FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Multi-modal threat detection framework
Student:	Bc. Jaroslav Hlaváč
Supervisor:	Ing. Martin Kopp, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Security
Department:	Department of Information Security
Validity:	until the end of summer semester 2021/2022

Instructions

The goal of this thesis is to create a working prototype of an easily extensible framework for user and entity behavioural anomaly detection (UEBA). The framework must support multiple different data sources, e.g.: network connection data (proxy logs, netflows), endpoint data (OS logs, registry modifications), and combine them into an embedding of observed entities (users, devices, etc.).

Analyze the state of the art statistical and behavioural anomaly detection systems in the computer security domain.
Based on the previous step, propose and compare embeddings for modelling various network entities. The quality of embeddings should be evaluated by means of anomaly detection.
Design a modular framework for processing different data sources by multiple behavioural models for various network entities.
Demonstrate viability of the framework on real data provided by the supervisor.

Electronically approved by prof. Ing. Róbert Lórencz, CSc. on 4 January 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Multi-modal Threat Detection Framework

Bc. Jaroslav Hlaváč

Department of Information Security
Supervisor: Ing. Martin Kopp, PhD.

May 6, 2021

Acknowledgements

First and foremost, I would like to thank my supervisor Ing. Martin Kopp, PhD. for his inspiring guidance and patience while explaining complex topics to me.

I would also like to thank my mother and father for their support in my studies and their tremendous help with proofreading the thesis.

Most importantly, I would like to thank my wife for continuous love and support during the process of writing this thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 6, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Hlaváč, Jaroslav. *Multi-modal Threat Detection Framework*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Detekce behaviorálních anomálií je široce používaná metoda pro ochranu počítačových sítí proti moderním útokům. Systémy behaviorální analýzy uživatelů a entit (UEBA) sledují chování entit a odhalují v něm vzorce běžné pro malware.

Pro monitorování počítačových sítí lze použít mnoho heterogenních zdrojů telemetrie. Mohou to být síťové toky, logy z koncových zařízení a logy z aplikačních serverů. Vytvořit kombinovanou telemetrii z těchto zdrojů je náročné, ale může to výrazně rozšířit detekční schopnosti systémů UEBA.

Tato práce navrhuje architekturu typu pipeline, která obsahuje tři nezávislé vrstvy pro zpracování kombinované telemetrie ze sítě i koncových zařízení. První vrstva kombinuje zdroje telemetrie do normalizovaného formátu. Vrstva Event generation ji předává modelům určeným pro detekci anomálií a ty z ní generují bezpečnostní události. Poslední vrstva používá vygenerované bezpečnostní události pro automatickou detekci hrozeb.

Implementace navrhovaného konceptu vrstvy Event generation byla úspěšně otestována na telemetrii z více než 25 různých privátních sítí. Velikosti sítí se pohybovaly od stovek do stovek tisíc zařízení. Jednalo se o sítě z různých odvětví, jako akademické sféra, finančnictví, doprava a medicína.

V rámci práce je zaveden nový postup pro vytváření embeddingů síťových entit pomocí Bag of Words modelů s použitím časových okének. Kombinace vytvořených embeddingů byla úspěšně otestována na problému sledování zařízení v průběhu času. Tento přístup rovněž umožňuje vytvořit nové typy modelů pro sledování náhlé změny chování zařízení.

Klíčová slova detekce behaviorálních anomálií, embedding entit, síťové chování, UEBA

Abstract

Behavioral anomaly detection is the state-of-the-art method for protecting computer networks against modern attacks. User and entity behavioral analytics (UEBA) systems track the behavior of entities to uncover patterns common for malicious software.

Computer network can be monitored using many heterogeneous types of telemetry sources. They include network flows, endpoint logs, and application-specific server logs. The combination of information from these sources is challenging, but it can vastly extend the detection capabilities of a UEBA framework.

This thesis proposes framework containing three independent layers for processing combined telemetry from both network and endpoints. The first layer combines the telemetry sources in a normalized format. The combined telemetry is forwarded to anomaly detection models in the event generation layer. Generated events are then used for automated threat detection in the last layer.

The proof-of-concept implementation of the event generation layer was successfully tested using telemetry from more than 25 different private networks. Sizes of these networks ranged from hundreds to hundreds of thousands of devices in multiple industries, including academia, finance, transportation, and medicine.

The thesis also introduces a new approach for embedding network entities by time window Bag of Words models in latent space. A combination of these embeddings was successfully tested for device tracking over time. It is also a prerequisite for a new type of behavioral models, such as a model for detection of a sudden change in behavior.

Keywords behavioral anomaly detection, entity embedding, network behavior, UEBA

Contents

List of Figures

List of Tables

Introduction

Protecting modern computer networks is a difficult task. The volume of data transferred and the number of devices online grow rapidly [1]. Networks change with each new service and each device update. This creates a space for mistakes in configuration and new attack vectors for malicious actors. Therefore, as the landscape transforms every second the security systems need to adapt dynamically to keep up with the changes.

The common way to detect threats in the network is to use Indicators of Compromise (IoCs). An IoC is a piece of data observed on the network. It can be defined by a signature such as a domain, file hash, IP address, etc., that was previously labeled as malicious. These signatures can be used to create alerts. To overcome a defense framework based solely on IoCs the attacker just needs to change the domain or one line in the code. Then he remains undetected until someone adds the changed signature to the list of known IoCs.

Behavioral analysis is a technique used to observe the activities of actors present in the network based on the context they appear in. It is used for modeling the behavior of entities in the network and helps to uncover complex or ever-changing threats. It is much harder to change the behavioral pattern of a malware than to change its IoC. No matter what program or domain is used to exfiltrate data, the behavioral pattern of unusual outgoing traffic will always be present (even though it might be hard to differentiate it from normal traffic).

For instance, seeing a network device access the domain `drive.google.com` does not indicate anything suspicious as it is commonly used public service. Yet, adding a context to the visit may drastically change that innocently looking activity. Imagine that the following sequence of events was observed in a short succession. An e-mail was received with a link from `secret-attack@yahoo.com`. A few seconds later, a script was run on the device modifying its Windows Registry [1]. Right after that the device started

[1] A database used for storing settings for the Windows OS.

communicating heavily with internal database server and a few other key network assets. A little bit later, huge amounts of data were uploaded to `drive.google.com`. Such stream of events immediately turns the innocently looking visit of a public cloud storage service into an important piece of evidence suggesting that sensitive data are being exfiltrated.

The example illustrated that having access to multiple sources of data and combining them on the behavioral level can be a powerful way to detect attacks. These different modalities can be endpoint logs from an operating system, network flows, e-mails, proxy logs, reputation databases of IP addresses or domains, and many more. The more information, the better.

Given the diverse nature of the data sources and the amount of data on the network, creating such a complete picture is a difficult task. The system needs to keep track of many features and also persist the previous state of the network to look for behavioral changes that indicate possible threats. With this approach, both known and unknown threats may stand out from the huge amount of traffic as illustrated by the example above.

The cost that comes with it is the complexity of behavioral analysis as it needs to dynamically adapt to the ever-changing state of the network. The widely used term for these methods is User and Entity Behavior Analytics (UEBA). The main challenge of UEBA is the creation of a normal profile of behavior of the entity, that can be used to look for abnormal actions. An **entity** is any identifiable object on the network. It can be a personal computer (PC), router, internet of things (IoT) device, or even a subnet, a group of devices (e.g. printers), etc.

A framework for user and entity behavioral anomaly detection was implemented as part of this thesis. A cross domain analytics (XDA) approach is used to combine information from different sources of telemetry that do not share a reliable common identifier. The models in the framework are designed to track and analyze the changes in behavior of entities on the network. The framework provides easily extensible anomaly detection capabilities on top of the combined telemetry.

The thesis also contains experiments with embedding devices into a latent space. Device representations in the latent space were used for device tracking over one week in a real network. The experiments had shown that using combined telemetry in modified bag of words device representation is a promising direction of representing the device behavior.

The rest of the work is structured in the following way. Chapter 1 explains the possible approaches to UEBA solutions and shows an overview of the proposed framework. Chapter 2 covers different data sources available on a computer network and points out the advantage of combining these data sources. The anomaly detection layer used for event creation is described in Chapter 3 and the threat detection layer follows in Chapter 4. Chapter 5 covers the architecture of implemented data processing framework and Chapter 6 covers the experiments conducted to compare different entity embeddings.

Goals

This thesis aims to create an easily extensible framework for user and entity behavioral anomaly detection that will serve as a part of an existing intrusion detection system Cognitive Threat Analytics developed by Cisco Systems, Inc. The focus is on anomaly detection using different models with several data sources.

The framework's input consists of network telemetry (network flows and proxy logs) and information gathered on the monitored network's endpoints (via Cisco Secure Endpoint[2]). The framework should investigate devices' behavior on the network by combining information from both data sources. Yet, the framework should be designed in such a way that it will allow additional data sources such as, e-mail or firewall logs.

The output should consist of events generated by behavioral anomaly detection models. These security events, need to be labeled by the device that the anomaly was observed on. They are consumed by subsequent modules of the IDS. The thesis should also demonstrate an approach for device behavior tracking over time in an embedding space.

The framework has to be lightweight in order to handle traffic from networks consisting of tens or even hundreds of thousands of devices. Moreover, it has to be extensible with new anomaly detection methods, behavioral model types, and data sources. The implementation is expected to serve as a proof-of-concept solution of the architecture.

[2]a client used for endpoint telemetry collection (`https://www.cisco.com/c/en/us/products/security/amp-for-endpoints/index.html`)

State of the Art

The traditional approach to computer security is to use multiple layers of defense. The first layer is on the edge of the network, and the last one is on the operating system level of the end device. To gain control of a device in the network, the adversary must get through the firewall, avoid detection in the internal network, gain access to the device and avoid detection there. For each of these layers, there exists a complex and robust solution. Yet, it is not enough to protect computer networks against modern threats. One of the reasons is the increasing sophistication of attacks. The attackers develop targeted attack vectors suited for specific networks, which are almost impossible to detect by traditional means. Another problem is the complexity of the correct configuration of sophisticated security tools [2].

To tackle these problems, security researchers invented a dynamic method that leverages the logging and detection capabilities of existing systems. They create the user and behavioral analytics as a sophisticated system to monitor modern computer networks (e.g. [3]).

UEBA solutions build standard profiles of entities (hosts, network traffic, etc.) on the network across time [4] to determine the entity's normal behavior. Traffic that differs from this baseline profile is then considered anomalous. The need for UEBA is well defined in [5]. UEBA system should support the human analyst (or software) to make decisions about potential threats. Their solution creates the baseline profile of the behavior from network traffic seen on a device and flags changes in the network communication as anomalies. The authors introduce a UEBA architecture of the system containing four parts: data preparation, feature extraction, behavior profiling, and anomaly detection. DINGfest [6] proposes a similar architecture that uses data collected by virtual machine inspection (VMI), creating periodic snapshots of the monitored system. Their approach focuses on permissions violations seen in the data stream. The anomalies are created by matching event streams to a database of malicious behavior fingerprints. DINGfest data sources lack the network data collected, thus decreasing the visibility into the network. Xi

et al. [7] again uses a similar architecture to find anomalies using endpoint logs. They use neural networks and isolation forests as anomaly detection models, and also introduce the idea of data enrichment from previously saved knowledge databases.

This thesis implements a proof-of-concept of UEBA framework similar to [5]. It is designed to be composed of different data source specific detectors. The PoC solution uses pattern matching models and histogram-based models [8] implemented with surprisal adjustments as described in [9]. It was further expanded by simple statistical anomaly detectors such as [10, 11]. These detectors are expected to consume network flows, network proxy logs, logs from the endpoint, with the possibility to add more data sources in the future. The sources are merged and then enriched with available information (e.g. GeoIP).

Chapter 1

Scope Definition of the Framework

To truly protect a computer network means maintaining confidentiality, integrity, and availability (CIA) of the data. Therefore, the ultimate goal of security solutions is to prevent any disruption of the CIA, whether it was caused by malicious intent or mere mistake. An example of malicious activity is sending a phishing e-mail to HR department in order to get access to the company network. A worker unaware of company policy copying confidential files to his personal Dropbox to work on them from home is a case of unintentional security policy violation by mistake.

In reality, protecting computer networks is a never-ending race between attackers and defenders. Network administrators try to stay at least one step ahead of the attackers in order to prevent most of the attacks. The security breaches need to be identified and patched as fast as possible. Once they are patched, the administrators use the knowledge to update the security so that the breach does not repeat. On the other side of the barrier, the attackers try to overcome any defenses put in front of them. They actively search for new loopholes and for known unpatched vulnerabilities in the network to achieve their malicious intents.

Over the years, many powerful intrusion detection techniques were developed to discover and prevent security breaches. This chapter covers defensive techniques that are often used within an intrusion detection systems (IDS). Parts of these systems target different stages of malware life cycle as described in section Section 1.1. A taxonomy of intrusion detection systems is covered in Section 1.2. Section 1.3 establishes the advantages that user and entity behavioral anomaly detection systems bring to the network security industry. Finally, Section 1.4 presents the overview of the proposed framework in the scope of an IDS.

