UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Real-Time Classification of Malware using a High-Interaction Honeypot with Machine Learning

Miguel Sanches Ganhão Cortinhal Faísco

Mestrado em Segurança Informática

Dissertação orientada por:
Profª. Doutora Ibéria Vitória de Sousa Medeiros
Prof. Doutor Hans Peter Reiser

2025

Acknowledgements

First and foremost, thank God.

I want to thank my family for giving me the means to do my dissertation in a foreign country far away from home, and for their unconditional support throughout the time I was away. I would like to thank my old friends in Portugal and the new friends I made in Iceland, both for their help with work and for pulling my head out of work from time to time and allowing me to rest.

I would like to thank my supervisors, Prof.ª Ibéria Medeiros and Prof. Hans Reiser, for sharing ideas and helping me perfect this work with their vast knowledge, and for giving me the opportunity to produce this work that I so much enjoyed, in every step of the way.

This work was supported by the Frostbyte Cybersecurity Lab, funded by the European Union under grant agreement No 101127453, and by FCT through the LASIGE Research Unit, ref. UID/00408/2023.

Dedicado aos meus pais.

Resumo

Hoje em dia, o público em geral é constantemente alertado de que as guerras do futuro serão travadas em torno de informação, tendo a *Internet* como principal campo de batalha e fonte, onde o controlo sobre dados pode determinar poder e influência. Desde pequenos ataques organizados por um grupo restrito de indivíduos com intenções maliciosas até operações totalmente financiadas por governos, envolvendo milhões em *hardware* e mão-de-obra, qualquer sistema e utilizador pode ser um alvo para benefício financeiro ou informacional. Neste novo formato de guerra, em que qualquer pessoa pode ser o alvo, a defesa assume um papel crucial — conhecer as táticas e ferramentas de potenciais adversários é essencial para o reforço dos sistemas e para a mitigação de ataques. Uma das melhores formas de observar este comportamento é com recurso a *honeypots* - redes ou sistemas concebidos especificamente para atrair atacantes, onde as ações dos adversários são gravadas e podem ser utilizadas para análise.

Outro campo de investigação que atualmente recebe particular atenção é o da Aprendizagem Máquina (*machine learning*, ML), tendo já vários projetos aplicados à informação relacionada com cibersegurança obtido sucesso. Neste campo de investigação têm emergido novos métodos de análise e a um ritmo elevado. É um facto de que tem havido um grande interesse e valor em combinar estes dois mundos distintos na investigação do comportamento e dos métodos de adversários, e por tal é pretenção deste trabalho explorar esta combinação.

Embora os *honeypots* sejam imensamente úteis para a recolha de *malware*, a maioria apresenta grandes limitações no campo da análise deste. Isto é especialmente verdade para os tipos mais básicos de *honeypots* (de baixa e média interação), visto que estes não possuem a complexidade necessária para realizar este tipo de análise sobre os ficheiros e eventos recebidos. Normalmente, esta tarefa acaba por ficar a cargo de um analista de segurança ou da equipa do Centro de Operações de Segurança (SOC) da organização que mantém a rede de *honeypots*, sendo esta responsável por analisar os *logs* recebidos pelos *honeypots*. Tal análise manual levanta os seus próprios problemas, pois assume que a equipa do SOC possui conhecimentos aprofundados na área de *malware* e segurança, e eficiência suficiente para acompanhar o volume de tráfego da rede, que é diariamente inundada por ligações.

Tendo estes problemas em consideração, e de forma a alcançar uma solução mais rentável, algumas organizações recorrem a ferramentas como o Hybrid Analysis para realizar esta análise automaticamente. Embora estas ferramentas apresentem informação útil para tipos de *malware* mais simples, que normalmente reutilizam elementos de outros já existentes, estas ferramentas podem não atender às necessidades de casos mais complexos que envolvem ataques direcionados ou

