog posts of antivirus companies. They are also used as a source of reports of minor activity changes without blog posts from more reliable sources such as a new phishing domain or detecting a change in the number of reported samples of a certain malware family.

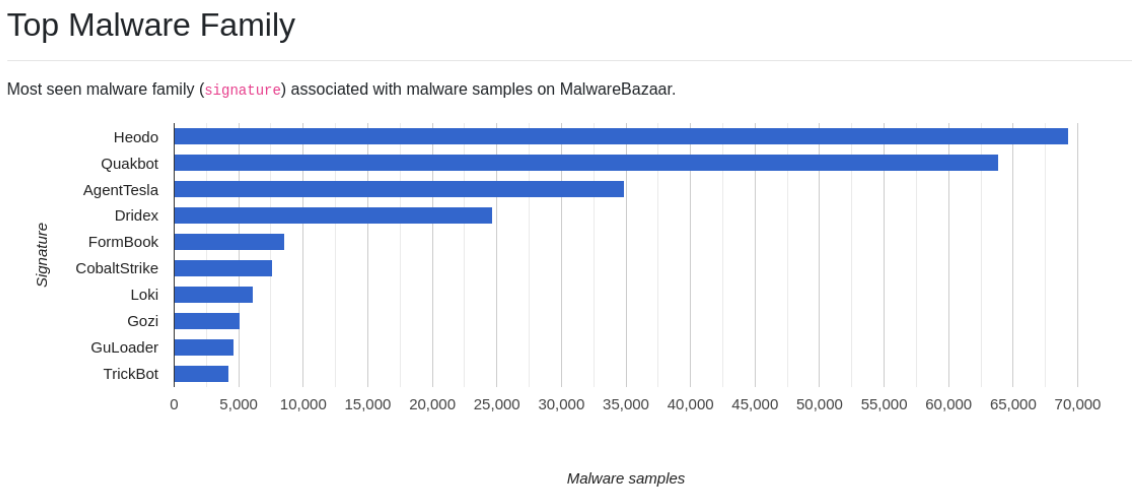

Figure 7.1: Top malware families reported by Abuse.ch

7.2 Campaign Detection Evaluation

The experimentation process of fine-tuning the detection rules has led to the normalization of the detection rules and changes in implementation so that the conditions are effectively evaluated. The detection capabilities of the service were evaluated several times. The evaluation was a subsequent iterative process that responded to new blog posts and newly detected campaigns by the service. This section provides the results of the final evaluation and describes the changes that were made during the experimentation process.

7.2.1 Threat-Data Sources Evaluation

Topics from which the relevant events are consumed do not only contain relevant data or events of the same message format. The feed of all events contains events reporting incidents that occurred at clients, but also other internal services. Such events need to be filtered. The list of filtered events is shown below:

- Events generated by additional products, not only monitored clients – Avast SecureLine (VPN service), Avast Forum, etc.
- Events unrelated to specific samples (except domain reporting sources)

[3] https://www.malwareurl.com/

- Events generated by internal tools such as analytical platforms and internal sandboxes. These data do not represent the current situation of clients.
- Top stars (very prevalent samples), clean samples according to the campaign score.

The fileless campaigns are related to a specific value and type of the artifact. Not all sources contain useful data for this type of campaign. The selected sources that detect these types of campaigns are shown in Figure 7.2.

Type of the fileless campaign	**Sources**
Domains	DNS feed, Domain feed
File paths	Anti-rootkit feed, Android feed

Figure 7.2: Sources used for detection of fileless campaigns

Feed from each source is evaluated independently. The evaluation includes their common coverage and the impact that each source has on the final detection of the analyzed campaigns. The evaluation is performed on several long-term deployments of the service. It is based on the observation of detected campaigns and specific results of the deployed version of detection rules. The following is a summary of the contributions of each source to campaign detection.

- **General incident reports**: This feed represents the main source of campaign-related data. It also partially covers data from other data feeds. In these cases, the incident is reported in a specific feed, such as the Anti-rootkit feed, and then reported again as a general incident report with additional information from other sources.
- **Reputation feed**: The reputation feed is especially useful for detecting campaigns of *clean* samples.
- **Loner feed**: The feed of loners (low prevalent samples) provides data about the emerging threats but mostly covers campaigns of clean samples similar to reputation feed.
- **Android feed**: A useful source of additional data on threats on mobile devices. Malware campaigns are reported based on this feed.
- **Anti-rootkit feed**: This feed is mostly covered by the feed of general incidents, but it is useful for separating related data, such as file paths, that are used to detect fileless campaigns.
- **Domain feed**: The originally used feed was replaced during the experimentation by the additional experimental source. The reasons are described in Section 7.3.
- **Network shield**: The feed reported by the network shield is the main source of networking-related incidents.

The importance of each feed for the detection of malware and clean campaigns is assessed separately. Malware campaigns are important in terms of mitigating identified threats and informing analysts about new campaigns. However, the rules were adjusted to detect clean campaigns as well. Their importance lies in the detection of an emerging threat, which is not yet covered by client detections and needs to be stopped in order to mitigate its

further spread. Figure 7.3 shows the importance of each individual source for the detection of clean campaigns and Figure 7.4 represents the malware campaigns. The importance is calculated based on the ratio of correctly identified campaigns of the given classification that are reported by the feeds.

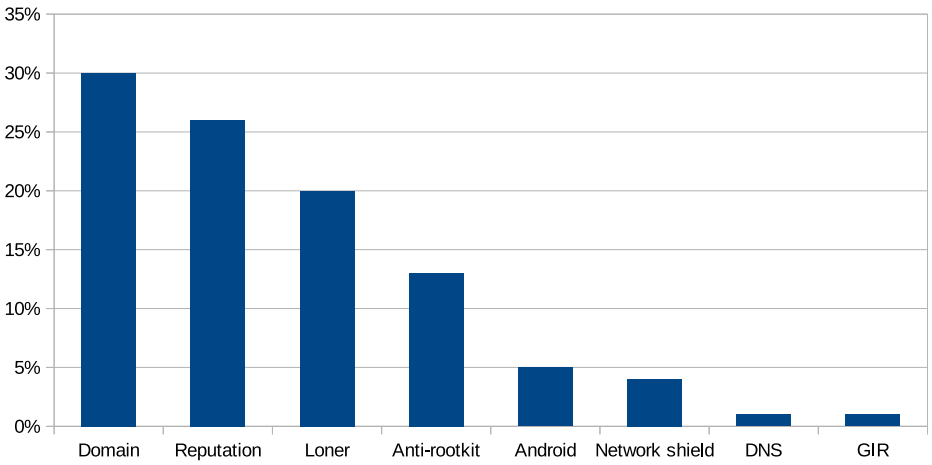

Figure 7.3: Source importance for detection of clean campaigns

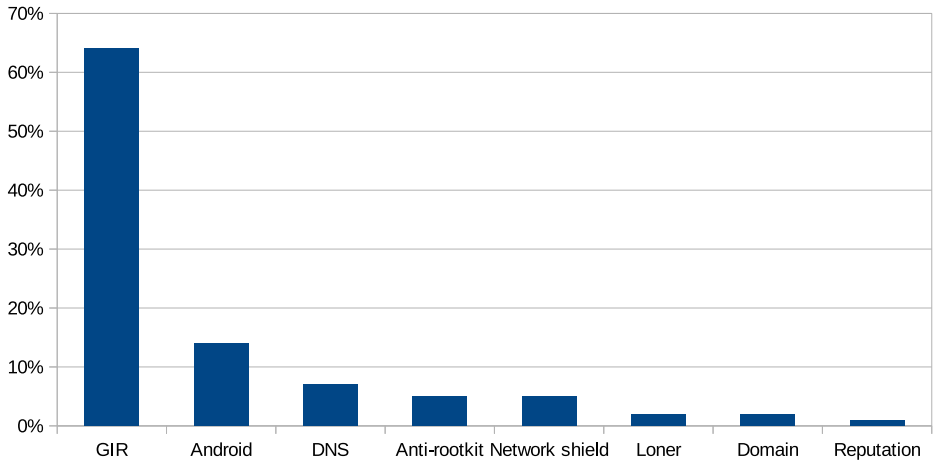

Figure 7.4: Source importance for detection of malware campaigns

Among the detected campaigns, the percentage of campaigns supported by multiple sources achieves 68%. This is a positive metric, meaning the majority of the campaigns are tracked by various independent data sources that provide additional information about the activity of sample or the malware family and their origin.