1.1 Persistent Threat Life-Cycle

The goal of an attacker is to extract value from the infected host or network in any possible way. There are many ways to exploit networks by malicious activity. A database with personal data (addresses, passwords, usernames, etc.) can be sold on the dark web. Stolen blueprints with a new prototype can be sold to a competitor. Another way is to mine cryptocurrency on the infected hosts or by sending spam e-mails from it. In order to extract value, these attacks need to gain control of a device in the network or at least of a program running on it. Once they are in control, they are able to run arbitrary code and achieve their goals. In case of an advanced persistent threat (APT) a persistence will be established (e.g. by writing into Windows Registry). Or some sensitive data will be stolen with all traces of the attack erased.

A malware uses seven steps to get into the network and complete its objectives. They are summarized as a malware kill-chain in [12]. This model is widely used by security software and personnel to mitigate the threats on different layers of the network. The stages of kill-chain paraphrased from [12] are as follows:

1. **Reconnaissance**: Conducting both passive and active research about the target. Searching public information for useful clues to be used in the attack, e.g. port scanning, web scraping.

2. **Weaponization**: Creating a package of malware exploiting a vulnerability found during reconnaissance. E. g. hiding malware inside of a macro in an Office365 document or a PDF document.

3. **Delivery**: Transferring the weaponized package to the targeted network host. E. g. a document sent via e-mail, a USB drive dropped in the company parking lot.

4. **Exploitation**: Triggering the malicious code in the delivered package by exploiting the targeted vulnerability. E. g. user opening a document received in a phishing e-mail, plugging the USB into his/her workstation.

5. **Installation**: Establishing persistence on the attacked system. It can be installed in the Windows Registry to start automatically after a system reboot.

6. **Command and Control**: Establish a communication channel with the outside world. The malware notifies a Command and Control server outside of the network of its state and may receive instructions on what to do next.

7. **Actions on Objectives**: Taking concrete steps toward completing the objectives. These steps are controlled remotely through the communica-

tion channel set up previously. An attacker might use the network host to ex-filtrate internal data or any other goal that he is after.

Each of the stages can be further divided to smaller steps, and each step can be achieved by multiple different techniques. A curated knowledge base of adversary behavior with a detailed description of individual techniques can be found in MITRE ATT&CK® [13].

Kill-Chain Phase	**Visibility**
Reconnaissance	Network
Weaponization	Usually not visible
Delivery	e-mail, (Network, Endpoint)
Exploitation	Endpoint
Installation	Endpoint
Command and Control (C2)	Network
Actions on Objectives	Network

Table 1.1: Areas of a computer network where different phases of a malware kill-chain are visible. Weaponization usually happens outside of the network on a system maintained by the attacker.

To successfully achieve the objectives, the attacker needs to go through at least some of the phases of the kill-chain. Often he needs to go through all of them. Each phase gives the security team several opportunities to disrupt the kill-chain and effectively stop the attack. Table 1.1 shows where each phase of the kill-chain happens, can be detected and disrupted. In the following chapters, the focus will be on the most common areas of visibility: network and endpoint. Other relevant data sources exist, like physical security (monitoring office entry logs) or e-mail, but those are beyond the scope of this thesis.

1.2 Intrusion Detection Systems

The best way to prevent a malicious actor from causing harm to a computer network is to prevent him from gaining access to it. A properly set up firewall can stop a majority of non-targeted attacks. However, as the networks need to communicate with the internet, there is always a loophole where the attacker can possibly get through. Any unauthorized action on the network that can cause damage is considered an intrusion. An intrusion detection system is a software or hardware solution that detects any malicious network traffic and computer usage that the firewall cannot block [14].

A typical IDS only passively monitors the network. In case of an anomalous or suspicious behavior, it notifies the person or software responsible for the security. Some systems enable actively modifying defenses according to the detections. Such a system is called an intrusion prevention system (IPS).

An IPS has a significant responsibility as it can act on the observed events [15]. The following example can explain the difference between IPS and IDS, together with the pitfalls.

Imagine that there is an intrusion detection system set up on the edge of a small company network. Suddenly it notices that the mail server starts to send hundreds of e-mails an hour, where it previously sent only about 300 e-mails a day. It clearly is a massive change in the behavior of the server and therefore should be investigated. The IPS could immediately stop all outgoing SMTP communication from the network to prevent any further harm. The IDS would only send an alert to the security log, where the security worker would investigate it and act accordingly. In case of actual infection, the IPS would be a better solution as it would stop the harmful activity in a matter of minutes. But what if it was only a new marketing campaign, where the company started sending weekly newsletters to its customers. Then it would mean that the IPS effectively stopped a value-generating process of the company.

The previous example leads to the main difference between IPS and IDS. An IPS needs to be very precise and should act **only** if it is sure of the assumption. Each false-positive intrusion prevention can harm the protected company. On the other hand, IDS focuses on not missing any actual threat, and it can safely create less precise events. IDS produces some false positives, but these can be filtered by the subsequent system or person responsible for the threat detection. This thesis uses the term IDS as the focus is on the anomaly detection part of the system, which is similar for both IDS and IPS.

There are three types of IDS distinguished by the technique use to detect intrusions: signature-based, anomaly-based, and hybrid.

A signature-based intrusion detection system tries to match patterns observed in the traffic against a database of known signatures, previously observed in known attacks. In case of a match, an intrusion has been detected. The system can identify the attack type according to the signature matched with high confidence. The signature-based approach is powerful in identifying already documented attacks such as Snort signatures [16]. The biggest weakness of signature-based IDS is the inability to identify the zero-day (previously unseen and unknown) attacks. Since there is no signature, each zero-day attack goes unnoticed. There even exists polymorphic malware, a type of malware that leverages this weakness by dynamically changing the signatures to avoid detection. This weakness underlines the need for anomaly-based detection systems, as are designed to detect zero-day attacks.

The anomaly-based IDS leverages unsupervised methods for finding possible intrusions. It looks for suspicious traffic that deviates from the normal behavior on the network. Often it has a training phase to create a baseline for the normal behavior and active phase when newly observed traffic is compared with the normal behavior to identify anomalies. The main strength of the anomaly-based approach is the ability to detect previously unobserved

attacks. The behavioral change can identify any previously unseen malicious action, that might become a threat to the network. Because of the unsupervised nature, anomaly-based IDS also creates false alerts (e.g. a legitimate server moving to a new IP address could be identified as an anomaly because of high number of connections to a previously unseen IP address) that need to be filtered later.

The signature-based and anomaly-based IDS complement each other. The best performance comes from their combination in a hybrid IDS [17]. Threat researchers can leverage proper usage of anomaly-based IDS to find new signatures for a signature-based IDS. Models implemented in the proposed framework use both anomaly-based and signature-based approach.

The main two places in the network where an IDS can run are an endpoint device and a network device. Host intrusion detection system (HIDS) runs on a single machine with access to fine-grained information such as API calls, file executions, and registry changes (e.g. [18] or antivirus software). Many detections can be created on the end device with the help of signature databases downloaded from the internet (or policies set up on the host). HIDS enables a granular insight into what is happening on one device, but it lacks a global view of the network.

Network intrusion detection system (NIDS) usually monitors network traffic of the whole computer system. NIDS looks for anomalies in the traffic observed on collection points in the network. The information about individual hosts in the network is not as rich as from HIDS, but it enables modeling relationships between internal and external hosts. Moreover, it allows monitoring of all devices that communicate over the network, even those that do not have a HIDS installed, such as IoT devices and personal appliances.

Information obtained from both HIDS and NIDS gives a good insight into what is happening in the network and enables complex detections.

1.3 Cross-Domain User and Entity Behavior Analytics

User and Entity Behavioral Analytics (UEBA) is method used in anomaly-based IDS. Its goal is to create **security events** based on anomalies found in the data. Previous section covered two domains, endpoint device and network, where anomalies are usually detected. Since there can be many such events, it is not expected that a real person would go through all raw events manually. For that reason, security information and event management (SIEM) systems were developed. It is a system that aggregates logs and reports from multiple sources. Well-presented and aggregated information in a SIEM can help the administrator get more insight into what is happening in the computer network and take the appropriate action.

A cross-domain analytics (XDA) empowers the approach above by combining the telemetry under one entity (e.g. device, user or subnet) prior to the anomaly detection. This greatly increases detection capabilities of the system as the behavioral anomaly detection can run on the entity level. The fact, that entities are determined prior to the anomaly detection phase simplifies the work of the person using the SIEM and enables better automation of the threat detection.

Ideally, the cross-domain UEBA system would be able to identify users and different assets on the network. Personal computer, router, IoT device, subnets, groups of devices (e.g. printers) and others can be monitored. However, identifying entities on the network is a complex problem due to the heterogeneous data sources available in computer networks. The sources usually do not have an identifier to tie all the information together and create a well-structured database where one could filter by username, MAC address or IP address. Users change their locations, IP addresses rotate because of DHCP and MAC addresses are not always available in the data. Creating combined telemetry for the UEBA system, covered in Chapter 2, is a non-trivial task. Using this combined telemetry to produce security events is covered in Chapter 3.

1.4 Proposed Framework Overview

This thesis proposes UEBA framework, used to create security events from multiple data sources. An overview of the proposed pipeline architecture is depicted in Figure 1.1. It starts with raw data collected from different sources available in the network. This data are combined and enriched with additional information (for details see Section 2.5). The combined telemetry is then forwarded to the anomaly detection layer where detection models run tracking both whole network and individual entities.

The anomaly detection layer scales well with more data as the models run in parallel. Each model is consuming data from one modality and anomalies found are forwarded to the threat detection layer as security events. The threat detection layer can be either a SIEM or an automatic threat detection algorithm.

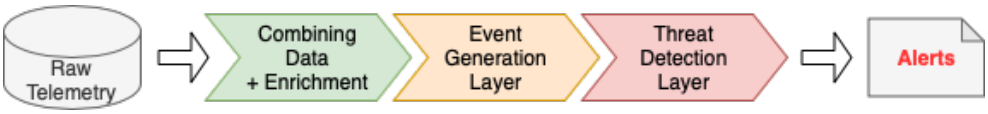


Figure 1.1: High-level overview of the framework. It uses a pipeline architecture starting with raw data and ending with threat alerts.

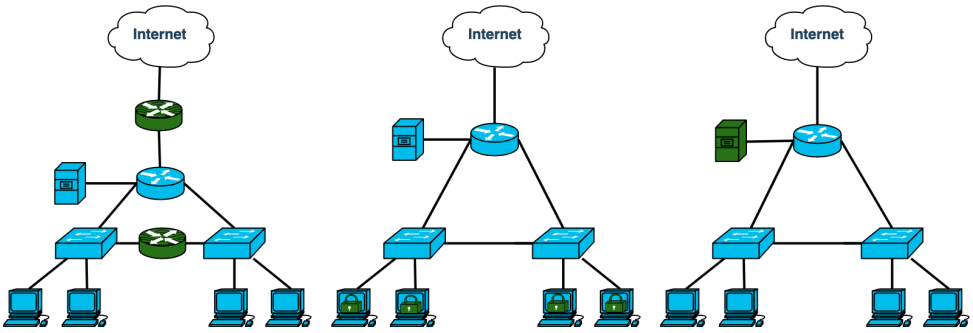
The main focus of the implementation part of this thesis was the event generation layer. Several anomaly-based and signature-based models were

used to create security events. The event generation layer was connected to systems implemented by other developers. The whole system was then tested as a successful proof of concept solution for UEBA deployed on data provided by Cisco.

Chapter 2

Data Sources

The detection capability of an IDS comes from the quality of the data collected, efficient filtering, and processing. Even a small high-speed computer network (hundreds of devices) can produce a considerable amount of telemetry. Thus collecting and storing the data is a problem that needs to be solved in both hardware (e.g. special network interface cards) and software solutions. A common assumption that more data means better results can become a liability while monitoring computer networks. To use accurate techniques, like deep packet inspection (DPI), the data needs to be prefiltered and stored in a concise form. It is important to note that the collected data contains highly confidential information. Therefore, it needs to be stored and processed in a secure environment without the possibility of it being ever exposed to unauthorized personnel.



(a) (b) (c)

Figure 2.1: Most common data collection points (depicted green) on the network are flow monitoring devices on the edge or inside of a network (a), a monitoring system running on the end devices (b) and server logs collection (c).

This chapter covers different data sources collected in a monitored com-

puter network. Figure 2.1 shows the most common places where an IDS can collect computer system telemetry. Having access to all of the data sources, brings the best results in defending the network. However, telemetry is not always collected on every device (e.g. IoT devices often do not have an endpoint client installed). Section 2.1 covers the network-based data sources, Section 2.2 gives an overview of endpoint data collection and Section 2.3 lists other possible data sources. The problem of merging the data under one identifier is laid out in Section 2.4. It also covers the approach of merging data from different data sources used in the deployment of the framework. Enrichment of the data from external databases is covered in Section 2.5.