malware concebido para evitar ambientes de análise. O *malware* desenvolvido com objetivos específicos, como o ataque a determinados sistemas ou redes, pode comportar-se de forma diferente quando analisado em ambientes externos (ou seja, por tais ferramentas), em comparação com o seu alvo original. Isto, em conjunto com técnicas de evasão, como o adiamento de certos comportamentos até que determinadas condições sejam cumpridas, ou a tentativa de contactar servidores de *Command and Control* (C&C) durante uma execução única, pode dificultar a análise quando realizada através de ferramentas externas. Face a todas estas questões, a importância de uma análise interna de *malware* ser efetuada dentro do *honeypot* e em (quase) tempo real torna-se cada vez mais evidente. Tal forma de análise leva a que as organizações possam realizar a análise internamente, e configurar os seus ambientes de análise de forma a reproduzir as suas próprias configurações de rede, o comportamento dos utilizadores e as especificações dos sistemas, captando assim o comportamento completo de *malware* mais complexo e exclusivo. A crescente complexidade e sofisticação dos ciberataques justificam a necessidade urgente de soluções de segurança que vão além das abordagens tradicionais. Embora as tecnologias de *honeypots* atuais se tenham revelado eficazes na atração e captura de atividade maliciosa, o seu verdadeiro potencial é frequentemente negligenciado no domínio das capacidades analíticas integradas.

Os *honeypots* são ferramentas já conhecidas na cibersegurança como modo de observar o comportamento de atacantes e recolher amostras de *malware*. Contudo, *honeypots* tradicionais, em especial os de baixa e média interação, são frequentemente identificáveis como "ambientes simulados" pelos atacantes, e o paradigma dominante de "capturar primeiro, analisar depois" falha em dar resposta a desafios críticos como:

- O *malware* sem ficheiros (fileless) não deixa artefactos físicos adequados (como ficheiros) para análise posterior;
- Ataques em múltiplas fases podem apenas revelar comportamentos maliciosos ou críticos em condições específicas ou em tempo real;
- O *malware* sofisticado evita a sua análise por apresentar um comportamento diferente daquele quando é executado num ambiente de *sandbox* com o objetivo de análise.

Esta dissertação procura responder a estas lacunas, desenvolvendo uma solução de *honeypot* que oferece análise integrada e classificação do *malware* recebido com base nas suas características e através de *machine learning*, permitindo assim uma melhor compreensão e defesa mais eficaz contra atacantes — sendo esta a principal motivação para o projeto. Ao integrar a análise de *malware* diretamente na infraestrutura do *honeypot*, o sistema revela comportamentos que ferramentas externas ou processos manuais poderiam não detetar, contribuindo assim para o avanço nesta área de investigação.

Tendo em consideração os objetivos e desafios anteriormente mencionados, propomos o sistema **MaCHoS** (**Ma**lware **C**lassifying **Ho**neypot **S**ystem), um *honeypot* com capacidades integradas de classificação de *malware* em (quase) tempo real, usando *machine learning*.

Enquanto o *honeypot* em si imita sistemas habitualmente utilizados, como serviços e sistemas operativos, de forma a garantir que os atacantes não os conseguem distinguir facilmente de

alvos legítimos, MaCHoS captura eventos detalhados dos ataques, através dos seus componentes de análise estática e de análise dinâmica, este último baseado nas chamadas de sistema (*system calls*) que recolhe por meio de pontos de instrumentação extended Berkeley Packet Filter (eBPF), incluindo artefactos em memória e ataques em múltiplas fases. O *malware* captado, bem como os eventos ocorridos da sua execução, são analisados por estes dois componentes em tempo real e enquanto o atacante ainda interage com o sistema. MaCHoS classifica o *malware* numa categoria de *malware* que conhece, com base na informação recolhida e recorrendo ao seu componente de aprendizagem máquina (*machine learning*).

O modelo de aprendizagem que integra foi treinado com instâncias de bases públicas de *malware* e processadas pelo VirusTotal, fornecendo-lhe assim verdade absoluta (*ground truth*) sobre os dados de treino. Este processo requeriu algum pré-processamento e redução da quantidade de colunas dos nossos dados, de modo a reduzir o tempo de treino do modelo sem sacrificar o seu desempenho. Para a escolha do modelo, embora estudos anteriores já tenham sugerido alguns modelos como sendo os mais adequados para tarefas de classificação de *malware*, decidimos conduzir o nosso próprio estudo utilizando vários modelos, treinando-os sobre os nossos dados, de forma a retirarmos as nossas próprias conclusões quanto ao modelo mais adequado para o nosso caso de uso. Deste estudo concluímos que o melhor modelo para o MaCHoS é o *Random Forest*.

