

UNIVERSIDAD POLITÉCNICA DE MADRID



**MÁSTER UNIVERSITARIO EN
CIBERSEGURIDAD**

TRABAJO FIN DE MÁSTER

***Automated detection of sensitive code
using transformer-based models for
binary code similarity detection***

Alejandro Barreiro Morante

2023

Alejandro Barreiro Morante

Automated detection of sensitive code using transformer-based models for binary code similarity detection

Proyecto Fin de Máster, Thursday 29th June, 2023

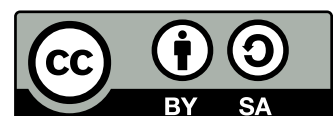
Director: Jorge Blasco Alís

E.T.S. de Ingenieros de Telecomunicación

Avenida Complutense, 30

28040, Madrid, España

This work is licensed under a [Creative Commons “Attribution-ShareAlike 4.0 International”](https://creativecommons.org/licenses/by-sa/4.0/) license.



Yo, **Alejandro Barreiro Morante**, estudiante de la titulación **Máster Universitario en Ciberseguridad** de la de E.T.S. de Ingenieros de Telecomunicación de la Universidad Politécnica de Madrid, como autor del Proyecto Fin de Máster titulado:

**Automated detection of sensitive code using transformer-based models for
binary code similarity detection**

DECLARO QUE

Este proyecto es una obra original y que todas las fuentes utilizadas para su realización han sido debidamente citadas en el mismo. Del mismo modo, asumo frente a la Universidad cualquier responsabilidad que pudiera derivarse de la autoría o falta de originalidad del contenido de la memoria presentada de conformidad con el ordenamiento jurídico vigente.

Madrid, a Thursday 29th June, 2023

Fdo.: Alejandro Barreiro Morante
Autor del Proyecto Fin de Máster

Resumen

Las Amenazas Persistentes Avanzadas (APTs) plantean importantes retos en ciberseguridad, que exigen enfoques innovadores para su eficaz detección y análisis. En este trabajo fin de máster presentamos el desarrollo de MAPTER (*Mapping APTs through Embedding Representation*), un sistema modular que aprovecha la detección de similitudes de código binario basada en aprendizaje automático para mejorar la comprensión y generar medidas de defensa contra las APTs.

MAPTER integra técnicas de distintas disciplinas para analizar binarios de malware de APTs. Se vertebra sobre SAFE [1], un modelo de aprendizaje profundo, utilizado para generar representaciones numéricas de funciones binarias desensambladas. Estas representaciones son tratadas para ser representadas gráficamente y así poder trazar análisis específicos.

Proponemos un análisis específico centrado en el estudio de grupos concretos. Para ello, se analizarán las correlaciones entre sus binarios, basándose en las funciones *entry point* y *main*. De esta forma, se espera obtener información valiosa sobre la evolución y el comportamiento de las APTs.

Realizamos experimentos con el grupo APT19 que validan la eficacia de MAPTER. El enfoque específico planteado, revela similitudes significativas entre algunos binarios, arrojando luz sobre los métodos de actuación de este grupo.

Este trabajo contribuye al campo de la ciberseguridad proponiendo un nuevo enfoque para el análisis de APT que aprovecha la detección de similitudes de código binario basada en el aprendizaje automático. El diseño modular de MAPTER permite la adaptación a nuevas tecnologías de aprendizaje automático, la incorporación de diversos conjuntos de datos y el desarrollo de enfoques de análisis específicos para APTs. Los hallazgos obtenidos mediante MAPTER potencian la comprensión, la detección de amenazas y facilitan la mitigación de APTs.

Palabras clave: APTs; análisis de malware; aprendizaje automático; BSCD

Abstract

Advanced Persistent Threats (APTs) pose significant challenges in cybersecurity, demanding innovative approaches for effective detection and analysis. In this master's thesis we present the development of MAPTER (Mapping APTs through Embedding Representation), a modular framework that leverages machine learning-based binary code similarity detection to enhance understanding and enable defense measures against APTs.

MAPTER integrates state-of-art techniques from various disciplines to analyze APT malware. It utilizes SAFE [1], a self-attentive deep learning model, to extract numerical representations from disassembled binary functions. These embeddings undergo dimensionality reduction techniques for subsequent visual representation so that targeted analysis can be performed.

We purpose a targeted analyses over specific APT groups. By studying correlations among their binaries based on entry points and main functions, insights into the origins, evolution, and behavior of APT groups are obtained.

We perform experiments on the APT19 group validate MAPTER's effectiveness. Correlation analysis reveals significant similarities and shared functions, shedding light on relationships and uncovering valuable insights.

Overall, this thesis contributes to the field of cybersecurity by proposing a new approach to APT analysis that leverages machine learning-based binary code similarity detection. The modular design of MAPTER enables adaptability to emerging machine learning technologies, incorporation of diverse datasets, and the development of targeted analysis approaches for specific APT families. The findings obtained through MAPTER empower defense strategies, enhance threat detection capabilities, and facilitate timely mitigation against APTs.

Keywords: APTs; malware analysis; machine learning; BSCD

Contents

Contents	i
List of Figures	ii
1 Introduction	1
1.1 Problem Statement and Objectives	3
1.2 Structure of the document	4
2 Background	6
2.1 Malicious Software	6
2.2 Malware Analysis	16
2.3 Machine learning	22
3 Related work	28
3.1 APT Analysis	28
3.2 APT malware analysis	30
3.3 Binary Code Similarity Detection	30
3.4 SAFE	34
4 MAPTER	38
4.1 MAPTER Overview	38
4.2 MAPTER: Targeted analysis	46
5 Results	53
5.1 Applying MAPTER	53
5.2 The target: APT19	57
5.3 Correlation Analysis	59
5.4 Use Case 1	61
5.5 Use Case 2	66

5.6	Use Case 3	71
5.7	Use Case 4	74
5.8	Use Case 5	76
5.9	Use Case 6	79
6	Research Conclusions and Future Work	83
6.1	Overview of Findings	83
6.2	Closing Thoughts	84
6.3	Future Research Directions	86
	Bibliography	88

List of Figures

2.1	Phases of Cyber Kill Chain.	16
3.1	The image showcases the various stages involved in the compilation process. Each gear symbolizes a specific point in the process, where any alteration results in a distinct binary code. The term "Modified" refers to optional changes that can be applied to the code, serving as a means of introducing obfuscation techniques.	31
3.2	Architecture of SAFE, extracted from the work of <i>Massarelli, Di Luna, Petroni</i> , et al. [1].	35
4.1	MAPTER different steps overview	39
4.2	MAPTER adaptation steps overview	47
5.1	Reduced embedding representations of APTs group binaries.	56
5.2	Reduced embedding representation of all main functions of the binaries labelled as APT19. The highlighted points indicate the cases where the embeddings overlap or present interesting behaviour. The dimensionality reduction technique used combines PCA and t-SNE. The <i>Binary Hash</i> refers to the first six characters of each binary's SHA-256 hash.	60

5.3	Reduced embedding representation of all entry point functions in the binaries labelled as APT19. The dimensionality reduction technique used combines PCA and t-SNE. The <i>Binary Hash</i> refers to the first six characters of each binary's SHA-256 hash.	61
5.4	Representation of all the function embeddings of the binaries belonging to the Use Case 1.	62
5.5	Different information of the binaries obtained from VirusTotal.	63
5.6	Representation of all the function embeddings of the binaries belonging to the Use Case 2.	66
5.7	Different information of the binaries obtained from VirusTotal.	67
5.8	Representation of all the function embeddings of the binaries belonging to the Use Case 3.	71
5.9	Different information of the binaries obtained from VirusTotal.	72
5.10	Representation of all the function embeddings of the binaries belonging to Use Case 4.	74
5.11	Different information of the binaries obtained from VirusTotal.	75
5.12	Representation of all the function embeddings of the binaries belonging to Use Case 5.	77
5.13	Different information of the binaries obtained from VirusTotal.	78
5.14	Representation of all the function embeddings of the binaries belonging to the Use Case 6.	79
5.15	Different information of the binaries obtained from VirusTotal.	80

1.

Introduction

The democratization of new technologies has enabled an increasingly interconnected world, evidenced by the proliferation of technological devices such as phones, computers, and even [Internet of things \(IOT\)](#) devices [2] which every day are more present in our lives [3].

However, as our reliance on these devices grows, the attack surface for cybercriminals has also increased, resulting in a rising prevalence of cyberattacks, so does the cost and impact of cyberattacks which are becoming increasingly significant [4] [5].

Malware, which refers to "malicious software", has become one of the most popular forms of cyber-attacks on systems. It encompasses all types of unwanted programs that are designed to cause harm to a target system's security and/or privacy, including viruses, Trojans, ransomware, and others. In recent years, the amount of malware present on systems has grown exponentially, as illustrated by the statistics compiled by the German AV-TEST institute. These show that the amount of malware has increased almost tenfold, from around 100 million samples in 2013 to more than 1 billion samples in 2023. [6].

Malware, like software, is constantly evolving from its beginnings: from the first malicious programs that simply rotated the monitor display, to highly complex programs designed by teams specialized in multiple fields. Therefore, to understand the evolving threat landscape, it is important to study the latest trends in malware and cybercrime.

One such trend is the development of specific and targeted attacks known as [Advanced Persistent Threats \(APTs\)](#). APTs are complex attacks carried out by adversaries with sophisticated levels of expertise and significant resources, who use multiple attack vectors to create highly customized and complex programs for the purpose of attacking a target without being detected [7]. The special attention given to APTs is important because, although attackers use known methods to enter their target's network, the tools they use to penetrate are not. Which is why they are given special attention: the tools may include new types of malware, that aren't usually detected by those

signature based antivirus or intrusion detection and prevention systems [8] [9]. Therefore, understanding APTs is crucial for effective malware analysis and detection, as they are among the most challenging types of attacks to detect and defend against.

Malware analysis is a critical field in the fight against cybercrime, which aims to understand the behaviour and functionality of malicious software. There are two primary methods for malware analysis, static and dynamic analysis. Static analysis involves examining the code of a malware sample without executing it. It can quickly identify the structural properties of the application and identify the malware variant within a large scope. However, static analysis cannot analyse the malware in-depth, as many variants use obfuscation techniques to evade detection and hide certain behaviours. In contrast, dynamic analysis involves executing and monitoring the malware in an isolated environment, which can overcome obfuscation and detect variants effectively. Nevertheless, dynamic analysis entails a trade-off between accuracy and efficiency, as it demands more computational resources (such as a simulated or isolated environment) and substantial human involvement to examine and comprehend the outcomes, resulting in a prolonged process. [10]

Considering the critical role of malware analysis in combating cyber threats, it is essential to explore methods and techniques that can enhance its effectiveness and efficiency. One promising direction is the application of machine learning-based approaches, which have gained popularity in recent years, as they can automate the malware analysis process without relying on handcrafted features. Furthermore, many cybersecurity experts believe that AI-powered anti malware tools will be able to cope with modern malware threats and improve scanning engines. This belief is corroborated by the abundant literature on machine learning methods for malware detection, as well as by the outstanding achievements of artificial intelligence in various domains, such as natural language processing with large-scale models like ChatGPT[11] or computer vision with Stable Diffusion[12]. [13]

Machine learning methods can also be categorized into the same groups as traditional malware analysis approaches, depending on the type of features they are trained with. However, another category of analysis that is worth considering is Hybrid analysis. This category refers to methods that combine features from both static and dynamic analysis, with the goal of leveraging the advantages of each approach and overcoming their limitations. [13]

Understanding APTs is crucial for effective malware analysis and detection, as they are among the most challenging types of attacks to detect and defend against. APTs pose a significant threat to organizations, governments, and critical infrastructure, and have been responsible for some of the most high-profile cyberattacks in recent years. These attacks have caused extensive harm, ranging from the sabotaging of Iran's nuclear program with Stuxnet to massive data breaches, such

as the Marriott Hotel's attack, which compromised the data of approximately 500 million guests [8], [14]. In this context, machine learning algorithms can provide a valuable tool for malware analysis and detection, as they can learn from large collections of malware samples and identify novel and unknown APTs based on their features and patterns. By applying machine learning, researchers and practitioners seek to improve the accuracy and efficiency of the malware analysis and detection process, and to mitigate the impact of APTs on the cyber domain.

1.1 Problem Statement and Objectives

Despite their widespread use and success, traditional malware analysis techniques are often considered time-consuming and tedious processes due to their reliance on manual analysis. In addition, these techniques can struggle to keep pace with the rapid growth in quality and quantity of malware, especially when it comes to detecting advanced persistent threats. They often use polymorphic and obfuscation techniques to avoid detection, making them difficult to identify using traditional methods.

To cope with the challenges of malware analysis, various techniques have been developed based on different paradigms, such as signature-based and behaviour-based detection methods, as well as machine learning-based approaches. These techniques aim to extract and learn features and patterns of malware from large datasets in order to enhance their capability to detect and classify malware. Nevertheless, these techniques are often inadequate in detecting APTs due to their advanced techniques for evasion. Therefore, there is a need for novel approaches to detect and analyse APTs that can keep pace with their evolving sophistication.

It is in this context that the current thesis is framed, whose main objective is to **propose a new approach to the analysis of APTs in order to better understand their functioning**. The set of objectives of the thesis can be summarized in the following points:

1. Investigate the potential of machine learning-based approaches, particularly deep learning models, for APT detection and analysis. This investigation will focus on exploring the capabilities of these models in directly working on disassembled binary functions without the need for manual feature extraction.
2. Develop MAPTER, a flexible and modular framework for visualizing APTs function embeddings, facilitating the identification of sensitive code fragments and enhancing the comprehension of APTs. The framework will prioritize modularity to ensure adaptability to

emerging machine learning technologies and enable the incorporation of new datasets and visualization techniques.

3. Explore and propose specific targeted analysis for the application of MAPTER, aiming to leverage the capabilities and potential of the proposed approach. Given the comprehensive nature of the approach and the vast amount of information it encompasses, there is a wide range of opportunities to be explored. By identifying and implementing at least one specific approach, we can extract valuable insights from APTs and gain a deeper understanding of their characteristics and behaviour. Through these targeted analysis, we aim to demonstrate the effectiveness and versatility of the proposed framework in addressing various aspects of APT analysis and providing actionable intelligence.
4. Conduct an empirical validation of the proposed specific targeted analysis and the developed tool by analysing the discovered correlations. This validation process aims to assess the significance and reliability of the identified correlations, thus demonstrating the efficacy of the proposed approach in revealing valuable insights into the structure and behaviour of APT malware. By validating the correlations and their implications, we can establish the credibility and robustness of the proposed framework and use cases, further enhancing its value in the field of APTs analysis.

By achieving these objectives, the thesis aims to contribute to the field of cybersecurity by offering a novel approach to APTs analysis, leveraging machine learning-based techniques and visualization methods to improve our understanding of these sophisticated threats.

1.2 Structure of the document

The structure of the thesis is organized into six chapters, which are summarized as follows:

- Chapter 1: Introduction. This chapter serves as the introduction, providing an overview of the research topic and its significance in the context of cybersecurity, malware, and APTs. It defines the problem statement and outlines the research objectives.
- Chapter 2: Background. This chapter offers the necessary background information to understand the work presented in the thesis. It delves into two key aspects: malware and machine learning, providing the foundational knowledge required for the subsequent chapters.

- Chapter 3: Related work. The third chapter focuses on conducting a thorough literature review, exploring the current state-of-the-art in [APTs](#) detection and analysis techniques, as well as [Binary Code Similarity Detection \(BCSD\)](#). The chapter critically analyses existing literature, establishing the theoretical framework for the thesis.
- Chapter 4 describes the research methodology employed in the study. It introduces MAPTER, the proposed framework for [APTs](#) analysis. The framework involves generating function embeddings using machine learning models and visualizing them for correlation analysis, serving as the backbone of the thesis.
- Chapter 5: Results. In this chapter, we present the findings and outcomes of the experimentation and analysis conducted using the MAPTER framework. The primary objective of this chapter is to showcase the effectiveness of MAPTER in identifying and analysing correlations among different [APT](#) samples, with a specific focus on the APT19 group.
- Chapter 6: Conclusions. The final chapter provides a summary of the research conducted and presents the conclusions derived from the findings. It discusses the contributions of the thesis to the field of malware analysis, the effectiveness of the proposed approach, and its implications for improving [APT](#) detection and analysis. Additionally, this chapter suggests future directions for further research and development in the field.

2.

Background

This chapter is structured around two key axes, malware and machine learning, which provide the essential context for comprehending the work presented in this thesis. It aims to cater to readers with varying levels of familiarity with the subject, ensuring that even those unfamiliar with the field can engage with the research effectively. By exploring the background information encompassing these two axes, readers will acquire a solid foundation in both malware and machine learning, equipping them with the necessary knowledge to delve into the subsequent chapters and grasp the intricate discussions and findings presented in this thesis.

2.1 Malicious Software

Over the years, the targets of malware have evolved, reflecting the changing landscape of technology and the motivations of malicious actors. In the early years of the internet, malware was primarily developed as a challenge or a diversion, exploiting vulnerabilities and behaviours in programs that had not been considered by software developers. These early forms of malware were often created as experiments or proof-of-concepts, rather than with malicious intent [15], [16].

As the internet became more prevalent and interconnected, the motivations behind malware shifted, and malicious actors began to develop malware with explicit malicious intent. Keyloggers emerged as a notable example, designed to surreptitiously record keystrokes and capture sensitive information such as passwords and personal data. Botnets also emerged, networks of compromised computers that could be remotely controlled by a central authority and used for various malicious purposes such as spamming, [Distributed Denial of Service \(DDoS\)](#) attacks, or stealing data. This type of malware generally had a fixed purpose and did not make significant efforts to hide itself [15], [16].

In recent times, the landscape of malware has witnessed significant changes, characterized by a

surge in the number, sophistication, and cost of malicious activities targeting the global economy. This new generation of malware is capable of executing highly destructive and unprecedentedly targeted and persistent attacks. Unlike traditional malware that served a single purpose and made minimal efforts to conceal itself, these advanced threats often employ multiple types of malware, each serving different purposes, throughout their attacks [15], [16].

Moreover, the motivations behind malware attacks have also diversified. While financial gain remains a prominent motivation, other objectives such as corporate espionage, political influence, and disruption of critical infrastructure have gained prominence. Advanced Persistent Threats exemplify the evolution of targeted and sophisticated attacks, typically attributed to state-sponsored actors or well-resourced criminal organizations. APTs employ a combination of techniques, including social engineering, zero-day exploits, and stealthy persistence, to infiltrate targeted systems and maintain long-term access without being detected [8], [15].

Malware Behaviours

In the field of cybersecurity, the classification and understanding of malware, have undergone significant evolution. As the threat landscape has expanded, cybersecurity professionals have sought to categorize and analyse malware based on its behaviour and characteristics. This subsection explores the concept of malware behaviour in contrast to malware families, shedding light on how these classifications have developed over time.

As the sophistication and diversity of malware increased, it became apparent to cybersecurity experts that relying solely on the traditional concept of malware families could be limiting. They realized that numerous samples could be classified into multiple malware families, as the lines between them blurred. Consequently, the definition of a malware family shifted to refer to a group of malware samples that share a common code base or structural similarities [17]. This traditional notion of malware families gradually became associated with the concept of malware behaviour.

Several significant malware behaviours have been extensively documented in previous researches [18], [19]. Building upon these valuable insights, this work aims to provide a concise summary of some of the most crucial and noteworthy malware behaviours. By exploring these behaviours, we can develop a deeper understanding of the **Tactics, Techniques and Procedures (TTPs)** employed by malicious actors.

Virus

A virus is a self-replicating code that attaches itself to host files or programs. When executed, it can corrupt or destroy files and disrupt the functioning of the infected computer. Viruses typically spread through network connections and can cause system performance degradation and denial of service attacks.

Worm

Unlike viruses, worms are standalone malicious programs that can replicate and spread independently. They exploit vulnerabilities in operating systems and propagate through networks and storage devices. Worms consume network and computer resources, leading to system performance degradation and denial of service incidents.

Trojan

A Trojan is a piece of software that appear benign but have hidden malicious purposes. They commonly spread through voluntary downloads from the internet. Trojans can steal sensitive information, observe user activities, and manipulate or delete files on infected systems. They often create backdoors that allow unauthorized access to malicious actors to the system.

Spyware

Spyware is a type of malware that operates covertly, collecting personal information and monitoring user activities without their knowledge or consent. It can be installed on a system without the user's awareness and secretly sends the gathered information to its creator. Spyware is often used for surveillance purposes or to obtain sensitive data such as login credentials. Some authors [19] include within this type of malware different subtypes such as Adware, Keyloggers, Trackware, Cookies, Riskware, Greyware or even Sniffers. More details on the subtypes can be found in the mentioned articles.

Ransomware

Ransomware is a type of malware that encrypts the data or locks down a system, restricting user access. The majority of ransomware variants demand monetary compensation (a so-called "ransom") from the victims in exchange for restoring the normal functionality of the affected system.

Ransomware poses a significant challenge to the internet industry, as it can hijack PCs, encrypt data, and extort money from the users, without any assurance of returning control.

Botnet

A botnet refers to a network of infected computers controlled by hackers or attackers without the owner's knowledge. Bots, which are malicious programs, enable the attacker to remotely control the compromised machines via a Command and Control channel connected to a [Command and Control \(C&C\)](#) server. Botnets can be utilized for various malicious activities, such as organizing [DDoS](#) attacks, conducting phishing fraud, and sending spam messages.

Rootkit

Rootkits aim to gain control over an operating system, concealing themselves or providing a secure environment for other malware to operate. They employ techniques to deceive antivirus software, making detection challenging. Advanced forms of rootkits, known as bootkits, infect the master boot record or volume boot record, enabling them to persist and remain active even after system reboots.

Reverse shell

A reverse shell is a malicious program that allows an attacker to gain unauthorized access to a remote computer by redirecting shell input and output from the target system. Consequently, they allow attackers to execute commands on the host as if they were local.

Backdoor

Backdoors are malware that install themselves and create hidden entrances, bypassing system authentication procedures and providing unauthorized access for attackers to conduct illegitimate activities. Backdoors are commonly employed in conjunction with other malware attacks, widening the attack surface without causing direct harm.

These different types of malware have distinct characteristics and behaviours that need to be understood for effective malware analysis, detection, and defence strategies. By identifying the specific features and functions of each type, organizations can enhance their system protection and reduce the risks caused by these malicious threats.

Malware Camouflage

One of the earliest forms of malware evolution was the emergence of camouflage techniques. The main condition for malware to be able to perform the purpose for which it was designed is that it should not be recognized as a malicious program. Therefore, there are different techniques for camouflaging malware. This subsection explores notable techniques used for malware camouflage, as presented by *Tahir and Akhtar* [18], [19], providing insights into each technique's characteristics.

Obfuscation

Obfuscation encompasses a range of strategies employed by programmers to make code intentionally more challenging to comprehend. These techniques are utilized by both malware developers seeking to conceal the objectives of their programs and software developers aiming to safeguard the intellectual property of their code. Commonly employed obfuscation techniques include:

- **Dead code insertion:** it involves adding irrelevant or redundant instructions that do not affect the functionality of the program, but make it more complex and confusing to analyse.
- **Instruction replacement:** it is another obfuscation technique, where original instructions are substituted with structurally different but functionally equivalent ones.
- **Register reassignment:** this obfuscation technique involves modifying the allocation and usage of registers within a program, making it harder to track the data flow and dependencies.
- **Code transposition:** is based on rearranging the order of instructions, thereby obscuring the logical flow of the code and complicating its understanding.
- **Subroutine reordering:** is an obfuscation technique that modifies the sequence of subroutine calls. By altering the order of these calls, additional complexity is introduced, making it more challenging to unravel the program's logic and intentions.

Packing

Packing is a technique employed in malware wherein executable files are compressed or encrypted within other programs. This process allows the packed file to remain concealed until specific conditions are met, triggering its release and execution of its intended behaviour.[19]

Encryption

Encryption is a technique that encrypts the malware code with a key and runs a decryption module to execute it at runtime. This technique is employed by some [Malware-as-a-Service \(MaaS\)](#) providers, who generate a specific type of malware and sell it to different customers, using different keys for each version to make them appear as distinct samples and evade some detection mechanisms.

Polymorphism

Polymorphic malware is designed to alter its appearance with each execution while retaining its original code. Unlike encryption techniques, polymorphic malware can employ a variety of encryption algorithms for mutation. The transformation engine, encrypted within the malware, generates new encryption algorithms and re-encrypts the engine and malware with a different decryption key. Polymorphic malware can embed malicious actions, making detection more challenging.

Metamorphism

Metamorphic malware, also known as body-polymorphic malware, mutates its code with each execution, creating unique instances that function similarly but have no resemblance to the original code. There are two categories of metamorphic malware: malware which mutates through communication with external sites over the internet, and malware that self-reprograms through binary code mutation or pseudocode representation.

These camouflage techniques demonstrate the sophisticated strategies employed by malware developers to evade detection, complicate analysis, and ensure the successful execution of their malicious objectives. By understanding these techniques, security professionals can enhance their detection and mitigation strategies against such evolving threats.

A new form of threats: APTs

There are a number of attacks that are known to take advantage of these cloaking techniques and incorporate multiple malware behaviours into their attacks. These are known as [Advanced Persistent Threats \(APTs\)](#). The [National Institute of Standards and Technology \(NIST\)](#) provides a widely accepted definition for [APTs](#), as follows:

"An adversary with sophisticated levels of expertise and significant resources, allowing it through the use of multiple different attack vectors (e.g., cyber, physical, and deception), to generate opportunities to achieve its objectives which are typically to establish and extend its presence within the information technology infrastructure of organizations for purposes of continually exfiltrating information and/or to undermine or impede critical aspects of a mission, program, or organization, or place itself in a position to do so in the future; moreover, the advanced persistent threat pursues its objectives repeatedly over an extended period of time, adapting to a defender's efforts to resist it, and with determination to maintain the level of interaction needed to execute its objectives". [7]

This definition allows us to distinguish APTs from other types of threats on several key points. Firstly, they are **advanced**, leveraging their extensive capabilities and utilizing the latest technologies throughout the various stages of an attack. Secondly, they are **persistent**, aiming to remain undetected within systems for an extended period. Lastly, they pose significant **threats** as their primary objectives typically involve exfiltration or damage to sensitive data within targeted systems. The table 2.1 summarizes the differences between traditional threats and APTs for some attack attributes:

Table 2.1: Comparison of traditional and APT attacks, extracted from the work of *Chen, Desmet, and Huygens* [9].