2.1 Network Data

Today, most threats come over the network. Therefore, the information about an ongoing attack can usually be found in the network telemetry. The attacker is generating traffic that would not normally be present on the network. The goal of the network monitoring system is to highlight such unwanted network traffic.

The smallest amount of data that is usually monitored is a packet. A network packet is a formatted piece of information traveling over the internet. It contains information used for transporting the packet across the network and the actual information carried to the destination. Theoretically, whole network communication can be reconstructed if all packets are captured. In reality, it is not possible because of the following reasons. Networks can easily generate tens of gigabytes of telemetry per second. Going through this amount of data in real-time is computationally unfeasible, and storing it would take an immense amount of space. Moreover, most of the traffic is encrypted by private keys available only to the communicating hosts.

As packets are not suitable for efficient network monitoring, they are aggregated into network flows. A **network flow** is a record containing important properties of an unidirectional communication between two network host. A lot of the information is lost in the aggregation process (especially the actual payload of the packets). However, the condensed information enables investigating the network flows in real-time (or close to real-time) and storing them in a database for later investigation. Many countries around the world require internet service providers to store flow information for a period of time to be used for forensic purposes by the authorities if there is a suspicion of a crime.

There are two commonly used format specifications for network flow, the NetFlow [19], and IPFIX [20]. Different implementations use different fields. However, the basic structure is very similar for both of them. The most common fields are the following:

1. Flow identifier

2. Source IP address
3. Destination IP address
4. Direction
5. IP protocol
6. Source port (for UDP or TCP)
7. Destination port (for UDP or TCP)
8. Bytes transferred
9. Start time
10. End time

Once captured, the flow record is usually processed and stored so that an IDS system can access it later. Sensors in different parts of the network can collect network flows, as depicted in Figure 2.1a. The most common spot for deploying the flow collector is the edge of the network, where all communication with the outside world can be captured. Routers used on edge often come with this capability. To enable monitoring of the internal communication, collectors also need to be placed inside the network. However, this creates additional work for the IDS as it needs to deduplicate some of the flows seen on multiple collection points.

Network flows are a good source of information for behavioral analysis. They can be further enhanced by other interesting parts of the communication, such as the initial data packet (IDP). IDP is the first packet sent as part of the communication that usually carries essential information about the connection. Network flows can also be grouped to bi-flows merging the two directions into one record, which further reduces storage space.

Proxy logs can be considered as another source of data (they could also be categorized as application-specific, see Section 2.3). A proxy is a server that acts as an intermediary between the user and the internet, especially the web. A proxy can serve as a simple security measure in the company network. They can **partially** hide the network identity of the end-user, block specific websites, etc. Each request to access a website goes through a proxy and is logged there. These logs contain important information about the connection: URL of the requested website, the username of the user that has created the request, and more depending on the logging setup.

Network telemetry data used in this thesis comes from both network flow collectors and proxy logs.

2.2 Endpoint Data

Even though most attacks come over the network, their target is an endpoint device. An endpoint is any physical (or virtual) device connected to the internet. It can be a server, personal computer with any operating system, mobile device, switch, router, or IoT device. Each of these endpoints is a potential target. Personal computers, servers, and mobile devices are often targeted by attacks trying to steal confidential information. They are usually better protected than IoT devices and small appliances like cameras and home routers. These are more often exploited as bots for sending spam e-mails or DDoS attack generation.

The attacker's goal is to infect the endpoint with malware and hide its presence. It tries to mislead the operating system and any antivirus software into believing it is a legitimate service. The malicious code exploits vulnerabilities in the host's software to gain control of it. Once it is in control, it can perform actions to complete its goal. An example of an obfuscation technique can be injecting malicious code into running processes (process hollowing), running solely in memory without any trace on the disk, and even deleting evidence of its presence when the goal is achieved.

An adequately set up endpoint is logging the actions happening on it. Logins, program installation, file download, file execution, registry changes, reboots, and even API calls can be logged. With this kind of information, the antivirus software running on the endpoint can look for malicious activity. It matches the hashes of files with its database to block any known threats. It can also monitor the Windows Registry changes and send suspiciously behaving software to the threat analysts for further investigation. Even more sophisticated detection mechanisms can run on the endpoint, such as monitoring the system calls [21].

Some of the logs can be sent to a secure cloud database accessible by an IDS, where more computation capacity is available. This also enables an overview of the health of the computer network from the view of the end hosts. For example, suppose malware starts appearing on multiple Windows devices in the network. In that case, the administrator can assume a virus is spreading through the network and act accordingly (e.g. quarantine these devices into a separate subnet to prevent the infection from spreading). Alternatively, it can analyze the freshly changed registry keys to discover and attempt to gain persistence on the machine.

The endpoint logs used for experiments in this thesis come from Cisco Secure Endpoint clients.

2.3 Application Specific Data

In addition to user-operated endpoints and network infrastructure devices, there is usually a service-providing server. These servers are an attractive target for the attackers as they hold important databases and provide services crucial for the operation of the network. Logs from these services are another data source useful for multi-modal detections. An example could be the Lightweight Directory Access Protocol (LDAP) server, where logins to the company network are being logged. If an LDAP server is compromised, it could allow the attacker access to a database of all users in the network. Logs from a network proxy, mentioned in Section 2.1, can be to some extent considered application-specific.

Another excellent example of application-specific telemetry are mail servers. Phishing e-mails are still one of the most prevalent sources of infection. Therefore, monitoring mail servers should be a priority. The best results are achieved when the security system can read whole e-mails and use natural language processing and other methods to find phishing and other threats. This approach requires particular caution as e-mails are highly sensitive. For this reason, detections on e-mail data often use only the header information.

Combining e-mail data with other sources of telemetry has become increasingly important. It is standard now that the attackers use legitimate services to deliver malware to the users. Malware code can be found on GitHub[3], or the binaries are being downloaded from Google Drive, Dropbox, and other public storage services. The link from a phishing e-mail might download a compressed zip file from Google Drive containing malware as well as a legitimate document sent by a coworker.

The framework implemented as part of this thesis does include only proxy logs as application-specific data. However, it was implemented with other data sources in mind. If e-mail or other application-specific data are available, they can be easily added to the framework to widen its detection capabilities.

2.4 Merging Data

Detecting anomalies in individual telemetries covered in previous sections is a well-used security technique. However, when the telemetries are merged, the quality of detections can increase significantly. For example, by looking only at the link or file attached to the e-mail, it is hard to decide whether it is malicious. Knowing that the executed file was received as an e-mail attachment came from a suspicious address, it tried to communicate with an IP address hosted in Zimbabwe and was found malicious by the antivirus software gives the security personnel a much better story of what has happened on the network.

[3] a provider of hosting software source code on the internet (`https://github.com/`)

Combining the telemetry from more data sources enhances current detections and opens new detection capabilities of the IDS. Data, merged by a unique key, can also be used as a complex behavioral fingerprint of an entity. Not only it can automate some of the processes that are usually done manually, but it can also bring up important information that would otherwise go unnoticed.

Combined telemetry is created by joining all sources of telemetry under one entity. An entity is any object on the network that can be logically separated from the others. It can be a user, a device, a subnet, a group of devices or users, etc. The choice of an entity depends on the data available and the nature of the protected assets. The rest of the thesis uses a device as an entity of choice. The approach for different entities would be similar.

A unique device identifier is needed to distinguish between devices and merge available telemetries. For endpoint data, this unique identifier can be the installation ID of the antivirus software or operating system or MAC address if it can be obtained reliably. An IP address is the closest thing that is available as the unique identifier in network telemetry. However, in private networks, an IP address is not necessarily unique. On top of that, IP addresses change over time, so the identification is valid only at the exact moment the IP was observed. Some telemetry cannot be accurately merged as some types of telemetry may not be available for some devices, e.g. IoT or mobile devices might not have antivirus software installed.

Fortunately, modern endpoint monitoring software also stores some network requests and logs the source IP address used for this request. One way to create the combined telemetry from the endpoint and network telemetry is to join network telemetry to the endpoint telemetry by matching IP addresses in a time window. Prolonging the time window lowers the certainty that the match has been created correctly. However, make the time windows too small, and less telemetry can be merged accurately. This approach requires the timestamps of different telemetries to be aligned.

Merging other types of telemetries is similar to the example above. There needs to be at least one common feature, to merge the telemetries correctly. The quality of the resulting data depends on the features available.

The approach explained above was used to create combined telemetry for this thesis. The resulting dataset consists of network and endpoint telemetries merged under a unique device identifier. Some parts of the telemetry could not be used for multi-modal detection as it could not be merged with high enough certainty. One way to improve the algorithm for data merging, is by optimizing the time window. However, it is out of the scope of this thesis.

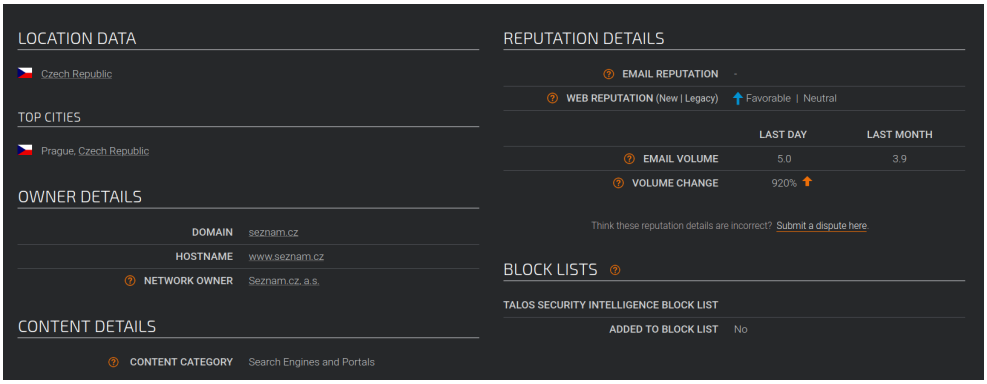


Figure 2.2: Response for a `www.seznam.cz` in Talos web interface.

2.5 Data Enrichment

The data collected and merged as explained in previous sections can be further enriched with even more information. Below are listed three sources commonly used for data enrichment. It can be information available on the internet via shared allow lists, blocklists, and databases. Other means are using an internet geolocation (GeoIP) database or passively gathering domain name system (DNS) queries.

Both public and private block and allow lists can be used to enhance network flows. For example, Cisco Talos Intelligence Group [22] maintains a whole reputation database that contains information whether Talos considers the queried domain, IP address, or URL favorable or not. A simple web view of such a query for `www.seznam.cz` can be found in Figure 2.2.

Gathering GeoIP information is a process used to determine the geographical whereabouts of the queried IP address. There are publicly available GeoIP databases available on the internet (e.g. [23]), and many companies collect their own databases suiting their needs. The easiest way to gather GeoIP is to create a `whois` query targeting an IP address. The `whois` command-line tool uses the WHOIS [24] text-based protocol designed for providing information about IP addresses and domains to internet users. The registration authority responsible for the IP maintains a server that responds to `whois` queries with a record containing the company's name, location, and other information about the IP address. An example of a response to the `whois` query can be found in Figure 2.3.

Another common way to store additional information is to create a passive DNS database. Passive DNS is an approach to store historical DNS resolution data into a database. In case of a security incident, this database can be referenced for past DNS queries. It can help uncover threats that use domain generation algorithms (DGA), for example, a malware using domains that exist only for a short amount of time. When the security expert investigates

```
inetnum:        77.75.74.0 - 77.75.74.255
netname:        SEZNAM-CZ
descr:          Seznam.cz
country:        CZ
admin-c:        SZN5-RIPE
tech-c:         SZN5-RIPE
status:         ASSIGNED PA
mnt-by:         SEZNAM-MNT
created:        2007-06-20T11:47:42Z
last-modified:  2007-06-20T11:47:42Z
source:         RIPE
role:           Seznam.cz IT department
address:        Radlicka 3294/10 150 00 Prague 5
                Czech Republic
phone:          +420 602 126 570
abuse-mailbox:  abuse@seznam.cz
admin-c:        PZ172-RIPE
tech-c:         SZN11-RIPE
tech-c:         SZN10-RIPE
nic-hdl:        SZN5-RIPE
mnt-by:         SEZNAM-MNT
created:        2007-05-06T15:50:27Z
last-modified:  2015-07-03T13:19:00Z
source:         RIPE # Filtered
% Information related to '77.75.74.0/24AS43037'
route:          77.75.74.0/24
descr:          SEZNAM - II
origin:         AS43037
mnt-by:         SEZNAM-MNT
created:        2007-10-04T08:21:30Z
last-modified:  2007-10-04T08:21:30Z
source:         RIPE
```

Figure 2.3: First part of the response from RIPE (Réseaux IP Européens) for the query `whois 77.75.74.172`, the IP address of `www.seznam.cz`. A lot of additional information can be found in the response.

a domain, the only record is in the passive DNS database as the domain is already nonexistent.