The average *multiplication rate* achieves 77%. This metric represents a probability that if an event is part of a particular campaign, the event cache contains an event reporting the same sample from another user or a different feed. This means that even if the event is missed or not processed, the campaign will most likely be detected due to data present in the event cache.

7.2.2 Detection Rules Evaluation

Each attribute of the detection rules described in Section 5.2 is tested as a single condition or in combination with other parameters. As a result, this section discusses the most successful combinations of the rules. Figure 7.5 shows the effectiveness of individual conditions in comparison with their mutual combinations. Most campaigns are detected based on the campaign score, the cache prevalence, and weighted graph database activity of the samples. The campaign score is not efficient enough to be used as a standalone metric, but it helps significantly in combination with other sources. The combination of all three properties is used the most by the service for campaign detection.

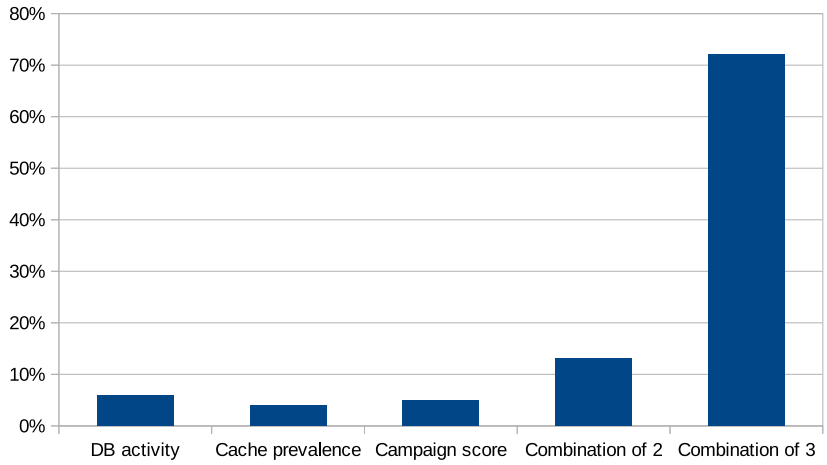

Figure 7.5: Types of the used detection rules

The following attributes are to have the greatest impact on the campaign score for the campaign detection. The default campaign score is set to a maximum value of one before the calculation, and the additional attributes described below affect the score. The score is normalized into the specified probability range after calculation.

Negative impacts on the campaign score:

- **Avast triggers** – These values represent incident triggering events when scanning attachments, detection by a behavioral shield monitoring individual processes, when detecting packets by a network shield, and the browser scanning detections. The mentioned triggers have often led to the detection of false-positive campaigns and the campaign score is decreased by 30-80% when encountered.
- **Avast shield** – The JavaScript scanning shield has led to an increased number of false-positive detections and the campaign score is decreased by 20% when encountered.
- **A valid certificate** – Samples with a valid certificate are generally less likely to be malware and they also led to an increase in false alarms. Therefore, their score is decreased by 30%.
- **Higher sandbox prevalence (SNX)** – A high sample prevalence presents several factors but can be thought of as a sample covered by detections. The high prevalence

also reduces the chances of a malware sample classification. If the value is higher than the specified threshold, the campaign score is decreased by 20%.

Positive impacts on the campaign score:

- **Avast triggers** – Campaigns detected on the basis of events from these sources: detections by the behavioral shield, on the execution of sample, and exploitation mitigation often contained related data and were less likely to be false positives. They increase the campaign score by 20-30%.
- **Invalid, expired or revoked certificates** – Sample with invalid, damaged or revoked certificates are more likely to be classified as malware. Increasing the campaign score by 30% leads to the detection of additional malware campaigns.
- **IRC protocol** – Application protocols can lead to detection of campaigns based on a certain type of communication, as explained in the malware family examples provided in Chapter 2. IRC is an example of such protocol due to C&C communication, and few undetected campaigns led to an increase of the campaign score by 10%.
- **Low sandbox prevalence (SNX)** – The low prevalence increases the chance of detection of emerging campaigns and the sample is more suspicious. The scan in the sandbox is determined by certain conditions. These events increase the campaign score by 20%.
- **DNS TXT response type** – DNS translation is heavy traffic, but TXT records, which allow the exfiltration of any arbitrary information by the malware DNS server, led to detection miss of a single related campaign. Based on this reason, the campaign score is increased by 20%.
- **Presence of Smart Home data** – The IoT campaigns were initially missed by the detection service. If the origin of the event is a smart home device, the campaign score is increased by 40% to increase the chances of campaign detection.

Other properties do not significantly affect the detection results after recalculating the campaign score based on their values. A similar approach is used to set thresholds for the activity of the observed objects in the graph database. The activity thresholds are split into four main categories representing the increase in activity over the last hour compared to the last day and the increase over the last day compared to the average daily activity of the last month. Each of these increases is determined individually for new, emerging threats and re-occurring samples. They are often manifested by a sudden increase of their activity. The activity of re-occurring malware samples and families can be observed in multiple time slices of the last month.

7.2.3 Detection Statistics

This section focuses on presenting the evaluated detection statistics for all types of campaigns detected by the implemented service.

A one-time evaluation of the currently stored campaigns in the production database is focused on the ratio between the various detected campaigns. This is shown in Figure 7.6. The highest number of campaigns detected is related to specific samples, followed by the fileless artifacts. The number of detected family campaigns is the lowest of all categories due to more strict detection rules and a limited number of malware families.

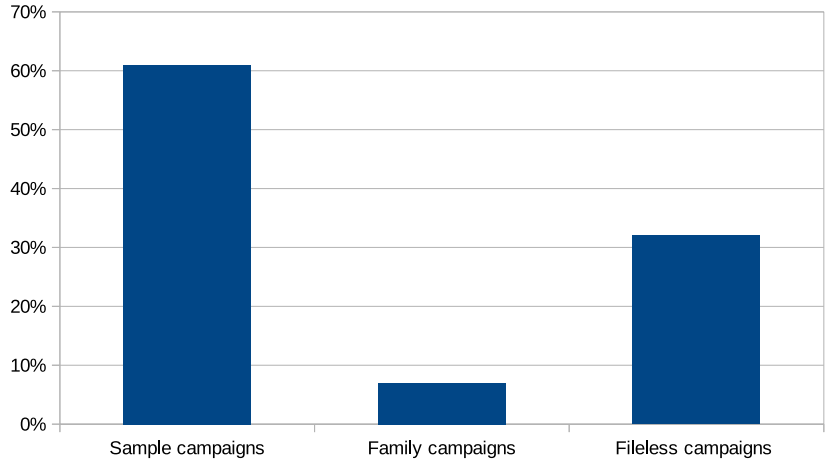

Figure 7.6: Ratio of the detected campaign types

In terms of continuous evaluation of the real-time detection, average numbers of the detected campaigns for the given category are presented in Figure 7.7. It confirms that the number of family campaigns detected is on average the lowest and that sample and fileless campaigns are more prevalent due to the diversity of their possible values. The number of campaigns detected varies slightly on different days, weekends, and different times of the day due to the world-wide user base and different numbers of clients installed across the world. The proportion of the detected fileless campaign types (path-to-domain ratio) was originally 13% of path campaigns and 87% of domain campaigns (with the older detection rules and usage of the original domain feed) compared to 3% and 97% respectively after the final evaluation. It means that the domain campaigns are more likely to be detected.