Quanto ao *honeypot* em si, este foi concebido com o objetivo de se assemelhar a um sistema real com credenciais fracas, permitindo que os atacantes se conectem facilmente a ele e realizem os ataques neste mesmo sistema. Simultaneamente, estes ataques são guardados no controlador do *honeypot*, bem como quaisquer ficheiros enviados pelos atacantes. Para isto, o *honeypot* utiliza o módulo de eBPF que deteta a execução de determinadas *system calls* (definidas previamente) ao nível do *kernel* e encaminha os eventos relacionados com a execução dessas *system calls* para o controlador, através de um *listener*. Os eventos relevantes e relacionados com o comportamento de *malware* são encaminhados para um analisador que os agrega num relatório de características semelhante aos presentes no conjunto de dados usado para treinar o modelo de *machine learning*. Quando a execução do *malware* é interrompida ou o *honeypot* é reiniciado, este relatório é enviado para o classificador, que fornece uma classificação para esse *malware*, cumprindo assim o seu propósito na arquitetura do sistema.

O sistema MaCHoS foi comparado com uma abordagem padrão, representativa das soluções tradicionais de *honeypot* e análise atualmente existentes. O MaCHoS superou a abordagem padrão na captura de atividade de atacantes e seus comportamentos, bem como na consideração de técnicas complexas de ataque, tais como ataques em múltiplas fases e *malware* sem ficheiros. A classificação das amostras de *malware* recebidas na abordagem proposta esteve também ao nível da abordagem padrão. Esta dissertação contribuiu para o campo de pesquisa em *honeypots*, com especial atenção à inclusão de análise na arquitetura dos mesmos, bem como no desenvolvimento de novas técnicas para análise de ataques e seus autores.

Palavras-chave: Honeypots, Classificação de Malware, Rastreamento de Eventos, Aprendizagem Máquina, Inteligência sobre Ciber-Ameaças

Abstract

As systems get more numerous, complex and robust, so do the attacks against said systems, which in turn requires further effort on the defensive side. The use of honeypots (decoy systems or networks designed to attract attackers) has become essential to analyse the behaviour and methodology of attackers as a way to learn about attacking trends and to develop new methods to more swiftly defend against said attacks. Additionally, with the current rise of machine learning, it is easier to automatically detect and classify attacks within log files, externally uploaded files, and other entry points to these systems, further aiding in system defence, and in acquiring intelligence on attackers. Current honeypot solutions do not take advantage of these techniques, being merely used for sample collection, and thus leaving analysis for external tools or to be done manually, which may prove ineffective against more complex threats. With these issues in mind, we propose MaCHoS (Malware Classifying Honeypot System), a novel honeypot system with local embedded analysis. This analysis is done by a machine learning based classifier, that classifies malware sent to the honeypot by its static and dynamic traces, which are extracted from the malware execution in the target system in real time. These dynamic features are extracted through system call monitoring powered by eBPF (extended Berkeley Packet Filter). The evaluation performed on our solution proved that it can be a solid competitor and surpass current approaches, in fields such as attacker attractiveness, behavioural capturing, complex malware considerations, and classifier effectiveness.

Keywords: Honeypots, Malware Classification, Behaviour Tracing, Machine Learning, Cyber Threat Intelligence

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Nowadays, the general public is constantly warned about how future wars will be waged over information, with the Internet as the primary battleground and source, where control over data can determine power and influence [1]. From small attacks organized by a small group of individuals with malicious intentions to fully fledged government funded operations using millions worth of hardware and labour, every system and user can be targeted for financial or informational gain. In this new form of warfare, where anyone can be targeted, defence has mesmerizing value – knowing the tactics and tools of potential adversaries is essential for the strengthening of systems and for anticipating attacks. One of the best ways to observe this behaviour is by using honeypots, networks or systems designed specifically to attract attackers, where adversary actions are recorded and can be used for analysis [2].

Another field of research that is currently receiving particular attention is machine learning (ML), with multiple successful projects applied to cybersecurity related data, with a very active landscape and novel methods of analysis appearing at a high rate. There appears to be great value in combining these two distinct worlds for research in attacker behaviour and methods, and that is exactly what this work intends to explore.

The following sections in this opening chapter outline the research problem we want to solve, the motivation behind the problem, the goals we set in order to achieve a solution for this same problem, the contributions of the disseration, and finally a short structure of the rest of the dissertation.