The data sources covered in this chapter open vast opportunities for both human and machine-operated intrusion detection systems. The private dataset used in experiments in this thesis consists of network flows, proxy logs, and endpoint telemetry gathered by Cisco Secure Endpoint logs merged by source IP address under Cisco Secure Endpoint client installation ID in one minute time windows. Public IP addresses are then enhanced with GeoIP data.

Chapter 3

Proposed Framework for Security Event Generation

The main focus of this thesis is the event generation layer of the framework. This layer leverages user and entity behavioral anomaly detection to find suspicious activity on the network that traditional signature-based approaches would not notice. In recent years all of the major security system vendors (e. g. [25], [3], [26]) were investing heavily into the development of UEBA systems as they are by design able to detect zero-day attacks [27]. This chapter covers the theoretical background for cross-domain UEBA framework consuming combined telemetry merged by the algorithm from Section 2.4.

A straightforward yet still powerful behavioral anomaly detection method is to create several different models, each handling a specific type of detection. For this reason, both anomaly-based and signature based models are deployed in the proposed framework. The findings are then combined in the subsequent threat detection layer.

Anomaly-based models create a **behavioral fingerprint** of the tracked entity and look for perturbances in the behavior. Signature-based models generate security events by matching signatures with the telemetry. With the combined telemetry as an input, these security events are easily assigned to a specific entity/device. Knowing the source entity that has created a security event, helps with threat detection and serves as evidence for investigation. Section 3.2 covers the details about individual implemented detection techniques.

Another type of behavioral fingerprint is a vector representation of entities embedded into a vector space. This approach opens several options useful in an anomaly detection framework. New types of anomaly detectors can be introduced by tracking the embedding over the vector space and looking for significant shift or a change in the group of an entity. Clustering methods can be used to create groups of similar entities. These groups can be monitored by models with specifically adjusted thresholds. Section 3.3 formalizes the

problem of embedding creation. The embeddings are used in experiments in Chapter 6 for tracking devices in time according to their behavior.

3.1 Anomaly Detection

Anomaly detection (also called outlier analysis) is a technique used to find data points that do not correspond to the observed group's expected pattern. Most of the data points likely originate from a normal data source. Whereas, anomalies are data points that do not adhere to the structure expected from the normal data source [28]. Therefore, there is a suspicion that the abnormal data points originate from a different source or the normal source started to behave differently. The goal of anomaly detection is to flag these abnormal data points as anomalous.

Anomaly detection is used in many different areas (i.e., industrial damage detection [29], medical anomaly detection [30], cyber-security anomaly detection [31], etc.) for details, see e.g. [32]. In some areas the normal data source is well defined, and anomalies are easily spotted. For example, a roller press producing 0.5 mm thick sheets of metal with a tolerance of 0.01 mm is easily monitored for anomalies. Any sheet that is thinner than 0.49 mm or thicker than 0.51 mm is considered anomalous. This abnormality would indicate a possible malfunction and would be reported to the operator.

In computer security usual actions of a legitimate user are the normal data source. The abnormal data are then generated by the attacker or accidental misuse by a normal user. Concrete properties of the data source's normal behavior are unknown and need to be estimated or learned from the data. A **model** of normal behavior needs to be crafted to serve as a baseline for anomaly detection. Examples of such models can be found in the next section. The quality of the model defines the relevance of the security events forwarded to the threat detection engine. The problem of defining normal behavior is difficult. Many different algorithms exist for normal model creation, such as histogram-based outlier score models [8], multi-variate statistical analysis [33], and others.

This thesis uses statistical unsupervised anomaly detection methods. A data model is learned from the observed data points without any prior knowledge of the data point distribution. A data point is considered anomalous if it appears out of the distribution created by most data points. The tricky part of this approach is selecting the features to be modeled and the entity to be modeled. The creation of multiple statistical models can solve this problem, by each model focusing on a specific feature and tracking different entities. Thus, the chance of some significant change in the data structure slipping unnoticed is decreased. Ideally, the amount of false positives needs to be as low as possible, while maintaining zero false negatives.

3.2 Behavioral Modeling

The proposed solution for UEBA framework uses multiple standalone anomaly detection models of different types to model the behavior of entities in the network. Each model can target a specific technique, tactic or procedure used in one or more phases of malware kill-chain [12]. The goal is to create comprehensive security events from different parts of the network. These events are later presented to the administrator preserving the context of a potential threat. The contextual information is crucial for the decision making of the administrator.

A few examples of anomalies are presented in Table 3.1. Security events, generated by models in this layer can be leveraged to create comprehensive behavioral signatures. There are many legitimate tools, that are used for both legitimate and malicious purpose. An example of such a tool is `nmap`[4]. However its simplicity also makes it an excellent tool for malicious purposes. Classic signature-based IDS would not flag `nmap` as malicious. In the proposed framework, low severity events are created to be later automatically processed to serve as evidence in threat analysis or discarded.

Investigating the anomalies from Table 3.1 separately does not give enough information to determine whether the anomaly needs to be considered a threat. The administrator would probably discard them as too weak and not actionable. However, their combination may create a very strong evidence of a device being compromised and tell the story how it happened. In the example case, if the events were observed on one device, the story could be the following. A signature-based detection was triggered by a `nmap` scan. A statistical model detected unusually high amount of internal hosts contacted. These two events might look suspicious, but they could easily be a part of a regular network administrator workflow. With the information added by a rule-based system, it is possible, that the scan was run by a PowerShell parented by Microsoft Word[5]. It is obvious, that the whole incident should be inspected further. The administrator should check, whether the host that executed `nmap` is not infected. The ability to combine the information is enabled by properly merging the data from different sources, as explained in Section 2.5.

The implemented event generation layer of the framework creates security events from several different anomaly detection models covered below. The input data is merged under a unique device identifier, which enables creating one complete story for the security analyst to support his decision making. Creating alerts, by combining the security events is beyond the scope of this thesis. However, to showcase the framework's value, a rule-mining algorithm was used to combine the security events. The goal was to find new rules specific

[4] a network scanner used for device discovery on the network (`https://nmap.org/`)

[5] a word processor developed by Microsoft (`https://www.microsoft.com/en-us/microsoft-365/word`)

Detection type	Example
statistics-based	unusually high amount of internal hosts contacted
signature-based	suspicious file hash
rule-based	opening a document spawned a PowerShell process

Table 3.1: Taxonomy of used anomaly detection methods with examples.

for predefined malware classes (e.g. dropper[6]). This experiment serving as a proof of concept for the whole framework is demonstrated in Section 4.2. The other option would be to feed a SIEM with the events from different sources and let the analyst to write the "rules" themselves.

One of the framework's main advantages is that it can be easily extended with new models. Newly created detection methods can be added and complement old ones instead of rewriting the whole framework. Moreover, the framework can consume events from existing systems to use all defense measures in the network to the fullest extent. This thesis divides the models into 2 main categories, **stateless models** and **stateful models**.

Stateless models are used where the anomaly can be determined within the context of the actual batch of data. They are often used for pattern-based models that filter the telemetry without the need for memorizing historical data. Stateful models are mainly used for statistical anomaly detection. They have to learn from the historical data before they start to generate anomaly events. These types of models are discussed in further detail below.

3.2.1 Stateless models

Stateless models are best described as a sophisticated signature-based or rule-based filter specifically crafted by a security expert. Each batch of telemetry is processed independently with no information stored in the model. The filter does not have to be as strict as a concrete signature of an attack. Instead, it should try to capture a behavior that is common for some type of malware. This behavior can be captured as a pattern to be matched (e.g. suspicious domain name generated by domain generation algorithm) or as a sequence of behavioral events (e.g. an Adobe Acrobat[7] application parenting a PowerShell[8] process).

An example of a stateless model is the registry change detection model. Commonly, malware uses Windows Registry to establish persistence on the device. However, installing legit programs also creates registry modifications.

[6] a software used to install malware on a device while avoiding detection by antivirus.

[7] a popular PDF document viewer (`https://acrobat.adobe.com/us/en/`)

[8] a command-line shell and scripting language developed by Microsoft (`https://docs.microsoft.com/en-us/powershell/`)

This model creates an event if a modification is made to one of the watched registries (e.g. `HKCU\Software\Microsoft\Windows\CurrentVersion\Run*`).

The advantage of stateless models looking for behavioral patterns is the simplicity of deployment. Once the hard part of finding the pattern common for the malware's behavior is finished, it literally takes minutes to deploy a stateless model to the framework. The downside is that these models can provide many false positives (in case the pattern is too broad) or no findings at all (if the behavioral pattern is too strict). However, security events from stateless models are vital for threat discovery in the network, as they help with explaining the story behind the breach.

3.2.2 Stateful Models

The idea behind stateful models is to create a baseline behavior of the modeled entity and then compare newly observed behavior to this baseline. From the standpoint of anomaly detection (Section 3.1), the baseline is learned from the normal data source. If the observed behavior suddenly changes, an anomaly is reported together with the change of behavior that caused it. The change in behavior can serve as crucial information for the person or software determining the severity of a possible incident.

The operation of a stateful model consists of two phases: warm-up phase and detection phase. During the **warm-up phase** the model learns the baseline of the behavior of the entity (e.g. device). The warm-up phase needs to be long enough to establish a proper baseline. Once the warm-up is over, the **anomaly-detection phase** starts. Incoming traffic is compared with the baseline to find abnormal behavior and create security events. It is expected that the behavior of the device changes over time because of legitimate reasons, e.g. new software is installed or the device's user starts to work from home more often. To take these changes into account, stateful models use a sliding window to adjust the baseline behavior to actual state of the network. The size of the sliding window is set by the **forget period** parameter. Any behavior older than the forget period is discarded from the model's baseline. Therefore, the forget period needs to be long enough to smooth out expected cyclic changes in the behavior of the network, like day and night, or work day and weekend.

The basic assumption is that the baseline behavior is the expected behavior of the device. In case of the device being infected prior to the deployment of a stateful model, the security events will not be generated, even though there is malicious behavior going on.

A stateful model's operation is demonstrated by the implementation of a histogram-based outlier score (HBOS) model [8] with updated surprisal score from [9]. This model learns a feature distribution over time and then calculates the **anomaly score** of all incoming values. If the observed value is beyond a probability threshold, it is considered anomalous. The internal state of

this model is kept in the histogram keeping track of the current distribution. The histogram is updated with each incoming batch of data. A normalised surprisal [9] is used to compute the anomaly score S for each bin after the update:

$$S(x) = \max\{0, -\log_2(p(x)) - H(B)\}, \tag{3.1}$$

where $p(x)$ is the probability of value x calculated from the current histogram, H is Shannon's entropy and B is the set of all bins. From the current state an **anomaly threshold** is calculated to determine which bins are considered anomalous usually by a probability of $p = 0.05$ or less. For each bin whose anomaly score exceeds the anomaly threshold, a security event is created.

To make sure that the histogram is populated, a minimal amount of records in the histogram can be set to create anomaly events. This prevents models with a low number of records from creating events for every single observation.

The HBOS models (and other stateful models) can model different entities. In this thesis, the models run on company and on device level. The **per company** mode creates a baseline from all traffic observed on the company network. A small change on the network can be lost in the company view due to noisiness of the baseline. To mitigate this problem, **per user** mode is also used, where a stateful model is maintained for every single entity/device. These individual baselines track behavior specific to the device. The downside is, that the device needs to have enough traffic to populate the baseline. Ideally, similarly behaving devices could be tracked in groups, creating baselines specific to the groups behavior. To enable such clustering is one of the goals of embedding creation covered in Section 3.3. An example of per user HBOS model histogram with destination ports contacted by a single device in one day is depicted in Figure 3.1. The model is looking for out-of-distribution ports which in the case of the depicted device would be ports 1900 and 123.

To underline the power of modeling different entities, two different scenarios are presented below. A newly accessed country (e.g. Kyrgyzstan) in a small US-based company could be a reason to create an anomaly on the company level and not on the device level. Especially when the device communicates to Kyrgyzstan, it does not have enough telemetry to populate the country code HBOS model because it mainly communicates to the internal network. A counterexample to showcase the advantage of using the per-user HBOS model would be a device (e.g. proxy server) suddenly communicating on the 445 via the Server Message Block (SMB) protocol port. This change would easily get lost in the per company model as SMB is a frequently used protocol and therefore would not be considered anomalous. From the perspective of this particular proxy server, it is an anomaly that needs to be logged.