	Traditional Attacks	APT Attacks
Attacker	Mostly single person	Highly organized, sophisticated, determined and well-resourced group
Target	Unspecified, mostly individual systems	Specific organizations, governmental institutions, commercial enterprises
Purpose	Financial benefits, demonstrating abilities	Competitive advantages, strategic benefits
Approach	Single-run, "smash and grab", short period	Repeated attempts, stays low and slow, adapts to resist defences, long term

The emergence of Advanced Persistent Threats was exemplified by the Stuxnet attack in 2010. Stuxnet was a sophisticated cyber-weapon that targeted Iran's nuclear program, signifying a strategic shift towards APTs. Unlike traditional malware, Stuxnet was designed to remain undetected for an extended period, reflecting the *persistent* nature of APTs. The attack employed a multi-stage strategy, utilizing a Windows zero-day exploit, a stealthy rootkit, and various propagation methods to spread within networks. This complexity indicated a well-resourced and highly skilled threat actor [8], [20].

One of the remarkable aspects of Stuxnet was its specialization in manipulating [Programmable Logic Controllers \(PLCs\)](#) used in industrial control systems. Specifically targeting Siemens' PLCs, the malware exploited vulnerabilities in their firmware. This demonstrated a deep understanding of the target systems and a strategic intent to cause physical damage, highlighting the *advanced* nature of [APTs](#) [20].

The Stuxnet attack served as a wake-up call, illustrating the emergence of [APTs](#) as a new form of cyber-weapon. The primary objective of [APTs](#) was not immediate financial gain or broad disruption, but rather achieving strategic goals through precise and prolonged infiltration of targeted systems. The attack showcased the capability of threat actors to develop advanced and highly tailored malware, specifically designed to exploit vulnerabilities in critical infrastructure. This event paved the way for further research and exploration into [APTs](#), their techniques, and effective countermeasures.

Organizing an attack

Before delving into the attack phases associated with Advanced Persistent Threats, it is crucial to introduce the Cyber Kill Chain model, which serves as a framework for analysing and understanding most cyberattacks. The Cyber Kill Chain model was initially presented by Lockheed Martin in 2011 and is widely used by incident response teams, digital forensic investigators, and malware analysts [21].

The Cyber Kill Chain model provides a systematic and sequential method for incident response. It entails understanding and examining the offensive tactics used by cyberattackers, which helps security analysts to devise effective defensive measures. Hence, security analysts who aspire to develop robust defence strategies and mitigation techniques need to study the Cyber Kill Chain. Each phase of the Cyber Kill Chain constitutes a significant research domain that demands thorough analysis and investigation [21], [22].

The Cyber Kill Chain framework enables analysts to decompose complex attacks into stages or layers. This layered approach facilitates the analysts to tackle smaller, more tractable problems, while also allowing defenders to create specific defences and mitigation strategies for each phase. The Cyber Kill Chain usually comprises seven phases, as shown in Figure 2.1. The following description of the phases is based mainly on the work of *Hutchins, Cloppert, Amin, et al. and Yadav and Rao* [21], [22].

1. Reconnaissance: This stage involves conducting research, identifying potential targets, and

gathering information about them. Attackers may crawl internet websites, conference proceedings, mailing lists, and other online sources to obtain email addresses, social relationships, and details about specific technologies. The information collected during reconnaissance is crucial for designing and delivering the payload in later stages.

2. **Weaponization:** In the weaponization stage, the attacker designs a backdoor and formulates a penetration plan based on the information gathered during reconnaissance. This typically involves combining a [Remote Access Trojan \(RAT\)](#) with an exploit to create a deliverable payload. Popular techniques include utilizing automated tools or customize existing malware to suit their needs.
3. **Delivery:** The delivery stage plays a critical role in a cyberattack. Attackers aim to efficiently and effectively deliver their malicious payload to the target environment. This often requires user interaction, such as downloading and executing malicious files or visiting malicious websites. However, some attacks can exploit network devices without user interaction. To minimize traceability, attackers frequently employ anonymous services, compromised websites, and compromised email accounts for delivery.
4. **Exploitation:** Once the weapon is delivered and the target engages with it, the exploitation phase begins. This is the stage where the attacker triggers the malicious tool to execute on the target's system. The tool may exploit a vulnerability in an application or operating system, or trick the user into running it. Successful exploitation allows the attacker to gain control over the compromised system.
5. **Installation:** With host-based security measures becoming more robust, attackers have developed innovative procedures to bypass these controls during the installation stage. Traditional infection vectors like infected removable media have given way to multistaged malware that rely on droppers and downloaders for a more sophisticated delivery of malware modules. The installation stage ensures the installation, update, and regulation of remote access trojans or backdoors on the victim's system, providing the attacker with persistent access.
6. **Command and Control (C&C):** The Command and Control stage involves establishing a channel between the compromised host and an external server controlled by the attacker. Compromised hosts often beacon outbound to this server, enabling the attacker to have hands-on access inside the target environment. The [C&C](#) system serves as a covert communication platform for issuing instructions to compromised machines and exfiltrating data.

7. Actions on Objectives: Finally, after progressing through the preceding stages, the attacker can execute actions to achieve their original objectives. This often involves data exfiltration, where the attacker collects, encrypts, and extracts information from the victim's environment. Other objectives may include violating data integrity or availability. Additionally, the compromised system can serve as a pivot point for lateral movement and further compromise within the network.

Several authors have drawn connections between the phases of the Cyber Kill Chain and the stages commonly associated with Advanced Persistent Threats attacks [8], [9], [23]. While the Cyber Kill Chain provides a comprehensive framework for analysing most cyberattacks, the APTs phases introduce additional considerations and techniques. These distinctions shed light on the unique characteristics of APTs attacks compared to the broader concept of the Cyber Kill Chain.

APTs attacks acknowledge the importance of reconnaissance, delivery, exploitation, and establishing command and control mechanisms like the Cyber Kill Chain. However, they also emphasize the stages of lateral movement within the target's network, post-exfiltration/post-impediment activities, and the secure transmission of stolen data, the last two phases occur mainly when the purpose of the attack is data related.

Lateral movement is a crucial stage in APTs attacks where attackers navigate within the target's network, expand control, and gather valuable data. They employ legitimate tools, crack or steal credentials, and employ stealthy tactics to maintain a low and slow profile. The Cyber Kill Chain does not explicitly address this stage but focuses on achieving the attacker's objectives after gaining access.

When the purpose of the attack is data related, APTs specifically address the exfiltration of stolen data, detailing the process of funnelling information to internal staging servers, compressing, encrypting, and using secure protocols or the Tor network for transmission. The Cyber Kill Chain does not explicitly cover data exfiltration but recognizes actions involving the collection, encryption, and extraction of information from the victim's environment.

Furthermore, the APTs phases introduce the post-exfiltration/post-impediment stage, where attackers take actions after data exfiltration or impairment. This may involve continued exfiltration, disabling critical components, or deleting evidence to cover their tracks and maintain persistence within the compromised network. The Cyber Kill Chain does not explicitly address these post-attack activities.

By considering these differences between the Cyber Kill Chain and APTs phases, we gain a deeper

understanding of the unique characteristics and techniques employed in APTs attacks, enabling us to develop more effective defensive strategies and countermeasures [8], [9], [23].

The work examines the five phases of the Cyber Kill Chain, with a particular emphasis on the weaponization phase. This phase is of utmost importance as it involves the analysis of the binary itself. However, it is crucial to acknowledge that other phases, such as exploitation, installation, command and control, and actions on objectives, may also be encompassed within this binary. Different payloads associated with these phases can be included, highlighting the interconnected nature of the APTs attack lifecycle and the need for comprehensive analysis across all phases.



Figure 2.1: Phases of Cyber Kill Chain.

2.2 Malware Analysis

Malware analysis is the process of studying malicious software to understand its structure, functionality, behaviour, and evolution. Malware analysis can be performed for various purposes, such as malware detection, classification, attribution, and mitigation. Malware analysis can be broadly categorized into different types that will be discussed within this section.

Static Analysis

Static analysis is a widely used technique in malware analysis. It involves examining an application without executing its code, aiming to detect potential malicious elements. Static analysis can be performed at various levels of abstraction, including binary features, assembly code, and even source or pseudo code. The depth and detail of the analysis determine its categorization into basic and advanced approaches.

Basic static analysis primarily focuses on extracting and analysing diverse features from executable files, such as strings, imports, exports, functions, directories, and file headers. These extracted features provide valuable insights into the behaviour and functionality of the executable files, aiding in the identification of potential malicious patterns.

For advanced static analysis, reverse engineering techniques are employed to extract an application's code, gain an understanding of its functionality, and identify possible malicious behaviour. Decompilers, disassemblers, and other specialized tools play a vital role in closely examining the application code to uncover key indicators of malicious activity, including modifications to sensitive elements or the presence of known malware signatures.

However, static analysis may have limitations in capturing the complete behaviour of an application. Since it relies on examining the code without executing it, certain malicious actions or effects may remain undetected. For instance, obfuscation techniques may be employed to conceal true behaviour, or specific actions may be triggered only under certain conditions. In such cases, static analysis alone may fail to identify the application's malicious behaviour.

Recognizing these limitations, it becomes imperative to complement static analysis with other techniques to achieve a comprehensive and accurate analysis of an application's behaviour. By combining static analysis with dynamic analysis and other complementary methods, a more robust analysis framework can be developed to effectively study APTs and gain deeper insights into their characteristics and objectives [16], [24].

Dynamic Analysis

Dynamic analysis plays a crucial role in the field threat analysis by providing insights into the behaviour of malware samples. It involves running the applications within a controlled environment and observing their real-time interactions with different sensitive resources. This information serves to identify malicious behaviour and validate the findings from static analysis [16], [24].

By running an application in a controlled environment, dynamic analysis allows for the monitoring of various aspects, such as system calls, network traffic, file access, and memory operations. This information can be used to identify malicious behaviour, such as unauthorized data exfiltration, network communication with suspicious domains, or modification of critical system files [24].

One of the advantages of dynamic analysis is its ability to capture the complete behaviour of an application, including actions that may be triggered under specific conditions. It provides a compre-

hensive view of the application's actions and interactions, helping to identify intricate malicious behaviour that may be hidden from static analysis.

Despite its benefits, dynamic analysis does have certain drawbacks. One of these drawbacks is the time and resource-intensive nature of the analysis process. Running applications on devices or emulators and observing their behaviour in real time requires substantial time and resources, making dynamic analysis more time-consuming compared to other analysis methods, such as static analysis. Additionally, specialized tools and technologies, such as forensic tools and sandbox environments, may be necessary, thereby increasing the complexity and cost associated with the analysis.

Another limitation of dynamic analysis is its inability to capture the complete behaviour of an application. Since dynamic analysis only observes the behaviour of an application within a specific environment, it may fail to detect certain malicious actions or potential effects. For example, an application may exhibit malicious behaviour only under specific conditions, such as when a wireless internet connection is available or when interacting with other applications. In such cases, dynamic analysis alone may not fully identify the malicious behaviour of the application.

In addition to dynamic analysis, various specialized tools and technologies are commonly employed in the analysis of [APTs](#). These include forensic tools used for data extraction and analysis from devices, network analysis tools for monitoring network traffic, and sandbox environments that enable the safe execution and observation of malicious applications [\[24\]](#).

Malware analysis is a complex and ever-evolving field. By gaining an understanding of the fundamental concepts and techniques of malware analysis, as well as the tools and technologies utilized in this domain, researchers and practitioners can develop the necessary skills and knowledge to effectively identify and protect against malware on devices in the context of [APTs](#) analysis [\[16\]](#), [\[24\]](#).

Disassemblers for Malware Analysis: Radare2, IDA, and Ghidra

Disassemblers play a crucial role in malware analysis and reverse engineering, allowing researchers to delve into the low-level assembly code of executable files and uncover potential malicious behaviour. In this subsection, we will explore three prominent disassemblers: Radare2, IDA, and Ghidra [\[24\]](#).

Disassemblers serve as tools to translate executable files into human-readable assembly code, aiding reverse engineers in comprehending the functionality and logic behind software and identifying possible malicious code and behaviour. These tools often provide additional features, such as graphing, scripting, debugging, and decompiling, to facilitate the reverse engineering process.

It is important to note that disassemblers may encounter challenges and limitations when dealing with [APTs](#) malware. Some obfuscation as well as architectures and file formats employed by [APTs](#) malware may not be fully supported by disassemblers. Moreover, the complexity and ambiguity of assembly code can introduce inaccuracies and inconsistencies in the disassembled output.

Therefore, it is crucial for reverse engineers to carefully select appropriate disassemblers for their analysis tasks and complement them with other techniques like static analysis, dynamic analysis, machine learning, and visualization. By combining different approaches, researchers can enhance their efficiency and effectiveness in [APTs](#) malware detection and classification. It is worth noting that licensing restrictions or availability issues may make certain disassemblers costly or inaccessible, so researchers must make informed decisions based on the specific constraints of their projects.

With a solid understanding of the strengths and limitations of disassemblers, coupled with a comprehensive arsenal of complementary techniques, reverse engineers can navigate the complex landscape of [APTs](#) malware and extract valuable insights for effective threat detection and analysis.

Radare2

Radare2 is a powerful open-source disassembler and reverse engineering framework that provides a wide range of features for binary analysis. It supports multiple architectures and various binary formats, making it adaptable to diverse scenarios. One notable aspect of Radare2 is its [Command-Line Interface \(CLI\)](#) and [Graphical User Interface \(GUI\)](#), which offer users the flexibility to choose the mode that best suits their preferences and requirements.

An additional advantage of Radare2 is its integration with Python, facilitated by its Python bindings. This integration enables users to programmatically interact with the framework, allowing for advanced analysis tasks and automation of repetitive processes. Leveraging Python's extensive ecosystem and libraries, researchers and developers can utilize data manipulation, visualization, and machine learning integration within their Radare2 workflow, enhancing their overall analysis process.

Radare2 excels in efficiently handling large binaries by supporting incremental analysis. Instead of analysing the entire file, users can selectively load and analyse specific portions of the binary, resulting in reduced analysis time and resource consumption. Additionally, Radare2 offers powerful scripting capabilities, enabling automation and customization of analysis tasks to suit individual needs.

By leveraging Radare2's versatility, Python integration, and efficient analysis capabilities, researchers and analysts can effectively explore and uncover insights from binary files. The combination of Radare2 with other tools and techniques within the Python ecosystem empowers users to enhance their analysis workflow and efficiently tackle complex reverse engineering challenges. [25], [26]

IDA

IDA (*Interactive DisAssembler*) is a widely used commercial disassembler renowned for its extensive capabilities in binary analysis. Supporting a broad range of architectures and file formats, IDA is suitable for analysing diverse types of software. It offers both a graphical interface and a command-line interface, providing flexibility to users based on their preferences and expertise.

IDA excels with advanced features like graph visualization, cross-references, and interactive debugging, making it a popular choice among experienced reverse engineers. The tool also boasts a rich plugin ecosystem, enabling users to extend its functionality and integrate additional analysis tools. IDA's comprehensive analysis capabilities, combined with its user-friendly interface, make it a valuable asset in the field of malware analysis.

One of the downsides of IDA is that the full version of the software that allows access to its most interesting features, IDA Pro, is a paid version, so it is not accessible to all users [27].

Ghidra

Ghidra is a robust open-source software reverse engineering framework developed by the [National Security Agency \(NSA\)](#). It offers an array of features for disassembly, decompilation, and analysis of binary files. With support for various architectures and file formats, Ghidra proves suitable for analysing diverse software types [28].

Ghidra's notable strength lies in its powerful decompilation capabilities, which transform assembly code into a higher-level programming language. This feature greatly aids in understanding the functionality and behaviour of complex binaries. Ghidra also offers advanced features like scripting, collaborative analysis, and plugin support, enhancing its flexibility and usability.

While Ghidra is a powerful tool for reverse engineering and binary analysis, it is essential to consider its downsides, particularly when using the [Command-Line Interface \(CLI\)](#) version and attempting to automate processes [28].

All three disassemblers, Radare2, IDA, and Ghidra, enjoy widespread use in the field of malware analysis and reverse engineering. Their comprehensive features, support for multiple architectures and file formats, and extensibility through plugins make them invaluable tools for researchers and analysts seeking to explore the inner workings of software and identify potential malicious behaviour. The choice of disassembler depends on the specific analysis requirements, the expertise of the analyst, and the available resources.

IOCs: Key Elements in Malware Analysis

In the field of Advanced Persistent Threats analysis, the identification and utilization of [Indicators of Compromise \(IOCs\)](#) play a crucial role in understanding and attributing cyber threat incidents. [IOCs](#) are artefacts or evidence that indicate the presence of malicious activities or the compromise of a system or network. These artefacts can be used to identify and track the [Tactics, Techniques and Procedures \(TTPs\)](#) employed by threat actors throughout different phases of a cyberattack [29], [30].

[IOCs](#) can be categorized into two main types: high-level [IOCs](#) and low-level [IOCs](#). High-level [IOCs](#) refer to broader patterns and characteristics observed in cyberattacks, such as the use of specific malware families, the exploitation of certain vulnerabilities, or the adoption of particular attack methodologies. These high-level [IOCs](#) provide valuable insights into the behaviour and actions of threat actors and help in profiling their activities [29].

Low-level [IOCs](#) are more specific and tangible artefacts that can be easily observed or detected within a system or network. These include IP addresses, domain names, file hashes, registry keys, network traffic patterns, and other technical indicators that directly point to the presence of malicious activities. Low-level [IOCs](#) are commonly used in security operations, where they are employed to detect and block malicious traffic, trigger alerts, or aid in forensic investigations [29].

While low-level [IOCs](#) offer immediate and actionable information for detecting and responding to cyber threats, they have limitations. Threat actors often employ various tactics to obfuscate or change their low-level [IOCs](#), such as using new servers, domain names, or IP addresses, which reduces the effectiveness of these indicators in the long term. Furthermore, low-level [IOCs](#) can be easily spoofed or anonymized, leading to inaccurate or biased attributions.

High-level [IOCs](#) provide a broader understanding of the overall attack campaign, linking multiple incidents and identifying common patterns across different [APTs](#) campaigns. By analysing high-level [IOCs](#) found in [Cyber Threat Intelligence \(CTI\)](#) reports and correlating them with previous threat incidents, security analysts can attribute cyber threat incidents to specific threat actors or groups. High-level [IOCs](#) focus on the Tactics, Techniques and Procedures employed by threat actors, allowing for a more comprehensive and proactive defence against [APTs](#).

To aid in the identification and utilization of [IOCs](#), including both high-level and low-level indicators, researchers and analysts often rely on tools like YARA rules. YARA is a powerful and flexible open-source tool used for pattern matching against files and processes. It allows the creation of custom rules that define patterns based on strings, byte sequences, or regular expressions. These rules can be used to scan files, memory, or network traffic for the presence of specific indicators associated with known threats or attack techniques. By leveraging YARA rules, analysts can automate the detection of [IOCs](#) and streamline the analysis process, contributing to a more efficient and effective defence against [APTs](#) [31], [32].

In conclusion, [IOCs](#) play a vital role in the analysis of Advanced Persistent Threats by providing tangible evidence and valuable insights into the Tactics, Techniques and Procedures employed by threat actors. The utilization of YARA rules enables the definition and detection of both high-level and low-level [IOCs](#) in malware samples, enhancing the identification and utilization of these indicators. Within the context of this thesis, our proposed framework focuses primarily on facilitating the identification of high-level [IOCs](#) [29], [30].

2.3 Machine learning

[Machine Learning \(ML\)](#) is a subfield of artificial intelligence that aims to emulate one of the human abilities: automatic learning from the environment. Learning refers to the capability of a system to improve its performance on a task by using data as input. A typical example could be a computer program that learns the basic rules of chess and improves its skills by playing games against itself [33].

Algorithms and examples

Machine learning techniques can be classified into three main categories, according to the nature of the problem they address and the feedback mechanism they use:

- **Supervised learning**, involves training machines using labelled datasets to learn a function that maps inputs to outputs. The typical approach is to divide the dataset into training and testing subsets, allowing the machine to learn from one portion and use the other for making predictions and assessing its performance. This type of learning is commonly applied to classification and regression tasks, where the machine learns to categorize or predict numerical values based on the provided labels. By leveraging labelled data, supervised learning enables machines to generalize patterns and make accurate predictions on new, unseen data.

Supervised learning can be employed in the detection of phishing emails, leveraging content and sender information. By training on labelled datasets containing legitimate and phishing emails, these algorithms can learn to differentiate between them based on features such as subject lines, sender addresses, spelling and grammar, tone and urgency, embedded links and attachments, and more. The trained model can then be utilized to scan new emails and flag potential phishing attempts [34], [35].

- **Unsupervised learning**, in which machines receive unlabelled data sets for training, so that they have to discover patterns and structures in the data. One of the challenges of these algorithms is how to measure their performance. Examples of unsupervised learning algorithms are clustering or dimensionality reduction.

Unsupervised learning can be used in cybersecurity for identifying new malware families based on code similarity and behaviour. By clustering malware samples according to their similarities, these algorithms can aid in labelling and naming them, contributing to the discovery of new or previously unknown malware families and improving understanding of their characteristics and capabilities [34], [36].

- **Reinforcement learning**, in which machines do not need labelled data sets for training, but rather interact with their environment and receive rewards based on their actions. In this case, the training and testing phases are intertwined, as the machine learns by trial and error. The goal of the system is to maximize its cumulative reward over a sequence of actions and interactions with the environment.

A use case of reinforcement learning is defending against distributed [Distributed Denial of Service \(DDoS\)](#) attacks by dynamically adjusting the network parameters and policies. Reinforcement learning algorithms can be used to learn how to optimize the network performance and security by interacting with the environment and receiving rewards or penalties based on their actions. The algorithms can then adjust the network parameters and policies, such as the bandwidth allocation, the traffic filtering, the load balancing, etc., in response

to the changing conditions and threats. This can help mitigate the impact of [DDoS](#) attacks and maintain the availability and functionality of the network [\[34\]](#), [\[37\]](#), [\[38\]](#).

Deep Learning

Going into a finer level of detail, this work makes use of an architecture based on a particular family of algorithms within the field of machine learning: [Deep Learning \(DL\)](#).

Traditional approaches to machine learning revolve around the use of algorithms that work with a set of features whose importance and constituents are defined by the person who decides to use them. These are extracted in a process of engineering on the raw data, known by various names, including feature extraction. This process involves obtaining learning representations, and the performance of machine learning algorithms is closely related to the quality of these representations.

Traditional approaches in deep learning involve the use of algorithms that rely on predefined sets of features, determined by the human experts that use the model, through a process known as feature extraction. This process involves deriving meaningful representations from raw data, and the performance of machine learning algorithms heavily relies on the quality of these representations.

However, traditional methods often encounter limitations when working with raw, unprocessed data, which necessitates pre-processing steps. In contrast, deep learning emerges as a subfield where the algorithm itself autonomously identifies and focuses on the most significant features through various mechanisms. This intrinsic feature discovery offers several advantages, reducing the workload of data analysts and potentially optimizing their analysis.

[DL](#) is particularly beneficial when working with challenging data formats, such as raw data encountered in image processing, natural language processing, audio processing, and even code processing. Raw data presents difficulties due to its complex nature, requiring human-like comprehension for interpretation. [DL](#) architectures, on the other hand, excel in extracting meaningful structures from such data.

This ability to work with raw data is one of the major contributions of deep learning, achieving interpretations of these characteristics through representations that are expressed in terms of other, simpler representations. In other words, it allows complex concepts to be constructed through simpler concepts thanks to artificial neural networks [\[39\]](#).

Machine learning-based approaches to malware analysis

In recent years, the field of malware analysis has experienced significant advancements, with machine learning techniques playing a pivotal role in enhancing detection and classification capabilities. Based on the works of *Ucci, Aniello, and Baldoni* and, *Aslan and Samet* [40], [41] this subsection provides a comprehensive review of the different machine learning-based approaches used in malware analysis.

Malware analysis poses numerous challenges due to the increasing complexity, diversity, and volume of malware samples. Malware authors employ various evasion techniques, making traditional analysis methods less effective. Moreover, the sheer number of malware samples requires automated and scalable solutions to keep up with the ever-changing malware landscape. Machine learning techniques offer several advantages for malware analysis, including:

- **Autonomous learning:** Machine learning models can learn patterns and features from data without the need for manual feature extraction or rule generation. This allows for more efficient analysis processes.
- **Handling complex data:** ML algorithms excel at handling high-dimensional and heterogeneous data types, such as binary code, system calls, network traffic, and more. This capability speeds up analysis and enables researchers to explore and extract valuable information from different data sources.
- **Generalization and adaptability:** Machine learning models can generalize to unseen data and adapt to new situations by updating their models with new information. This adaptability, combined with autonomous learning, helps improve the defence systems of organizations.

However, machine learning-based approaches in malware analysis also face challenges and limitations. They require large amounts of high-quality data for training and testing, and issues such as data imbalance, noise, redundancy, and inconsistency can impact their performance. Adversarial attacks pose a threat as they can manipulate data or models to evade detection or cause misclassification. Computational costs and complexity may also limit scalability and efficiency.

One common approach is to use machine learning algorithms for malware detection. These algorithms can be trained on large datasets of known malware samples, allowing them to learn patterns and characteristics that distinguish malicious code from benign code. Various features can be extracted from malware samples, such as opcode sequences, API calls, system call traces, and

byte-level n-grams, to represent the behaviour and structure of the code. As noted in Section 2.3 classifiers, such as decision trees, support vector machines, and neural networks, are then trained on these features to classify new samples as either malware or benign.

Another important application is malware family classification. This involves categorizing malware samples into different families or groups based on their similarities. ML algorithms are trained on labelled datasets of known malware families, learning to recognize common features and patterns specific to each family. Feature selection and dimensionality reduction techniques are often employed to handle the high-dimensional nature of malware data.