Type of the campaign	**First evaluation**	**Final evaluation**
sample campaigns	300 / hour	45-80 / hour
family campaigns	3 / hour	3 / hour
fileless campaigns	30-100 / hour	30-100 / hour

Figure 7.7: Average numbers of the detected campaigns

Among the most common reasons for missing a campaign detection (**false negatives**) are:

1. **Unknown classifications/tag** – The sample does not have any classification in the internal systems, as it could be a newly emerged sample or the company has not yet obtained the given samples from any source. This is a common reason for missing family malware campaigns that require accurate classification in order to be detected.

2. **Conflicting classifications** – Conflicting classifications are a reason for not classifying a fileless property that requires a unified classification of the connected samples. A commonly observed phenomenon is the unification of the classification of different samples with the correct malware type, but the family classifications differs.

3. **No data/few samples/stochastic approach** – Samples do not have to be sent by Avast clients for further analysis and only a certain fraction of the available data is processed. If the sample is not obtained from the client or other sources, the missing analysis data about the samples complicates both campaign detection and verification.

4. **Marginal cases of overly strict rules** – The rules are designed to find a compromise between the different perspectives of campaign detection. The increase in missed campaigns is caused by increasing the thresholds to lower the number of false positives.

The sample campaigns have proven themselves in the scenarios above by indirect detection of campaigns (as a different family, no classifications, etc.). The sample campaign does not require a precise classification to be detected. The estimated false positive campaign rate is 80%. This number depends on the preferences of users and their subjective *campaign* definitions. The presented number is calculated based on an additional campaign confirmation by any external campaign reporting source. Due to the amount of the processed data and the variety of the analyzed feeds, the calculated value is a success for campaign detection. This value is expected because external sources, such as blogs, often do not cover smaller campaigns.

The most common reasons for detection of **false positive** campaign are:

1. **Grey zone, a common campaign, no verification** – Some campaigns cannot be confirmed because the samples analyzed are not yet submitted and analyzed by the internal tools of the company.

2. **High current volatility of the graph database activity** – Some campaigns are detected due to the high volatility of the graph database activity, which means that they are significantly active, but in waves of very short duration. The detection rules fail to recognize a singular campaign of longer duration but detect multiple small campaigns, which is a less preferable result.

3. **Clean data** – The main purpose of clean campaign detection is to find an emerging threat that is not yet covered by various analysis tools and detections. However, the amount of processed clean data is higher than the number of malware incidents. Better filtering is one of the possible enhancements in the future.

4. **Generic classifications** – When the samples are correctly classified as malware with a common malware type, it is not enough to detect the family campaign. Moreover, classifications obtained from the embedded sources, such as detections, are often more general than the final classification estimated by Tagger. However, they are present in most of the incident reports and represent the fastest way of obtaining classification of the sample.

5. **Marginal cases of overly mild rules** – To identify certain types of campaigns, such as emerging samples, rules were made more general. This has lead to an increase in false positives from other types of sources.

The number of undetected campaigns in relation to the **size of event cache** is also evaluated. In the case of this metric, no significant detection differences are spotted once the duration has achieved a threshold of fifteen minutes. The differences are shown in Figure 7.8. If the size of the event cache is larger, it only slightly reduces the speed of the database operations, but it does not help improve the detection of missed campaigns. For this reason, a limit of fifteen minutes is selected as the final cache size. The recurrence rate of malware campaign-related sample events in the cache has achieved 87%.

In the case of sample campaigns, the **ratio of blocked and passed threats** is evaluated in terms of campaign importance. If the incident is suppressed for any reason, a

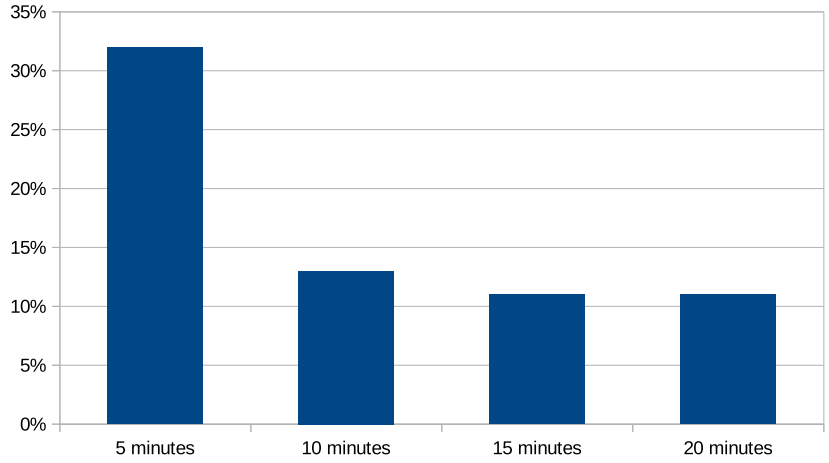

Figure 7.8: Relation of event cache size and the number of undetected campaigns

passed threat is reported. Detection that results in threat mitigation is reported as the blocked threat. The higher the value of this metric, the higher the volatility of the sample. The reasons why the incident is suppressed for different clients vary. Campaigns with these metrics higher than zero are also less likely to be false positives. The proportion of campaigns that contained threat-blocked ratio conflicts for samples in the same campaign was initially 29%. Following the increase of threshold in the rules, the proportion of such campaigns falls to 3%.

The impact of using **the weighted activity** obtained from the graph database instead of the general one is also a measure. The difference in campaign detection is estimated at 2%, which means that the difference between the two metrics does not have a significant effect on the process. A decrease in the false-positive rate is observed when using *Day vs. Daily Average activity* instead of using the overall monthly activity by 6.7%.

The most common reasons for network-related campaign detection are blocks generated by the active component of *webshield* that permanently scans the traffic. The second most common reason is a block of URLs. With these types of incidents, a specific URL is obtained from the events. The lack of detections of these types of campaigns is caused by the low conjunction of Avast's and Malware URL's domains lists during evaluation. Slow updates of the lists of the active campaigns (approximately once a week) have complicated the consequent evaluation.

Whether a campaign is classified or detected correctly is not a sufficient metric for evaluating the real-time detection service itself. For this reason, **campaign detection delays** are also evaluated based on multiple factors. The main metric is the analysis of the campaign using the retrospective data obtained from other internal systems and event cache. The purpose is to find the first potential event that should have been included in the detected campaigns. The campaign detection time is then compared to this number and the difference is noted. Figure 7.9 shows the average campaign detection time. Most campaigns are detected within 20 minutes of the first campaign-related event that did not result in campaign detection.