More stateful models can be added to the framework (e.g. models based on cumulative sum [34], σ-distance models similar to the ones used in [5]). They

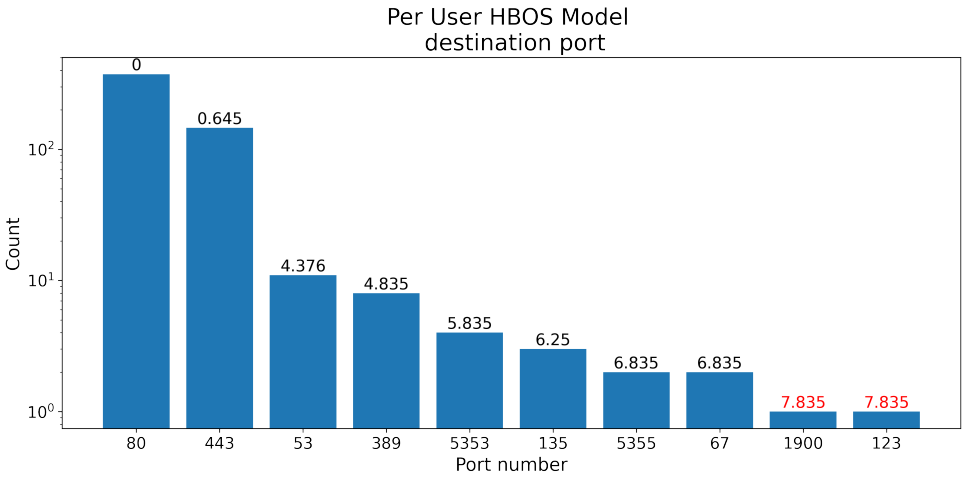


Figure 3.1: Distribution of registered ports contacted by a single device in one day. Anomaly scores are listed above bars corresponding to different port numbers. The red bars exceed the $p = 0.05$ anomaly threshold. This type of model is more sensitive to small changes on the device level that could be missed on the company level.

calculate the standard deviation from normal distribution on given feature and when a change bigger than several standard deviation occurs, a security event is created. These models are used to detect bursts on different features (e.g. sudden burst on a specific port number). The σ-distance models were implemented by another developer and successfully used in the PoC framework.

3.3 Entity Embedding

An **entity embedding** is a vector representation of the entity in a latent space. A **latent space** is a vector space defined by the dimension of the embeddings. Ideally, the data points representing the entities in a latent space are similar to each other, if the entities behave similarly. Embeddings that preserve the similarities of entities in the latent space bring new possibilities to work with the data.

The devices cannot be represented by vectors in the raw feature space as it is not a metric space. Therefore, the devices can be represented only as sets of feature values. Only pairwise set similarities can be used for working working with these sets, which is computationally expensive. The usage of the embedding transformation to the latent space overcomes the heterogeneity of the raw feature space. The vector representations in the metric latent space can leverage the properties of metric spaces.

The entities change dynamically, therefore their embedding also changes and corresponding data points move in the latent space. A stateful model can track these changes in time by comparing the embedding vectors created for individual entities in time. If a significant shift occurs in the latent space, an anomaly is reported. The embeddings can be used to find groups of similarly behaving entities. Finding that an entity has changed its group can be treated as an anomaly. Another possibility is to track the entity in time without need of a unique identifier. The behavioral fingerprint could serve as the identifier. This would enable, for example, identifying the device even after the change of device identifier (e.g. due to reinstalling the endpoint client). Several of these possibilities are explored in Chapter 6.

The creation of entity embedding is a transformation $e\colon X \to L$ from heterogeneous non-metric feature space X to a latent space L where each entity is represented by a vector. The goal is to create behavioral fingerprint of the entity. The embedding should be comparable to other entities using a similarity metric, while retaining the structural information of the network. There are two requirements for the embeddings:

- **Requirement 1**: Embeddings of an entity in two subsequent time periods need to be similar to each other. This can be expressed as following formula for average self-similarity:

$$r_1 = \frac{1}{N}\sum_{i}^{N} \mathrm{sim}\left(e(x_{it_1}), e(x_{it_2})\right), \tag{3.2}$$

  where N is the number of entities in the set, sim is an arbitrary similarity function, e is the embedding transformation, $x_i \in X$ is the entity representation in feature space and t_1 and t_2 are the subsequent time periods.

- **Requirement 2**: Different entities need to be dissimilar and distinguishable by their embeddings. This can be expressed as the following formula for average dissimilarity between different entities:

$$r_2 = \frac{2}{N(N-1)}\sum_{i}^{N}\sum_{j}^{i-1}\left(1 - \mathrm{sim}(e(x_{it}), e(x_{jt}))\right) \tag{3.3}$$

  where N is the number of entities in the set, sim is an arbitrary similarity function with a range of [0, 1], e is the embedding transformation and $x_i, x_j \in X$ are different entities.

Using only r_1 would not be enough as a function transforming all entities into one point in the latent space would not bring meaningful results, even though it would fulfill the requirement. Therefore, requirement r_2 is introduced.

With the requirements above, finding the embedding transformation e can be formulated as an optimization of the following function:

$$e^* = \arg\max_{e \in \mathcal{E}} (w_1 r_1 + w_2 r_2), \tag{3.4}$$

where $\mathcal{E}$ is the set of all embedding transformations e, w_1 and w_2 are weights assigned to the requirements r_1 and r_2. The embedding function e should also regularize the vectors. The experiments in Chapter 6 were evaluated by metrics corresponding with the requirements above.

3.3.1 Bag-of-Words Representation

This thesis uses a **bag-of-words (BoW)** representation for proof-of-concept of device embedding. BoW is a information retrieval technique originating in document classification. It is used to represent a document in a vector space by computing the number of term occurrences in the document and discarding the structure of the document. A term is usually a word or n-gram. The dimension of the representation space is determined by the number of unique terms (called vocabulary) in the set of documents that are being compared.

Having network flows and endpoint logs at disposal, the bag is constructed from all values (terms) observed in one feature in a given time window, i.e. all executable hashes used by a device in one day (treated as a"document".) would be added to the bag. The vocabulary would then be all the hashes used by the devices in the network in an extended time window (e.g. day, week).

Using only counts of occurrences in individual features does not work well for many of the features as usually few values occur significantly more often than others. Thus, all vectors look similar, because of this frequent feature. In the case of executable hashes, this could be Google Chrome, as it is the most common browser. Therefore, tf-idf (term frequency - inverse document frequency) [35] is used to weight the vector by the amount of information each term brings. If the term is very common, the idf value is small, reducing the impact of the term in the resulting vector.

The frequent features can have several orders of magnitude more occurrences than the other features. In the document classification problems, the most frequented words (is, are, with, the, a, an etc.) can simply be removed from the vocabulary. Such is not the case in the network and endpoint telemetry as the most frequented terms can change (e.g. update of a program changes the executable's file hash). Or they can contain information useful for entity identification (e.g. the most frequented autonomous systems (AS) contacted by Windows machines are maintained by Microsoft, distinguishing them from Linux machines).

According to our experiment even using tf-idf to re-weight features is not enough. Therefore, a different approach needs to be used to re-weight the

feature in the combined telemetry. This thesis uses time window BoW (tw-BoW) representation, where each feature is counted only once for each time window it occurred in. This creates a constraint on maximal value of each component of the vector (e.g. 288 for a representation of one day split into 5 minute windows). This vector is then re-weighted by tf-idf.

Chapter 6 covers the experiments comparing BoW and tw-BoW representations.

Chapter 4

Threat Detection Layer

Security events created by the event generation layer are forwarded to the threat detection layer. The goal of threat detection is to combine and filter the security events to find actual threats and create an alert for the security operations center (SOC). A SOC is a specialized workplace where dedicated employees use technical security solutions to monitor the security of a company.

This chapter briefly covers two approaches that can be used in a threat detection layer. One is combining the data from a security information and event management system (SIEM) via handcrafted rules and the other is to automate the rule creation process by an algorithm.

4.1 Security Information and Event Management System

Security information and event management systems such as Splunk [36] or Solarwinds Security Event Manager [37], are a system where all the security events, anomaly detections, antivirus and other logs are normalized into a common representation. The events should also contain information about the current state of the network. The system gives the SOC the ability to monitor and analyze events transpiring in the network in real-time [38]. Based on the findings, the SOC is able to write rules targeting specific threats. Stored events also helps with post hoc investigation. SIEMs are the state-of-the-art solution used by security analysts for tracking and responding to potential threats in the network.

Originally, SIEMs started as an enhanced logging tool to combine logs from different security tools running in the network. Over the years the ability to manually create detection rules was added, so that security analysts could create alerts fulfilling their needs. Nowadays some SIEMs are able to automate the process of rule creation. The automatic approach for finding rules is

covered in the next section.

4.2 Rule Mining

To showcase the new capabilities opened by using combined telemetry for anomaly detection a rule mining approach was used. Security events were forwarded to the rule mining algorithm implemented as part of another project. The rule mining algorithm is designed to help threat analysts to discover new behavioral rules for known threats. The rules are mined for malware families, groups of similarly behaving malware. An example of such a malware family could be ad injector or dropper. Ad injector is software used for injecting adds to web pages via browser. Injected ads are not monetized by the official owner of the website. Instead, they are sold by the malicious actors controlling the ad injector. A dropper is the software responsible for the delivery phase of malware kill-chain.

Frequent item set mining algorithm FP-growth algorithm [39] with slight modifications was used to look for behavioral patterns used by specific malware families. The algorithm leverages frequency based prefix tree to discover frequent security event sets that lead to a confirmed detection. In case of previous example, the item sets found are behavioral patterns of the malware family. Found patterns are inspected by a security analyst and can be used as new signatures for malware detection. This approach was not implemented as part of this thesis, however the security events created by event generation layer were used as an input to the existing solution to see whether such patterns can be found in combined telemetry.

Telemetry Source	**Event Description**
Network	unusual user-agent for given domain
Endpoint	unusual file hash
Network	connection check
Endpoint	device fingerprinting
Network	persistent communication with unusual domain
Network	unusual user-agent

Table 4.1: Example of a rule, mined for the dropper malware type. Two of the behavioral security events come from endpoint telemetry and the rest is coming from network telemetry.

Table 4.1 shows an example of one such rule, mined for the dropper malware type. The rule was mined as a set but an actual threat was confirmed with the listed succession of events. First a known domain was accessed with an user-agent, that was not previously used to contact this domain. Then an unusual file hash was flagged by endpoint client on the device. Later it

was confirmed, that `curl`[9] was used to download the file. Security events detecting connection checks and collection of information about the device started to appear. A persistent communication with unusual domain, again using unusual user-agent were also observed. After deeper inspection, it was discovered that events were caused by the dropper Shlayer [40], that was not found by other means.

The tests with mining rules from security events showed, that not all rules are meaningful. Therefore, the rules are currently used in the human-in-the-loop scenario. A threat analyst checks selected rules for validity before deploying them to the threat detection layer. The rule mining algorithm helps the threat analyst to automatically discovering new rules, that would either take a significantly higher time to write or would slip by unnoticed.

[9] An open source software used to transfer files over the internet

Chapter 5

Implementation of the Event Generation Pipeline

This chapter covers the architecture and implementation of the framework, which is designed to consume data from any source covered in Chapter 2. Implemented pipeline consumes network flows, proxy logs and endpoint logs merged by source IP address under the endpoint client installation ID. The event generation layer is responsible for anomaly detection on the combined telemetry and produces security events consumed by the threat detection layer (e.g. by the rule-mining algorithm) and also stored for further reference. One of the main contributions of this thesis is the implementation of the event generation layer pipeline.

In a production deployment the framework would run in a batch streaming mode. One batch consists of real-time telemetry collected for a small time window. To mimic batch streaming, the proof-of-concept pipeline runs on persistently stored historic data. Batches are loaded and processed in 5-minute time windows as they would arrive in production deployment.

5.1 Architecture

The framework uses pipeline architecture consisting of three independent layers: Combining Data and Enrichment layer, Event Generation layer and Threat Detection layer. A diagram of architecture can be found in Figure 5.1. The first layer merges raw data from various sources and converts them into a normalized format. The combined telemetry is then saved to cloud storage in a columnar format partitioned by time. This partitioning provides the possibility of loading batches for specified time windows efficiently, to mimic the batch streaming mode expected of the deployment version of the pipeline. The batches with combined telemetry are then forwarded to anomaly detection models running in distributed environment. Every model then creates

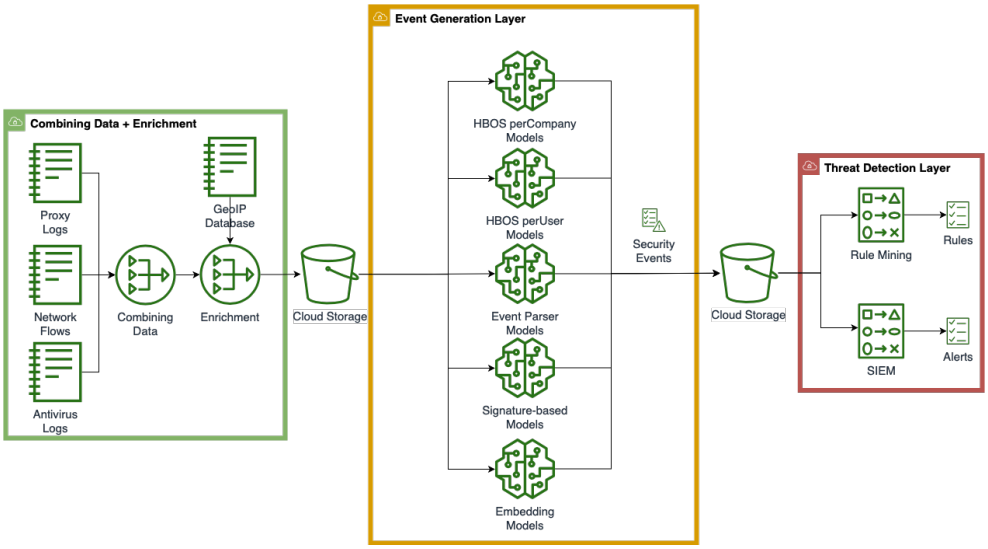


Figure 5.1: Overview of whole framework using multiple modalities. Data from separate telemetry sources (the leftmost green box) is merged and stored in cloud. Event generation layer (the middle yellow box) loads the stored telemetry and feeds it to individual anomaly detection models. Security events are then stored in cloud. Threat detection layer (the rightmost red box) uses security events to generate alerts.

security events of specific type. The events are again stored in cloud where they can be accessed by the threat detection layer.