Furthermore, machine learning can be leveraged for malware attribute extraction. This involves extracting specific attributes or characteristics of malware samples, such as the presence of certain API, system calls, or code obfuscation techniques. ML algorithms can learn to identify these attributes, automating the extraction of valuable information from malware samples.

Overall, machine learning has proven to be a valuable tool in malware analysis, facilitating faster and more accurate detection, classification, and attribute extraction. However, challenges such as adversarial attacks, data imbalance, and evolving malware techniques continue to pose significant obstacles to the effectiveness of machine learning-based approaches.

Dimensionality reduction for Data Visualization

Dimensionality reduction techniques play a significant role in this thesis, as we try to visualize the vector representations of binary functions which are known for having high dimensionality. Therefore, we use dimensionality reduction techniques with the aim of reducing the complexity and dimensionality of the data while retaining the most relevant information. These techniques aid in visualizing and understanding the relationships between different features or data points, facilitating the detection of patterns and clusters within APTs binaries.

PCA

Principal Component Analysis (PCA) is a widely used linear dimensionality reduction technique. It transforms the high-dimensional data into a lower-dimensional representation by identifying the principal components that capture the maximum variance in the data. These principal components are orthogonal to each other and ordered by the amount of variance they explain. By selecting a subset of the top principal components, we can effectively reduce the dimensionality of the data while retaining a significant amount of its information [42].

t-SNE

t-Distributed Stochastic Neighbour Embedding (t-SNE) is a nonlinear dimensionality reduction technique that is particularly effective at preserving the local structure of the data. It models the high-dimensional data as probabilities in a way that similar points have higher probabilities of being chosen as neighbours. It then constructs a lower-dimensional representation in which similar points have a higher probability of being close to each other [42].

The Fusion of PCA and t-SNE

A common practice in data visualization is to use **PCA** as a preliminary step to lower the dimensionality of the data before applying **t-SNE**. The combination involves using a pipeline based on **PCA** as a preprocessing step to reduce the dimensionality of the data before applying **t-SNE**. This approach takes advantage of the strengths of both techniques. **PCA**, being a linear technique, efficiently captures the major sources of variation in the data. By applying **PCA** first, we can retain the most important components that explain the majority of the variance in the data. These chosen components are then used as input for **t-SNE**, which is more suitable for detecting nonlinear relationships and showing local structures in the data.

The integration of **PCA** and **t-SNE** provides a robust framework for visualizing complex datasets. This combination helps address certain limitations of **t-SNE**, such as its sensitivity to the perplexity parameter and the potential loss of global structure in the resulting visualizations. By incorporating **PCA** as a preprocessing step, we can reduce the dimensionality of the data while retaining crucial information, thereby potentially leading to more meaningful visualizations through **t-SNE**.

The work by *Kobak and Berens* [42] offers insights into practical considerations and best practices when employing **PCA** and **t-SNE** together. The paper discusses the importance of selecting an appropriate number of principal components to retain from **PCA** and provides guidance on parameter choices for **t-SNE**. The authors emphasize an iterative approach, where different combinations of **PCA** and **t-SNE** can be explored to identify the most informative and interpretable visualizations.

To summarize, the combination of **PCA** and **t-SNE** harnesses the strengths of both techniques for effective visualization and exploration of high-dimensional data. By utilizing **PCA** to capture global variation and **t-SNE** to reveal local structures, researchers can gain valuable insights and discover patterns in complex datasets.

3.

Related work

In this chapter, we present a comprehensive review of the existing literature on [APTs](#) analysis, with a particular focus on [APTs](#) malware analysis supported by machine learning techniques. The literature review provides a solid foundation for understanding the current state of research in the field and identifies the gaps and challenges that exist in the field. By building upon this knowledge, we aim to develop and evaluate the effectiveness of our proposed MAPTER framework for [APTs](#) analysis.

3.1 APT Analysis

Advanced Persistent Threat analysis is a crucial area of research in cybersecurity, aiming to detect, analyse, and mitigate sophisticated and persistent cyber threats. [APTs](#) analysis involves studying the attack vectors, techniques, capabilities, and motivations of these adversaries. This helps security professionals and researchers gain insights into their tactics, techniques, and procedures and develop effective countermeasures, improve threat intelligence, and enhance the overall security of organizations.

The literature on [APTs](#) analysis covers various topics, such as malware analysis, network analysis, attribution, and threat intelligence. Researchers have explored different approaches to detect, analyse, attribute, and defend against [APTs](#) using diverse data sources such as network traffic, system logs, malware samples, and intelligence reports.

In general, the most common approach to [APTs](#) malware detection is network analysis [43]. It focuses on examining network traffic and communication patterns. By analysing network logs, packet captures and network behaviour, security professionals can detect signs of lateral movement, command-and-control communication and data exfiltration associated with [APTs](#) activities. This analysis helps identify potential Indicators of Compromise and anomalous network behaviour indicative of [APTs](#).

For instance, in a study conducted by *Do Xuan, Dao, and Nguyen* [44], a novel method for detecting advanced persistent threat attacks is proposed, leveraging network flow data and deep learning models. The approach involves analysing network traffic and organizing it into IP-based network flows. Deep learning models are then utilized to extract relevant features from the flow data, enabling the identification of IP addresses involved in APTs attacks. This innovative approach enhances the ability to detect APTs attacks and contributes to the overall understanding of APTs analysis within the field of network analysis.

Threat intelligence plays a crucial role in the analysis of Advanced Persistent Threats, involving the collection and analysis of information related to these threats. It involves studying known APTs campaigns, groups, their motivations, targets, and tactics. Threat intelligence provides valuable insights into the geopolitical context, attribution, and strategic implications of APTs activities. This information aids in understanding the characteristics, trends, and emerging threats associated with APTs, providing a comprehensive view of the threat landscape [8], [9].

By leveraging threat intelligence, security professionals can enhance their understanding of the characteristics, trends, and emerging threats associated with APTs. The work of [8] acknowledges the role of threat intelligence as a valuable resource for APTs analysis, enabling security professionals to stay updated on evolving APTs techniques and the behaviours of threat actors. It serves as a foundation for identifying potential IOCs, abnormal network behaviour, and other patterns associated with these threats.

Attribution is a challenging aspect of APTs analysis, focused on identifying and attributing APTs attacks to specific individuals, groups, or nation-states. It requires a combination of technical analysis, intelligence collaboration, and geopolitical analysis. Accurate attribution informs defensive strategies, diplomatic actions, and law enforcement efforts [45].

For example, in the research conducted by *Rosenberg, Sicard, and David* [45] they explore the advancements in attribution techniques, specifically focusing on the utilization of deep neural networks for nation-state APTs attribution. It recognizes that successful attribution enhances the ability to develop effective defensive measures against APTs, facilitates international collaboration in countering threat actors, and supports diplomatic initiatives to address cyber threats. The findings of the paper likely reinforce the importance of accurate attribution in APTs analysis, which informs defensive strategies, diplomatic actions, and law enforcement efforts in countering APTs attacks.

This project primarily contributes to the field of malware analysis, while drawing on threat intelligence and attribution to complement the APTs analysis. By focusing on analysing APTs-related

malware samples, this project aims to enhance understanding of APTs behaviours, capabilities, and mitigation strategies.

3.2 APT malware analysis

Malware analysis plays a significant role in APTs analysis, involving the examination of APTs-related malware samples. Static and dynamic analysis, reverse engineering, and code analysis are conducted to understand the functionality, capabilities, and evasion techniques employed by these threats. This analysis helps to identify the tools and methods used by APTs actors to compromise systems and maintain their persistence, as well as to understand the tactical techniques and procedures used by different APTs groups.

Liras, Soto, and Prada [43] propose a framework for data-driven APTs-related malware discrimination. The main goal of the framework is to identify the most discriminatory features to distinguish malware samples that belong to their campaigns from generic non-APTs malware samples. The framework uses features of three types of analysis: static, dynamic and network-related (only the country of origin of the traffic generated by the sample).

Machine learning has emerged as a powerful tool for APTs analysis. By applying machine learning algorithms and models, it becomes possible to automatically detect and classify APTs-related activities, identify patterns and anomalies in large datasets, and generate actionable intelligence. Deep learning models, in particular, have shown promise in malware analysis of these threats by directly processing disassembled binary functions and learning complex representations of APTs behaviour. Building upon the capabilities of deep learning in APTs analysis, the next section sets the stage for the concept of binary code similarity.

3.3 Binary Code Similarity Detection

Binary Code Similarity Detection (BCSD) refers to the task of comparing two or more pieces of binary code, quantifying the similarity between them. It plays a crucial role in various domains, including software analysis, reverse engineering, malware detection, plagiarism detection, and software clone detection. Binary code similarity analysis provides information on code reuse, code evolution and potential security vulnerabilities, making it a powerful tool for APTs malware analysis.

Problem Overview

When comparing binary code from different code fragments, we have to pay special attention to the process of obtaining this code: the compilation process. This process introduces various transformations that modify the representation of binary code derived from the same source code. These transformations can lead to semantically equivalent but different binary programs. Figure 3.1 illustrates several factors that can be modified by an author to produce such variations. Some of these factors are intrinsic to the standard compilation process, such as the choice of compiler or optimization level, which optimize the binary code for efficiency. Other factors are associated with the target platform, such as the operating system or CPU architecture, which influence the instruction set and structure of the binary code. Additionally, deliberate obfuscation techniques, including control flow and data flow obfuscation, can further obscure the original code and generate polymorphic variants [46].

An important characteristic of binary code similarity approaches is their ability to detect similarities among binary code derived from the same source, despite these transformations. The robustness of a binary code similarity approach is determined by its capability to handle various compilation and obfuscation transformations while accurately detecting similarity [46].

Extended Compilation Process

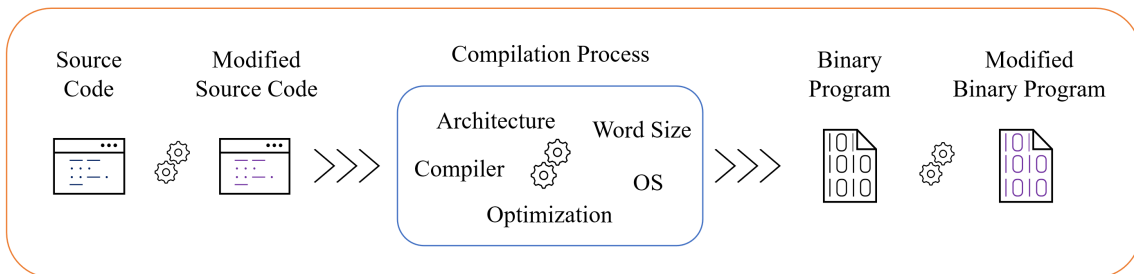


Figure 3.1: The image showcases the various stages involved in the compilation process. Each gear symbolizes a specific point in the process, where any alteration results in a distinct binary code. The term "Modified" refers to optional changes that can be applied to the code, serving as a means of introducing obfuscation techniques.

Furthermore, the comparison of the code can be done from different angles, the choice of which is at the expense of the purpose of the research. Following the study by *Haq and Caballero* [46], next sections summarize the key characteristics that define the nature of the comparison

Comparison Type

The comparison type determines the level of similarity being assessed between two pieces of binary code. It can be categorized as identical, similar, or equivalent.

- **Identical:** This type of comparison determines whether two binary code pieces are exactly the same, including all instructions and attributes.
- **Similar:** Two pieces of binary code can be considered similar if their syntax, structure, or semantics are similar. It sits between syntactic and semantic similarity.
- **Equivalent:** The equivalent comparison determines if two binary code pieces have the same semantics: if they have the same functionality or perform the same operations, regardless of the implementation details.

Granularity

Granularity refers to the level of detail at which the comparison is performed, determining the size and composition of the code pieces being compared. Different granularities include:

- **Instructions:** The comparison is performed at the finest level, focusing on individual instructions within the code.
- **Basic Blocks:** The comparison is conducted at a higher level, considering groups of instructions that form basic blocks within the code.
- **Functions:** The comparison occurs at the highest level, taking into account entire functions within the code.

Number of Inputs

The number of inputs defines the comparison scenario, specifying how many code pieces are involved in the comparison process. The scenarios can be categorized as:

- **One-to-One:** A single pair of binary code pieces is compared.
- **One-to-Many:** One binary code piece is compared against multiple others.

- Many-to-Many: Multiple binary code pieces are compared with each other, forming pairwise comparisons among all possible combinations.

By considering these comparison characteristics, binary code similarity analysis can be tailored to different scenarios and provide a comprehensive analysis of APTs malware.

Evolution

The following section presents an overview of the evolution and different approaches to BCSD, building upon the work done by Wang, Qu, Katz, et al. [47] and Haq and Caballero [46].

Traditional BCSD Approaches

Analogous to malware analysis, BCSD can be categorized into static and dynamic methods, depending on the source of code feature extraction.

Static methods for BCSD focus on identifying structural differences in binary code. These methods analyse the binary code at a static level, considering the arrangement and characteristics of instructions and basic blocks. Approaches such as KamIn0[48], Tracy [49] and BinSequence [50] fall into this category. They leverage techniques like categorized operands or instructions, instruction sequences, or graph/tree edit distance of basic block expression trees to measure the similarity between binary functions. However, these static methods often have limitations in capturing the semantics and relationships between instructions, resulting in lower accuracy.

Dynamic analysis methods, on the other hand, measure binary code similarity by analysing the runtime behaviour of the code. They assume that logically similar code exhibits similar runtime behaviour. Techniques like BinHunt [51], iBinHunt [52], and Genius [53] employ dynamic analysis by utilizing graph matching algorithms based on CFGs graph-isomorphism theory. These methods compare the similarity of binary functions by examining their runtime behaviour. However, dynamic methods tend to be computationally expensive and may not be suitable for large-scale BCSD due to long-running times.

ML BCSD Approaches

Inspired by the advancements in Natural Language Processing (NLP) and deep learning techniques, machine learning approaches have been applied to BCSD, aiming to capture both the structural and semantic information of binary code functions.

ML-based BCSD approaches utilize real-valued vectors, called embeddings, to represent the semantic information of binary functions. These approaches leverage deep learning algorithms to learn the embeddings and measure the similarity between different binary functions based on vector distances.

Many ML-based BCSD methods employ Siamese networks, such as SAFE [1], Gemini [54], Bin-Shot [55], and OrderMatters [56]. These methods require ground-truth mappings of equivalent binary functions for training. They use deep learning models, including Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM) networks, or Graph Neural Networks (GNNs), to learn the embeddings of binary functions or basic blocks. By computing vector distances in the learned embedding space, these approaches approximate the similarity between binary functions.

Some ML-based approaches, like Asm2Vec [57], have explored unsupervised learning for BCSD. Asm2Vec generates instruction sequences using Control-Flow Graphs (CFGs) but does not rely on ground-truth mappings. It employs unsupervised algorithms to learn the embeddings of binary functions. However, the performance of unsupervised learning methods may not be as good as supervised learning approaches.

ML-based BCSD approaches offer scalability and computational efficiency, making them suitable for large-scale BCSD tasks. However, challenges remain in accurately capturing both the structural and semantic information of binary functions, as some approaches neglect certain aspects such as instruction semantics or inter-basic block instruction co-occurrence. Future research in this field aims to address these limitations and further improve the accuracy and effectiveness of BCSD techniques.

3.4 SAFE

In this section, we introduce SAFE: Self-Attentive Function Embeddings for Binary Similarity, a machine learning-based framework that we utilize in this thesis to generate embeddings for the functions of different binaries in our dataset. SAFE was proposed by *Massarelli, Di Luna, Petroni*, et al. in 2019 [1] as a novel solution to the binary similarity problem.

The focus of the SAFE is a specific version of the binary similarity problem, where two binary functions are considered **similar** if they result from compiling the same original source code with different compilers. SAFE employs a binary code lookup to compute a query from **one** binary function to **many** other binary functions.

The SAFE approach leverages self-attention mechanisms, inspired by the field of natural language processing, to learn embeddings that capture both the structural and semantic information of binary code functions. By representing binary code functions as real-valued vectors, known as embeddings, SAFE aims to measure the similarity between different functions based on vector distances.

SAFE Overview

The SAFE framework consists of a two-phase embedding model, comprising a word2vec model and a Self-Attentive Neural Network, responsible for generating embeddings. This integrated approach enables the effective analysis of binary similarity by generating function embeddings that encompass both structural and semantic information. The overall architecture of the SAFE framework is depicted in Figure 3.2.

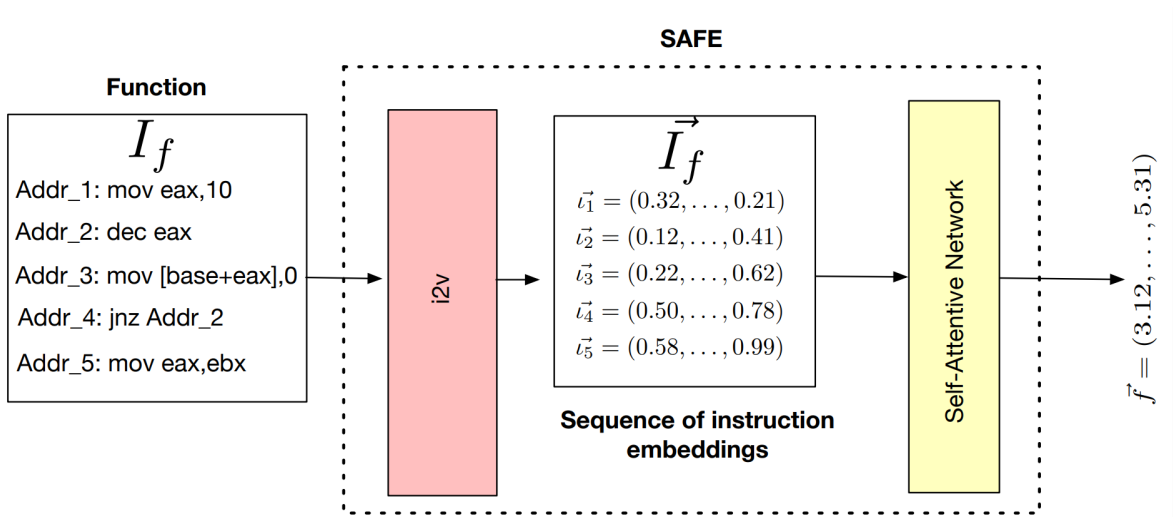


Figure 3.2: Architecture of SAFE, extracted from the work of Massarelli, Di Luna, Petroni, et al. [1].

In the first phase, the i2v component maps each function instruction to a real-valued vector using the word2vec model. This model is trained with a large corpus of instructions, leveraging the instruction embedding model, i2v. The skip-gram method is employed, where the current instruction is used to predict the surrounding instructions [58]. The output of the word2vec model is a vocabulary of instruction vectors that can be used to represent any binary function as a sequence of vectors.

The second phase involves the Self-Attentive Neural Network, which is based on a network proposed by Lin, Feng, Santos, et al. [59]. The neural network is used to encode the sequence of in-

struction vectors into a single function vector that captures both the structural and semantic information of the function. The Self-Attentive Neural Network is composed of an encoder layer, and an attention layer. The encoder layer applies a bidirectional recurrent neural network with gated recurrent units (GRU) [60] to process the sequence of embedded vectors and produce hidden states that encode the context of each instruction. The attention layer applies a self-attention mechanism to compute an attention score for each hidden state based on its relevance to the function representation. The attention scores are then used to compute a weighted sum of the hidden states. The resulting vector is flattened using a two-layers fully connected network with ReLU activation, resulting in a single function vector of size 100 that represents the function [1].

The data used to train the SAFE model consists of a large corpus of disassembled binaries from two training corpora: one for the instruction set of ARM and another for AMD64. The training corpora were built by disassembling multiple UNIX executables and libraries with IDA PRO [27]. To ensure uniqueness and avoid duplication, a duplicate detection mechanism was used by computing a hash of all function instructions. The data is preprocessed to extract the instruction mnemonics and operands from the binary code and to remove irrelevant or noisy information. The number of instructions within each function is truncated to a maximum value of $m = 150$. This truncation strikes a balance between training time and accuracy, as the majority of functions in the datasets fall below this threshold (over 90% of the functions) [1].

Related Work on BCSD

In addition to the SAFE model, several state-of-the-art models have been analysed for the task of generating embeddings in the field of Binary Code Similarity Detection (BCSD). In this subsection, we provide a brief review of two recent models: jtrans and BinShot.

jtrans [47] is a Transformer-based approach that utilizes self-attention mechanisms to capture both sequential and branching information (jumps) in binary code functions. It extends the Transformer-based language model BERT [61] by incorporating control flow information. jtrans introduces a novel jump-aware tokenization scheme that splits instructions into tokens based on jump types and operands. It also employs a jump-aware positional encoding scheme to encode the relative positions of tokens within functions and across different branches. The token vectors are then fed into the transformer model to obtain function embeddings, which are compared using cosine similarity. Experimental results demonstrate that jtrans outperforms state-of-the-art approaches on the BinaryCorp dataset, which is a large-scale and diverse benchmark for BCSD released on the same work [47]. However, we believe it is important to note that when jtrans is evaluated with different models, SAFE's results come close to the performance obtained by jtrans, despite being

published four years apart, indicating SAFE's robustness in tackling the BCSD problem.

BinShot [55] is another Transformer-based approach that leverages Graph Neural Networks (GNNs) to capture the structural information of binary code. It represents binaries as CFGs, where nodes correspond to basic blocks and edges represent control-flow transitions. BinShot applies a GNN to learn node embeddings that incorporate both local and global features of the CFGs. Additionally, it utilizes a modified version of BERT [61] to learn instruction embeddings that capture the semantic information of basic blocks. By combining the node embeddings and instruction embeddings, BinShot obtains a unified representation of the binary code. However, BinShot's evaluation section is limited in scope, and it does not compare its performance with state-of-the-art models like SAFE or jtrans. Furthermore, BinShot uses a smaller training dataset of 1,400 binaries compared to jtrans dataset of around 48,000 binaries.

BinShot [55], is a Transformer-based approach that utilizes Graph Neural Networks (GNNs) to capture the structural information of binary code. It represents binaries as Control-Flow Graphs (CFGs), where nodes represent basic blocks and edges represent control-flow transitions. BinShot applies a GNN to learn node embeddings that incorporate both local and global features of the CFGs. Additionally, it employs a modified version of BERT [61] to learn instruction embeddings that capture the semantic information of basic blocks. By combining the node embeddings and instruction embeddings, BinShot obtains a unified representation of the binary code.

It is worth noting that when comparing jtrans with different models, the results obtained by SAFE approach are comparable, despite being published four years apart. This indicates the robustness of SAFE in tackling the BCSD problem, suggesting its effectiveness as an alternative solution. BinShot's evaluation section has a limited scope and does not compare its performance with state-of-the-art models like SAFE or jtrans. Furthermore, BinShot uses a smaller training dataset of 1,400 binaries compared to the jtrans dataset, which consists of around 48,000 binaries.

It is important to highlight that both jtrans and BinShot rely on the paid version of IDA as a disassembler, which restricts accessibility and availability for researchers and practitioners. This limitation goes against the principles of openness and reproducibility in the field.

In conclusion, jtrans and BinShot are two recent models that have shown promising results in the field of BCSD. While jtrans outperforms existing approaches on the BinaryCorp dataset, the performance of SAFE comes close to that of jtrans in certain evaluations. BinShot, lacks comprehensive comparisons with state-of-the-art models. The reliance on paid disassemblers in both approaches hinders their accessibility and reproducibility.

4.

MAPTER

This chapter embarks on a comprehensive exploration of MAPTER, Mapping [APTs](#) through Embedding Representation, the purposed framework that presents a fresh approach to analysing [APTs](#). The main objective of this chapter is to introduce the framework and provide a clear understanding of the methodology that will be employed throughout our study. By following this methodology and conducting detailed case studies, our aim is to enhance our knowledge of [APTs](#) groups and contribute to the development of effective defence strategies against sophisticated cyber threats.

To commence, an overview of the proposed framework is presented, highlighting its key components and the rationale behind its design. Subsequently, we delve into different subsections that guide the application of MAPTER, outlining a step-by-step process that researchers and practitioners can follow to effectively utilize the framework in their investigations. We also provide a case study to demonstrate the effectiveness of MAPTER in the field of [APTs](#) analysis.

4.1 MAPTER Overview

In this section, we provide an overview of MAPTER, highlighting its main features and the purposed adaptation. In pursuit of a deeper understanding of Advanced Persistent Threats, this thesis aims to propose a fresh approach to their analysis. To accomplish this objective, we leverage MAPTER: a flexible framework designed to adapt to diverse environments and incorporate advancements in Machine Learning and malware analysis for [APTs](#) analysis. Utilizing a modular approach, each step of the framework can be easily customized without the need to reinvent the entire system.

Figure [4.1](#) depicts the overall structure of the MAPTER framework, serving as a visual representation of its components and workflow. In the following sections, we provide a concise description of each step involved in MAPTER.