Once a campaign is detected, its duration represents the time of initial detection until the time of campaign archivation based on the detection rules. The average times of campaign duration are shown in Figure 7.10 for various campaign types. For most sample campaigns,

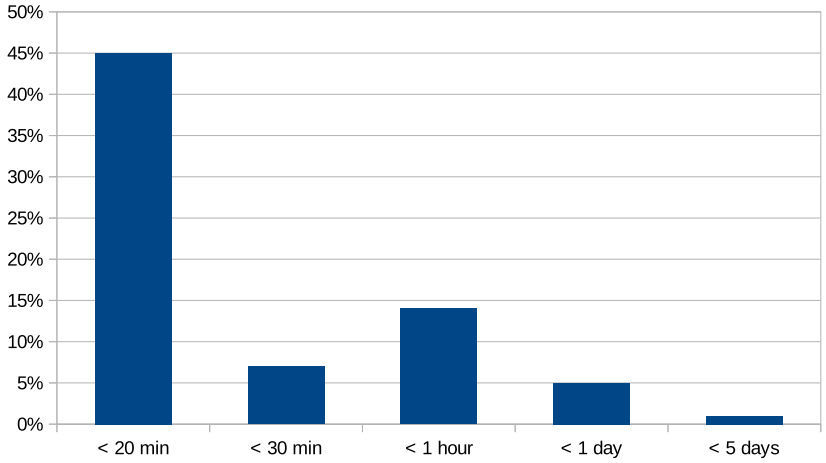

Figure 7.9: Comparison of campaign detection delays

the campaign ends after one day. The family campaigns have a higher duration. Fileless campaigns end on average up to an hour.

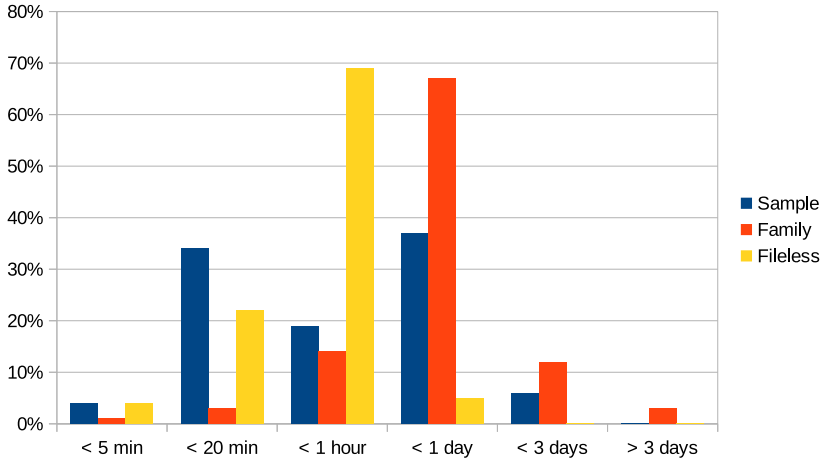

Figure 7.10: Comparison of average campaign duration

7.2.4 Real World Campaigns

For the reasons described in Section 7.1.2, the service was evaluated on the basis of the detection of confirmed campaigns by external sources by the researchers of the given institutions. If the campaign was not detected, the reason for the campaign miss was analytically examined with a focus on why the current rules were not sufficient or why the campaign was not detected at all. The list of campaigns evaluated in this way is described below [15, 17, 39].

- **Casbaneiro** – Casbaneiro is a malware family that typically operates in Latin America and serves as a banking trojan. In addition to other features such as mouse and keyboard emulation, Casbaneiro can take screenshots and upload them to the C&C server. The campaign took place mainly in Mexico in March 2021 and originally

was reported by ESET. The analyzed samples revealed that the campaign was indirectly detected as a family campaign and a sample campaign, which means that the classifications of the detected families differed but were correctly detected under the common malware type *banker*.

- **Mekotio** – Mekotio is similar to Casbaineiro in terms of being a banking trojan, which mainly attacks Latin America. It is interesting that this banker uses a shared SQL database as its C&C server. It has been active since at least 2015, but in January 2021, there was a recent campaign targeting Spain. It was detected indirectly as both a sample campaign and a family campaign and was originally reported by ESET.

- **Grandoneiro** – Grandoneiro has similar traits to the above-mentioned campaigns, but CaDet was unable to detect the campaign in February 2021. Analysis of the data showed that clients did not report any such data, which means that the detection miss was probably caused by insufficient data or user base disjunction between different antivirus companies.

- **Danabot** – A Danabot campaign, a banking trojan that attempts to steal credentials and private data, took place in January 2021 and was detected as a sample campaign, but not as the malware family campaign. This was due to an unknown internal classification for the observed samples. While a malware family campaign requires the knowledge of precise classification, sample campaigns do not initially require classification to be able to detect similar threats.

- **Bazarloader** – Bazarloader is written in C++ and tries to exploit communication using external tools, such as mails and a communication platform Slack, to spread. It serves for downloading additional modules and can be used by other malware types, such as Ryuk, for deployment. This campaign was directly identified as a sample and family campaign in January 2021.

- **Minebridge** – Minebridge was detected as a sample campaign due to a generic tag, which means that the observed samples were correctly classified as malware, but the family information was missing. Minebridge is a remote access Trojan whose campaign was observed in February 2021 by exploiting documents of Microsoft Word.

- **Masslogger** – Masslogger is a credential stealer targeting .NET platform. The anti-debugging techniques used are simple and can be bypassed by the analyst, leading to decryption and payload generation. The campaign was reported in February 2021 and detected as both sample and family campaigns by CaDet.

- **VGriefferS** – This family is based on ransomware principles, although it does not rename the encrypted files, as most ransomwares do. It informs victims about encryption and the steps that need to be taken in order to decrypt the data. The campaign of this malware was observed in February 2021 and was detected at the sample campaign layer because generic tags correctly classifying the threat as malware were missing the family classification.

- **LodaRAT** – LoraRAT is an AutoIT script-type remote access trojan with a deeper infection chain, which also acts as a keylogger. The campaign was reported by Talos Intelligence and was targeting states of North and South America. The detected samples were correctly classified as keyloggers, but the family classifications differed.

- **Vovalex** – Vovalex was also noticed in February 2021 and this family is a type of ransomware. The threat was discovered by MalwareHunterTeam and it is considered to be the first ransomware written in D programming language. Vovalex is distributed mainly from the pirated versions of downloadable software, such as various torrent clients, unzippers, etc. The encrypted files can be easily recognized because each file has an appendix `.vovalex`. It was recognized at the sample campaign layer, but due to its novelty, it was missing an exact classification.

- **Flubot** – Flubot is a campaign in contrast with previous campaigns because it is spread primarily via SMS messages that encourage users to install the malware application on their phones. The campaign of this type of malware was spotted in February 2021 and was correctly classified on both campaign detection layers.

- **Turla** – The campaign of this malware utilizes various sophisticated spreading techniques like vulnerabilities of the Flash service, document attachments, attacks on the older Java versions, etc. Its campaign was observed in January 2021 and detected on both types of layers.

- **Emotet** – The Emotet campaign was reported by multiple sources and it represents a commonly visible malware family of bankers and password stealers. The campaign was detected by earlier versions of this service in October 2020 and confirmed by sample analysis.

- **Coinminer** – Coinminers focus on mining cryptocurrencies. A campaign of this generic family was reported in February 2021 by two independent sources. It was successfully detected as a sample and family campaign.

- **Vadokrist** – Vadokrist is a Latin American banking trojan that first appeared in 2018, but its campaign was discovered in February 2021. It shares the most important traits with other banking families presented in this list and the program was written in Delphi. ESET first announced this campaign that was detected on the sample campaign layer by CaDet under a different classification.

- **Lokibot** – The Lokibot campaign was discovered in January 2021 and was detected as both sample campaign and malware family campaign. Lokibot is known to employ trojan malware to steal sensitive information from the victims and uses obfuscated files and software packing as its defense evasion techniques. The campaign in February 2021 was caused by the impersonation of a launcher of the popular computer game Fortnite.