The independent deployment of each layer in a distributed environment is important for such a framework. It offers the possibility of easily reusing the layers in different networks and with different sources of telemetry, while retaining the format of security events used for threat detection. The PoC framework was successfully used for security event creation on telemetry from more than 25 real private networks. University, transportation, financial and other industrial networks were present in the dataset. The number of devices present in the networks ranged from few hundred to hundreds of thousands.

5.1.1 Combining Data and Enrichment

First, raw network flows, proxy logs and endpoint logs need to be loaded and transformed to a normalized format that the models in the event generation layer are able to consume. In the implemented PoC version of the pipeline, the individual telemetries are loaded from cloud storage, yet in the production the input will be in the form of real-time data stream. The endpoint client installation ID from the endpoint logs is used to uniquely identify devices on the network. The rest of the thesis uses the term device ID for this unique

identifier. Different telemetry sources are then merged using the source IP address.

Source IP address was chosen for merging, because it is the only feature available in all telemetry sources. However, many of the endpoint logs contain only files executed and modified with no IP address, which makes the merging process complicated. With the network dynamically changing, certainty that an IP address is assigned to a specific device ID decreases rapidly from the time a network communication was logged on the endpoint.

When a network connection is logged, it is certain that the device ID has the source IP address used for the connection. If a device with ID A communicated from an IP $a.b.c.d$ in time t and in time $t + \Delta$ a flow record was observed with source IP $a.b.c.d$, it is **not** certain that the communication originated from device A. The main cause of this problem is that IP address is not an unique identifier. It changes in time because of DHCP leases, the device reconnecting on different network (e.g. using VPN) or other reasons.

As explained above, the certainty of the *(device ID, device IP)* pair being valid is highest right after the observation of both identifiers in one endpoint network communication log. The algorithm used considers the *(device ID, device IP)* pair to be valid for 1 minute after the observation. When the minute has expired, the algorithm waits for another observation containing both IP address and device ID. Depending on the density of network communication logs, this approach may not be able to match some of the network traffic. The devices not running the endpoint software cannot be merged at all. Unmerged network telemetry can still be used for anomaly detection, but the anomalies found cannot be attributed to a concrete device.

The combined telemetry is stored in a columnar format in a cloud storage. The format chosen to store this data is Apache Parquet [41], as it is versatile and programming language independent. Parquet data format also enables compression and the possibility to load only selected columns or partitions as well as storing arbitrary objects. This is useful in the event generation phase of the pipeline, where each model needs to access only a subset of all the features stored in the parquet.

5.1.2 Security Event Generation

Security events are generated from signature-based and anomaly-based models. A model is an independent module, consuming a subset of combined telemetry needed for detecting suspicious behavior for devices in the network. These models are the core of the framework. Each model uses different features from the combined telemetry. Thus, each model can load only the features it needs from the storage, leveraging the columnar storage format. All models operate in a batch mode, consuming several minutes of telemetry at a time. Each model can use a different sized time window.

The models implemented are histogram-based outlier score (HBOS) models [8], signature-based models and event parser models. HBOS models and signature-based models are explained in detail in Section 3.2. Event parser model is a simple model used to translate existing detections from other security systems running in the network to the security event format used further in the pipeline. A security event is a timestamped tuple, identified by the device ID and flagged with an *event_type* specific for the model type. The events have the following format: *timestamp, device_id, event_id, count, event_type*. The individual field descriptions are below:

- **timestamp:** timestamp of the end of batch in which the event occurred
- **device_id:** unique device identifier taken from endpoint telemetry (the installation ID of an endpoint client)
- **event_data:** subject of the event, for ports it would be the port number, for hashes it would be a specific hash, etc.
- **count:** number of occurrences in given batch, used to group the same events together
- **event_code:** code identifying the anomaly detection model that produced the event

Parameters of each model need to be adjusted to improve the quality of detections and to regulate the number of events generated by each model. They are tuned with the same objective as described in Section 3.1: To keep the number of false positives low, and ideally to produce zero false positives. The pattern-matching models are adjusted by the patterns themselves. If the pattern is too general the output gets cluttered by false positive events. Coming back to the example of change in Windows Registry signature-based detector a regular expression pattern `HKCU\*` would generate many more events than a specific `HKCU\Software\Microsoft\Windows\CurrentVersion\Run*` pattern. The latter is commonly used for setting up persistence on the system, as software listed there is started on user logon.

Adjusting parameters of the HBOS models is more complex. Warm-up period, forget period, and the minimal population of models need to be set as well as the anomaly threshold used to generate events in populated histograms. Most of the detectors use anomaly threshold of 0.99, which means that only 1% of the most anomalous observations is reported as security events. In the data used, this corresponds to 5-10% of the devices in the network generating events. The minimal population of the histogram needed to start generating events was set to 50 observations per forget period. The warm-up period for the models is set to 24 hours, to capture the behavior for whole day before the model starts generating events. To ensure smooth transitioning between days and to mitigate small perturbances, the forget period is set to 48 hours.

`StatefullModel` and `StatelessModel` abstract classes provide an interface that makes the implementation of completely new detection model families simple. The only requirement is that they consume combined telemetry and generate security events in the required format.

5.2 Technology Used

This section lists the technology used for developing the prototype version of the pipeline. The programming language of choice was Python (version 3.85) because of its flexibility and many useful data science libraries. The data format and libraries used in the framework were picked according to the expected rewriting of the prototype version into production code using Apache Spark [42] with Java or Scala programming languages. Apache Parquet [41] was chosen as the storage format and the library used for distributed computing is Dask [43]. Dask is an open source library able to use common data science libraries like Pandas [44] and NumPy [45] (both used in the implementation) in a distributed environment natively, while providing similar features as Apache Spark.

The framework was designed to run both manually (either in a Jupyter notebook [46] or as an application) and automatically using Apache Airflow [47] as a workflow automation library. Airflow enables easy deployment by running Docker[10] containers on Kubernetes[11] clusters for each of the anomaly detection models used by the framework.

[10] a service used for virtualization on the OS level via containers (`https://www.docker.com/`)

[11] a system for automated deployment of containerized applications (`https://kubernetes.io/`)

Chapter 6

Entity Embedding Experiment

This chapter covers experiments with different entity embeddings, that can be used to extend the basic pipeline covered in Chapter 5. The main goal was to compare different embeddings for tracking entities in the network over time. For the purpose of these experiments, a device was selected as the represented entity. It was an obvious choice (from the data available), as devices are directly identifiable by the endpoint client installation ID.

The problem lies in the fact, that not all devices on the network have an identifier assigned. IoT devices or personal appliance cannot or do not have an endpoint client installed. The ultimate goal of embedding creation would be the creation of a behavioral embedding that could serve as unique identifier for all devices on the network. This ideal embedding would create a unique fingerprint for each device, that would enable differentiation between them based solely on their behavior. The experiments performed as part of this thesis are the first step in this direction. The endpoint client installation IDs were used as labels for the purpose of device tracking, thus omitting the devices that produced only network telemetry.

Two possible embedding creation approaches based on the bag-of-words (BoW) method explained in Section 3.3.1 are compared in the experiments. The first approach is BoW representation using raw features weighted by tf-idf. In classic BoW, feature frequencies are computed from all term occurrences (e.g. for ports, each access on destination port 80 counts as one occurrence). tf-idf is later used to re-weight each feature according to the frequencies observed in the network.

The second approach is the **time window BoW** representation (again weighted by tf-idf), where each feature counts only once for each time window it occurred in (e.g. for ports, no matter how many times in a time window device accessed port 80, it counts as only one occurrence). For this experiment we used 5-minute time windows, therefore each vector component ranges from 0 to 288 (number of 5 minute windows in 24 hours). Smoothing the vector by this method enables less significant values for given feature to have bigger im-

pact on the final vector. Otherwise the most frequent features could outrange others even after tf-idf smoothing.

6.1 Embedding Evaluation

The embeddings were evaluated according to their ability to track the device in time in the latent space. This evaluation criterion is formalized by the requirements in Section 3.3. To evaluate the embedding quality, similarities between all devices appearing in one day, and all devices appearing in the other day were computed. Total of $N * M$ similarities were computed for every two of days, where N is the number of devices in the first day and M is the number of devices in the second day. The M similarities in each row i of the matrix were ranked according to:

$$\text{rank}_i = 1 + |\{j|\text{sim}(e(x_{it_1}), e(x_{jt_2})) > \text{sim}(e(x_{it_1}), e(x_{it_2})\}|, \tag{6.1}$$

where sim is a similarity measure, $e(x_{it_1}), e(x_{jt_2}) \in L$ are embedding of different devices in the latent space L and times t_1, t_2 are two consecutive days.

The following metrics were used to compare the quality of the embeddings:

- **Mean rank R** in which each device embedding appeared in the second day:

$$R = \frac{\sum_i^M \text{rank}_i}{M}, \tag{6.2}$$

  where M is the number of devices in the second day and rank_i is the rank from Equation 6.1. The lower mean rank, the better the representation. In the best case the mean rank would be 1, allowing to precisely track all devices over time.

- **Percentage of precise hits A** is the defined as:

$$A = \frac{\sum_i^M \mathbb{I}[\text{rank}_i = 1]}{M} \tag{6.3}$$

  where M is the number of devices in the second day, $\mathbb{I}$ is the indicator function which is 1 if the rank is equal to 1 and zero otherwise and rank_i is the rank from Equation 6.1.

  The value of A shows the portion of devices that could be uniquely identified. If there are multiple devices tied with the same highest similarity, it does not count as precise hit, because the device cannot be identified uniquely (cf. Requirement 2 in Section 3.3).

- **Cumulative distribution function** (CDF) of the device appearing on rank N or lower for each device:

$$f(x) = P(\text{rank}_i \leq x) \tag{6.4}$$

  where the right-hand side represents the probability of randomly selected device x having higher rank than rank_i. This metric is useful for comparison between different embedding transformations.

These metrics enable comparison of both BoW and time window BoW approaches as well as comparing the different features.

6.2 Experimental Setup

The experiments use three different features, listed in Table 6.1, to test the viability of the BoW approach. Private destination IPs are addresses falling into ranges defined in RFC1918 [48]. This feature comes from the endpoint telemetry due to the fact, that network telemetry available in the dataset was gathered on the edge of the network. That is why endpoint logs were used as a source for private IP address. Private IP address was chosen because it is one of the few that captures behavior on the internal network.

The file hash is a string uniquely identifying a file. Files that were created, opened or executed on the endpoint are supplied to a hashing function to compute the hash. In endpoint security, the hashes are queried against a database of known files to check their legitimacy. The telemetry used in this thesis come from logs of these queries.

Autonomous system number comes from enriching the network telemetry with information from a GeoIP database. It is expected that a single network in one location will mostly communicate with several autonomous systems. The dominant autonomous systems will be similar for each device in the network. The experiment is designed to test whether the remaining less frequent ASNs can serve to distinguish devices.

Feature	**Description**	**Telemetry**
Private destination IP	Private range IP address	Endpoint
File Hash	Hash of the inspected file	Endpoint
Autonomus System Number	ASN of the destination IP	Network

Table 6.1: Features tested for device embeddings.

The experiments were performed on a week of real telemetry (Jan 11 to Jan 18, 2021) from a corporate network. Table 6.2 shows the numbers of devices present in the network for each day in the week. The difference between numbers of devices seen at the endpoint and in the network is mainly

caused by endpoint devices that communicate only within the private network. Therefore, the traffic wasn't observed on a proxy.

During weekdays, the number of devices changed by several percent as some of the devices present one day did not appear in the other. This can be due to many reasons. For example the device was not powered on during that day. Or, the device was a virtual machine instance and each new instance receives new endpoint client installation ID. During the weekend, a significant drop in device number is observed because less employees are at work.