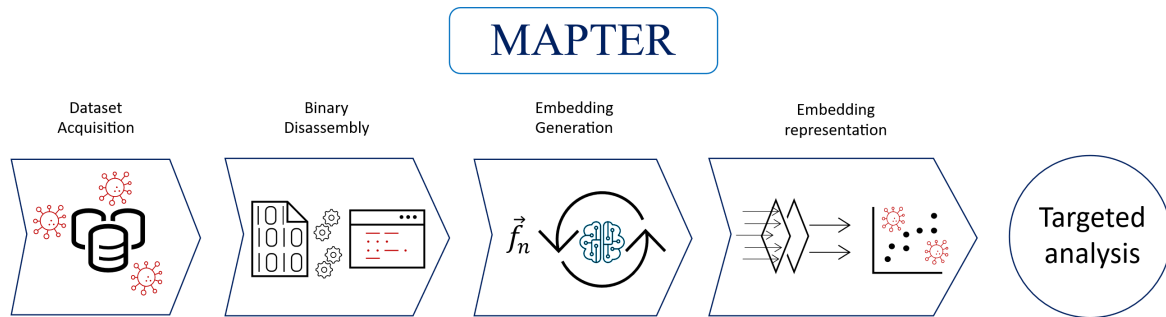


Figure 4.1: MAPTER different steps overview

1. **Dataset Acquisition.** Obtain a sufficiently large dataset of advanced persistent threats malware samples. The dataset should encompass a variety of binaries to ensure the analysis provides generalizable conclusions.
2. **Binary Disassembly.** Utilize a disassembler tool to extract the assembly code from the malware binaries. This is achieved using a disassembler tool selected based on project requirements and constraints, taking into account its compatibility with the embedding generator model. The chosen disassembler tool should effectively assist in extracting the desired code fragments from the assembly code, enabling the generation of embeddings for further analysis.
3. **Embedding Generation.** Generate vector representations of the extracted code using machine learning techniques for binary analysis. The embeddings are numerical representations that capture the semantic and syntactic features of the code. The embeddings can then be used for various tasks such as clustering, classification, or similarity search.
4. **Embedding representation.** As the generated embeddings typically have high dimensionality and are not easily visualizable, apply dimensionality reduction techniques to visualize the embeddings effectively is recommended. This step aids in understanding the relationships and patterns within the code embeddings.
5. **Targeted analysis.** This step focuses on conducting a comprehensive analysis of a targeted concept within one or multiple [APTs](#), enabling a thorough examination and a deep understanding of the subject.

In addition to introducing a general approach, this thesis places emphasis on adapting MAPTER by integrating advanced tools to conduct an extensive study on advanced persistent threats. By incorporating state-of-the-art tools, the framework leverages the latest advancements in [APTs](#)

analysis, carefully chosen for their effectiveness, reliability, and compatibility with the proposed methodology. Furthermore, an open-source approach is embraced, promoting collaboration among the research community and ensuring transparency throughout the analysis process, thereby benefiting researchers and practitioners alike. The following sections describe in depth each of the steps covered by MAPTER.

Dataset Acquisition

In the process of selecting an appropriate dataset for MAPTER, we conducted an examination of various [APTs](#) datasets available in the research community. Our aim was to identify a dataset that aligns well with our research objectives and provides the necessary characteristics for comprehensive analysis.

The ideal [APTs](#) dataset for our study should possess the following characteristics:

1. **Size and Diversity:** The dataset should encompass a substantial number of malware samples related to a diverse range of [APTs](#) groups. This diversity is crucial for capturing the varying techniques, tactics, and procedures employed by different [APTs](#) actors.
2. **Attribution Information:** It is desirable for the dataset to include information about the [APTs](#) groups responsible for each malware sample. This attribution information allows for a more accurate understanding of the characteristics and behaviours associated with specific [APTs](#) groups.
3. **Coverage of Nation-States:** The dataset should encompass malware samples attributed to a wide range of nation-states involved in [APTs](#) campaigns. This ensures a broader representation of geopolitical factors and motivations behind [APTs](#) activities.
4. **Temporal Relevance:** A dataset that includes recent and up-to-date samples is essential for capturing the current trends and dynamics of [APTs](#) attacks. This ensures that the analysis is conducted on malware samples that are more representative of the evolving threat landscape.
5. **Availability and Accessibility:** Easy accessibility to the dataset is crucial for reproducibility and collaboration within the research community. Open availability, such as being publicly accessible or easily obtainable through established channels, promotes transparency and facilitates further research.

After a comprehensive analysis, we have chosen to utilize the APTMalware dataset [62] due to its extensive size and rich content. The APTMalware dataset provides a substantial collection of data that is well-suited for our research objectives. It encompasses over 3,500 malware samples that are related to 12 APTs groups, which allegedly are sponsored by five different nation-states. A summary of the APTMalware dataset is presented in table 4.1.

This dataset covers a wide range of malware families and attack techniques that can be used to identify and characterize APTs activities. Moreover, the APTMalware dataset is publicly available on GitHub, which makes it easy to access and use for research purposes. Additionally, the APTMalware dataset has been analysed and used for benchmarking different machine learning approaches, which demonstrates its relevance and usefulness for the academic community [63]–[67]. By leveraging this dataset, we aim to obtain meaningful insights into the characteristics and behaviour of APTs [62].

However, the APTMalware dataset also has some limitations and challenges that need to be addressed. One of them is that the dataset may not reflect the current trends and dynamics of APTs attacks, as it was created in 2018 and may not include the latest malware samples and variants. Moreover, the dataset may not be representative of all APTs groups and nation-states, as it only covers 12 APTs groups and five nation-states, while there are more actors involved in APTs campaigns.

Table 4.1: The table illustrates how the samples of the APTMalware dataset are distributed [62].

Country	APT Group	Family	No. of Binaries
China	APT 1	-	405
China	APT 10	i.a. PlugX	244
China	APT 19	Derusbi	32
China	APT 21	TravNet	106
Russia	APT 28	Bears	214
Russia	APT 29	Dukes	281
China	APT 30	-	164
North-Korea	DarkHotel	DarkHotel	273
Russia	Energetic Bear	Havex	132
USA	Equation Group	Fannyworm	395
Pakistan	Gorgon Group	Different RATs	961
China	Winnti	-	387
Total	12		3,594

Binary Disassembly

The *Binary Disassembly* step plays a crucial role in our methodology, as it is responsible for extracting the assembly code of the binary. By disassembling the binaries, we can obtain the low-level representation of the code, which is essential for further analysis and embedding generation.

When selecting a suitable disassembler tool for the application of MAPTER, it is essential to conduct a comparative analysis of the available options in the research community. The objective is to find a disassembler that aligns well with the research goals and offers the necessary functionality for extracting the desired code fragments from binary files.

During the selection process, it is advisable to consider open-source disassembly tools and libraries that are widely utilized in the field of reverse engineering. These open-source tools provide transparency, reproducibility, and flexibility for research purposes. They can be customized and adapted to specific requirements, ensuring a tailored disassembly process. Additionally, the availability of these tools encourages collaboration and knowledge sharing within the research community.

As noted before, the selected tool should have the capability to efficiently extract the desired code fragments from the assembly code, enabling the generation of embeddings. In our research, we have specifically focused on **working with functions**, as they are widely used elements for generating code embeddings. By concentrating on the functions, we aim to generate better quality embeddings and to gain deeper insights into the behaviour and characteristics of the analysed malware samples.

After careful consideration, we chose to utilize Radare2 as our disassembler tool. This decision was based on several factors, including its robust set of features, open-source nature, compatibility with the function extraction process and with *SAFE*, the model utilized for generating the embeddings from the code. Its integration with Python allowed for seamless incorporation into our workflow.

With Radare2 as our chosen tool, we proceeded to extract functions from the binary files, extracting different associated features for further examination. We briefly explain each feature as follows:

- **Size:** The "size" feature refers to the size of the function in bytes. It indicates the total number of bytes occupied by the function's instructions and data within the binary. The functions in the group exhibit significant variation in size, with some functions being much

larger or smaller than the average. This suggests that the functions within this group are relatively larger in, containing a significant number of instructions and potentially more complex logic

- **Stackframe:** The "stackframe" feature represents the size of the stack frame used by the function. It indicates the amount of stack space allocated by the function for storing local variables and other data.
- **Nbbs:** "Nbbs" stands for the number of basic blocks. A basic block is a sequence of instructions without any branch or jump instructions within it. The "nbbs" feature represents the count of such basic blocks in the function. This suggests that the functions in APT19 have a moderate level of complexity with a reasonable number of control flow branches.
- **Edges:** "Edges" refers to the number of control flow edges in the function. It represents the number of connections or transitions between basic blocks within the function. Higher values indicate more branching and potentially more complex control flow patterns.
- **Callrefs:** The "callrefs" feature represents the number of function call references within the function. The functions in the group make a moderate number of calls to other functions. Functions with a higher number of call references may have more dependencies on other functions or perform complex operations involving function calls.
- **Datarefs:** "Datarefs" refers to the number of data references within the function. It represents the count of memory locations accessed or modified by the function. The low value suggests that these binaries might not heavily rely on external data manipulation.
- **Codexrefs:** The "codexrefs" feature represents the number of cross-references to the function's code. It indicates how many times the function is referenced or called from other parts of the code.

It is worth noting that we deliberately excluded the extraction of subroutines from the binaries. This approach was adopted to avoid introducing unnecessary noise that may arise from common subroutines across different binaries [26]. By focusing solely on relevant functions, our analysis process aimed to enhance accuracy and effectiveness.

By employing Radare2 as our chosen binary disassembler, we successfully extracted valuable insights and efficiently obtained functions, which played a crucial role in facilitating the generation of embeddings, as described in the next section.

Embedding Generation

Generating numerical representations, or embeddings, for disassembled binary code is a critical challenge in binary similarity analysis. These embeddings allow us to compare code using simple and efficient geometric operations, such as cosine similarity or Euclidean distance. Additionally, embeddings provide the advantage of graphical representation, enabling visual exploration and analysis of the code, which enhances our understanding of its characteristics and relationships.

To select an appropriate model for embedding generation, it is essential to conduct a comprehensive analysis of the state-of-the-art techniques in this field. This process should be similar to the investigation outlined in Chapter 3, aiming to achieve efficient and accurate embedding generation for disassembled binary code.

We have selected SAFE as our preferred approach for embedding generation due to its alignment with open-source principles and its ability to deliver optimal performance. SAFE is a deep learning-based model specifically designed to extract embeddings from disassembled binary functions.

One of the compelling aspects of SAFE is its integration with the open-source disassembly software Radare2, which offers multiple benefits. This integration ensures accessibility and enables seamless integration into our workflow. Furthermore, it showcases impressive speed and resource efficiency, allowing us to process large volumes of binary code efficiently.

In addition to its integration with Radare2, SAFE has consistently demonstrated superior performance compared to many current models in the field. Despite being proposed in 2018, its effectiveness has been validated in various studies, including [47]. This track record of good performance further reinforces our decision to use SAFE to generate function embeddings.

To incorporate SAFE into our project, we utilize the pre-trained models available in the *SAFEtorch* GitHub repository [68] which is the PyTorch [69] implementation of SAFE. We employ both the word2vec model, which transforms instructions into vectors, and the *SAFEtorch* model, which further transforms the vectors of each function into embeddings.

Our goal is to generate the embeddings of all the functions of the binaries in a dataset. Therefore, we modify the `test.py` from the repository to iterate over all the functions of each binary and to store the generated embeddings in separated files.

In summary, *SAFEtorch* is an ideal tool for generating function embeddings in our research. It leverages the power of Radare2 to produce embeddings that are both fast and accurate. It also

outperforms existing solutions in terms of predictive accuracy. The fusion of speed and accuracy makes SAFE a formidable tool for generating feature embeddings in our research.

Dimensionality Reduction

To facilitate the analysis and visualization of the embeddings obtained from the previous step, it is essential to address the challenge of high dimensionality. Dimensionality reduction techniques play a vital role in reducing the complexity of the data while preserving its meaningful structure and patterns.

We studied different methods of dimensionality reduction and found that combining [Principal Component Analysis \(PCA\)](#) and [t-Distributed Stochastic Neighbour Embedding \(t-SNE\)](#) improved results, as reflected in Section 2.3 [42]. By integrating both techniques, we were able to enhance the separation of different clusters and gain a more comprehensive understanding of the relationships within the dataset.

For the implementation of [PCA](#) and [t-SNE](#), we utilized the Scikit-learn [70] library in Python, which offers a convenient and efficient implementation of these algorithms. We conducted experiments with different parameter settings to optimize the performance of both [PCA](#) and [t-SNE](#) and to achieve meaningful visualizations of the function embeddings.

Since the function embeddings produced by SAFE consist of 100 dimensions, we applied [PCA](#) with $n_components = 40$. This reduction step allowed us to capture the most significant information while reducing computational complexity. Subsequently, we applied [t-SNE](#) with $n_components = 2$ to obtain a 2D representation of the embeddings. This choice of 2D representation enables a better analysis and interpretation of the data compared to 3D representations. To determine the perplexity value, we carefully considered the parameter's impact and conducted various visualization experiments with our adapted version of MAPTER, ultimately selecting a fixed $perplexity = 50$.

To visualize the resulting features, we employed Plotly [71], a versatile and adaptable visualization library. Plotly enables the incorporation of additional dimensions, such as colour, size, or shape, into the visualization, enhancing our insights into the characteristics of the [APTs](#) malware samples.

By applying dimensionality reduction techniques and utilizing Plotly for visualization, we can effectively explore and analyse the function embeddings, gaining valuable insights into the structure and behaviour of the [APTs](#) malware samples. These visualizations allow us to uncover hidden

patterns, cluster relationships, and further our understanding of APTs activities.

4.2 MAPTER: Targeted analysis

The proposed MAPTER in this thesis serves as a versatile tool that can be utilized for various analyses and purposes. Its modular and adaptable nature enables the exploration of multiple use cases, offering a range of applications and insights into the Advanced Persistent Threats analysis. Some of the ideas to which it can be applied range from enabling the identification of common characteristics, the exploration of subgroups inside APTs, or even detection of shared code or relationships between different APTs. By leveraging the framework's modules and techniques, analysts can gain valuable insights into the behaviour, tactics, and techniques of APTs, enabling more proactive and effective defence strategies against these sophisticated cyber threats.

Analyse APT groups

To assess the effectiveness of MAPTER, a focused analysis has been devised to **examine individual APTs groups**. This targeted approach aims to provide valuable insights into the behaviour and characteristics of specific APTs groups. The following outlines the methodology for this use case, which is summarized in Figure 4.2. Some of the key advantages and use cases of this analysis are described in the following paragraphs.

By utilizing the purposed target analysis of an individual APT group, analysts can identify distinct subgroups within an APTs. APTs often consist of multiple units or teams, each with their own objectives and methodologies. Through correlation analysis and pattern recognition techniques, MAPTER can help to uncover subtle variations and distinct characteristics within the APT group. This knowledge aids in understanding the internal structure, hierarchy, and specialization within the APTs, allowing analysts to tailor their defensive strategies accordingly.

This target analysis with MAPTER provides the means to verify the attribution of samples within an APT group. By comparing the function embeddings and other features of different samples, analysts can determine if they belong to the same APT group or if there are variations that suggest the involvement of different actors or subgroups. This verification process strengthens the accuracy of APTs attribution, which is crucial for effective threat intelligence and response.

Another valuable application is the tracing of malware adaptation across different campaigns conducted by an APT group. APTs frequently modify their malware to suit specific targets, objectives,

or operational requirements. By analysing the function embeddings and correlating them with campaign data, analysts can identify commonalities and variations in the malware used across different campaigns. This information sheds light on the evolution and adaptability of the [APT](#) group's toolset and provides insights into their strategic decision-making processes.

MAPTER aids analysts in gaining a better understanding of the construction of malware within an [APT](#) group and the tools they rely on. By examining the function embeddings, code patterns, and other characteristics of malware samples, analysts can discern common design patterns, shared code snippets, or even the reuse of specific tools or frameworks. This analysis enables analysts to grasp the technical capabilities, development practices, and tool preferences of the [APT](#) group. Such insights are invaluable for enhancing threat intelligence, building more accurate behavioural profiles, and developing effective defensive measures.

In summary, the targeted analysis of individual [APTs](#) groups using MAPTER offers significant advantages to analysts. It facilitates the identification of subgroups within an [APTs](#), verifies sample attribution, traces malware adaptation across campaigns, and enhances the understanding of malware construction and tool usage among others. By leveraging the capabilities of this adaptation of MAPTER, analysts can gain deeper insights into the behaviour, tactics, and techniques of [APTs](#) groups, leading to more effective threat detection, attribution, and response strategies.

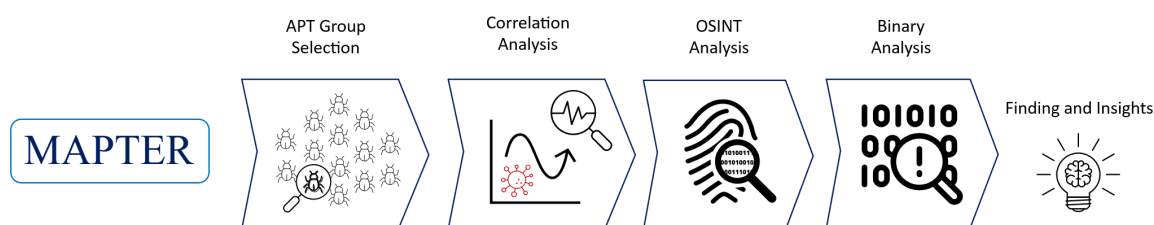


Figure 4.2: MAPTER adaptation steps overview

1. **Apply MAPTER.** Start by applying the general framework to the dataset acquired for analysis.
2. **APT Group Selection.** In this step, an [APT](#) group is selected from the dataset used on MAPTER. The selection process involves considering [APTs](#) groups that have characteristics that can satisfy the research objectives. For this specific adaptation, we recommend electing a group that have sufficient representation within the dataset. This ensures that the analysis is conducted on a group that provides significant insights into the characteristics and behaviour of the [APT](#).

3. **Correlation Analysis.** The correlation analysis step involves examining the filtered functions and binaries based on the chosen criteria. In this specific work, the analysis focuses on identifying correlations and relationships between different binaries by plotting the embeddings of the main functions and entry points. The closeness of points in the visualization indicates similarities and potential connections between the binaries. By performing correlation analysis, it becomes possible to uncover patterns, similarities, and potential dependencies between the analysed binaries, providing valuable insights into the structure and behaviour of the selected **APT** group.
4. **OSINT Analysis.** Conduct **OSINT** analysis to search and gather additional information about the correlated samples from online sources, such as reports and VirusTotal. This can help us validate our findings, enrich our knowledge, or discover new insights about the **APT** group's activities, tactics, techniques, or procedures. The **OSINT** analysis complements the technical analysis by providing a broader context and a deeper understanding of the **APT** group's motivations and strategies.
5. **Binary Analysis.** Conduct a detailed examination and exploration of the filtered samples using reverse engineering tools. Reverse engineering techniques, such as disassembly, are employed to analyse the binaries at a deeper level. This process involves examining the code, identifying functionalities, understanding obfuscation techniques. The binary analysis step aims to extract comprehensive information about the **APT** group's techniques, capabilities, and potential targets.
6. **Findings and insights.** Summarize the findings obtained from the targeted analysis of the selected **APT** group, drawing conclusions from them. Provide a comprehensive overview of the group's characteristics, tactics, and behaviours based on the correlation analysis, **OSINT** analysis, and binary analysis conducted. The conclusions draw upon the collected evidence and observations to provide a deeper understanding of the **APT** group's activities and to highlight any significant findings or patterns identified during the analysis process.

By following this framework and conducting case studies, the effectiveness of the proposed framework can be tested and evaluated.

APT Group Selection

In the *APT Group Selection* step, an **APT** group is carefully chosen from the dataset for targeted analysis. The selection process involves considering several factors, including the availability of

sufficient representation of the [APT](#) group within the dataset and a balanced number of samples to allow us to draw conclusions when applying the proposed targeted analysis.

The primary objective of this step is to identify an [APT](#) group that can provide valuable insights into the characteristics and behaviours of [APTs](#). The selected group should have a substantial but not excessive number of associated malware samples in the dataset, allowing for a comprehensive analysis of their techniques, tactics, and procedures within the time constraints associated with a Master's thesis.

Once the [APT](#) group is selected, it serves as the focal point for the subsequent analysis steps, such as *Correlation Analysis*, *OSINT Analysis*, and *Binary Analysis*. The findings and conclusions drawn from these steps are centred around the characteristics and activities of the chosen [APT](#) group, providing a focused and targeted analysis of their behaviour in the dataset.

Correlation Analysis

The *Correlation Analysis* step plays a crucial role in filtering and analysing the data to facilitate the subsequent steps of the analysis. Its ultimate purpose is to identify correlations and dependencies among the functions and binaries associated with the selected [APT](#) group. By doing so, it aims to lessen the amount of data to be analysed and promote a clearer understanding of the structure and behaviour of the [APT](#) group's malware.

The correlation analysis of this work is conducted based on main and entry point criteria. The main criterion involves examining the embeddings of the main functions of the binaries. By plotting the embeddings of these main functions, it becomes possible to identify similarities and patterns among the [APT](#) group's malware samples. Binaries with closely located points in the embedding space indicate a higher likelihood of sharing similar characteristics or being related.

Similarly, the entry point criterion focuses on the embeddings of the entry points of the binaries. The entry point is the location where the execution of the binary code begins. By analysing the embeddings of entry points, relationships between binaries that exhibit similar entry point behaviour can be identified.

In order to complete the findings, we purpose to pay special attention to the cases where the main and the entry point functions of different binaries overlapped in the embedding space. This allows us to directly correlate those binaries and then visualize and compare all their functions to spot similarities and differences and get useful insights for later steps.

In addition to the main and entry point criteria, we suggest plotting the embeddings of all binary functions to identify those that significantly differ from the others. By adjusting the plot to incorporate additional characteristics like function size, a more comprehensive analysis can be conducted. This aids in examining how binaries are constructed within the [APT](#) group and allows for drawing conclusions about their internal structure.

The main objective of the correlation analysis step is to reveal clusters, groups, or associations among the functions and binaries linked to the chosen [APT](#) group. By identifying these correlations, it becomes feasible to simplify the analysis by concentrating on specific areas of interest. Moreover, these correlations offer valuable insights into the connections between various malware samples, uncovering shared code, common techniques, or even distinct subgroups within the [APT](#) group. Overall, the *Correlation Analysis* plays a crucial role in guiding the subsequent steps of the analysis and providing a deeper understanding of the selected [APT](#) group.

OSINT Analysis

The *Open-Source Intelligence (OSINT) Analysis* step is designed to obtain additional information about the filtered binaries and highlighted functions derived from the previous *Correlation Analysis*. Its ultimate purpose is to gather relevant data from publicly available sources to enhance the understanding of the [APT](#) group and its associated malware.

[OSINT](#) analysis involves gathering information from various sources such as news articles, blogs, research papers, public reports or malware analysis engines. These sources can provide valuable context about the [APT](#) group, including their known targets, attack techniques, infrastructure, and any public attribution information.

During the [OSINT](#) analysis, it is crucial to cross-reference and verify the information obtained from multiple sources to ensure accuracy and reliability. The analysis may involve studying past campaigns and attacks attributed to the [APT](#) group, identifying their modus operandi, and understanding their historical development and evolution.

Additionally, [OSINT](#) analysis include examining the [APT](#) group's tools, malware samples, and Indicators of compromise that have been publicly reported or shared by cybersecurity organizations. This can help in understanding the technical capabilities of the [APT](#) group and identifying potential overlaps with the dataset used in the study.

The findings from the *OSINT Analysis* can provide valuable context and enrich the understanding of the [APT](#) group's behaviour and motivations. This information is used to complement the

technical analysis conducted in the *Binary Analysis* step, 4.2, and provide a more comprehensive picture of the [APT](#) group's activities.

Binary Analysis

The *Binary Analysis* step plays a crucial role in obtaining detailed insights and extracting valuable information from the filtered binaries and features highlighted in the previous *Correlation Analysis* and *OSINT Analysis*. Its primary objective is to delve deeper into the selected malware samples, uncovering differences, and gaining a better understanding of their functionality, behaviour, and potential capabilities.

To conduct the binary analysis, we purpose using Ghidra. Ghidra provides readable pseudocode representations of the underlying code. This facilitates the analysis of the malware's logic, structure, and individual functions. By utilizing Ghidra's capabilities, we can explore the binaries at a more granular level, examining their function calls, control flow, and data structures. This allows us to identify distinct patterns, algorithms, or techniques employed by the [APT](#) group in their malware samples.

The *Binary Analysis* step also enables us to compare similar binaries within the selected [APT](#) group. By identifying differences in the code, we can uncover variations in functionality, feature sets, or customization specific to certain targets or campaigns. This information provides valuable insights into the [APT](#) group's adaptability, level of sophistication, and potential evolution over time.

Furthermore, Ghidra allows us to annotate and document our findings, making it easier to share knowledge and collaborate with other researchers. By leveraging the collective expertise of the cybersecurity community, we can gain additional perspectives and collectively contribute to the understanding of [APT](#) groups and their activities.

In summary, the *Binary Analysis* step, powered by Ghidra, empowers researchers to perform in-depth analysis of the filtered binaries and highlighted features obtained from the *Correlation Analysis* and *OSINT Analysis*. By examining the pseudocode representations of the binaries, we can uncover differences, understand functionality, and identify potentially malicious behaviours. This step is instrumental in revealing the inner workings of the [APT](#) group's malware samples and contributes to the overall understanding of their techniques, capabilities, and potential impact.

Findings and Insights

The *Findings and Insights* step serves as the final step in the targeted analysis of a specific [APT](#) group within MAPTER. Its purpose is to synthesize and present the findings and insights gained throughout the entire analysis process, providing a comprehensive understanding of the [APT](#) group's activities, techniques, and potential implications.

Based on the results obtained from the *Correlation Analysis*, *OSINT Analysis*, and *Binary Analysis* steps, we can draw conclusions and make informed assessments about the targeted [APT](#) group. This includes identifying commonalities, distinguishing characteristics, and unique traits exhibited by the group's malware samples.

The *Findings and Insights* step enables us to consolidate the knowledge acquired during the analysis, highlighting the [APT](#) group's tactics, techniques, and procedures, as well as their potential motives and objectives. It allows us to evaluate the group's level of sophistication, the scale of their operations, and the potential impact on targeted entities or industries.

The *Findings and Insights* step also provides an opportunity to identify areas for further research and investigation. By pinpointing gaps in knowledge or areas of uncertainty, we can guide future studies and refine our understanding of [APTs](#) activities. Furthermore, the conclusions drawn from the analysis can inform defensive strategies, threat intelligence efforts, and the development of countermeasures against [APTs](#) attacks.

It is important to note that the conclusions presented are based on the available data, analysis techniques, and the specific scope of the targeted analysis. As the threat landscape evolves and new information becomes available, ongoing monitoring and analysis are necessary to stay abreast of emerging [APTs](#) trends and developments.