1.1 Problem

Honeypots, as great as they are for malware collection and logging, most are strongly lacking in the field of malware analysis. This is specifically true for the most basic type of honeypots (i.e. with low and medium interaction), since these do not have enough complexity to perform this type of analysis on the samples and logs received. This usually leaves the analysis to be made by the security analyst or the Security Operations Center (SOC) team within the organization that owns the honeypot network to analyse the logs that these honeypots receive. This has its own set of problems, as the SOC team must have extensive knowledge in the field of malware and security,

and be efficient enough to keep up with the amount of samples reaching the network, which is inundated by connections on a daily basis.

With these problems in mind, and in order to reach a better cost-effective solution, some resort to tools like Hybrid Analysis [1] to perform this analysis automatically. And, while these tools do provide useful information for more simple types of malware that usually use elements copied from other already existing malware, they may not fully address the needs of more complex cases involving targeted attacks or analysis evading malware [3]. Malware designed with specific objectives, such as targeting particular systems or networks, may behave differently when analysed in external environments, compared to its intended target. This, along with incorporated evasion techniques, such as delaying certain behaviours until specific conditions are met, or attempting to contact Command-And-Control (C&C) servers during a single-use execution, can difficult the analysis if made through the use of external tools [4].

With all these issues in mind, the importance of (near) real-time internal malware analysis becomes increasingly apparent, and by performing the analysis internally, organizations can configure their analysis environments to mimic their own network configurations, user behaviour, and system specifics, to capture the full operational behaviour of complex, environment-specific malware. With this, to therefore solve the overarching problem, the main goal of this dissertation is described as:

> *Developing a high-interaction honeypot system with an real-time integrated malware classification component using machine learning, trained on a dataset composed of recent malware from trusted sources.*

1.2 Motivation

The increasing complexity and sophistication of cyberattacks highlight the urgent need for solutions that go beyond traditional security approaches. While current honeypot solutions have proven effective at attracting and capturing malicious activity, their full potential is overlooked in the field of integrated analytical capabilities.

Honeypots have long been established tools in cybersecurity for observing attacker behaviour and collecting malware samples. However, traditional honeypots, especially low- and medium-interaction types, are often identifiable as simulated environments by attackers, and the dominant "capture first, analyse later" paradigm fails to address critical challenges:

- Fileless malware does not leave suitable physical artefacts (like malware files) for later analysis;
- Multi-stage attacks may reveal critical payloads or behaviours only under specific real-time conditions;
- Sophisticated malware evades analysis by behaving differently when later executed in an analysis sandbox.

[1] Crowdstrike Hybrid Analysis, https://hybrid-analysis.com

This dissertation addresses this gap, by developing a honeypot solution that provides integrated analysis and machine learning based classification of received malware through its features, enabling a better understanding, and ultimately defence against attacker behaviour, and this is where the motivation for the work stems from. By integrating malware analysis directly within the honeypot infrastructure, the system uncovers behaviours that external tools or manual processes might miss, such as the nuanced strategies employed by advanced persistent threats, thus advancing the state of research in this field.

1.3 Objectives

As previously explained, the overarching goal of this work is to design and implement a honeypot architecture that provides deeper insights into cyberattacks, through the analysis of their behaviour and the capabilities of the malware the attackers use. To achieve this, the dissertation defines the following specific research questions, which will be addressed further into the work:

- **RQ1:** What data sources and selection process is necessary to build a ML-model ready dataset comprised of Linux binaries?
- **RQ2:** What steps and methodologies should be taken into account when building a machine learning model for the purpose of classifying malware samples based on their static and dynamic analysis features?
- **RQ3:** What architectural and functional design choices need to be considered to build a high-interaction honeypot with an embedded static and dynamic analysis component, machine-learning based malware classification, and stealthiness capabilities?
- **RQ4:** Can a machine-learning based malware classification system integrated in a high-interaction honeypot provide more effective and insightful malware analysis than classical approaches that separate collection and subsequent analysis with third-party services?
- **RQ5:** Can system call (syscall) tracing based dynamic analysis provide additional insight on malware / attacker behaviour and help with its respective classification?

1.4 Contributions

With our goal and research questions defined, the contributions of the present dissertation are as follows:

- *Study on Relevant Topics:* A study on relevant topics, such as Linux malware and its families, static and dynamic analysis methodologies and tools, honeypot systems, machine learning based malware classification, and virtual machine monitoring based attacker behaviour tracing and analysis.