Figure 7.11 shows a summary of the number of sources that mentioned the individual campaigns in their reports, whether the campaigns were confirmed by AnyRun sandbox or Malware Bazaar and the number of IoC (samples, artifacts) manually analyzed in order to find relations with the detected campaigns. This summary confirms that the additional data sources, such as AnyRun reports, are in some cases able to validate a campaign. It also summarizes which campaigns were detected by CaDet (SC – sample campaign, FC – family campaign).

Campaign	Analyzed IoCs	No. reports	CaDet det.	AnyRun det.	Abuse.ch det.
Casbaneiro	6	1	SC + FC	no	yes
Mekotio	3	1	SC + FC	no	no
Grandoneiro	7	1	no	no	no
Danabot	18	2	SC	yes	no
Bazarloader	9	1	SC + FC	no	no
Minebridge	30	1	SC	no	no
Masslogger	16	1	SC + FC	no	no
VGriffersS	10	1	SC	no	no
LoraRAT	20	1	SC	no	yes
Vovalex	12	1	SC	no	no
Flubot	8	4	SC + FC	no	yes
Turla	14	1	SC + FC	no	no
Emotet	30	3	SC + FC	yes	yes
Coinminer	10	2	SC + FC	no	yes
Vadokrist	5	2	SC	no	no
Lokibot	20	3	SC + FC	no	no

Figure 7.11: Campaign reports from external sources

7.3 Domain Feed Comparison

Domains campaign detection has proven that many possible values require reduction by filtering the most common values. This is necessary to reduce possible false positive detections as well as high data rates.

Experiments with the original feed reporting domain incidents have found that most of the reported domains and URLs can be classified as clean. This poses several problems because malware campaigns are more difficult to identify among the clean values. There is also a high detection rate of common campaigns without any significant or related information valuable for threat-intelligence purposes. Example of such a domain is `youtube.com`.

This issue is resolved by creating a custom domain list that is inspired by other filtering lists of the internal services, such as Clusty or the graph database. Some values are added to the blacklists based on the common detections by CaDet. The list consists of the blacklisted domains and URL prefixes.

The next step is to consume data from the newly created experimental feed instead of the original domain feed. The new experimental feed contains events related to the domain that is evaluated as suspicious at the client stations. The evaluation is based on static rules and the randomized clean domains previously obtained from the original data feed are not present. This reduces the amount of data processed and helps focus on malware campaigns. The percentage of campaigns reported from the new source drops to 18% of the original value. However, the proportion of malware-related campaigns remains at 99% of the original value. This means that the decrease is caused by the reduction of the clean campaign detections, which is an acceptable metric. The proportion of manually confirmed fileless malicious campaigns by their analysis using internal tools is estimated to be 8%.

Chapter 8

Testing

The designed and implemented detection platform was tested with a suite of unit, integration and end-to-end tests to verify and validate its overall functionality. The test suite represents a set of related test cases. All types of mentioned test types focus on different aspects and verify the functionality by using different testing techniques, such as mocking or generating an artificial data feed. All tests type are a common combination for testing of production services. The total number of the implemented automated tests is 342.

8.1 Testing Interface and Scope

All tests were implemented using the Python 3 module *unittest*. This module provides a high-level interface for the implementation of automated test cases and test suites. A Python testing framework *pytest* is used for the execution of the implemented test cases. Pytest provides a more pleasant user interface and can be easily extended with different plugins [31]. It also supports the configuration of the test interface using a configuration file. All tests are fully automated and do not need user interaction during the test execution. A short statistic of passing and failing tests is printed to the standard output after the execution of all tests is completed. Test coverage and statistics are generated into the prepared folder in a human-readable representation of an HTML document. To verify that code meets the standard Python criteria defined by PEP rules, the Flake8 module is used to check the source code and print any violations of the validated rules. This includes a maximum line length, usage of single vs. double quotes, order of the imported modules, etc.

8.2 Unit Tests

Unit tests focus on testing the functionality of specific modules, classes or functions [12]. They do not test the overall functionality of the application or cooperation between different modules. All tests are sorted into multiple categories and can be run separately for each category or as a whole set. The structure of unit test categories reflects the module structure of the project itself, so it is easier to navigate and find a certain test that will require an update once the implementation changes in the future and vise versa. Each module may be further split into multiple subcategories. A basic description of the main categories is provided below. Other tests that have a separate module due to the project structure include campaign detection tests that test whether detection rules are triggered based on

predefined conditions, classification tests and validation of the parsed detections. They also include verification that the consumers are set correctly and return valid attributes. The filtering module is also tested to verify that the event properties or entire events are properly filtered. The report displayed after test execution is shown in Figure 8.1.

- **Alerting** – This set of tests verifies that the alerting and monitoring interface was set up successfully for the related services Grafana and Slack, mocks the networking methods to simulate the communication and validates the constructed messages based on the message schemes provided by the platform documentation.
- **Campaigns** – Tests are verifying that the attributes of the initialized campaigns are set correctly.
- **DB** – Tests for the successful setup of the database interface.
- **Event Records** – Validation of transformation of the raw parsed events based on protobuf schema into an internal representation of the analyzed events.
- **External Services** – These tests mock responses from the external services, such as Tagger and the graph database, based on the specification of their outputs and validates that the mocked responses are processed correctly.
- **Utils** – Testing of all utility functions, formatting, localization, etc.

```
❯ pytest tests/unit/
================================== test session starts ==================================
platform linux -- Python 3.9.5, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: /home/patrik/projects/cadet_dp/cadet/cadet, configfile: pytest.ini
plugins: cov-2.11.1
collected 163 items

tests/unit/campaign_detection_tests.py ..................                        [ 11%]
tests/unit/classification_tests.py .....                                         [ 14%]
tests/unit/event_consumer_tests.py ..............                                [ 23%]
tests/unit/event_handler_tests.py .                                              [ 23%]
tests/unit/filtering_tests.py ....                                               [ 26%]
tests/unit/alerting/grafana_tests.py ............                                [ 33%]
tests/unit/alerting/slack_tests.py .........                                     [ 39%]
tests/unit/campaigns/family_campaign_tests.py .                                  [ 39%]
tests/unit/campaigns/fileless_campaign_tests.py .                                [ 40%]
tests/unit/campaigns/sample_campaign_tests.py .                                  [ 41%]
tests/unit/db/event_db_tests.py .                                                [ 41%]
tests/unit/db/family_campaign_db_tests.py .                                      [ 42%]
tests/unit/db/fileless_campaign_db_tests.py .                                    [ 42%]
tests/unit/db/sample_campaign_db_tests.py .                                      [ 43%]
tests/unit/db/general/mongo_tests.py .                                           [ 44%]
tests/unit/event_records/android_record_tests.py ..                              [ 45%]
tests/unit/event_records/antirootkit_record_tests.py ..                          [ 46%]
tests/unit/event_records/dns_record_tests.py ..                                  [ 47%]
tests/unit/event_records/domain_tests.py ..                                      [ 49%]
tests/unit/event_records/gir_record_tests.py ..                                  [ 50%]
tests/unit/event_records/network_feed_tests.py ..                                [ 51%]
tests/unit/event_records/record_tests.py .....                                   [ 54%]
tests/unit/event_records/reputation_system_tests.py ..                           [ 55%]
tests/unit/external_services/graph_db_tests.py ......................           [ 69%]
tests/unit/external_services/tagger_tests.py ....................                [ 82%]
tests/unit/utils/configuration_tests.py .......                                  [ 86%]
tests/unit/utils/environment_tests.py .                                          [ 87%]
tests/unit/utils/general_tests.py ....                                           [ 89%]
tests/unit/utils/locale_tests.py ....                                            [ 92%]
tests/unit/utils/serialization_tests.py ..                                       [ 93%]
tests/unit/utils/formatting/markdown_tests.py ....                               [ 95%]
tests/unit/utils/network/connections_tests.py ...                                [ 97%]
tests/unit/utils/network/udp_client_tests.py ....                                [100%]

---------- coverage: platform linux, python 3.9.5-final-0 ----------
Coverage HTML written to dir tests/reports/code_cov/


================================== 163 passed in 3.11s ==================================
```

Figure 8.1: Report shown after the execution of unit tests

Unit tests are not dependent on any data from the configuration file, any access to the external services or other servers. A concept called *mocking* is used to mimic the dependencies [36]. It allows the replacement of any object with an artificial mock that provides an interface for checking whether and how many times a certain method was called. It also allows setting a specific return value of any method or function. For example, to test the response of the unit that is dependent on the configuration file, the parsing method of the configuration file is mocked with an object with a predefined return value independent of the current contents of the file itself. This helps to easily isolate the tested scope. Manually predefined data are generated for each test. The total number of implemented unit tests is 163.