In summary, the *Findings and Insights* step serves as the culmination of the targeted analysis, integrating the findings from the preceding steps to deliver a comprehensive understanding of the targeted [APT](#) group. It enables us to draw conclusions, assess the group's capabilities and motives, and inform future research and defensive strategies in the ongoing effort to mitigate the threats posed by [APTs](#) activities.

5.

Results

The *Results* chapter presents the findings and outcomes of the research conducted using MAPTER for an individual [APT](#) target analysis. This chapter provides a comprehensive overview of the results obtained from each module of the framework, highlighting the key observations, discoveries, and insights gained throughout the analysis process. The presentation of results follows a logical sequence that aligns with the steps of MAPTER, ensuring clarity and reproducibility.

Within this chapter, we delve into the correlation analysis, which played a crucial role in identifying eight cases of binary code similarity. To enhance clarity and coherence, we have organized these correlated cases into separate sections. Within each section, we further distinguish the steps of [OSINT](#) analysis, binary analysis, and findings and insights, as proposed in the methodology.

This organizational structure enables us to provide a comprehensive and systematic presentation of the target analysis process. By detailing the steps taken in each correlated case, readers can follow the progression of the analysis and discern the importance of each step in contributing to the overall findings. Such organization ensures that the results are presented in a coherent manner, facilitating understanding and interpretation.

5.1 Applying MAPTER

In this section, we describe the application of the proposed adaptation of MAPTER. The analysis involved a series of steps, with each corresponding to a specific module within the MAPTER framework.

1. **Dataset Acquisition.** The APTMalware dataset [62] was selected as the primary dataset for this analysis due to several key factors. Firstly, it offers a large-scale collection of 3,594 malware samples. Secondly, the dataset encompasses diverse and rich content, representing 12

APT groups allegedly sponsored by five distinct nation-states. Thirdly, the dataset provides desirable temporal coverage, having been published in 2019. Moreover, its availability on GitHub ensures easy accessibility for research purposes. Finally, the dataset's relevance to the research objectives solidifies its suitability for conducting the analysis.

2. **Binary Disassembly:** The disassembly of the binary files was conducted using the Radare2 disassembly software. Radare2 was chosen due to its robust features, open-source nature, and compatibility with the SAFE model used for generating function embeddings. To ensure Python compatibility and extract relevant information, we made adaptations to the Radare2 software. This enabled us to extract code functions and their associated characteristics, such as offset, name or size among others. In order to streamline the correlation work and remove noise, we excluded subroutines functions from the extracted features. These repetitive and pre-defined methods did not provide significant insights for our analysis.
3. **Embedding Generation.** The function embeddings were generated using the SAFE model, a deep learning-based model specifically designed to extract embeddings of disassembled binary functions. The pretrained version of the SAFE model, implemented in PyTorch, SAFETorch, was utilized for this purpose. The code was adapted to iterate through all the files in the dataset, generating embeddings for each function, and storing them in a file for further analysis.
4. **Embedding representation.** The embeddings generated by the SAFE model, consisting of 100 dimensions, were subjected to dimensionality reduction techniques for better analysis and visualization. Principal Component Analysis and t-SNE were employed for this purpose. We have implemented both models in a pipe, a PCA with n_components=40 to capture the most significant information while reducing computational complexity. Subsequently, a t-SNE was applied with n_components=2 to obtain a 2D representation of the embeddings and a fixed perplexity=50, selected after performing several experiments.
5. **Targeted analysis.** The adaptation of the targeted analysis step purposed on this work focuses on analysing specific APT groups within the dataset. We have selected the APT19 and conducted the presented steps on Figure 4.2 to gain insights into the group's activities, relationships, and characteristics. The process followed is detailed throughout the remaining sections of this chapter.

Before delving into the results, it is essential to provide an overview of the hardware specifications used during the execution of the tool for operations such as malware feature extraction. The tool was executed on a server configuration comprising an Intel i9 processor, 64 GB of RAM, a 1 TB

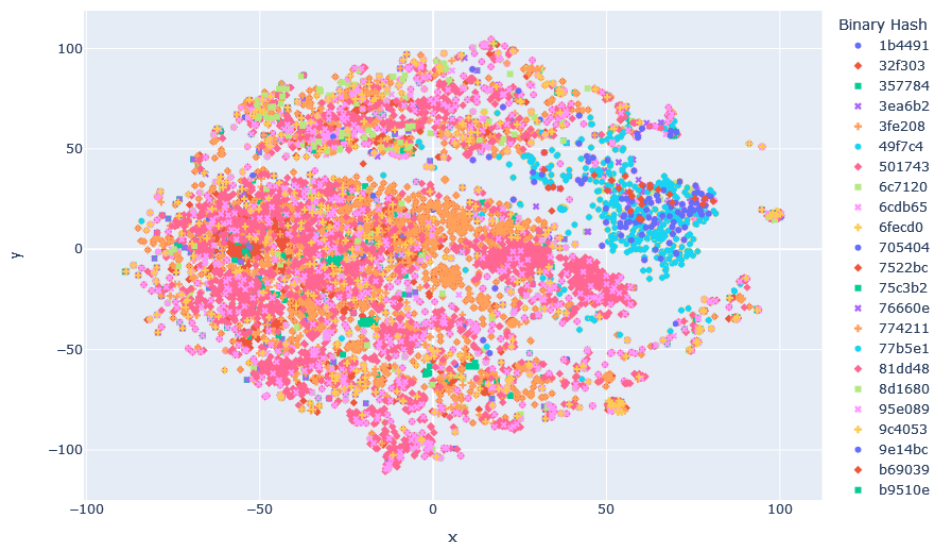
SSD, an Nvidia RTX 3080 graphics card, and the Ubuntu 20.04 operating system. However, for smaller, safer and less computationally demanding operations, the tool was run on the student's personal computer.

The Table 5.1 presents a comprehensive summary resulting from the application of MAPTER to the selected dataset. It includes the names of the APT groups under analysis, the total number of binaries examined, the number of functions extracted from these binaries, and the corresponding execution time. The execution time denotes the duration between the feature extraction process carried out using Radare2 and the subsequent generation of SAFE embeddings. This metric provides insights into the time required for processing and analysing the binaries associated with each APT group.

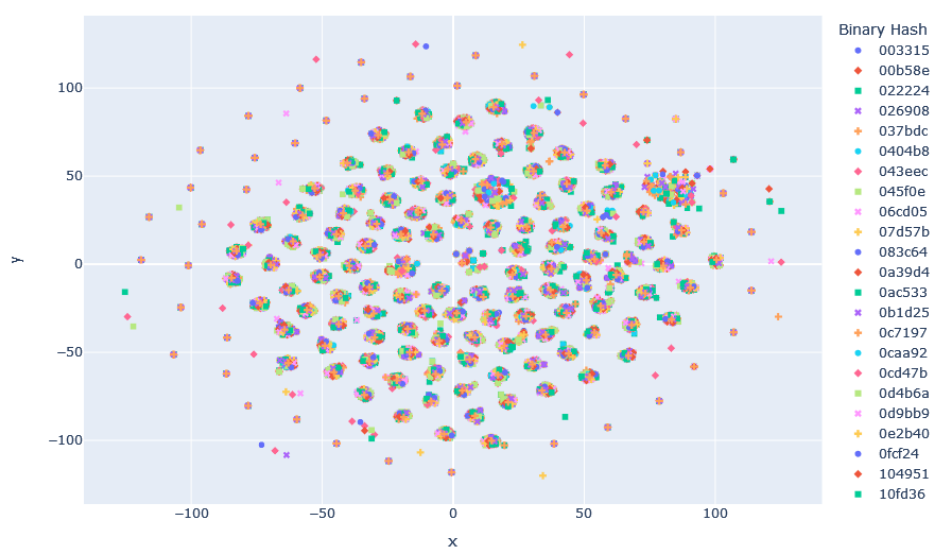
Table 5.1: Statistics resulting from the application of Binary Disassembly and Embedding Generation using Radare2 and SAFEtorch, respectively. The data used corresponds to the APTMalware dataset [62]. The execution time is presented in *HH:mm:ss* format.

APT Group	No. of Binaries	No. of Extracted Functions	Execution Time
APT 30	164	15,134	00:07:08
APT 10	244	90,545	00:34:59
APT 1	405	56,816	00:31:44
Gorgon Group	961	252,886	01:58:56
Energetic Bear	132	175,647	01:21:12
Equation Group	395	64,830	00:50:10
APT 29	281	97,257	01:02:37
APT 19	32	14,718	00:08:05
APT 28	214	66,604	00:39:12
APT 21	106	10,686	00:08:24
Winnti	387	79,447	01:01:09
Dark Hotel	273	113,213	01:10:54
Total	3,594	1,037,783	09:34:30

To illustrate the effectiveness of the proposed framework in enhancing the analysis of different APT groups, we present Figures 5.1a and 5.1b depicting the visual representation of binary functions from various APTs. These images showcase distinct characteristic shapes associated with each APT, providing visual evidence of the framework's potential. By visually capturing the unique patterns and structures of APTs, this framework offers valuable insights that can aid in the analysis and understanding of these sophisticated threats.



(a) Reduced embedding representation of all functions in the binaries associated with APT19 group. The dimensionality reduction technique used combines PCA and t-SNE. The *Binary Hash* refers to the first six characters of each binary's SHA-256 hash.



(b) Reduced embedding representation of all functions in the binaries associated with Equation Group APT group. The dimensionality reduction technique used combines PCA and t-SNE. The *Binary Hash* refers to the first six characters of each binary's SHA-256 hash.

Figure 5.1: Reduced embedding representations of APTs group binaries.

5.2 The target: APT19

The APT19 group is an Advanced Persistent Threat group that has been selected for targeted analysis in this study. We selected this group based on two main criteria: the number of binaries (32) and functions (14,718) that characterize its malware that allows an in depth analysis, as well as and the presence of various reports documenting its attack campaigns. Table 5.2 summarizes the different type of binaries contained on this group.

Table 5.2: Distribution of File Types of the samples labelled as APT19

File Type	Number of Files
PE32 Executable - EXE	19
PE32 Dynamic Link Library - DLL	12
Executable and Linkable Format - ELF	1
Total	32

By directing our attention to APT19, our objective is to obtain valuable insights into its distinctive characteristics, techniques, and behavioural patterns. Through this focused analysis, we aim to contribute to a deeper understanding of APT19 and its impact in the cybersecurity landscape.

APT19 has been active since at least 2013. It is also known by various other names, such as Shell Crew, Deep Panda, C0d0so, Codoso Team, Sunshop Group, WebMasters, KungFu Kittens, PinkPanther, Black Vine, Group G0073... Different security researchers and vendors have assigned these names based on their own analysis of the group's tactics, techniques, procedures, malware, targets, and infrastructure [72]–[75].

One of the challenges in attributing cyberattacks to specific threat groups is the inconsistency and ambiguity of the classification assigned by different sources. This issue is exemplified in the case of APT19, where different security researchers and vendors have assigned distinct names and classifications to the group based on their individual analyses. Notably, two significant names associated with APT19 are Shell crew/Deep Panda and C0d0so, which may or may not pertain to the group labelled as APT19 [73]–[75]. In this approach, we adopt a unified perspective by grouping them within the same APT19 category. Our decision is supported by the presence of samples classified as APT19 in our dataset, which exhibit connections to both subgroups.

APT19 is suspected to be sponsored by the Chinese government and has targeted a variety of industries, including defence, finance, energy, pharmaceutical, telecommunications, high-tech, education, manufacturing, and legal services. Some of the notable operations attributed to APT19 are the breaches of Forbes in 2014 [76], Anthem health insurance company in 2015 [72], [77], and several law and investment firms in 2017 [78].

APT19 uses various techniques to compromise and maintain access to victim networks. The group typically relies on spear phishing emails with malicious attachments or links to deliver initial exploits or payloads [73], [78]. The group has also used watering hole attacks to compromise legitimate websites visited by their targets [76]. APT19 relies on a mix of custom and publicly available malware and tools. Some of the malware variants that APT19 has used are trojans like Sakula/Sakurel, Derusbi, and Bergard and various web shells such as China Chopper. APT19 employs techniques such as DLL side-loading, command obfuscation, indicator removal, hidden windows, and registry run keys to evade detection and persistence [73]–[75].

Table 5.3 represent the average and standard deviation of various function features that have been extracted within the APT19 binaries of the dataset. We briefly explain each feature as follows:

- **Size:** The size feature refers to the size of the function in bytes. It indicates the total number of bytes occupied by the function's instructions and data within the binary. The functions in the group exhibit significant variation in size, with some functions being much larger or smaller than the average. This suggests that the functions within this group are relatively larger in, containing a significant number of instructions and potentially more complex logic
- **Stackframe:** The stackframe feature represents the size of the stack frame used by the function. It indicates the amount of stack space allocated by the function for storing local variables and other data.
- **Nbbs:** Nbbs stands for the number of basic blocks. A basic block is a sequence of instructions without any branch or jump instructions within it. The nbbs feature represents the count of such basic blocks in the function. This suggests that the functions in APT19 have a moderate level of complexity with a reasonable number of control flow branches.
- **Edges:** Edges refers to the number of control flow edges in the function. It represents the number of connections or transitions between basic blocks within the function. Higher values indicate more branching and potentially more complex control flow patterns.
- **Callrefs:** The callrefs feature represents the number of function call references within the function. The functions in the group make a moderate number of calls to other functions. Functions with a higher number of call references may have more dependencies on other functions or perform complex operations involving function calls.

- **Datarefs:** Datarefs refers to the number of data references within the function. It represents the count of memory locations accessed or modified by the function. The low value suggests that these binaries might not heavily rely on external data manipulation.
- **Codexrefs:** The codexrefs feature represents the number of cross-references to the function's code. It indicates how many times the function is referenced or called from other parts of the code.

Table 5.3: APT19 function statistics of the different binaries present on [62]. They have been extracted using Radare2 reversing framework. For more details about each feature, refer to Section 4.1

	Size	Stackframe	nbbs	edges	callrefs	datarefs	codexrefs
Mean	560	67.78	11.78	16.56	5.48	0.05	14.31
Std	7,267.80	212.50	23.06	36.23	29.87	0.23	24.91

These statistics provide an overview of the distribution and variability of function features within APT19 functions. They can be used to understand the characteristics, complexity, and relationships of the functions, aiding in the analysis and comprehension of the [APT](#) group as a whole.

5.3 Correlation Analysis

The Correlation Analysis module aimed to simplify the data and reveal the structure and behaviour of the APT19 malware by finding correlations and dependencies among its functions and binaries. We used two criteria to conduct the analysis: the main function and the entry point of each binary.

To distinguish between the main function and the entry point, one must understand their respective roles in program execution. The main function is a user created entry point that typically receives control from the entry point. The entry point is a compiler-generated function that performs various tasks such as setting up the runtime environment or allocating memory before invoking the user-defined main function.

By plotting the embeddings of entry and main functions, we were able to identify close points in the embedding space, indicating similarities or related features among the corresponding binaries. Focusing on cases where the representation of these functions overlapped in the embedding space allowed us to establish direct correlations between the binaries. To further analyse these correlations, we visually represented and compared all the functions of the correlated binaries,

identifying both similarities and differences. These insights gleaned from the analysis will prove valuable for subsequent modules. It is worth noting that Figure 5.1a already showcases the function embedding representation of all the APT19 binaries.

Main Function Analysis

We analysed two cases where the embeddings of the main functions overlapped. Figure 5.2 showcases the representation of the main functions of the APT19 binaries.

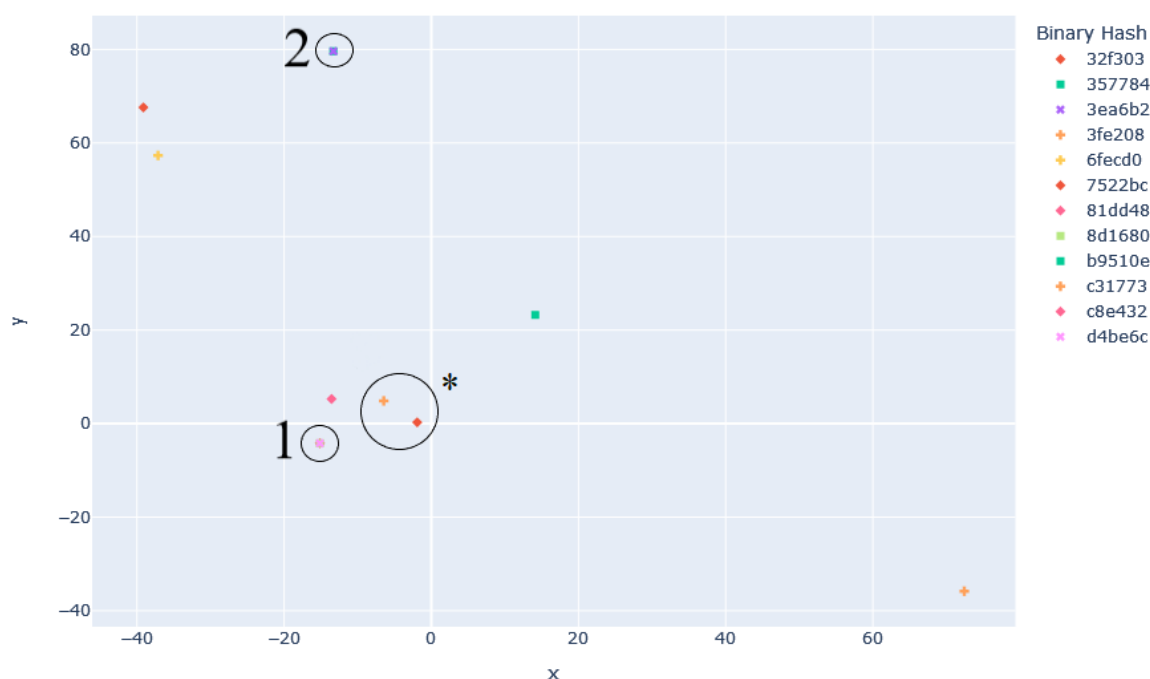


Figure 5.2: Reduced embedding representation of all main functions of the binaries labelled as APT19. The highlighted points indicate the cases where the embeddings overlap or present interesting behaviour. The dimensionality reduction technique used combines PCA and t-SNE. The *Binary Hash* refers to the first six characters of each binary's SHA-256 hash.

Entry Point Analysis

The analysis of the APT19 dataset identified six cases where the entry point embeddings overlapped. Figure 5.3 presents a visual representation of the entry points of the APT19 binaries.

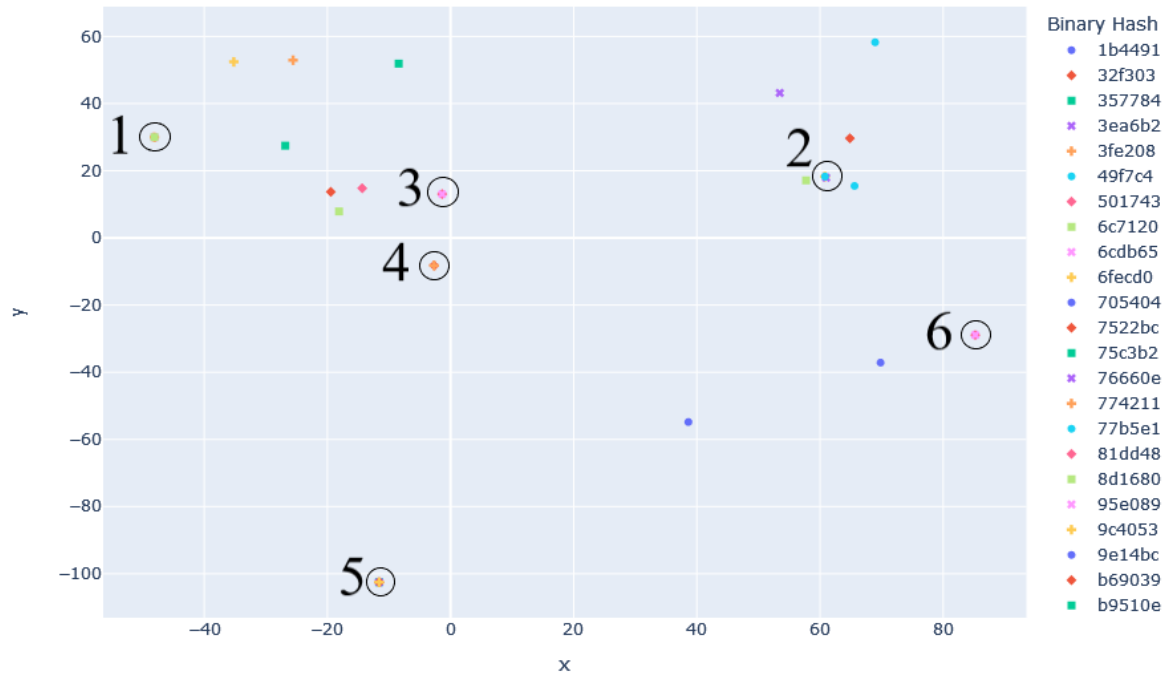


Figure 5.3: Reduced embedding representation of all entry point functions in the binaries labelled as APT19. The dimensionality reduction technique used combines PCA and t-SNE. The *Binary Hash* refers to the first six characters of each binary’s SHA-256 hash.

To ensure clarity and coherence, we have structured the correlated cases into distinct sections, following the proposed methodology depicted in Figure 4.2. This organization enables a systematic representation of each step involved in the target analysis. By presenting the analysis in this manner, readers can easily track the progression of the investigation and understand the significance of each step in uncovering the overall findings.

5.4 Use Case 1

In Use Case 1, we examine the relationships between three binaries, identified by their SHA-256 hashes:

- binary1: "8d168092d5601ebbaed24ec3caef7454c48cf21366cd76560755eb33aff89e9"
- binary2: "d4be6c9117db9de21138ae26d1d0c3cfb38fd7a19fa07c828731fa2ac756ef8d"

- binary3: "81dd48ed812d571c700c0c097c97a207eb5ac950fcf3c34309cedf9e88b1405d"

This analysis focuses on two significant aspects: the overlap of the main function among the three binaries, depicted as "1" in Figure 5.2, and the shared entry points between binary2 and binary3, denoted as "3" in Figure 5.3.

To provide a visual representation of the involved binaries, Figure 5.4 displays all the function embeddings relevant to this Use Case.

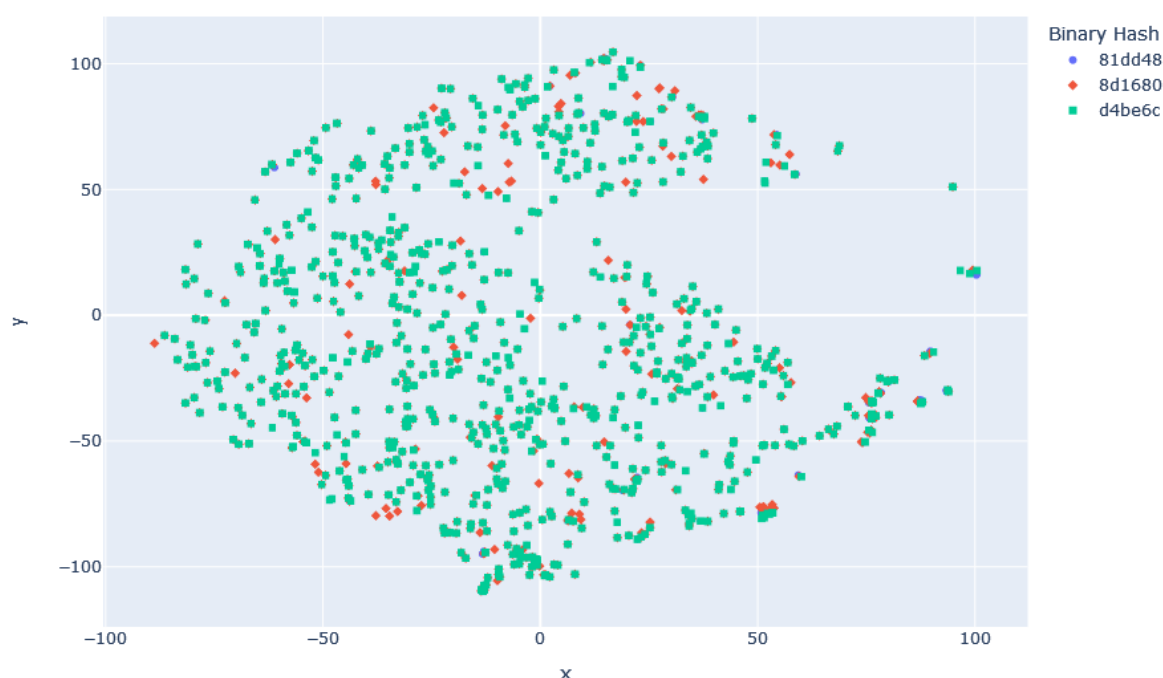



Figure 5.4: Representation of all the function embeddings of the binaries belonging to the Use Case 1.


OSINT Analysis

In Figure 5.5, a comprehensive overview of the details of the binaries obtained from VirusTotal for Use Case 1 can be found. For a more in-depth understanding, the links to the reports are provided: [Binary1](#), [Binary2](#) and [Binary3](#). Additionally, a new binary, [Binary4](#), was added due to the found relationships explained below. These links will redirect you to the respective VirusTotal reports for each binary, providing additional information and analysis.