Date	**Number of Devices Endpoint Telemetry**	**Number of Devices Network Telemetry**
2021-01-11	1764	1057
2021-01-12	1802	1065
2021-01-13	1796	1087
2021-01-14	1790	1068
2021-01-15	1754	1056
2021-01-16	1667	979
2021-01-17	1154	714
2021-01-18	1801	1064

Table 6.2: Number of devices observed in different telemetries on the network

The embeddings were created for one feature at a time using the BoW and time window BoW approach covered in Section 3.3.1. One day period was selected to create an embedding, with the assumption that it contains most of the regular behavioral routines of the device and is small enough to detect the behavioral change as soon as possible. The dimensions of the embedding spaces (defined by the vocabulary size) changed between different days. They were ~ 3500 for `dstIpPrivate`, and ~ 12000 for `fileHash`, and ~ 850 for `autonomusSystemNumber` changing slightly every day.

To test the device tracking in time, two embeddings (one for each day) were created for every two consecutive days. The vocabulary used for embedding creation contains all terms (observed feature values) that occurred during these two days. A days embedding was created for each device by counting feature occurences and re-weighting the resulting vector by tf-idf (using scikit-learn [49] library). Cosine similarity between embeddings from consecutive days was used to evaluate the quality of embeddings. This process was repeated for each day.

6.3 Results

Figure 6.1 shows similarity distributions to self and the most similar device using the available features independently. The plotted histograms represent

similarities of device embeddings from Monday January 11 and Tuesday January 12, for other days the distributions are similar. The orange color depicts the similarity to self in the second day. Better representation of devices can be seen from the amount of orange bars that are higher than the blue ones. By looking at the plots in Figure 6.1, several observations can be made:

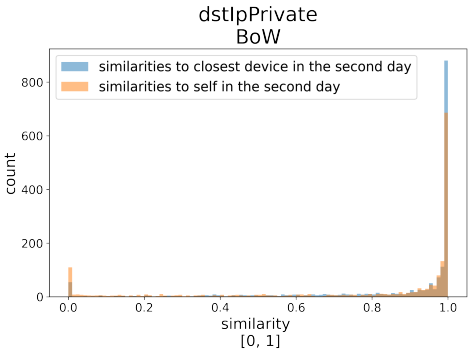
- Time window BoW representation performs better, with more devices being most similar to themselves rather than to another device.
- File hashes has the highest number of devices with high self-similarity followed by autonomous system numbers and destination IP address.
- Private destination IP address has a number of devices with similarity 0 to itself, indicating, that some devices cannot be tracked by this feature at all.

To compare the ability to track devices in time, cumulative distribution functions from Equation 6.1 are plotted in Figure 6.2a. Bigger area under the CDF indicates better representation. Time window BoW representations outperform the raw BoW approach for all the tested features. File hash shows the best results, with 75% of devices being in top ten ranks in the second day embeddings. Autonomous system number performs the worst of the three features. This indicates, that infrequently accessed autonomous systems are not enough to differentiate between devices. Private destination IP address slightly outperforms the ASN feature. After inspecting the data, most of the network communications were to several addresses that belong to load-balanced servers.

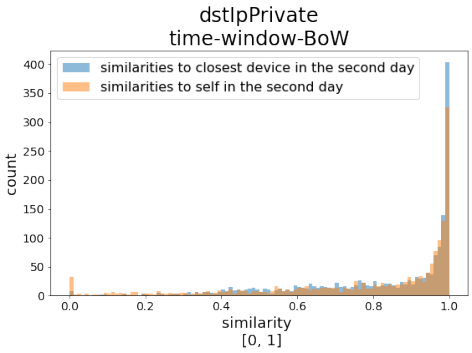
Complete results averaged over the whole week are listed in Table 6.3. Best values for each category are highlighted in bold. In all three tested features the time window BoW representation has shown better results and file hash proved to be the best feature for device tracking.

Feature	**Embedding**	**Mean Rank**	**Precise Hit Ratio**
dstIp Private	BoW	142.55 ± 21.96	0.27 ± 0.06
	tw-BoW	86.16 ± 15.20	0.34 ± 0.07
file hash	BoW	55.82 ± 10.98	0.35 ± 0.06
	tw-BoW	**25.65 ± 4.99**	**0.47 ± 0.10**
ASN	BoW	119.71 ± 21.49	0.22 ± 0.01
	tw-BoW	49.30 ± 10.20	0.42 ± 0.05

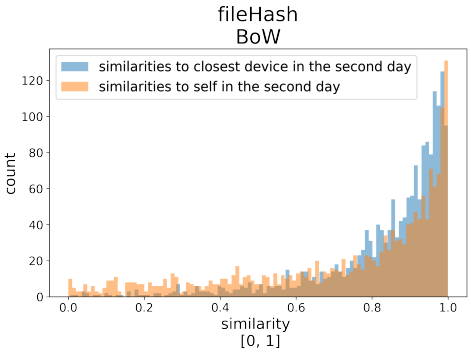
Table 6.3: Comparison of mean rank and mean similarity of BoW and time window BoW embedding creation methods. Mean rank shows the efficiency of tracking individual users (the lower the better). Precise hits ratio shows the percentage of devices that were tracked accurately over time.



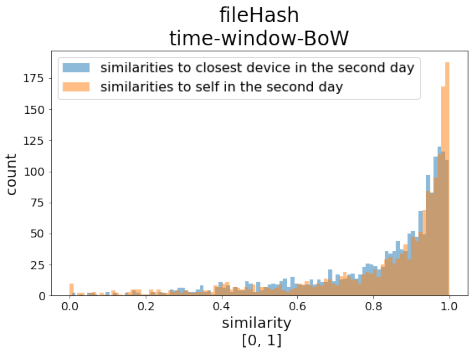
(a) dstIpPrivate: BoW



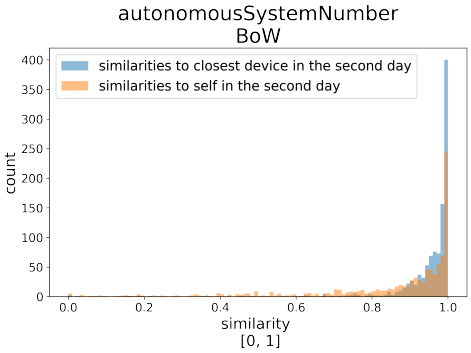
(b) dstIpPrivate: tw-BoW



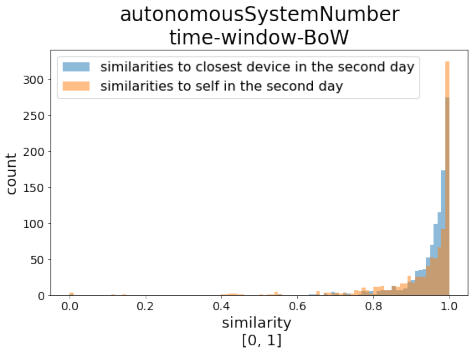
(c) fileHash: BoW



(d) fileHash: tw-BoW

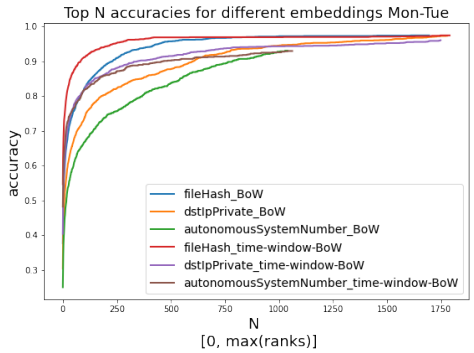


(e) autonomousSystemNumber: BoW

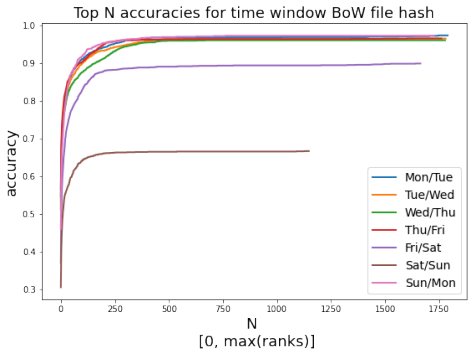


(f) autonomousSystemNumber: tw-BoW

Figure 6.1: Histograms of similarity distributions for device embeddings between Monday January 11 and Tuesday January 12, 2021.



(a) Comparison of all embedding representations with the cumulative distribution functions for probability of the device being within the top N ranks of similarities the next day. This graph was created for tracking between Monday Jan 11 and Tuesday Jan 12, 2021.



(b) CDF for the file hash feature in different days of the week. The quality of embeddings deteriorates significantly over the weekend as the number of devices drops significantly and their behavior changes due to employees of the company being at home.

Figure 6.2: Comparing CDFs of devices appearing on N-th rank between different features (a) and different days of the week (b).

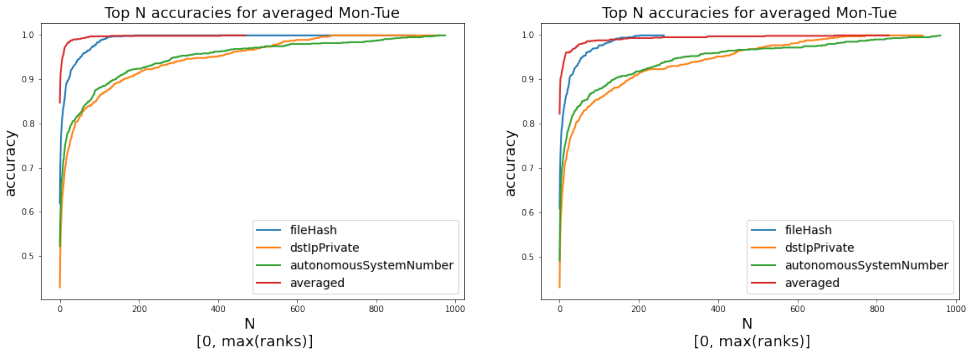
Figure 6.2b shows how the CDFs change over the course of the week for file hashes. Device tracking during work days is quite stable. During the weekend, the behavior changes and accuracy is significantly lower. The drop from Friday to Saturday is smaller than from Saturday to Sunday. This is probably caused by the time shift as the data is timestamped in GMT but the network is located in GMT-6 timezone. The accuracy is high again from Sunday to Monday because the devices running over the weekend are probably servers. Their behavior is maintained over time. The drop in accuracy over the weekend could be mitigated by tracking devices over regular week days separately from weekends.

The last experiment performed was the use of combined embeddings for device tracking. The already pre-computed embeddings were used to compute similarity matrices to all devices and then the three similarities were averaged:

$$\mathrm{sim}_{avg}(x_i, x_j) = \frac{\mathrm{sim}_{ASN}(x_i, x_j) + \mathrm{sim}_{dstIpPrivate}(x_i, x_j) + \mathrm{sim}_{fileHash}(x_i, x_j)}{3}, \tag{6.5}$$

where sim is the similarity function used with the suffixed feature and x_i, x_j are the devices compared.

Only devices that were present in all three telemetries were used in this experiment, which significantly reduces the number of devices for endpoint. Figure 6.3 shows the CDFs for ranks using common devices. Using the average



(a) CDF with average for subset of 981 devices for Jan 11 and Jan 12, 2021

(b) CDF with average for subset of 988 devices for Jan 12 and Jan 13, 2021

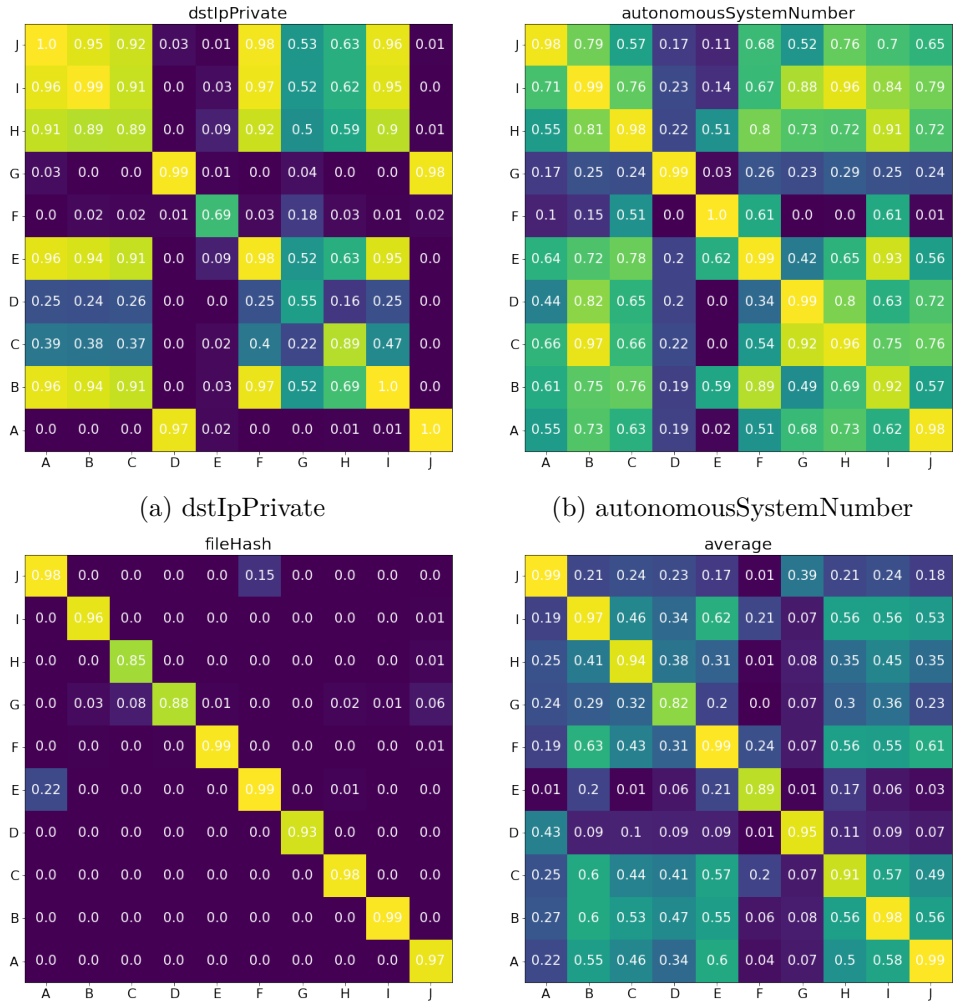
Figure 6.3: Comparing the combined averaged similarity with different features over 2 days.

similarity shows an improvement, increasing the precise hit ratio significantly. Concrete numbers can be found in Table 6.4.