8.3 Integration Tests

Integration tests are meant to verify the integration of modules and various parts of the service to function properly using the real database access and web interface [51]. Integration tests also verify the communication with external services. They load all credentials and necessary data from the configuration file. Many of them require a running instance of the whole service. If the given test suite requires such an instance, an abstract class for the given test suite was implemented. It ensures that the service is executed as a sub-process that is terminated after all tests in the test suite are finished. Execution of integration tests takes a long time due to the initialization of the whole service and connection to external services. Tests focused on the web interface are performed using the Firefox browser. Because the browser can be downloaded, an automated script that downloads and sets up the browser together with its webdriver was implemented to limit the storage of obtainable data. The external services might have an outage during the test execution or the tests may be dependent on other factors that cannot be influenced by the developer. The testing framework provides a way of skipping the test when the criteria are met. For example, the test is skipped when the external service is disabled by the configuration file.

The webdriver is responsible for communication between the browser and Selenium, a testing framework for the browser automation [48]. Selenium was initially implemented as a Java library but its support was extended into multiple languages. It provides an interface for browser manipulation, page navigation, manipulation of the DOM structure and other useful operations. It also provides many methods for finding and altering the present and displayed elements, styles or scripts. It allows supports a connection to a VNC server for visualization of the testing process. A basic description of the main categories of integration tests is shown below. The total number of implemented integration tests is 176.

- **DB** – Verification that the detected campaigns and parsed events are correctly inserted, stored and found in the underlying database.
- **External Services** – Validation of the obtained results from all external services via direct communication using the provided API. The tests verify the real analysis results for the queried samples and properties.
- **Web** – Web tests are split into two main subcategories, API tests and Web UI tests. API tests verify that the service returns correct JSON responses on the endpoint access. Web UI verifies the document structure and the results displayed by the web service. It also tests all searching parameters by emulation of the user input on the web page. Error responses in case of an unexpected input are also tested.

Among other tests that have a separate module are event consumption tests. The tested consumers are executed for a limited amount of time or number of consumed messages. The tests verify that the consumed events were stored successfully into in underlying database. Tests for integration with the protobuf schema and validation of the constructed messages into internal representation are also present.

8.4 End-to-End Tests

End-to-end tests do not focus on a specific unit, module or single dependency of their integrated functionality, but an overall functionality of the test scenarios that verify multiple possible outputs of the process [51]. The implemented end-to-end tests focus on the process of campaign detection by generating an artificial data feed of the predefined messages that should lead to the campaign detection by the service. Service responses are tested by a unique scenario for each type of campaign (sample campaigns, family campaigns or fileless campaigns). They verify that the feed was correctly processed and that all expected campaigns were detected. The campaign must be stored in the underlying database, and the service should respond with the correct results by displaying the detected campaigns in the dashboard, campaign details and provides the correct data about them via API.

Chapter 9

Future Improvements

This chapter suggests possible improvements based on the implementation phenomena described in Chapter 6 and detection evaluation provided in Chapter 7. All enhancements are based on the current state of the service and focus on improving the results already achieved by detecting all types of malware campaigns. The user experience and visualization are also assessed to further improve the use of the service as a platform to alert cybersecurity researchers and analysts about the ongoing events recently reported by the antivirus clients.

9.1 Property Classification and Artifacts

This work implemented, evaluated and analyzed the fileless campaign detection of two types of artifacts – domains and file paths. Each type of artifact includes different interpretations, e.g., the file path can represent a file name or a sample path on the protected system. However, internal analysis tools support many different types of artifacts [32]. Such artifacts include mutexes, imports and exports of the binary executable files, API calls, resources of the binary samples, names of the file sections, etc.

Mutexes represent signaling objects on a given execution platform and are set to state *signaled* when they are owned by a thread of the running process [15]. They can be used to detect if the executed malware is already running on the infected computer. Imports of the binary executable files represent names of the libraries and functions that are included and loaded before the execution of the code to ensure the functionality of the program due to dependencies [29]. Less common sample attributes, such as these, can contain unique values used by a specific strain of malware. This can help the analysis tools to classify the threat.

Integrating such artifacts with CaDet is possible through the usage of an API or regularly updated database used by the service. It can help to expand the list of supported artifacts and estimate their classification based on the samples that use the given artifacts. Monitoring the prevalence of the above-mentioned attributes can help to detect campaigns of a certain malware family that usually uses the observed values, or identify an unseen malware family.

9.2 Campaign Connections

At present, the individual events are interconnected and processed based on the client context and values of the observed property, hash of the sample, family name or an artifact. Once the campaign has been detected and displayed by the service, more detailed data can be obtained from other internal services of Avast Software so that a common pattern can be found between the different campaigns.

The first external service, still under development, would allow CaDet to acquire semantically identical malware families. Their names are different because the samples were obtained from various sources that use unique naming conventions. This phenomenon is called aliasing of malware families and is a common problem challenging the automated systems used for threat-intelligence purposes. This system could help CaDet identify a diverse campaign of the same family with different naming conventions and either merge them into one or provide users with information that such a connection exists.

The second approach is to extend the current integration with Clusty as an internal sample clustering service in order to find relations between the detected campaigns based on the clusters into which they were clustered. The related connections can be shown to users together with the relevant data representing the reasons for campaign connection due to property similarities that are evaluated by Clusty.

9.3 Honeypots Integration

A honeypot is a service, system or environment that is designed to attract potential attackers by exposing a known vulnerability, port, etc. Honeypots are able to collect additional data about the monitored environment and observe the activity of attackers who try to exploit the exposed system [27]. Internal honeypots share information about current events by creating Slack alerts and producing Kafka messages that may be consumed by other systems. This allows CaDet to integrate honeypotting as an additional potential data source and try to detect campaigns in ongoing honeypot activities. Currently deployed honeypots collect information based on the current events of the exposed environments and are not affected by the artificially generated samples or sample exchanges between the companies, which supports the general motivation of the implemented service.

9.4 Alerting and Report Generation

CaDet currently provides an interface for creating Slack messages and posting them to the selected Slack channel based on the webhook specified in the configuration file. Potential expansion in the future could include integration of the user base, e.g., by using an OAuth authentication and storing the user data into the database. Various users, such as malware analysts, can then specify their own privacy policies, rules for notifications or provide their own Slack webhooks to stay informed about the current events. Example of such an extension would allow a potential user to be notified when a specific malware family is detected based on a selected rule. This will increase the flexibility of notifications for the implemented service and increase interaction with individual users.