- Binary4 = “3fe208273288fc4d8db1bf20078d550e321d9bc5b9ab80c93d79d2cb05cbf8c2”




Binary 1

8d168092d5601ebbaed24ec3caeef7454c48cf21366cd7656075... | Size 496.22 KB | Last Analysis Date 9 months ago | 

SecureInput.exe

peexe signed overlay

Compilation Timestamp 2013-12-20 01:34:53 UTC

Popular threat label  trojan.sakelua/sakurel Threat categories trojan Family labels sakelua sakurel

Binary 2

d4be6c9117db9de21138ae26d1d0c3cfb38fd7a19fa07c828731fa... | Size 438.82 KB | Last Analysis Date 1 day ago | 

Juniper SSL VPN ActiveX.exe

peexe malware overlay runtime-modules signed detect-debug-environment checks-network-adapters long-sleeps

Compilation Timestamp 2014-05-23 08:07:49 UTC

Popular threat label  trojan.mivast/sakelua Threat categories trojan Family labels mivast sakelua

Binary 3

81dd48ed812d571c700c0c097c97a207eb5ac950fc3c34309ce... | Size 595.32 KB | Last Analysis Date 9 months ago | 

Secure Microsoft ActiveX Control.exe

peexe overlay runtime-modules signed long-sleeps direct-cpu-clock-access checks-user-input

Compilation Timestamp 2014-10-27 02:15:34 UTC

Popular threat label  trojan.sakurel/diofopi Threat categories trojan Family labels sakurel diofopi

Binary 4

3fe208273288fc4d8db1bf20078d550e321d9bc5b9ab80c93d79d2... | Size 11.82 KB | Last Analysis Date 28 days ago | 

VirusShare_02fab24461956458d70aedd1a028eb9c

peexe overlay runtime-modules signed checks-network-adapters armadillo direct-cpu-clock-access spreader

Compilation Timestamp 2013-12-07 07:09:30 UTC

Popular threat label  trojan.graftor/dapato Threat categories trojan droppe Family labels graftor dapato

Figure 5.5: Different information of the binaries obtained from VirusTotal.

After extensive research, it has been established that the analysed binaries are associated with the threat group commonly referred to as “Black Vine” or “Deep Panda” [79]–[81]. These different names are attributed to the group responsible for the well-known Anthem attack [74]. The binaries in question are related to the mentioned attack and represent different variants of a [Remote Access Trojan \(RAT\)](#) known as “Sakula” (for more information about Sakula, refer to MITRE’s [software profile](#)) [81], [82]. The VirusTotal labels shown in Figure 5.5 corroborate these associations.

During our investigation, we encountered a widely recognized YARA rule that has been mentioned in multiple sources [80], [83]–[85]. This YARA rule serves as a vital link connecting Binary1, Binary2, and introducing a new binary, Binary4, into our analysis. Binary4, labelled as APT19, is included in the dataset and will be subject to further scrutiny.

The description provided within the YARA rule highlights several significant strings. For instance, in Binary1, the presence of the URL `http://142.91.76.134/p.dat` suggests the potential for downloading a data section that may exploit hidden behaviour. Furthermore, our attention is drawn to the digital signature "DTPROTOOLZ", which has been identified in other sources [81]. This signature is associated with both Binary1 and Binary4, further reinforcing their connection to the Anthem attack [81], [82].

Before proceeding with the Binary Analysis, we conducted a Correlation Analysis to verify the correlation between Binary4 and the other binaries. By plotting the four binaries visually, it became evident that Binary4 exhibited clear differences. Both the entry point and main function were significantly distant from the others. Additionally, the function map of Binary4 was considerably smaller and had minimal overlap with the other binaries. As a result, we decided not to conduct further in-depth analysis of Binary4 and excluded it from our investigation.

Binary Analysis

The initial aim of the Binary Analysis was to explore the potential of MAPTER, particularly its graphical correlation capabilities, in revealing the similarities among the three remaining binaries. Therefore, this analysis focused in depth on comparing closely related or overlapping functions.

To perform this analysis, we adopted a sequential approach that followed the normal execution flow of the binaries. We designated the entry point as the first function and examined the subsequent function calls from it. During this process, we identified the main function in all three binaries and a function labelled as *AfxWinMain* by Ghidra. After consulting various sources [86], [87], we discovered that *AfxWinMain* serves as the entry point function typically used in [Microsoft Foundation Classes \(MFC\)](#) applications developed with Microsoft Visual C++. It acts as the starting point for the execution of [MFC](#)-based Windows applications. [MFC](#) is a framework provided by Microsoft that simplifies the development of Windows applications, particularly those with graphical interfaces.

At this point, our sequential analysis reached a halt, since conducting such an analysis on an application with a graphical interface would not yield practical insights. Consequently, we proceeded

with an arbitrary search for functions, focusing on references to imports of specific Windows functions such as *CreateProcess* or *CreateThread*.

Once we compiled a collection of these functions, we analysed the assembly code and pseudocode using Ghidra. Our analysis revealed that the entry point, main function, and *AfxWinMain* function in the three binaries were almost identical, differing only in memory addresses. Furthermore, all three binaries shared a common function that invoked *CreateProcess*, which exhibited almost identical instructions and functionality. Regarding *CreateThread*, we found that it was invoked by a single function in the case of Binary1 and Binary2, with the primary difference being a string reference to two URLs, in Binary1 http://extcitrix.wellpoint.com/v_0042d090 and in Binary2 <http://sharepoint-vaeit.com/login.php?ref>.

To further explore differences, we employed a graphical representation of the three binaries, incorporating size as an additional dimension. This allowed us to focus on the most significant functions. Upon examining the code of these visually prominent functions, we observed that they were primarily Visual Studio library functions and did not display notable distinctions.

Findings and Insights

In conclusion, our investigation on the Use Case 1 has confirmed that the analysed binaries are indeed associated with the threat group Black Vine or Deep Panda, known for their involvement in the Anthem attack. The presence of a widely recognized YARA rule further solidified the connections between Binary1 and Binary2. Notably, Binary4 was excluded from further analysis due to significant dissimilarities found during the Correlation Analysis.

The correlation analysis conducted using MAPTER demonstrated its effectiveness in identifying the similarities among the binaries, particularly Binary1 and Binary2. These binaries exhibited shared functions that strongly indicate their association with different variants of the Sakula Remote Access Trojan. Additionally, the presence of URLs within the binaries suggests their utilization as Command and Control Centres.

One notable discovery from our Binary Analysis was the use of the Microsoft Foundation Classes framework in conjunction with Microsoft Visual C++ in the development of these binaries. This finding raises intriguing questions about the motives behind APT19's choice of utilizing [MFC](#). It is possible that they employed this framework to disguise their malicious activities by imitating legitimate applications or to exploit vulnerabilities present in the [MFC](#) libraries, thereby evading detection.

These findings provide valuable insights into the techniques and strategies employed by APT19, shedding light on their malware development processes and potential avenues for mitigation.

5.5 Use Case 2

In Use Case 2, we examine the relationships between three binaries, identified by their SHA-256 hashes:

- binary5 = “3ea6b2b51050fe7c07e2cf9fa232de6a602aa5eff66a2e997b25785f7cf50daa”
- binary6 = “3577845d71ae995762d4a8f43b21ada49d809f95c127b770aff00ae0b64264a3”
- binary7 = “b690394540cab9b7f8cc6c98fd95b4522b84d1a5203b19c4974b58829889da4c”
- binary8 = “de33dfce8143f9f929abda910632f7536ffa809603ec027a4193d5e57880b292”

The analysis focuses on two key aspects: the overlap of the main function among two binaries: Binary5 and Binary6, represented as “2” in Figure 5.2, and the shared entry points of the four binaries, denoted as “5” in Figure 5.3.

To visually represent the function embeddings of all the binaries involved in this section, we provide Figure 5.6.

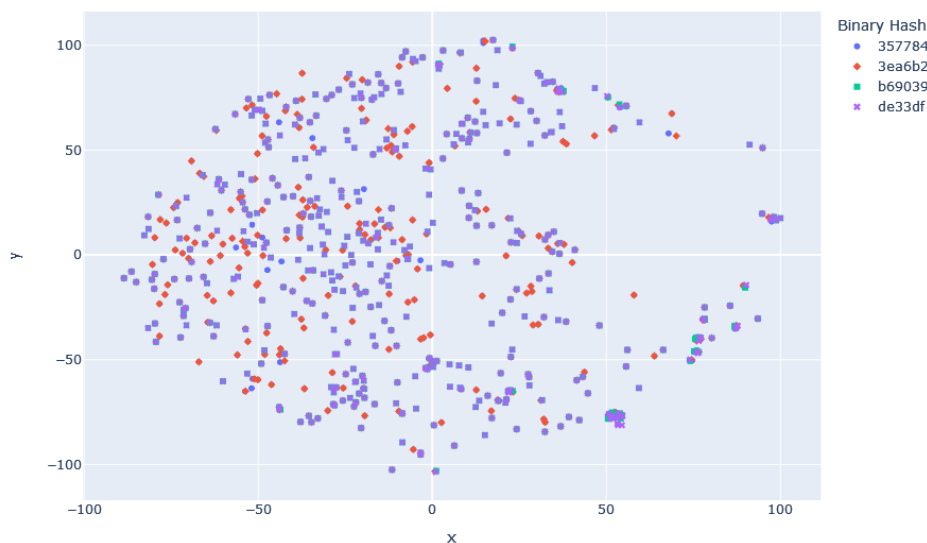


Figure 5.6: Representation of all the function embeddings of the binaries belonging to the Use Case 2.

OSINT Analysis

In our *OSINT analysis*, we obtained detailed information about the binaries from VirusTotal for case 2 and case 5, as shown in Figure 5.7. The reports for each binary can be accessed through the following links: [Binary5](#), [Binary6](#), [Binary7](#) and [Binary8](#).

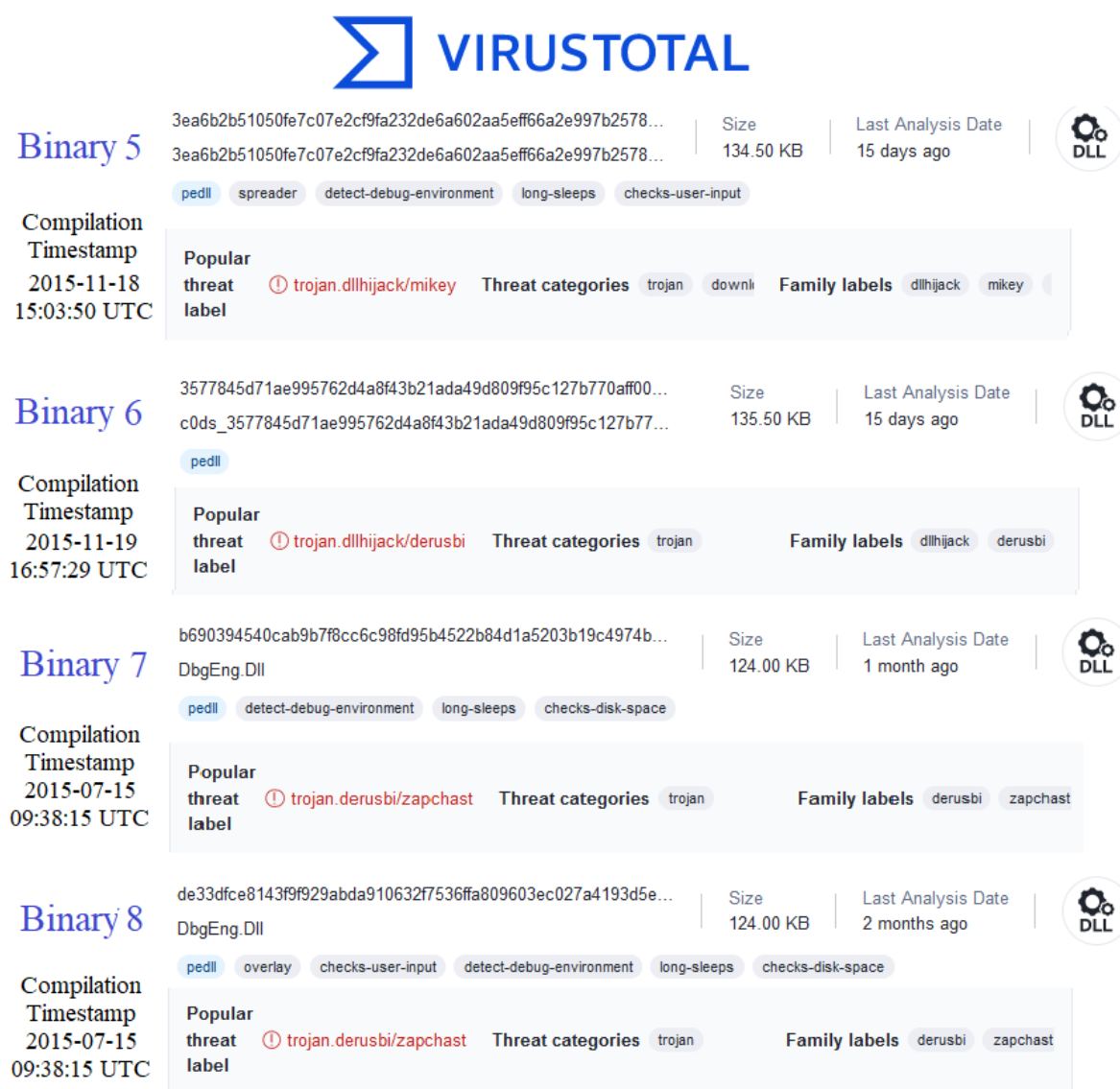


Figure 5.7: Different information of the binaries obtained from VirusTotal.

Upon analysing the reports generated by VirusTotal, we discovered indications suggesting that Binary7 and Binary8 are the same. These indications include similarities in various features such as the MD5 hash-based sections of the binary, Vhash and Imphash hashes, file size, and tagged filename.

Different sources [76], [88] place the analysed binaries with the "Codoso" group, which, as mentioned in section 5.2, is one of the aliases used by the APT19 group. This relationship is materialized thanks to details found in a report [76], which links the binaries to the watering hole attack on *forbes.com* in 2014 carried out by the APT19 group using a variant of the Derusbi trojan [73], [76].

The report by *Grunzweig and Lee* [76] provides a comprehensive analysis linking the four binaries and describing their different behaviours which are common among Derusbi malware variants. Within the four binaries, three domain names are identified: *jbossas.org* in Binary5, *supermanbox.org* in Binary6, and *microsoft-cache.com* in both Binary7 and Binary8. By resolving the IP addresses corresponding to these domains, it was discovered that all three domains point to the same IP located in Hong Kong (121.54.168.230), which is believed to serve as the Command and Control Center.

The first two binaries (along with another one that is not present in the dataset) are framed under a variant characterized by communicating via port 22 with the C&C. They suggest that this variant appears to be more recent and that was delivered under the name of *McAltLib.dll*, configured to be side-loaded with the legitimate McAfee *mcs.exe* executable. What supports the theory that Binary5 and Binary6 are an evolved version of the other two is the existence of references to HTTP strings still found within the sample itself. This is most likely due to code reused by attackers. Notably, Binary5 and Binary6 show a strong relationship with the 2014 attack on *forbes.com*, as a comparison of the binaries from both attacks reveals numerous similarities.

The remaining two binaries, along with other samples not included in the dataset, belong to the same variant that communicates via HTTP with the C&C. This variant masquerades as a seemingly benign product that generates keys for the AVG antivirus.

Both variants aim to establish persistence and exfiltrate victim details, such as MAC addresses and IP addresses, among other information. They receive a DLL file with the same exports from the C&C centre, which attempts to download additional malware.

Furthermore, a YARA rule [88] links Binary5 and Binary6 together. The description within the YARA rule highlights different strings related to network management commands.

During our investigation, we came across a detailed report [89] that thoroughly analyses Binary8, providing further insights into its behaviour and corroborating the findings described by *Grunzweig and Lee* [76] in the preceding paragraphs.

Binary Analysis

The analysis approach employed involves examining the similarities between the four binaries, and subsequently analysing the similarities and differences among pairs of binaries that exhibit a higher degree of similarity. The analysis focuses on exported functions, since DLLs are being analysed. Based on this approach, two sets of similarities are expected to emerge: one between Binary5 and Binary6, and the other between Binary7 and Binary8. In fact, comparing the exported functions already reveals this distinction, as the first two binaries share all exported functions with each other, while the second pair only shares the *entry* and *ServiceMain* functions in general.

The analysis begins with the examination of the *entry* function, which serves as the entry point for all four samples and is found to be identical in each case. Subsequently, the *ServiceMain* function, corresponding to the main functionality of the files, is analysed. Here, differences start to emerge between pairs of binaries. Binary5 and Binary6 share similar code in this function, with minor differences in memory addresses. Conversely, Binary7 and Binary8 exhibit exactly the same code, including identical memory addresses. Differences between the two groups of binaries are characterized by calls to different functions, marking the initial point of divergence. Further exploration within the first group reveals several identical functions with minor variations in memory addresses. In contrast, the second group of binaries features numerous identical functions. No additional similarities are identified between the two groups.

The analysis continues by examining the exports shared within the first group, specifically focusing on the *McInitMISPAAlert* function. Here, the first difference is observed: Binary6 includes a function call (0x10005e10) that is not present in Binary5. This particular function is responsible for dynamically importing libraries. Moreover, Binary6 implements the *McShowMISPNotification* function in a distinct manner compared to Binary5, which references the code of the *McInitMISPAAlert* function. All other exports remain the same.

Binary7 and Binary8 share identical exports, including memory addresses. To investigate the differences between these two files, the PEBear program is utilized to compare their contents. The analysis reveals that the only discrepancy lies in the overlay of Binary8, which contains noise likely added to evade signature-based checks.

Then, we proceed to explore the differences using the graphical representation of the binaries. This allows us to identify several functions added to Binary6 versus Binary5 such as 0x10005fd0 (*McShowMISPNotification*), 0x10005e10 (imports libraries dynamically), 0x100075d3 and 0x10005020 among others.

In the case of the second group, differences are only found in the representation of very small functions (6-20 bytes). Upon closer examination, these functions are found to exist in both binaries in the same manner, suggesting that the SAFE tool performs suboptimally when working with very small functions.

The differences between the two groups of binaries are substantial, making it challenging to conduct an in-depth analysis of code fragments. However, the graphical representation in Figure 5.6 reveals the reuse or similarity of certain functions.

Unfortunately, no links were found within the binaries, as they are purportedly encrypted using an XOR cipher [76].

Findings and Insights

Through our investigation of Use Case 2, different connections have been drawn between the analysed binaries and the notorious threat group known as "Codoso" (an alias for APT19) for their relation with the *forbes.com* attack on 2014 [76].

The analysis of Binary5, Binary6, Binary7, and Binary8 has revealed distinct patterns and relationships. They all exhibit overlapping entry points, as represented in Figure 5.3 as "7". Notably, Binary5 and Binary6 share commonalities in their main functions, as depicted in Figure 5.2 as "2". Additionally, the comparison between Binary7 and Binary8, utilizing the PEBear program, exposes a sole difference: an overlay within Binary8 containing noise, potentially intended to evade signature-based checks. This finding underscores the threat actors' efforts to obfuscate their activities and highlights the importance of employing advanced techniques to detect and analyse malware. These observations suggest potential connections and collaboration among these binaries, warranting further investigation into the motives and objectives of the threat actors involved.

The overall findings from Use Case 2 provide valuable insights into the behaviour and tactics of the "Codoso" group, revealing their inclination towards code reuse and evolution of malware. Our framework has been able to identify different binaries whose relationship has been confirmed by different reports that place these samples as variants of the Derusbi malware. These insights leverage the potential of MAPTER and helps security professionals to better understand APT19's *modus operandi*.

5.6 Use Case 3

In Use Case 3, we examine the relationships between three binaries, identified by their SHA-256 hashes:

- binary9 = “95e08990fdf11251e9ee935f0b2e075667133758bc68c4d390e82f041a54e4b3”
- binary10 = “77421106548e69e9666c538ad628918cad7cfcf8f6aa7825f71a4fc39e522a7d”
- binary11 = “705404d6bbf6dae254e2d3bc44eca239976be7f0dc4d49fe93b0fb1d1c2704fe”

This section specifically focuses on the relationships between these binaries where their entry point representations overlap, represented as “1” in Figure 5.3. To visualize the function embeddings of the binaries involved in Use Case 3, refer to Figure 5.8.



Figure 5.8: Representation of all the function embeddings of the binaries belonging to the Use Case 3.

OSINT Analysis

In Figure 5.9, a comprehensive overview of the details of the binaries obtained from VirusTotal for Use Case 3 can be found. For a more in-depth understanding, the links to the reports are provided: [Binary9](#), [Binary10](#) and [Binary11](#).

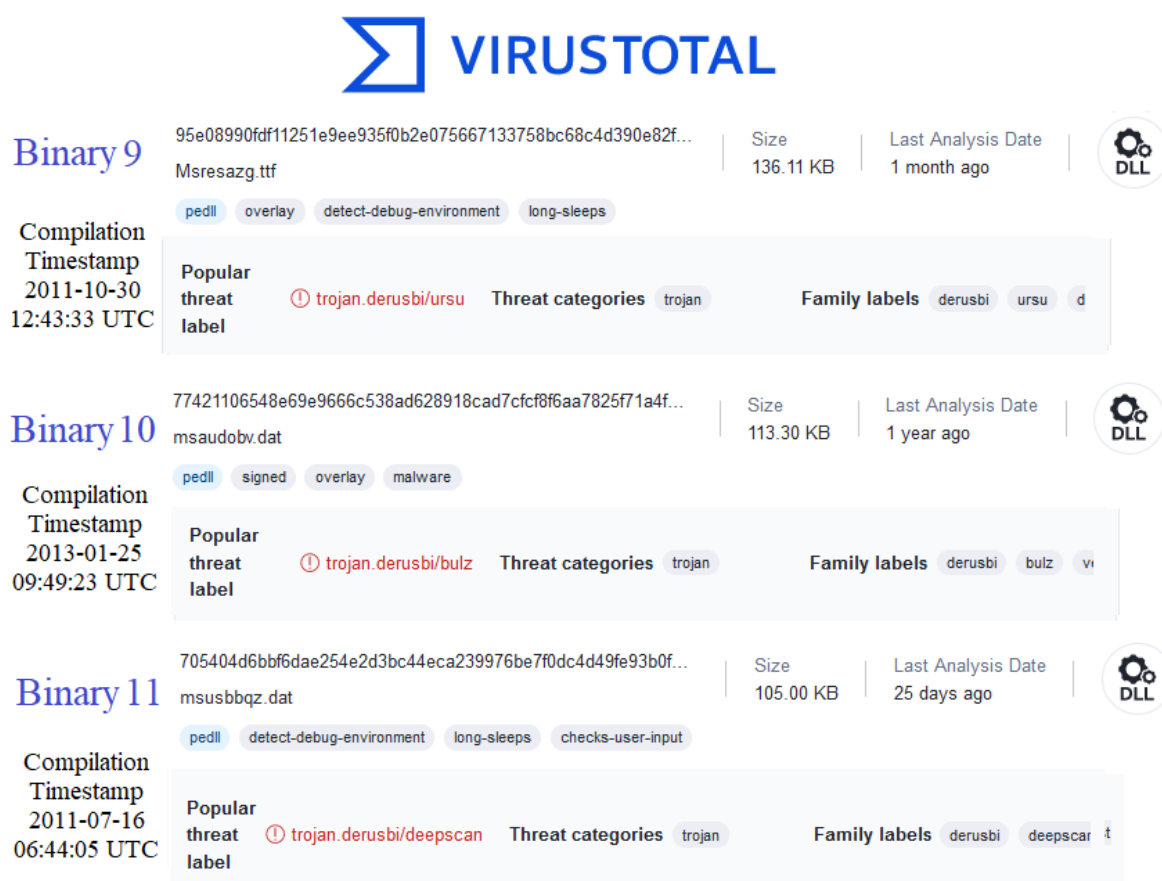


Figure 5.9: Different information of the binaries obtained from VirusTotal.

Unfortunately, there is limited published information available for these three binaries. Only binary10 is mentioned in the report [81], which links it to the DeepPanda (APT19) attack on Anthem. According to the report, binary10 is a variant of the Derusbi trojan family, digitally signed as "DPTOOLZ". It is worth noting that this signature also appears in some binaries of the Use Case 1 5.4.

Binary Analysis

In this analysis, we adopt a sequential approach based on the exported functions of the binaries, since we are analysing DLLs. We begin by examining the *entry* function, which serves as the en-

try point for all three binaries. The assembly instructions of this function are almost identical, excepting the memory addresses.

Subsequently, we analyse the *ServiceMain* function, which corresponds to the main function of the files. Binary9 and Binary10 exhibit only one difference in this function, the addition of "iphlpvc" to what appears to be the environment variables. However, we encountered difficulties in obtaining the complete code of the *ServiceMain* function for Binary11. Further analysis of the functions and exports of Binary11 revealed similar issues, indicating possible obfuscation techniques employed in this binary. Therefore, we set aside the analysis of Binary11 for the time being.

We continue with *WUServiceMain*, function, specific to Windows Update services, we find that it serves as the entry point for a service DLL that provides Windows Update functionality. In both Binary9 and Binary10, this function simply jumps to the *ServiceMain* function. Notable differences are observed in the *DllRegisterServer* and *DllUnregisterServer* functions. While these functions share some code fragments, there are differences in functional code fragments, indicating potential code reuse with modularization into separate functions in Binary9.

We then explore the graphical representation of the binaries. Compared to previous use cases, we observe a higher number of non-overlapping functions, although they are still closely located. Upon analysing some of the larger functions, we find that despite being located far apart, their content is very similar. Occasionally, differences arise in the order of execution of certain instructions, indicating signs of code reuse.

It is worth noting that Binary11, despite having a limited number of functions, follows a similar distribution pattern to Binary9 and Binary10. This suggests that it may be a similar binary but obfuscated.

Findings and Insights

Our investigation of Use Case 3 has provided us with some findings and insights into the analysed binaries and the activities of the threat actors behind them. Through the analysis, we have identified a possible connection between the binaries and the threat group "Deep Panda" in the Anthem attack.

The correlation analysis conducted using MAPTER has demonstrated its effectiveness in identifying the similarities among the binaries based on the entry point criteria, particularly Binary9 and Binary10. These two binaries share common exported functions and exhibit similar patterns in their code, indicating a high likelihood of them being variants of the same malware. Further-

more, the visual representation of the binaries has revealed a significant level of code reuse and similarity, suggesting a common development or evolution process.

5.7 Use Case 4

In Use Case 4, we examine the relationships between three binaries, identified by their SHA-256 hashes:

- binary12 = "9c4053485b37ebc972c95abd98ea4ee386feb745cc012b9e57dc689469ea064f"
- binary13 = "c6f1a8f9ea60286b24db87d6022991a4342bea473a520569b996a5883332788c"

This section specifically focuses on the relationships between these binaries where their entry point representations overlap, represented as "2" in Figure 5.3. To visualize the function embeddings of the binaries involved in Use Case 4, refer to Figure 5.10.