Feature	**Mean Rank**	**Precise Hit Ratio**
average	3.32	0.85
fileHash	9.56	0.62
dstIpPrivate	53.00	0.43
autonomousSystemNumber	50.73	0.50

Table 6.4: Results of using average similarity for computing similarity between a subset devices present in both network and endpoint telemetry between Jan 11 and Jan 12, 2021.

Lastly, to visualize the difference between all approaches, Figure 6.4 shows confusion matrices for 10 randomly selected devices. Lighter tile color means higher similarity. The difference between Figure 6.4c and Figure 6.4d shows that even device I and J, that use similar hashes can be distinguished by using the average similarity. Private destination IP address does not seem to be a good feature for individual user tracking. However, groups of devices behaving very similarly might be harvested from the data. For example, devices B, E, H, I, J from 6.4a are behaving similarly. Looking at the raw data, has revealed that the similarity comes mainly from few common IP addresses. These addresses are LDAP and HTTP servers in the network. However, as these servers use load balancing, the clusters are not stable in time as devices communicate to different IP addresses.



(a) dstIpPrivate

(b) autonomousSystemNumber

(c) fileHash

(d) average

Figure 6.4: Confusion matrices for 10 randomly selected devices common for both telemetries.

6.4 Takeaways

The results above correspond with the expectation, that less frequent values are crucial for definition of behavioral fingerprint. For example, in the case of autonomous system numbers, many devices are expected to frequently access Microsoft's autonomous systems. The difference between users shows up when the vector is limited by using the time-window approach. Then, less frequent ASNs like the ones where the actual users' favourite website is hosted have higher impact on the embedding.

Even though file hashes show promising results for embedding creation, it is not enough to confidently track the device over time. Averaging the similarities from different feature embeddings significantly increased the number of precisely tracked devices. The downside is that both network and endpoint telemetries have to be present. Further research should lead to combining the embedding from different features even if one feature is missing. Also dividing the embeddings to workday and weekend embeddings should be tested.

One of the possible next steps in the direction of behavioral fingerprinting would be creation of clusters of devices, that differentiates between different groups of devices. These groups could be human operated devices, printers, network devices, etc. The clusters can be used for targeted anomaly detection (e.g. looking for a device changing its group, models tailored for a specific device type).

Conclusion

This thesis presented an architecture and a proof of concept implementation of a user and entity anomaly detection framework. Anomaly detection models run in parallel in a distributed environment which makes the whole framework scalable. The framework combines telemetries from multiple data sources and is also open for extension with more telemetry types. Signature-based models are used to detect suspicious patterns in the telemetry. Whereas histogram-based outlier score (HBOS) serves as a behavioral fingerprint of the modeled entity. Furthermore, HBOS models are designed to track the whole network as well as individual devices.

The framework was successfully used to detect anomalies in real telemetry from more than 25 private networks. The number of devices in these networks ranges from hundreds to hundreds of thousands. The dataset spans multiple industries, including academia, finance, transportation, and medicine. The rule mining algorithm confirmed the quality of security events produced by the framework. Leveraging security events created by the UEBA framework, the algorithm was able to find new behavioral rules for known malware families.

Another contribution is the novel research in the direction of device embedding. Time window bag of words embedding transformation in combination with average similarity across different features was able to track 80% devices over multiple days uniquely. From the results, it can be concluded that such embeddings are good enough to be used in the sudden change of behavior detector in the event generation layer of proposed framework.

There is much to be done as future work. The embeddings will be leveraged in stateful anomaly detection models. Also, there are a plethora of additional features that can be used to create similar embeddings. Weighted average and other combinations of embeddings will be tested for device tracking. The direction of clustering on top of the embeddings shows promise for network asset identification.

Bibliography

[1] Internet live stats. `https://www.internetlivestats.com/`. Accessed: 2021-02-15.

[2] Artem Voronkov, Leonardo Horn Iwaya, Leonardo A Martucci, and Stefan Lindskog. Systematic literature review on usability of firewall configuration. *ACM Computing Surveys (CSUR)*, 50(6):1–35, 2017.

[3] The importance of user behavior analytics for cloud service security. `https://www.oracle.com/assets/user-behavior-analytics-3497541.pdf`. Accessed: 2021-03-23.

[4] Gorka Sadowski, Avivah Litan, Toby Bussa, and Tricia Phillips. Market guide for user and entity behavior analytics. 2018.

[5] Madhu Shashanka, Min-Yi Shen, and Jisheng Wang. User and entity behavior analytics for enterprise security. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1867–1874. IEEE, 2016.

[6] Florian Menges, Fabian Böhm, Manfred Vielberth, Alexander Puchta, Benjamin Taubmann, Noëlle Rakotondravony, and Tobias Latzo. Introducing dingfest: An architecture for next generation siem systems. 2018.

[7] Xiangyu Xi, Tong Zhang, Dongdong Du, Guoliang Zhao, Qing Gao, Wen Zhao, and Shikun Zhang. Method and system for detecting anomalous user behaviors: An ensemble approach. In *SEKE*, pages 263–262, 2018.

[8] Markus Goldstein and Andreas Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: Poster and Demo Track*, pages 59–63, 2012.

[9] Martin Kopp, Martin Grill, and Jan Kohout. Community-based anomaly detection. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6. IEEE, 2018.

[10] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. A geometric framework for unsupervised anomaly detection. In *Applications of data mining in computer security*, pages 77–101. Springer, 2002.

[11] Yoohwan Kim, Wing Cheong Lau, Mooi Choo Chuah, and H Jonathan Chao. Packetscore: Statistics-based overload control against distributed denial-of-service attacks. In *IEEE INFOCOM*, volume 4, pages 2594–2604. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), 2004.

[12] Eric M Hutchins, Michael J Cloppert, Rohan M Amin, et al. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1(1):80, 2011.

[13] Mitre att&ck®. `https://attack.mitre.org/`. Accessed: 2021-03-08.

[14] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1):1–22, 2019.

[15] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[16] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[17] Ratinder Kaur and Maninder Singh. A hybrid real-time zero-day attack detection and analysis system. *International Journal of Computer Network and Information Security*, 7(9):19–31, 2015.

[18] Suresh N Chari and Pau-Chen Cheng. Bluebox: A policy-driven, host-based intrusion detection system. *ACM Transactions on Information and System Security (TISSEC)*, 6(2):173–200, 2003.

[19] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954, IETF, October 2004.

[20] B. Claise, B. Trammell, and P. Aitken. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011, IETF, September 2013.

[21] Ming Liu, Zhi Xue, Xianghua Xu, Changmin Zhong, and Jinjun Chen. Host-based intrusion detection system with system calls: Review and future trends. *ACM Computing Surveys (CSUR)*, 51(5):1–36, 2018.

[22] Cisco Talos Intelligence Group. `https://talosintelligence.com/reputation_center`. Accessed: 2021-03-14.

[23] MaxMind. GeoLite2 Free Geolocation Data, 2021. available at: `https://dev.maxmind.com/geoip/geoip2/geolite2/`.

[24] L. Daigle. WHOIS Protocol Specification. RFC 3912, IETF, October 2004.

[25] Cloud user security. `https://www.cisco.com/c/dam/en/us/products/collateral/security/cloudlock/cisco-cloudlock-user-security-datasheet.pdf`. Accessed: 2021-05-03.

[26] User and entity behavior analytics. `https://www.fortinet.com/products/ueba`. Accessed: 2021-05-03.

[27] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. Zero-day malware detection. In *2016 sixth international symposium on embedded computing and system design (ISED)*, pages 171–175. IEEE, 2016.

[28] Charu C Aggarwal. *Outlier analysis second edition*. Springer, 2016.

[29] Daniel Toth and Til Aach. Improved minimum distance classification with gaussian outlier detection for industrial inspection. In *Proceedings 11th International Conference on Image Analysis and Processing*, pages 584–588. IEEE, 2001.

[30] Soumi Ray, Dustin S McEvoy, Skye Aaron, Thu-Trang Hickman, and Adam Wright. Using statistical anomaly detection models to find clinical decision support malfunctions. *Journal of the American Medical Informatics Association*, 25(7):862–871, 2018.

[31] Gilberto Fernandes, Joel JPC Rodrigues, Luiz Fernando Carvalho, Jalal F Al-Muhtadi, and Mario Lemes Proença. A comprehensive survey on network anomaly detection. *Telecommunication Systems*, 70(3):447–489, 2019.

[32] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[33] Nong Ye, Syed Masum Emran, Qiang Chen, and Sean Vilbert. Multivariate statistical analysis of audit trails for host-based intrusion detection. *IEEE Transactions on computers*, 51(7):810–820, 2002.

[34] Tomáš Čejka, Lukáš Kekely, Pavel Benáček, Rudolf B Blažek, and Hana Kubátová. Fpga accelerated change-point detection method for 100gb/s networks. In *9th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS)*, 2014.

[35] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. Citeseer, 2003.

[36] Splunk siem. `https://www.splunk.com/en_us/cyber-security/siem.html`. Accessed: 2021-05-04.

[37] Solarwinds security event manager. `https://www.solarwinds.com/security-event-manager`. Accessed: 2021-05-04.

[38] Sandeep Bhatt, Pratyusa K Manadhata, and Loai Zomlot. The operational role of security information and event management systems. *IEEE security & Privacy*, 12(5):35–41, 2014.

[39] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM sigmod record*, volume 29, pages 1–12. ACM, 2000.

[40] Malware authors trick apple into trusting malicious shlayer apps. `https://www.bleepingcomputer.com/news/security/malware-authors-trick-apple-into-trusting-malicious-shlayer-apps/`. Accessed: 2021-03-23.

[41] Deepak Vohra. Apache parquet. In *Practical Hadoop Ecosystem*, pages 325–335. Springer, 2016.

[42] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[43] Matthew Rocklin. Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th python in science conference*, volume 126. Citeseer, 2015.

[44] Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, 14(9):1–9, 2011.

[45] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[46] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. *Jupyter Notebooks-a publishing format for reproducible computational workflows.*, volume 2016. 2016.

[47] Apache Airflow Documentation. Apache airflow documentation-airflow documentation, 2019.

[48] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918, IETF, February 1996.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Appendix A

Acronyms

APT Advanced Persistent Threat

AS Autonomous System

ASN Autonomous System Number

BoW Bag of Words

DDoS Distributed Denial of Service

DGA Domain Generation Algorithm

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

DPI Deep Packet Inspection

HBOS Histogram-Based Outlier Score

HIDS Host Intrusion Detection System

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secured

IDS Intrusion Detection System

IDP Initial Data Packet

IoC Indicator of Compromise

IoT Internet of Things

IP Internet Protocol

IPS Intrusion Prevention System

MAC Media Access Control

NIDS Network Intrusion Detection System

OS Operating System

PC Personal Computer

LDAP Lightweight Directory Access Protocol

SIEM Security Information and Event Management

SMB Server Message Block

SMTP Simple Message Transfer Protocol

SOC Security Operations Center

TCP Transmission Control Protocol

tf-idf Term Frequency - Inverse Document Frequency

tw-BoW time window Bag of Words

UDP User Datagram Protocol

UEBA User and Entity Behavior Analytics

URL Uniform Resource Locator

VMI Virtual Machine Inspection

VPN Virtual Private Network

Appendix B

Contents of enclosed drive

- `readme.txt` ..... file with contents description
- `src` ..... directory of source codes
  - `event-generation` ..... models source code with pipeline notebook
  - `experiments` ..... experiments with enitity embedding
  - `thesis` ..... directory of LaTeX source codes of the thesis
- `text` ..... thesis text directory
  - `thesis.pdf` ..... thesis text in PDF format