- *System Architecture:* Designing a novel honeypot system with embedded syscall tracing, static and dynamic analysis, and machine learning based malware classification.
- *MaCHoS System:* Development and implementation of the aforementioned system and its 3 main components: the honeypot used for behaviour capturing, the dataset used to train the machine learning classifier, and the classifier itself.
- *System Evaluation:* A multi-layered evaluation of the finalised system based on tracing, classification, and luring capabilities, with the use of a classical approach based system to use as a baseline for comparison throughout the evaluation.

So far, the work and results from this dissertation have been presented in the form of:

- A paper and poster titled "Towards Real-Time Malware Classification Through Honeypot Analysis" presented at the 20th European Dependable Computing Conference (EDCC), Lisboa, Portugal [5], in which the poster won the Distinguished Poster Award.
- A paper titled "Evaluating eBPF as an Alternative to Virtual Machine Introspection for High-Interaction Honeypot Implementation", at the 14th Latin-American Symposium on Dependable and Secure Computing (LADC) to be presented in October in Valparaíso, Chile [6].
- A presentation in the Frostbyte Cybersecurity Workshop, at Reykjavík University, Iceland.
- Malguessr, a malware classifying game presented at the IT event UTmessan 2025 in Reykjavík, Iceland.

1.5 Structure of the document

The following chapters of the document will be organised as follows: We will start by addressing background and related work regarding relevant topics required to understand the proposed solution (Chapter 2). We will then proceed to describe the proposed solution architecture along with a few challenges faced while developing it (Chapter 3). The following chapter will describe in greater detail the process behind creating and implementing the various components of our solution, as well as any challenges met along the way (Chapter 4). Finally, we will present the evaluation of our solution (Chapter 5), and extracting conclusions of this same evaluation, along with the project as a whole (Chapter 6).

Chapter 2

Background & Related Work

This chapter provides insight into the essential concepts to understand the context of the current proposal and forthcoming work. The chapter starts by explaining what malware is, exemplifying with different families, and methodologies in malware analysis (Section 2.1). Next, Section 2.2 describes honeypots, and how they are classified, followed by Section 2.3 explaining the role of machine learning for extracting information and classifying malware. Lastly, Section 2.4, provides an explanation about the role of virtual machine (VM) analysis solutions in capturing malware behaviour from a live machine.

2.1 Malware

One of the easiest ways of compromising a vulnerable system is to get a previously made malware sample to be executed on the said system. Malware has become rampant among the current web landscape, and every user has to be aware of what files are being sent to them, in order to preserve their machines and systems as much as possible. The term malware stems from the combination of the words “malicious” and “software”, and is used to define any type of unwanted software [7]. From a corporate perspective, malware can also be defined as “any code that an organization is willing to pay money to remove” [8]. There are a plethora of different malware types and families, each with different methodologies, techniques, and goals, forcing the defence into becoming a jack of all trades, in an attempt to protect against as many malware kinds as possible [9]. The following list will present a description of selected Linux malware families which will be addressed later in our implementation. One aspect to note is that some of these families have been adapted from their Windows counterparts, which have had much more extensive analysis. In cases where the documentation of the Linux version of the malware is not sufficient or non-existent, the supporting documentation will be focused on the Windows version of the malware, with the same capabilities.

Table 2.1: Malware Families and their respective description

Family Name	Description
ares	A Python-based Remote Access Trojan (RAT) with capabilities such as command execution, second-stage payload downloading, and screen capturing[1]
berbew	Infostealer backdoor targeting financial institutions[2]
dofloo	Used to create DDoS botnets and mining cryptocurrency[3]
gafgyt	IoT targeting botnet for DDoS attacks, with command execution and second-stage payload downloading capabilities.[4]
kaiji	GoLang-based DDoS malware that spreads through SSH brute force attacks[5]
local	Exploits vulnerabilities on older Linux systems, such as CVE-2009-2698[6]
mayday	Used to create botnets, all bots connected by P2P[7]
mirai	Successor of Gafgyt, IoT targeting botnet for DDoS attacks, with command execution and second-stage payload downloading capabilities.[8]
multiverze	Generic detection label for variants of kaiji, mirai, etc.[9]
prometei	Multi-stage cryptocurrency mining botnet[10]
setag	Trojan downloader exploiting Log4Shell vulnerability in Java-based applications[11]
tsunami	Used to create DDoS botnets, utilizes IRC to communicate with attacker[12]
xmrig	Utilizes system CPU to mine Monero cryptocurrency, usually bundled with legitimate software[13]
xorddos	Trojan for endpoints and servers, with credential stealing, second stage payload downloading, rootkit installing and DDoS botnet capabilities[14]

2.1.1 Types of Malware Analysis

Malware analysis is a critical process within cybersecurity, aimed at understanding the behaviour and techniques of malicious software. This analysis helps to develop strategies to mitigate and defend against threats. There are two main types of malware analysis: *static analysis*, and *dynamic analysis*.