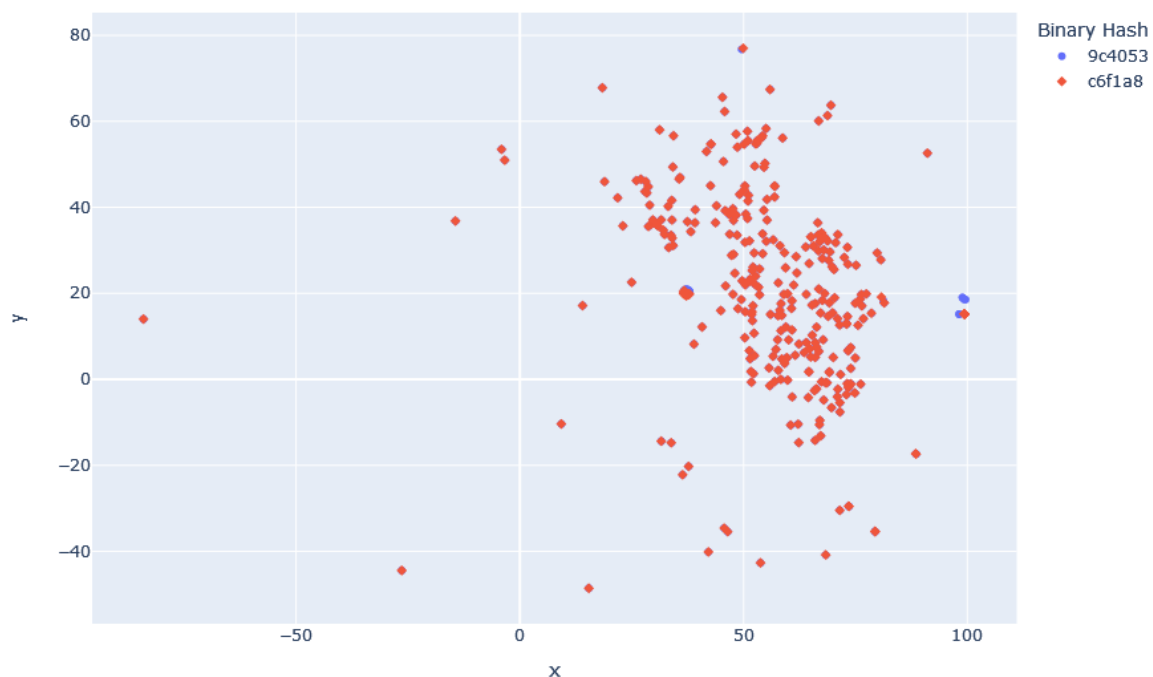


Figure 5.10: Representation of all the function embeddings of the binaries belonging to Use Case 4.

OSINT Analysis

Our [OSINT](#) analysis provides an overview of the details obtained from VirusTotal for case 4, as depicted in Figure 5.11. To access the reports for binary12 and binary13, use the following links: [Binary12](#) and [Binary13](#).

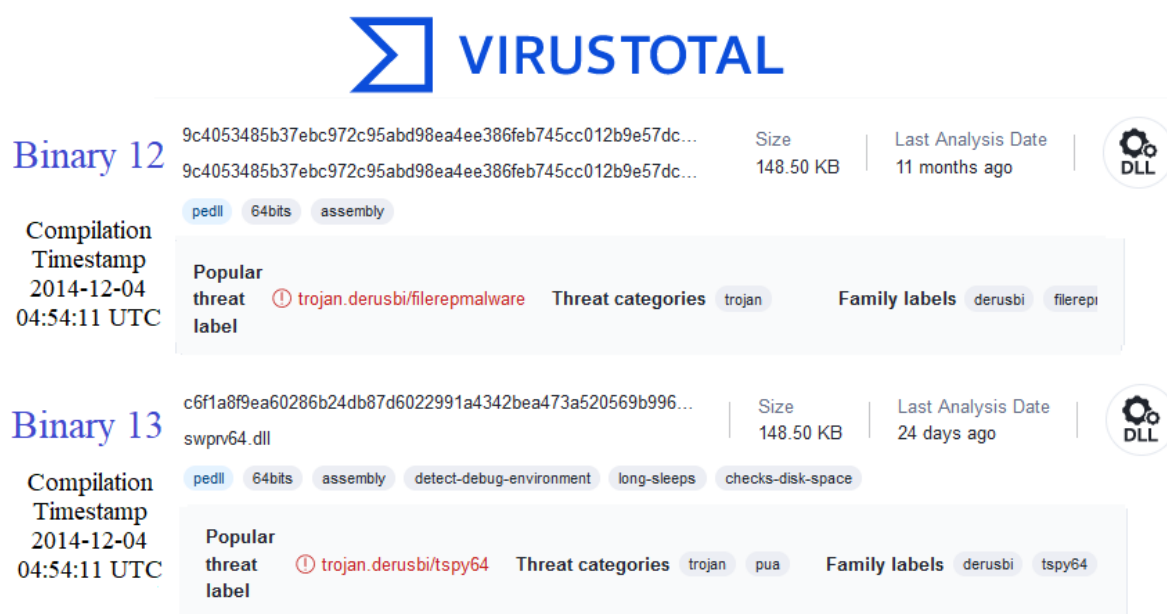


Figure 5.11: Different information of the binaries obtained from VirusTotal.

In this particular case, there is no publicly available information on these two binaries beyond what is provided on VirusTotal. Analysing the reports, we noticed remarkable similarities between the binaries. They share features such as the compilation date, size, Imphash, and Vhash of the samples. Additionally, their section sizes and MD5 hashes are identical, except for the ".text" and ".data" sections.

Binary Analysis

In this analysis, we focus on two DLLs and adopt a sequential approach based on the exported functions. Following this approach we find that all the exports: the *entry*, the *ServiceMain* and the *Start* are exactly the same, even in memory addresses.

We then proceed to analyse the visual representation of the binaries. In this representation, we observe differences in small functions that appear to have suffered problems in the decompilation process or have been intentionally obfuscated. These differences are particularly evident in the ".text" and ".data" sections, suggesting that obfuscation techniques may have been employed.

Findings and Insights

Our investigation of Use Case 4 has yielded limited findings and insights. While the analysis using MAPTER has successfully identified similarities in functions through the entry point, we have not been able to extract substantial conclusions or insights from these findings.

Due to the nature of the analysis and the limited information available, it is challenging to draw significant conclusions about the purpose, functionality, or relationships of the analysed binaries. The observed similarities in functions may indicate some level of code reuse or commonality, but without further context or additional analysis, it is difficult to make definitive assertions.

The visual representation of the embeddings showcases a curious and distinct pattern. This unique pattern raises the possibility that these binaries belong to a specific subgroup or variant within the [APT](#), potentially indicating a specialized focus or a specific objective.

In cases like these, where the findings are inconclusive, it is important to consider alternative approaches or complementary analyses to gain a deeper understanding of the binaries and the threat actors behind them. Further research or additional data sources may provide additional insights that can contribute to a more comprehensive analysis.

5.8 Use Case 5

In Use Case 5, we examine the relationships between three binaries, identified by their SHA-256 hashes:

- binary14 = "c317733322bd1c42601cefb6428e72eec2623ca2c0bfcaf8fb4d7256208f8748"
- binary15 = "32f303469030ee68b20c03e239ff9fdc801a747ba5148f36032d94ee41dcdf56"

This case focuses on the overlapping entry points of both binaries, represented by the number "4" in Figure 5.3. Additionally, the embedding representation of their main functions exhibits proximity, denoted by an asterisk in Figure 5.2. Figure 5.12 presents the graphical representation of the function embeddings for all the binaries involved in Use Case 5.

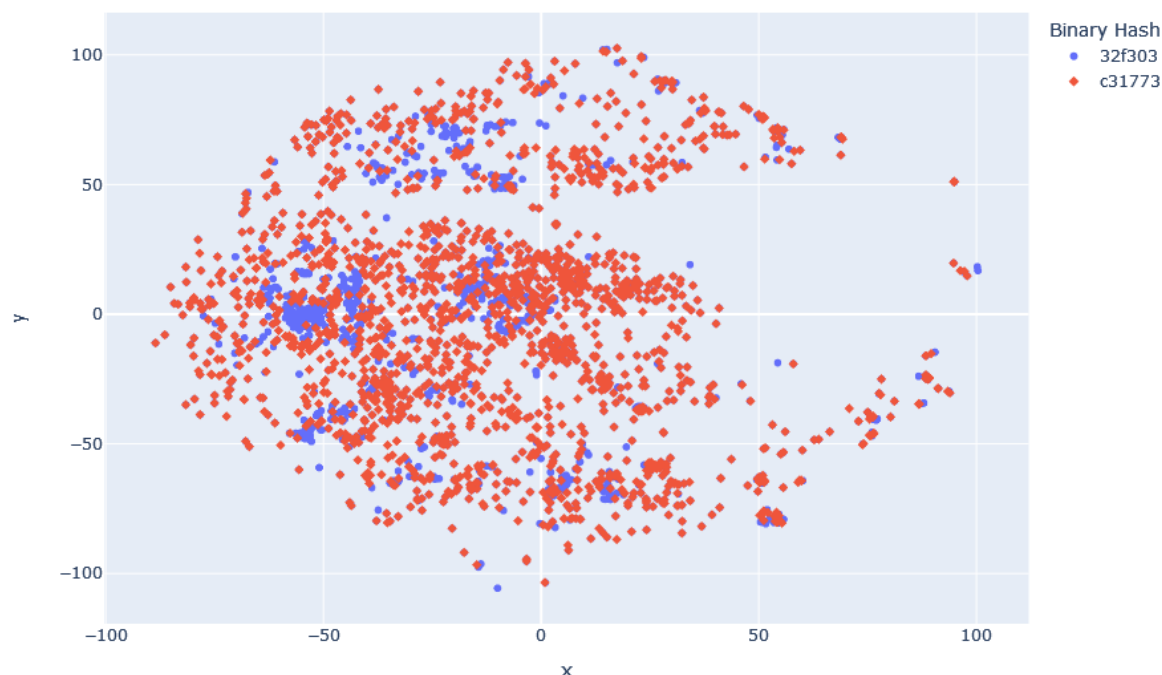


Figure 5.12: Representation of all the function embeddings of the binaries belonging to Use Case 5.


OSINT Analysis

In Figure 5.13, an overview of the details of the binaries obtained from VirusTotal for case 6 can be found. For a more in-depth understanding, the links to the reports are provided: [Binary14](#) and [Binary15](#).

Again, there is no publicly available information on these two binaries beyond what is provided on VirusTotal. The reports mention that both binaries were uploaded with the same name, "upnpshots.exe".

Binary Analysis

In this analysis, we focus on executable binaries and perform a sequential analysis starting from the entry point. We find that both the main and entry point functions are the same, with the only difference being the memory addresses. Continuing the analysis, we examine functions 0x4041a0 and 0x402cd0, both invoked from the main function.




Binary 14

c317733322bd1c42601cefb6428e72eec2623ca2c0bfc8fb4d7...
upnpshots.exe

Size
725.00 KB

Last Analysis Date
1 year ago



Compilation
Timestamp
2015-10-19
13:51:09 UTC

peexe

Popular
threat
label

trojan.derusbi/mikey

Threat categories
trojan


Family labels
derusbi mikey

Binary 15

32f303469030ee68b20c03e239ff9fdc801a747ba5148f36032d94...
upnpshots.exe

Size
807.00 KB

Last Analysis Date
2 months ago



Compilation
Timestamp
2015-09-10
09:12:54 UTC

peexe

Popular
threat
label

trojan.derusbi/midie

Threat categories
trojan

Family labels
derusbi midie

Figure 5.13: Different information of the binaries obtained from VirusTotal.

These two functions exhibit the same functionality in both binaries, but Binary15 encapsulates part of the operation within calls to other functions, such as 0x4051e7. It is within these functions that we find relevant clues about the purpose of the samples, specifically related to [Universal Plug and Play \(UPnP\)](#) devices. UPnP devices are network-enabled devices that conform to the UPnP protocol, allowing seamless discovery, control, and communication within a local network environment. These devices encompass a wide range of consumer electronics, appliances, smart home devices, network devices, and multimedia devices. The presence of strings related to UPnP in the analysed functions suggests that the malware may attempt to exploit vulnerabilities associated with these devices.

The visual representation of the binaries shows slight differences in the code, but when analysed in depth, no major variations in terms of functionality have been found.

Findings and Insights

In Use Case 5, our investigation has revealed significant findings and insights regarding the analysed binaries and the activities of the threat actors. The analysis indicates a connection between Binary14 and Binary15, with both binaries targeting vulnerabilities related to UPnP devices. Despite minor differences, the overall functionality remains similar, suggesting they are variations of the same malware.

Overall, the findings from Use Case 5 highlight the importance of analysing the correlations on main function and entry point to identify similarities and potential relationships between binaries, providing a starting point for further investigation and analysis.

5.9 Use Case 6

In Use Case 6, we examine the relationships between three binaries, identified by their SHA-256 hashes:

- binary16 = "50174311e524b97ea5cb4f3ea571dd477d1f0eee06cd3ed73af39a15f3e6484a"
- binary17 = "1b449121300b0188ff9f6a8c399fb818d0cf53fd36cf012e6908a2665a27f016"
- binary18 = "6cdb65dbfb2c236b6d149fd9836cb484d0608ea082cf5bd88edde31ad11a0d58"

This case focuses on the overlapping entry points of both binaries, represented by the number "6" in Figure 5.3. Figure 5.14 showcases the representation of all the function embeddings of the binaries involved in Use Case 6.

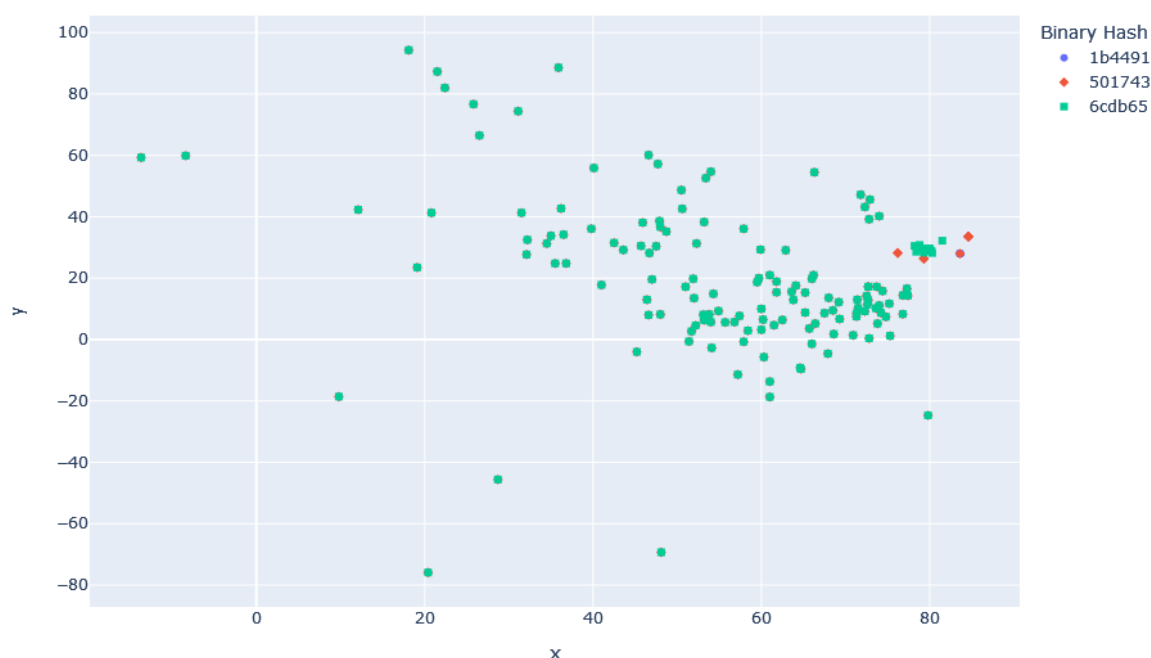
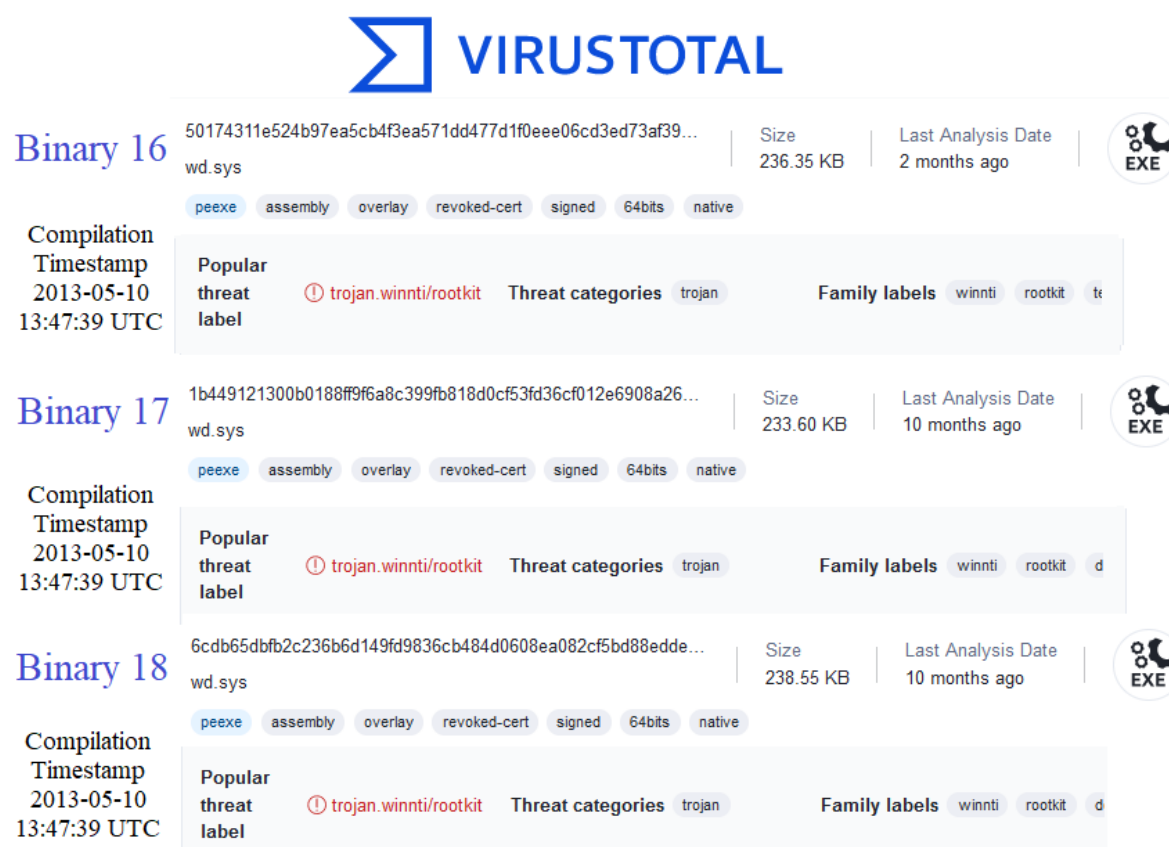




Figure 5.14: Representation of all the function embeddings of the binaries belonging to the Use Case 6.



OSINT Analysis

In Figure 5.15, a comprehensive overview of the details of the binaries obtained from VirusTotal for case 8 can be found. For a more in-depth understanding, the links to the reports are provided: [Binary16](#), [Binary17](#) and [Binary18](#).



VIRUSTOTAL

Binary 16
 50174311e524b97ea5cb4f3ea571dd477d1f0eee06cd3ed73af39...
 wd.sys | Size: 236.35 KB | Last Analysis Date: 2 months ago | 
 peexe assembly overlay revoked-cert signed 64bits native
 Compilation Timestamp: 2013-05-10 13:47:39 UTC
 Popular threat label:  trojan.winnti/rootkit | Threat categories: trojan | Family labels: winnti, rootkit, te

Binary 17
 1b449121300b0188ff9f6a8c399fb818d0cf53fd36cf012e6908a26...
 wd.sys | Size: 233.60 KB | Last Analysis Date: 10 months ago | 
 peexe assembly overlay revoked-cert signed 64bits native
 Compilation Timestamp: 2013-05-10 13:47:39 UTC
 Popular threat label:  trojan.winnti/rootkit | Threat categories: trojan | Family labels: winnti, rootkit, d



Binary 18
 6cdb65dbfb2c236b6d149fd9836cb484d0608ea082cf5bd88edde...
 wd.sys | Size: 238.55 KB | Last Analysis Date: 10 months ago | 
 peexe assembly overlay revoked-cert signed 64bits native
 Compilation Timestamp: 2013-05-10 13:47:39 UTC
 Popular threat label:  trojan.winnti/rootkit | Threat categories: trojan | Family labels: winnti, rootkit, d

Figure 5.15: Different information of the binaries obtained from VirusTotal.

Upon reviewing the reports, a notable observation is the similarity in build dates and names between the two binaries, prompting us to conduct a more thorough analysis. Upon closer examination, we discovered that all sections of the binaries, except for ".data" and ".edata," exhibit identical SHA-256 hashes. However, Binary18 differs from the other two in the ".rsrc" section, potentially accounting for the slight variations in sizes.

According to a report by *Perigaud* [90], these files are linked to a new version of the Derusbi trojan, which aligns with the classification by VirusTotal. In this version, the malware disguises itself as a Windows x64 driver, using names like wd.sys or udfs.sys. To function as a [Remote Access Trojan \(RAT\)](#), these binaries rely on DLLs packed in the ".data" section, which could account for the observed differences in the VirusTotal section. These DLLs enable various network behaviors,

including keylogging, VPN functionality, and remote desktop capabilities.

Furthermore, it is worth noting that VirusTotal labels these samples as "Winniti", which aligns with the YARA rules defined by *Project* [91]. "Winniti" is an alias associated with another APT group, known for its links to China, adding further intrigue to these findings.

Binary Analysis

In this analysis, we focus on executable binaries, therefore we will perform a sequential analysis starting from the entry point. Interestingly, we find that both the entry points and the main functions are exactly the same, including memory addresses.

The differences in the exports region (".edata") appear to be reconciled during the decompilation process. As for the differences in the ".data" section, we suspect they are a result of internal obfuscation techniques employed by the malware.

One notable difference lies in the resource section (".rsrc"), specifically an added icon in Binary18 that is absent in the other binaries.

When examining the visual representation of the binaries, we observe minimal differences between Binary16 and Binary17, indicating a high degree of similarity. However, Binary18 exhibits slight variations compared to the other two binaries. Despite these differences, they do not appear to be significant in terms of functionality. We can conclude that the three binaries are likely adaptations or obfuscated versions of the same underlying malware.

Findings and Insights

The analysis of Binary16, Binary17, and Binary18 has revealed strong similarities, indicating a close relationship between these binaries. The entry points and main functions of all three binaries are identical, further supporting their association. These findings suggest that Binary16, Binary17, and Binary18 are likely adaptations or obfuscated versions of the same malware.

Interestingly, the graphical representation of the embeddings for these binaries exhibits a unique shape, distinct from other use cases. This peculiar shape suggests that Binary16, Binary17, and Binary18 may belong to a subgroup or share a common lineage within the APT group under investigation.

The differences observed in the ".rsrc" section, specifically the presence of an additional icon in

Binary18, further support the notion of variations or modifications within the malware family. These findings underscore the threat actors' efforts to disguise their activities and evade detection by employing obfuscation techniques.

In summary, the findings and insights from Use Case 6 highlight the potential of MAPTER in discovering similarities between binaries through entry point analysis. Furthermore, the curious shape of the graphical representation suggests the presence of a distinct subgroup within the [APT](#), warranting further investigation. This knowledge equips security professionals with a valuable tool to uncover connections and enhance their understanding of the threat landscape.

6. Research Conclusions and Future Work

This chapter presents the culmination of the research conducted in this thesis, providing a comprehensive overview of the key findings and conclusions. It reflects on the achievements of the study, the insights gained, and the implications for the field of APT analysis. Furthermore, this chapter outlines the future directions, aiming to advance the understanding and detection of [APTs](#) through MAPTER. By examining the research conclusions and exploring potential avenues for future investigations, this chapter contributes to the ongoing efforts to enhance cybersecurity and combat the evolving threats posed by [APTs](#).

6.1 Overview of Findings

Through our comprehensive analysis of various use cases, we have uncovered significant findings and gained valuable insights into the field of malware analysis. Our investigation using MAPTER has provided evidence of connections between different binaries, revealing the presence of distinct threat actors and their techniques. Here is a summary of the main findings and contributions:

1. Use Case 1: We confirmed the connection between the analysed binaries and the threat group known as "Black Vine" or "Deep Panda". Additionally, our investigation revealed the utilization of the Microsoft Foundation Classes framework in conjunction with Microsoft Visual C++ in these binaries. This finding not only provides insights into the development methodology employed by APT19, but also raises intriguing questions about their potential hidden intentions and motivations.

2. Use Case 2: We uncovered connections between analysed binaries and the "Codoso" group (also known as APT19), highlighting code reuse and obfuscation techniques employed by the threat actors. The presence of shared functions and variations in code implementation pointed to the evolution of the Derusbi malware.
3. Use Case 3: Our analysis of DLLs revealed similarities and differences among the exported functions, providing insights into code reuse, obfuscation, and potential obfuscated binaries within the sample set.
4. Use Case 4: While limited conclusions could be drawn from this Use Case, the performance of MAPTER in identifying similar functions via the entry point was confirmed. The unique shape of the graphical representation hinted at a distinct subgroup within the APT, warranting further investigation.
5. Use Case 5: The similarities among Binary14 and Binary15 indicated a close relationship, potentially representing adaptations or obfuscated versions of the same malware. We uncovered significant findings and insights regarding the potential exploitation of vulnerabilities in UPnP devices.
6. Use Case 6: The analysis of Binary16, Binary17, and Binary18 suggested their association as adaptations or obfuscated versions of the same malware. The presence of an additional icon in Binary18 highlighted variations within the malware family. Use case 6 highlights the potential of MAPTER in discovering similarities between binaries through entry point analysis

These findings contribute to a deeper understanding of the behaviour, tactics, and motives of threat actors, enabling the development of effective mitigation strategies and improved malware analysis techniques.