The second possible enhancement in terms of reporting is the addition of support for generating regular reports about the current detection statistics. For example, a monthly report will be automatically generated in the form of a document showing individual statis-

tics about the campaign detected in the evaluated time period. Information such as this can be valuable for threat intelligence and confronting different internal services, identifying potential flaws in the detection process, or being shared by the company on social networks.

9.5 Scalability and Deployment

The system is currently deployed on a single test server with an integrated database. The consumers are based on the specification of individual containers in the Docker configuration files. Usage of an external database server might help with database concurrency. Deploying consumers among multiple servers and increasing their counts in the specified consumer groups supports scalability in terms of event processing. The robustness of the service also increases because the service does not depend on one server. For this purpose, it is possible to use an external task-processing broker to distribute the generated jobs across multiple deployed workers deployed by the service and thus distribute the current processing flow. Example of such tools is *Celery*. Celery provides a highly available and fast approach to task queue management [49]. Queues can be filled with analysis and evaluation tasks invoked by individual consumers and event handlers with the purpose of task flow distribution. It has native support for multiple servers, which further increases the scalability.

9.6 Event Processing and Bindings

A processing speed up might be required to keep up with the amount of produced events, which is expected to increase in the future together with the expansion of the user base and newly emerging threats. Approaches to increase processing speed can help reduce the stochastic approach. To ensure that the service keeps up with the flow, it is possible to implement Python and C++ bindings in the application. The main logic of campaign detection and event processing can be rewritten in C++ and the acceleration is achieved through the bindings. A language extension Cython, an integration of C++ and Python, can also help achieve similar results [11].

An alternative approach to increase the speed of event processing is to create an event processing cache where events are not processed sequentially as a single transaction. However, they are processed in a bulk of the consumed messages as a whole. This will help reduce the number of database operations and speed up the process.

Chapter 10

Conclusion

The goal of this thesis was to design and implement a service for real-time malware campaign detection to mitigate the need for retrospective querying of internal databases and threat-intelligence systems and to alert authorities when noticeable events occur without an avoidable delay. For this reason, it was necessary to study the internal systems of the Avast Software company and the topics of threat intelligence, malware classification and real-time event processing. Based on the provided information and documentation, a system named CaDet (Campaign Detector) was designed and implemented.

The system is able to process event and incident reports originating at protected end-user systems of the clients, filter the valuable information and estimate the emergence or ending period pattern of malware campaigns. It is designed to have a general support of events and file formats as well as to be able to report campaigns from incident reports that have occurred on personal computers, network traffic or mobile devices. CaDet supports three types of detected campaigns: sample campaigns related to a specific file, malware family campaigns and fileless campaigns of a specific property value, such as domain. Information about the campaigns can be obtained by using the provided API or accessing the interactive web application. The web interface provides threat-intelligence information in the form of an interactive map, statistics about the selected countries and details about the detected campaigns. Data provided include campaign detection sources, target countries, or recent threat activity. The service was successfully integrated with other internal systems.

The experimentation has confirmed that this service is able to detect malware campaigns. The rules, based on which is the detection based, can be altered to increase the efficiency of the implemented service and its detection capabilities. CaDet was able to detect campaigns reported by internal tools of the company as well as external sources, such as public blog posts of threat-intelligence researchers. The proposed detection rules rely on the recent activity obtained from the internal graph database, the prevalence of events in the fifteen-minute event cache and campaign scores. Campaign score is a metric calculated based on the properties of the obtained events, such as the validity of the certificate or the application protocol.

The service was also thoroughly tested with a set of unit, integration and end-to-end tests to verify its overall functionality. The functionality of the service can be enhanced by the addition of honeypots as a new data source, integration with other internal systems for family aliasing and campaign clustering, or implementation of the user base.

Bibliography

[1] *2020 Vulnerability Statistics Report*. EdgeScan. [Online; cit. 1.1.2021]. Available at: https://cdn2.hubspot.net/hubfs/4118561/BCC030%20Vulnerability%20Stats%20Report%20(2020)_WEB.pdf.

[2] *Protocol buffers*. Google developers. [Online; cit. 10.12.2020]. Available at: https://developers.google.com/protocol-buffers.

[3] *State of Malware Report 2019*. Malwarebytes Labs. [Online; cit. 1.5.2021]. Available at: https://resources.malwarebytes.com/files/2019/01/Malwarebytes-Labs-2019-State-of-Malware-Report-2.pdf.

[4] *State of Malware Report 2020*. Malwarebytes Labs. [Online; cit. 1.5.2021]. Available at: https://resources.malwarebytes.com/files/2020/02/2020_State-of-Malware-Report-1.pdf.

[5] Python Kafka Client Benchmarking. Activision Game Science. 2017. [Online; cit. 15.2.2021]. Available at: http://activisiongamescience.github.io/2016/06/15/Kafka-Client-Benchmarking/.

[6] *Investigation: WannaCry cyber attack and the NHS*. National Audit Office. April 2018. [Online; cit. 14.11.2020]. Available at: https://www.nao.org.uk/wp-content/uploads/2017/10/Investigation-WannaCry-cyber-attack-and-the-NHS.pdf.

[7] *The Threat intelligence Handbook: Moving Toward a Security Intelligence Program*. 2nd ed. CyberEdge Group, 2018. ISBN 978-1-948939-06-5.

[8] *PE Format*. Microsoft Corporation. 2018. [Online; cit. 12.12.2020]. Available at: https://docs.microsoft.com/en-us/windows/desktop/Debug/pe-format.

[9] Alam, S., Traore, I., Sogukpinar, I. and Coady, Y. In-Cloud Malware Analysis and Detection: State of the Art. *ACM International Conference Proceeding Series*. September 2014, vol. 2014. DOI: 10.1145/2659651.2659730.

[10] Amir, A., Salman, N., Babak, S. and David, B. Malware Dynamic Analysis Evasion Techniques: A Survey. November 2018. DOI: https://doi.org/10.1145/3365001.

[11] Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. et al. Cython: The Best of Both Worlds. *Computing in Science & Engineering*. May 2011, vol. 13, p. 31–39. DOI: 10.1109/MCSE.2010.118.

[12] Buffardi, K., Valdivia, P. and Rogers, D. Measuring Unit Test Accuracy. February 2019, p. 578–584. DOI: 10.1145/3287324.3287351.

[13] Chebotko, A., Kashlev, A. and Lu, S. A Big Data Modeling Methodology for Apache Cassandra. June 2015, p. 238–245. DOI: 10.1109/BigDataCongress.2015.41.

[14] Chodorow, K. and Dirolf, M. *MongoDB: The Definitive Guide*. 1st ed. O'Reilly Media, Inc., 2010. ISBN 1449381561.

[15] Elisan, C. C. *Advanced Malware Analysis*. McGraw-Hill Education, 2015. ISBN 978-0-07-181975-6.

[16] Ellis, B. *Real-time Analytics: Techniques to Analyze and Visualize Streaming Data*. Wiley, July 2014. ISBN 978-1-118-83791-7.

[17] Eslahi, M., Salleh, R. and Anuar, N. Bots and botnets: An overview of characteristics, detection and challenges. *Proceedings - 2012 IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2012*. November 2012, p. 349–354. DOI: 10.1109/ICCSCE.2012.6487169.

[18] Gallagher, S. They're back: inside a new Ryuk ransomware attack. SophosLabs. October 2020. [Online; cit. 10.5.2021]. Available at: https://news.sophos.com/en-us/2020/10/14/inside-a-new-ryuk-ransomware-attack/.

[19] Gerck, E. Overview of Certification Systems: X.509, PKIX, CA, PGP & SKIP. *The Bell Newsletter*. July 2000, ISSN 1530-048X, Vol. 1, p. p. 8. DOI: 10.13140/RG.2.1.1274.2489.