Static Analysis

Static analysis, in the context of malware analysis, is a method that examines binary code without executing it. Through the use of tools for disassembly and decompilation, it is possible to more closely analyse how the software interacts with the system in a human-interpretable, assembly

[1] https://www.broadcom.com/support/security-center/protection-bulletin/linux-variant-of-ares-rat-among-various-payloads-delivered-in-latest-campaigns-by-sidecopy-apt
[2] https://otx.alienvault.com/pulse/59e76d9a22f0e07a549552b0
[3] https://malpedia.caad.fkie.fraunhofer.de/details/elf.dofloo
[4] https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/gafgyt
[5] https://malpedia.caad.fkie.fraunhofer.de/details/elf.kaiji
[6] https://otx.alienvault.com/malware/Exploit:Linux/Local/
[7] https://otx.alienvault.com/malware/TrojanDownloader:Linux/Mayday/
[8] https://github.com/jgamblin/Mirai-Source-Code
[9] https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Linux/Multiverze
[10] https://malpedia.caad.fkie.fraunhofer.de/details/elf.tsunami
[11] https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor:Linux/Setag.C
[12] https://malpedia.caad.fkie.fraunhofer.de/details/elf.tsunami
[13] https://malpedia.caad.fkie.fraunhofer.de/details/elf.xmrig
[14] https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Linux/Xorddos!rfn

language representation of the program and its respective logic. Static analysis is also useful for acquiring information from executable file binaries, such as header and section information (such as size, entropy, starting address), strings, imports and exports, shared library usage, etc. Next, we present an overview of some of forms of static analysis.

- **File Information**
  Consists on analysing the files header and other metadata. This type of analysis is important, since a lot of malware has distinctive characteristics in its header, such as lack of common file information. A file's metadata can reveal other important information, such as imports and exports, which can reveal important information about the capabilities of a file without having to look at any code, and the file's sections and their respective names, entrypoints, size and entropy (randomness), which if unorthodox may be a strong indicative of obfuscation or self-encryption [10].

- **Pattern Matching**
  There are many possible ways of performing pattern matching when analysing an executable. One type involves extracting readable strings from a binary file, which can range from plaintext, to file paths, URLs, function names, commands, encryption keys, among others. On Linux, the *strings* utility is a easy way to extract any strings from a file[15], which can then be manually analysed. Another way of performing pattern matching uses YARA Rules[16] or regular expressions to find specific malware patterns or signatures.

- **Disassembled Code Analysis**
  Uses methods based on disassembly to analyse specific features of the programs code. These may include uncommon usage or sequencing of opcodes (which can be analysed using techniques such as n-grams [11, 12, 13]), API or system calls, or through the programs control-flow graph. Similarities to known malicious software in any of these are strong indicators of the programs illegitimacy, and can greatly help in detection and classifying of the said program.

Dynamic Analysis

Dynamic analysis traces a program's behaviour through its execution in a controlled environment, allowing for real-time monitoring of the program's interactions with other files, registry, services, shells, devices and networks. Some applied dynamic analysis techniques are as follows:

- **Network Traffic Analysis**
  Possible with on-VM tracing through the use of tools like *tcpdump* or *tshark* in the machine

[15] https://linux.die.net/man/1/strings
[16] https://virustotal.github.io/yara/

the malware is running on, or with off-host capturing at the hypervisor level, or through dedicated network monitoring equipment. Captures communication data such as packet tracing (pcap) or NetFlow records.

- **Host-Level Dynamic Tracing**
  Low-level behaviour capturing, such as system calls (syscalls) or API calls, providing detailed behavioural insight of the malware. Possible through the use of utilities and syscalls such as *strace* and *ptrace*.

- **Snapshot-Based System State Analysis**
  Involves capturing a system snapshot before, after, and possibly during the malware's execution, to identify changes made to it and their sequence. This includes any changes to the file system, main memory, registry, and other possible changes. Although one can argue this belongs in the static analysis section, the aim of the analysis is still the programs behaviour during execution, making it fitting for this section.