6.2 Closing Thoughts

In this master thesis, we have proposed a modular and adaptable framework with the aim of improving the analysis of APTs malware. After the development of this project and having presented the results obtained, we can affirm that the framework developed, MAPTER: Mapping APTs through Embedding Representation, has proven to be effective in identifying and understanding the characteristics and behaviour of APTs.

During the exploration of various use cases, MAPTER has showcased its effectiveness in detecting similarities and patterns among APTs samples. The integration of machine learning-based approaches, specifically deep learning models, has demonstrated promising results in directly analysing disassembled binary functions without the requirement for manual feature extraction. This approach has significantly enhanced the efficiency and accuracy of malware variants detection and APTs analysis, even in the presence of evasion techniques employed by threat actors.

While the results obtained from deep learning are encouraging, it is important to acknowledge that there is still room for improvement. Deep learning should be utilized as a supportive tool rather than a definitive solution. Combining AI-based technologies with human expertise can provide a more comprehensive and reliable approach to APTs analysis. Human analysts can leverage their domain knowledge and contextual understanding, to interpret and validate the insights generated by MAPTER, ensuring accurate and actionable intelligence.

The specific targeted analysis conducted using MAPTER has enabled the visualization and correlation of APTs function embeddings, facilitating the identification of sensitive code fragments and enhancing the comprehension of the APT19 group. The empirical validation conducted to assess the significance and reliability of the identified correlations has further strengthened the credibility and robustness of the MAPTER framework. The validation process has confirmed the efficacy of MAPTER in revealing valuable insights into the structure, behaviour and traceability of malware variants within APTs malware.

During the OSINT process utilizing VirusTotal, we have recognized the significance of this tool in malware analysis. However, it is essential to exercise caution when relying solely on the labels assigned by antivirus engines integrated into VirusTotal, as they may not always accurately reflect the true nature of the analysed samples. Therefore, it is crucial to validate and cross-reference the information provided by VirusTotal with additional reliable sources to ensure accurate and reliable insights into the analysed samples.

In conclusion, the MAPTER framework has addressed the challenges associated with APTs analysis by providing a novel approach that combines machine learning-based techniques, correlation analysis through visualization, and targeted analysis. The insights gained from this research contribute to the field of cybersecurity, enabling a better understanding of APTs and empowering organizations to detect, analyse, and mitigate these sophisticated threats effectively. Future developments of MAPTER will focus on extending its capabilities, incorporating new machine learning algorithms and designing more specific targeted analyses that will provide deeper insights into the characteristics and behaviour of APTs. By staying at the forefront of developments in the APTs landscape, MAPTER will continue to provide valuable insights and empower organizations in

their fight against APTs.

6.3 Future Research Directions

Our research has laid the foundation for future exploration in the field of APTs malware analysis. Building upon the MAPTER framework, there are several promising research directions that can be pursued:

- Extend the application of MAPTER to newer and more diverse datasets of APTs. By analysing a wider range of samples, including those from emerging threat groups, we can enhance the effectiveness and generalizability of the framework.
- Investigate the adoption of advanced machine learning models to the embedding extraction step, that can better capture the intricacies of code fragments within APTs binaries. While MAPTER has shown promising results, exploring alternative models that can handle small or large functions more effectively could further improve the accuracy and granularity of the analysis.
- Conduct a more in-depth examination of the unique shape of graphical representations within subgroups of APT19. By delving into these subgroups and their distinct graphical patterns, we can gain deeper insights into their characteristics, behaviours, and potentially identify distinguishing features that can aid in attribution and detection.
- Apply the proposed targeted analysis to other APT groups beyond APT19. By leveraging the capabilities of MAPTER, we can uncover valuable insights and correlations within different APTs families, expanding our understanding of their tactics, techniques, and procedures.
- Develop and conduct new specific targeted analyses using MAPTER. By tailoring the analysis to specific research questions or objectives, we can extract further insights from APTs and delve into specific aspects of their operations, such as lateral movement, persistence mechanisms, or data exfiltration techniques.
- Explore function classification algorithms to identify interesting groups of functions and facilitate the analysis of correlations between different binaries. By categorizing functions based on their behaviours or functionality, we can enhance our understanding of the relationships and connections within APTs samples.

- Develop enhanced machine learning models and algorithms specifically designed for malware analysis, taking into account the identified patterns, shared functions, and variations observed in the analysed binaries. These advancements can improve the accuracy, efficiency, and interpretability of malware detection and attribution.

By pursuing these research directions, we can continue to advance the field of APTs malware analysis, improve our ability to detect and attribute APTs attacks, and develop robust countermeasures to mitigate the impact of these sophisticated threats.

Bibliography

- [1] L. Massarelli, G. A. Di Luna, F. Petroni, R. Baldoni, and L. Querzoni, “Safe: Self-attentive function embeddings for binary similarity,” in *Detection of Intrusions and Malware, and Vulnerability Assessment: 16th International Conference, DIMVA 2019, Gothenburg, Sweden, June 19–20, 2019, Proceedings 16*, Springer, 2019, pp. 309–329. [Online]. Available: <https://arxiv.org/pdf/1811.05296.pdf>.
- [2] Cisco, *Cisco annual internet report (2018–2023) white paper*, Accessed: 2023-05-04, 2020. [Online]. Available: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>.
- [3] M. M. Rashid, J. Kamruzzaman, M. M. Hassan, T. Imam, and S. Gordon, “Cyberattacks detection in iot-based smart city applications using machine learning techniques,” *International Journal of environmental research and public health*, vol. 17, no. 24, p. 9347, 2020. [Online]. Available: <https://www.mdpi.com/1660-4601/17/24/9347>.
- [4] F. Cremer, B. Sheehan, M. Fortmann, *et al.*, “Cyber risk and cybersecurity: A systematic review of data availability,” *The Geneva Papers on Risk and Insurance-Issues and Practice*, vol. 47, no. 3, pp. 698–736, 2022. [Online]. Available: <https://link.springer.com/article/10.1057/s41288-022-00266-6>.
- [5] M. H. Uddin, M. H. Ali, and M. K. Hassan, “Cybersecurity hazards and financial system vulnerability: A synthesis of literature,” *Risk Management*, vol. 22, no. 4, pp. 239–309, 2020.
- [6] AV-TEST GmbH. “Malware statistics.” Accessed: 2023-05-20. (), [Online]. Available: <https://www.av-test.org/en/statistics/malware/>.
- [7] National Institute of Standards and Technology. “Apt definition(s).” Accessed: 2023-05-20. (), [Online]. Available: <https://csrc.nist.gov/glossary/term/apt>.

- [8] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, "A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1851–1877, 2019. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8606252>.
- [9] P. Chen, L. Desmet, and C. Huygens, "A study on advanced persistent threats," in *Communications and Multimedia Security: 15th IFIP TC 6/TC 11 International Conference, CMS 2014, Aveiro, Portugal, September 25-26, 2014. Proceedings 15*, Springer, 2014, pp. 63–72. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-662-44885-4_5.
- [10] K. Shaukat, S. Luo, and V. Varadharajan, "A novel deep learning-based approach for malware detection," *Engineering Applications of Artificial Intelligence*, vol. 122, p. 106 030, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0952197623002142>.
- [11] OpenAI, *Chatgpt: Optimizing language models for dialogue*, Accessed: 2023-05-05, 2022. [Online]. Available: <https://openai.com/blog/chatgpt/>.
- [12] S. AI, *Stable diffusion*, Accessed: 2023-05-05, 2022. [Online]. Available: <https://github.com/Stability-AI/stablediffusion>.
- [13] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *Journal of Network and Computer Applications*, vol. 153, p. 102 526, 2020, ISSN: 1084-8045. DOI: <https://doi.org/10.1016/j.jnca.2019.102526>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1084804519303868>.
- [14] T. West and A. Zentner, "Threats and major data breaches: Securing third-party vendors," *Available at SSRN 3532024*, 2019. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3532024.
- [15] Ö. A. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249–6271, 2020. DOI: [10.1109/ACCESS.2019.2963724](https://doi.org/10.1109/ACCESS.2019.2963724). [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8949524>.
- [16] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, "The evolution of android malware and android analysis techniques," *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, pp. 1–41, 2017. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3017427>.
- [17] G. D. S. AG, *Malware family naming hell*, Accessed: 2023-06-01. [Online]. Available: <https://www.gdatasoftware.com/blog/malware-family-naming-hell>.

- [18] R. Tahir, "A study on malware and malware detection techniques," *International Journal of Education and Management Engineering*, vol. 8, no. 2, p. 20, 2018. [Online]. Available: <http://www.mecs-press.net/ijeme/ijeme-v8-n2/IJEME-V8-N2-3.pdf>.
- [19] Z. Akhtar, "Malware detection and analysis: Challenges and research opportunities," *arXiv*, 2021. [Online]. Available: <https://arxiv.org/abs/2101.08429>.
- [20] M. Baezner and P. Robin, "Stuxnet," ETH Zurich, Tech. Rep., 2017. [Online]. Available: <https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/200661/1/Cyber-Reports-2017-04.pdf>.
- [21] E. M. Hutchins, M. J. Cloppert, R. 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*, vol. 1, no. 1, p. 80, 2011. [Online]. Available: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=ca18aa98d4d1d434802eec54c2ba6ea8cf493b88#page=123>.
- [22] T. Yadav and A. M. Rao, "Technical aspects of cyber kill chain," in Springer International Publishing, 2015, pp. 438–452. DOI: [10.1007/978-3-319-22915-7_40](https://doi.org/10.1007/978-3-319-22915-7_40). [Online]. Available: <https://arxiv.org/pdf/1606.03184.pdf>.
- [23] S. S. Chakkaravarthy, D. Sangeetha, and V. Vaidehi, "A survey on malware analysis and mitigation techniques," *Computer Science Review*, vol. 32, pp. 1–23, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1574013718301114>.
- [24] M. Sikorski and A. Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*, 1st. USA: No Starch Press, 2012, ISBN: 1593272901.
- [25] R. Team, *Radare2 github repository*, Accessed: 2023-05-20, 2017. [Online]. Available: <https://github.com/radare/radare2>.
- [26] R. Team, *Radare2 Book*. GitHub, 2017, Accessed: 2023-04-28. [Online]. Available: <https://radare.gitbooks.io/radare2book/content/>.
- [27] I. Guilfanov and Hex-Rays, *Interactive disassembler*, Accessed: 2023-05-20, 1991. [Online]. Available: <https://hex-rays.com/ida-free/>.
- [28] N. S. Agency, *Ghidra - software reverse engineering framework*, Accessed: 2023-05-20, 2019. [Online]. Available: <https://github.com/NationalSecurityAgency/ghidra>.
- [29] U. Noor, Z. Anwar, T. Amjad, and K.-K. R. Choo, "A machine learning-based fintech cyber threat attribution framework using high-level indicators of compromise," *Future Generation Computer Systems*, vol. 96, pp. 227–242, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167739X18326141>.

- [30] O. Catakoglu, M. Balduzzi, and D. Balzarotti, "Automatic extraction of indicators of compromise for web applications," in *Proceedings of the 25th international conference on world wide web*, 2016, pp. 333–343. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/2872427.2883056>.
- [31] E. Raff, R. Zak, G. Lopez Munoz, *et al.*, "Automatic yara rule generation using biclustering," in *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, 2020, pp. 71–82. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3411508.3421372>.
- [32] VirusTotal, *Yara: The pattern matching swiss knife*, Accessed: 2023-05-18, 2021. [Online]. Available: <https://virustotal.github.io/yara/>.
- [33] T. M. Mitchell, *Machine Learning*. New York, NY: McGraw-Hill, 1997.
- [34] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. The MIT Press, 2012, ISBN: 026201825X.
- [35] E. G. Dada, J. S. Bassi, H. Chiroma, S. M. Abdulhamid, A. O. Adetunmbi, and O. E. Ajibuwa, "Machine learning for email spam filtering: Review, approaches and open research problems," *Heliyon*, vol. 5, no. 6, e01802, 2019, ISSN: 2405-8440. DOI: <https://doi.org/10.1016/j.heliyon.2019.e01802>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2405844018353404>.
- [36] S. Pai, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "Clustering for malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 13, pp. 95–107, 2017. [Online]. Available: <https://link.springer.com/article/10.1007/s11416-016-0265-3>.
- [37] Y. Feng, J. Li, and T. Nguyen, "Application-layer ddos defense with reinforcement learning," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, 2020, pp. 1–10. DOI: [10.1109/IWQoS49365.2020.9213026](https://doi.org/10.1109/IWQoS49365.2020.9213026). [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9213026>.
- [38] Google, *Different approaches to machine learning*, 2022. [Online]. Available: https://storage.googleapis.com/gweb-news-initiative-training.appspot.com/upload/Lesson3%5C_1.pdf.
- [39] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: <http://www.deeplearningbook.org>.
- [40] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computers & Security*, vol. 81, pp. 123–147, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167404818303808>.

- [41] Ö. A. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249–6271, 2020. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8949524>.
- [42] D. Kobak and P. Berens, "The art of using t-sne for single-cell transcriptomics," *Nature communications*, vol. 10, no. 1, p. 5416, 2019. [Online]. Available: <https://www.nature.com/articles/s41467-019-13056-x>.
- [43] L. F. M. Liras, A. R. de Soto, and M. A. Prada, "Feature analysis for data-driven apt-related malware discrimination," *Computers & Security*, vol. 104, p. 102 202, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167404821000262>.
- [44] C. Do Xuan, M. H. Dao, and H. D. Nguyen, "Apt attack detection based on flow network analysis techniques using deep learning," *Journal of Intelligent & Fuzzy Systems*, vol. 39, no. 3, pp. 4785–4801, 2020. [Online]. Available: <https://content.iospress.com/articles/journal-of-intelligent-and-fuzzy-systems/ifs200694>.
- [45] I. Rosenberg, G. Sicard, and E. David, "Deepapt: Nation-state apt attribution using end-to-end deep neural networks," in *Artificial Neural Networks and Machine Learning–ICANN 2017: 26th International Conference on Artificial Neural Networks, Alghero, Italy, September 11–14, 2017, Proceedings, Part II* 26, Springer, 2017, pp. 91–99. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-68612-7_11.
- [46] I. U. Haq and J. Caballero, "A survey of binary code similarity," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–38, 2021. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3446371>.
- [47] H. Wang, W. Qu, G. Katz, *et al.*, "Jtrans: Jump-aware transformer for binary code similarity," *arXiv preprint arXiv:2205.12713*, 2022. [Online]. Available: <https://arxiv.org/abs/2205.12713>.
- [48] S. H. Ding, B. C. Fung, and P. Charland, "Kam1n0: Mapreduce-based assembly clone search for reverse engineering," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 461–470. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/2939672.2939719>.
- [49] Y. David and E. Yahav, "Tracelet-based code search in executables," *Acm Sigplan Notices*, vol. 49, no. 6, pp. 349–360, 2014.
- [50] H. Huang, A. M. Youssef, and M. Debbabi, "Binsequence: Fast, accurate and scalable binary code reuse detection," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 155–166. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3052973.3052974>.

- [51] D. Gao, M. K. Reiter, and D. Song, "Binhunt: Automatically finding semantic differences in binary programs," in *Information and Communications Security: 10th International Conference, ICICS 2008 Birmingham, UK, October 20-22, 2008 Proceedings 10*, Springer, 2008, pp. 238–255. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-88625-9_16.
- [52] J. Ming, M. Pan, and D. Gao, "Ibinhunt: Binary hunting with inter-procedural control flow," in *International Conference on Information Security and Cryptology*, Springer, 2012, pp. 92–109. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-37682-5_8.
- [53] X. Zhu, L. Jiang, and Z. Chen, "Cross-platform binary code similarity detection based on nmt and graph embedding," *Mathematical Biosciences and Engineering*, vol. 18, no. 4, pp. 4528–4551, 2021. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/2666356.2594343>.
- [54] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural network-based graph embedding for cross-platform binary code similarity detection," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 363–376. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3133956.3134018>.
- [55] S. Ahn, S. Ahn, H. Koo, and Y. Paek, "Practical binary code similarity detection with bert-based transferable similarity learning," in *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022, pp. 361–374. [Online]. Available: <https://dl.acm.org/doi/abs/10.1145/3564625.3567975>.
- [56] Z. Yu, R. Cao, Q. Tang, S. Nie, J. Huang, and S. Wu, "Order matters: Semantic-aware neural networks for binary code similarity detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 1145–1152. [Online]. Available: <https://doi.org/10.1609/aaai.v34i01.5466>.
- [57] S. H. Ding, B. C. Fung, and P. Charland, "Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization," in *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 472–489. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8835340>.
- [58] Authors. "Word2vec skip-gram implementation in tensorflow." Accessed: 2023-05-29. (), [Online]. Available: <https://www.tensorflow.org/tutorials/text/word2vec?hl=en>.
- [59] Z. Lin, M. Feng, C. N. dos Santos, *et al.*, *A structured self-attentive sentence embedding*, 2017. arXiv: 1703.03130 [cs.CL]. [Online]. Available: <https://arxiv.org/abs/1703.03130>.

- [60] K. Cho, B. Van Merriënboer, C. Gulcehre, *et al.*, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014. [Online]. Available: <https://arxiv.org/abs/1406.1078>.
- [61] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018. [Online]. Available: <https://arxiv.org/abs/1810.04805>.
- [62] C. Research, *Aptmalware*, Accessed: 2023-05-01, 2019. [Online]. Available: <https://github.com/cyber-research/APTMalware>.
- [63] A. Sharma, B. B. Gupta, A. K. Singh, and V. Saraswat, “Orchestration of apt malware evasive manoeuvres employed for eluding anti-virus and sandbox defense,” *Computers & Security*, vol. 115, p. 102 627, 2022.
- [64] D. Sahoo, “Cyber threat attribution with multi-view heuristic analysis,” *Handbook of Big Data Analytics and Forensics*, pp. 53–73, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167404822000268>.
- [65] D. Sahoo, “Cyber threat attribution with multi-view heuristic analysis,” *Handbook of Big Data Analytics and Forensics*, pp. 53–73, 2022. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-74753-4_4.
- [66] N. Xu, S. Li, X. Wu, W. Han, and X. Luo, “An apt malware classification method based on adaboost feature selection and lightgbm,” in *2021 IEEE Sixth International Conference on Data Science in Cyberspace (DSC)*, IEEE, 2021, pp. 635–639. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9750513>.
- [67] J. Gray, D. Sgandurra, and L. Cavallaro, “Identifying authorship style in malicious binaries: Techniques, challenges & datasets,” *arXiv preprint arXiv:2101.06124*, 2021. [Online]. Available: <https://arxiv.org/abs/2101.06124>.
- [68] L. Massarelli, G. A. Di Luna, F. Petroni, L. Querzoni, and R. Baldoni, *Safetorch: Pytorch version of the safe neural network*, Accessed on 2023-06-04, 2019. [Online]. Available: <https://github.com/facebookresearch/SAFEtorch>.
- [69] A. Paszke, S. Gross, F. Massa, *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.

- [70] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, “Scikit-learn: Machine learning in python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: <https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf?ref=https://>.
- [71] P. T. Inc. “Collaborative data science.” Accessed on 2023-06-04. (2015), [Online]. Available: <https://plot.ly>.
- [72] A. Lemay, J. Calvet, F. Menet, and J. M. Fernandez, “Survey of publicly available reports on advanced persistent threat actors,” *Computers & Security*, vol. 72, pp. 26–59, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167404817301608>.
- [73] M. ATT&CK, *Apt19*, Accessed: 2023-05-27. [Online]. Available: <https://attack.mitre.org/groups/G0073/>.
- [74] M. ATT&CK, *Deep panda*, Accessed: 2023-05-27. [Online]. Available: <https://attack.mitre.org/groups/G0009/>.
- [75] F. Roth, *Apt groups and operations spreadsheet*, Accessed: 2023-05-27. [Online]. Available: <https://apt.threattracking.com>.
- [76] J. Grunzweig and B. Lee, *New attacks linked to c0d0s0group*, Accessed: 2023-06-05. [Online]. Available: <https://unit42.paloaltonetworks.com/new-attacks-linked-to-c0d0s0-group/>.
- [77] S. S. Response, *The black vine cyberespionage group*, Accessed: 2023-05-30. [Online]. Available: https://web.archive.org/web/20170823094836/http://www.symantec.com/content/en/us/enterprise/media/security%5C_response/whitepapers/the-black-vine-cyberespionage-group.pdf.
- [78] I. Ahl, *Phished at the request of counsel*, Accessed: 2023-05-30. [Online]. Available: <https://www.mandiant.com/resources/blog/phished-at-the-request-of-counsel>.
- [79] F. Roth, *Ioc database for loki scanner*, Accessed: 2023-06-03. [Online]. Available: <https://raw.githubusercontent.com/Neo23x0/signature-base/master/iocs/hash-iocs.txt>.
- [80] F. Roth, *Yara rules fr*, Accessed: 2023-06-03. [Online]. Available: https://github.com/Neo23x0/signature-base/blob/master/yara/apt%5C_scanbox%5C_deeppanda.yar.

- [81] T. I. R. Team, *The anthem hack: All roads lead to china*, Accessed: 2023-06-03. [Online]. Available: https://paper.seebug.org/papers/APT/APT%5C_CyberCriminal%5C_Campagin/2015/Anthem%5C_hack%5C_all%5C_roads%5C_lead%5C_to%5C_China.pdf.
- [82] S. F. Users, *Anthem attack*, Accessed: 2023-06-03. [Online]. Available: <https://forums.spybot.info/archive/index.php/t-71985.html>.
- [83] AlienVault-OTX, *Alien vault yara rules*, Accessed: 2023-06-03. [Online]. Available: <https://raw.githubusercontent.com/AlienVault-OTX/OTX-Python-SDK/master/examples/yara.rules>.
- [84] alankrit29, *Deep panda yara*, Accessed: 2023-06-03. [Online]. Available: https://github.com/alankrit29/signature-base/blob/master/apt%5C_scanbox%5C_deeppanda.yar.
- [85] Y. Project, *Malware yara rules*, Accessed: 2023-06-03. [Online]. Available: https://github.com/Yara-Rules/rules/blob/master/malware/MALW%5C_Miscelanea.yar.
- [86] T. Whitney, *Mfc desktop applications*, Accessed: 2023-06-03. [Online]. Available: <https://learn.microsoft.com/en-us/cpp/mfc/mfc-desktop-applications?view=msvc-170>.
- [87] N. Nishant, *Mfc under the hood*, Accessed: 2023-06-03. [Online]. Available: <https://www.codeproject.com/Articles/1672/MFC-under-the-hood>.
- [88] Y. Project, *Yara rules for c0d0s0 group*, Accessed: 2023-06-06. [Online]. Available: https://github.com/Yara-Rules/rules/blob/master/malware/APT%5C_Codoso.yar.
- [89] CyberMasterV, *Analyzing apt19 malware using a step-by-step method*, Accessed: 2023-06-06. [Online]. Available: <https://cybergeeks.tech/analyzing-apt19-malware-using-a-step-by-step-method/>.
- [90] F. Perigaud, *Newcomers in the derusbi family*, Accessed: 2023-06-07. [Online]. Available: https://paper.seebug.org/papers/APT/APT%5C_CyberCriminal%5C_Campagin/2015/2015.12.15.Newcomers%5C_in%5C_the%5C_Derusbi%5C_family/Newcomers-in-the-Derusbi-family.pdf.
- [91] Y. Project, *Winnti malware*, Accessed: 2023-06-07. [Online]. Available: https://github.com/Yara-Rules/rules/blob/master/malware/APT%5C_Winnti.yar.

Acronyms

AI Artificial Intelligence. [2](#), [85](#)

API Application Programming Interface. [25](#), [26](#)

APT Advanced Persistent Threat. [3–5](#), [46–56](#), [59](#), [76](#), [81](#), [82](#), [84](#), [86](#)

APTs Advanced Persistent Threats. [ii](#), [1–5](#), [7](#), [11–13](#), [15–19](#), [22](#), [26](#), [28–30](#), [33](#), [38–41](#), [45–47](#), [49](#), [52](#), [55](#), [56](#), [83–87](#)

BCSD Binary Code Similarity Detection. [5](#), [30](#), [33](#), [34](#), [36](#), [37](#)

C&C Command and Control. [9](#), [14](#)

CFGs Control-Flow Graphs. [33](#), [34](#), [37](#)

CLI Command-Line Interface. [19](#), [21](#)

CNN Convolutional Neural Networks. [34](#)

CTI Cyber Threat Intelligence. [22](#)

DDoS Distributed Denial of Service. [6](#), [9](#), [23](#), [24](#)

DL Deep Learning. [24](#)

GNN Graph Neural Network. [37](#)

GNNs Graph Neural Networks. [34](#), [37](#)

GRU Gated Recurrent Units. [36](#)

GUI Graphical User Interface). [19](#)

IOCs Indicators of Compromise. [21](#), [22](#), [29](#)

IOT Internet of things. [1](#)

LSTM Long Short-Term Memory. [34](#)

MaaS Malware-as-a-Service. [11](#)

MFC Microsoft Foundation Classes. [64](#), [65](#)

ML Machine Learning. [22](#), [25](#), [26](#), [34](#)

NIST National Institute of Standards and Technology. [11](#)

NLP Natural Language Processing. [33](#)

NSA National Security Agency. [20](#)

OSINT Open-Source Intelligence. [48–53](#), [67](#), [75](#)

PCA Principal Component Analysis. [ii](#), [iii](#), [26](#), [27](#), [45](#), [54](#), [56](#), [60](#), [61](#)

PLCs Programmable Logic Controllers. [13](#)

RAT Remote Access Trojan. [14](#), [63](#), [80](#)

ReLU Rectified Linear Unit. [36](#)

t-SNE t-Distributed Stochastic Neighbour Embedding. [ii](#), [iii](#), [27](#), [45](#), [54](#), [56](#), [60](#), [61](#)

TTPs Tactics, Techniques and Procedures. [7](#), [21](#)

UPnP Universal Plug and Play. [78](#)