[20] Ghanei, F. Statsd Metrics Documentation. August 2018. [Online; cit. 24.4.2021]. Available at: https://readthedocs.org/projects/statsd-metrics/downloads/pdf/latest/.

[21] Highland, H. J. The brain virus: Fact and fantasy. *Computers and Security*. 1988, vol. 7, no. 4. ISSN 0167-4048.

[22] Holop, P. *Classification of Potentially Malicious File Clusters via Machine Learning*. Brno, CZ, 2019. Bachelor thesis. University of Technology Brno, Faculty of information technology. Available at: https://www.fit.vut.cz/study/thesis/21927/.

[23] Hossin, M. and M.N, S. A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*. March 2015, vol. 5, p. 01–11. DOI: 10.5121/ijdkp.2015.5201.

[24] Hrádek, I. *Štruktúra APK súboru na OS Android*. 2015. Master's thesis. Masaryk University, Faculty of informatics. [Online; cit. 15.10.2020]. Available at: https://is.muni.cz/th/uiuub/thesis.pdf.

[25] Jajoo, A. *A study on the Morris Worm: Optimal Co-flow based scheduling in Data Center*. Purdue University. May 2018.

[26] Jung, H., Lee, H. and Choi, J. Efficient Malicious Packet Capture Through Advanced DNS Sinkhole. *Wireless Personal Communications*. March 2017, vol. 93. DOI: 10.1007/s11277-016-3443-1.

[27] Kinariwala, R. M. and Jethava, G. B. Research on Various Next Generation Honeypot Systems. IJERT. December 2012. ISSN 2278-0181.

[28] Kolias, C., Kambourakis, G., Stavrou, A. and Voas, J. DDoS in the IoT: Mirai and other botnets. *Computer*. January 2017, vol. 50, p. 80–84. DOI: 10.1109/MC.2017.201.

[29] Koret, J. and Bachaalany, E. *The antivirus hacker's handbook*. John Wiley and Sons, Inc, 2015. ISBN 978-1-119-02875-8.

[30] Kováč, P. *Overview of the internal graph database. Internal documentation of Avast Software*.

[31] Krekel, H., Oliveira, B., Pfannschmidt, R., Bruynooghe, F., Laugher, B. et al. *Pytest x.y*. 2004. Available at: https://github.com/pytest-dev/pytest.

[32] Křoustek, J. and Zemek, P. *What is Clusty? Internal documentation of Avast Software*.

[33] Lakhani, J. Protocol Buffers: an Overview (Case Study in C++). *International Journal of Scientific Research in Computer Science Applications and Management Studies*. January 2014, vol. 3.

[34] Langley, A., Iyengar, J., Bailey, J., Dorfman, J., Roskind, J. et al. The QUIC Transport Protocol: Design and Internet-Scale Deployment. August 2017, p. 183–196. DOI: 10.1145/3098822.3098842.

[35] Lipovčan, R. *Monero usage and mining from usable security point of view*. 2019. Master's thesis. Masaryk University, Faculty of informatics. [Online; cit. 30.3.2021]. Available at: https://is.muni.cz/th/pvbu5/Thesis.pdf.

[36] Mackinnon, T., Freeman, S. and Craig, P. Endo-Testing: Unit Testing with Mock Objects. December 2001.

[37] Manos Antonakakis, e. a. Understanding the Mirai Botnet. *26th USENIX Security Symposium*. August 2017. [Online; cit. 10.4.2021]. Available at: https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-antonakakis.pdf.

[38] Merkel, D. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*. 2014, vol. 2014, no. 239, p. 2.

[39] Namanya, A. P., Cullen, A., Awan, I. U. and Disso, J. P. The World of Malware: An Overview. In: *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*. August 2018, p. 420–427. DOI: 10.1109/FiCloud.2018.00067.

[40] Narkhede, N., Shapira, G. and Palino, T. *Kafka The Definitive Guide: Real-time data and stream processing at scale*. O'Reilly, July 2017. ISBN 978-1-491-99065-0.

[41] Paquet Clouston, M., Haslhofer, B. and Dupont, B. Ransomware Payments in the Bitcoin Ecosystem. *Journal of Cybersecurity*. April 2018, vol. 5. DOI: 10.1093/cybsec/tyz003.

[42] Pawlak, P. Cyber security woes: WannaCry? European Union Institute for Security Studies (EUISS). May 2017. ISSN 2315-1129. [Online; cit. 2.5.2021]. Available at: https://www.iss.europa.eu/sites/default/files/EUISSFiles/Alert_13_Cyber.pdf.

[43] Pawlak, P. Mirai Botnet Loader Campaign. IBM X-Force Exchange. May 2017. [Online; cit. 1.4.2021]. Available at: https://exchange.xforce.ibmcloud.com/collection/Mirai-Botnet-Loader-Campaign-7e8131a283d50a0f13d43ae5f1d0058b.

[44] Plaskoň, P. *Vysoce výkonná platforma pro účely výzkumu malwaru*. Brno, CZ, 2019. Master's thesis. University of Technology Brno, Faculty of information technology.

[45] Plaskoň, P. and Zemek, P. *Tagger oveview. Internal documentation of Avast Software.*

[46] Plíva, R., Kuznetsov, D., Chrysaidos, N., Jursa, D., Vejmelka, M. et al. *Overview of threat-data systems. Internal documentation of Avast Software.*

[47] Rodriguez, C., Baez, M., Daniel, F., Casati, F., Trabucco, J. et al. REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices. June 2016, p. 21–39. DOI: 10.1007/978-3-319-38791-8_2.

[48] Salunke, S. S. *Selenium Webdriver in Python: Learn with Examples*. 1st ed. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2014. ISBN 1497337364.

[49] Stigler, S. and Burdack, M. A Practical Approach of Different Programming Techniques to Implement a Real-time Application using Django. *ATHENS JOURNAL OF SCIENCES*. February 2020, vol. 7, p. 43–66. DOI: 10.30958/ajs.7-1-4.

[50] Trautman, L. and Ormerod, P. Wannacry, Ransomware, and the Emerging Threat to Corporations. *SSRN Electronic Journal*. January 2018. DOI: 10.2139/ssrn.3238293.

[51] Turnquist, G. *Python Testing Cookbook*. Packt Publishing, 2011. ISBN 978-1-849514-66-8.

[52] Virgilio, R., Maccioni, A. and Torlone, R. Model-Driven Design of Graph Databases. October 2014, p. 172–185. DOI: 10.1007/978-3-319-12206-9_14.

[53] Wirawan, P., Riyanto, D., Nugraheni, D. and Yasmin, Y. Graph Database Schema for Multimodal Transportation in Semarang. *Journal of Information Systems Engineering and Business Intelligence*. October 2019, vol. 5, p. 163. DOI: 10.20473/jisebi.5.2.163-170.

[54] Zhauniarovich, Y., Khalil, I., Yu, T. and Dacier, M. A Survey on Malicious Domains Detection through DNS Data Analysis. May 2018.

Appendix A

DVD Contents

- **cadet/** - Source code of the application
- **tests/** - Unit, integration, and end-to-end tests
- **scripts/** - Supporting and deployment scripts
- **support/** - Data necessary for deployment of the service
- **dt_documentation/** - Source and PDF file of technical report
- **requirements.txt** - List of all required Python packages
- **config.ini** - The main configuration file
- **Dockerfile** - Configuration of the steps for image builds
- **docker-compose.yml** - Configuration of the containers
- **pytest.ini** - Configuration of the test framework
- **Makefile**
- **README.md**