Static and Dynamic Analysis Tools

Multiple tools have been developed to effectively perform both static and dynamic analysis, for various types of files and operating systems.

From a static analysis standpoint, tools such as Ghidra[17], IDA Pro[18] and Radare2[19] are among the most commonly used. All of these tools have disassembling capabilities, with some also supporting decompiling as well. Both of these techniques are important in the static analysis process, as they help acquiring information about a binary without executing it. Disassembling is the process of converting machine code into assembly language, thus making it human-readable, while decompilation is the process of translating machine or assembly code into a high-level programming language[20]. With this, it is possible to acquire plenty of information about a compiled binary, such as section and program headers, the programs symbol table, as well as its dependencies. Furthermore, it is also possible to capture function information and flow, as well as system calls, hardcoded strings and data references.

On the other dynamic analysis side, tools for this purpose work by observing a given sample behaviour during its execution in a isolated and protected VM (called sandbox), monitoring how it interacts with the system and environment its being executed on. Monitoring system calls and network activity is also common in dynamic analysis. Some examples of open-source dynamic analysis tools include:

- **Cuckoo 3** - Arguably one of the most well-known open-source malware analysis systems, Cuckoo is a common candidate for both static and dynamic analysis of malicious binaries. Although Cuckoo 2 had a wider range of supported systems (including Windows, macOS,

[17] National Security Agency. *GHIDRA Software Reverse Engineering Framework*, https://ghidra-sre.org/
[18] Hex-Rays. *IDA Pro Disassembler*, https://hex-rays.com/ida-pro/
[19] Radare2, https://rada.re/n/
[20] https://ctf101.org/reverse-engineering/what-are-disassemblers/

Linux, and Android), it stopped being officially supported and is incompatible with Python 3. The newest version, Cuckoo 3 has recently been revived under cert.ee[21], although with some limitations[22].

- **CAPEv2** - Derived from Cuckoo, CAPEv2 is a Python 3 based malware sandbox capable of analysing Windows and Linux files. Trusted by many, although less famous than Cuckoo itself, a version of it is even used by VirusTotal as one of the chosen sandboxes for its own dynamic analysis reports[23].
- **LiSa** - A relatively unknown tool compared to its competitors, LiSa is a dedicated analysis tool to analyse Linux binaries. It uses Radare2[24] for performing static analysis and Systemtap[25] kernel modules to perform dynamic analysis. It is also the base for ELF Digest[26], a service dedicated to analysis of Linux binary files, also used by VirusTotal for dynamic analysis reports[27].

2.2 Honeypots

Honeypots are resources that attempt to mimic a real computer system with the intention of being compromised by an attacker [14]. The attacker performs their techniques on the system, wrongly thinking it is a legitimate system, while these attacks are being recorded by defenders to be posteriorly analysed.

2.2.1 Honeypot classification

Honeypots range from simple services to complete operating systems and can usually be classified into one of three types (based on their interaction with the user):

- **Low-Interaction Honeypots**: These honeypots have the least amount of interaction for a connecting adversary, they have no operating system, usually consist of an open port in a network that accepts and logs connections.
- **Medium-Interaction Honeypots**: Being the type with the widest range and flavours, these honeypots usually attempt to emulate a system shell or specific service, but still without an operating system (they can, for example, receive malware samples from different kinds of botnets and command-line input to inspect the target system).
- **High-Interaction Honeypots**: These honeypots have the highest odds of trapping a human adversary, but also have the highest maintenance effort, without separation between the honeypot and the operating system. Malware infection as above and crypto miners are

[21] cert.ee, https://www.cert.ee/
[22] Cuckoo3, https://github.com/cert-ee/cuckoo3
[23] CAPEv2, https://github.com/kevoreilly/CAPEv2
[24] https://github.com/radareorg/radare2
[25] https://wiki.ubuntu.com/Kernel/Systemtap
[26] ELD Digest, https://elfdigest.com/
[27] LiSa (Linux Sandbox),https://github.com/danielpoliakov/lisa

good examples of common findings, usually turned into proxies for subsequent attacks if successfully infected.

Other lesser-known classification types can be used to further distinguish honeypots from each other, such as purpose (research or production), implementation (physical or virtual), activities (passive or active), running sides (server-side or client-side), operation (static or dynamic), and uniformity (homogeneous or heterogeneous) [15].

2.2.2 Classical Solutions and their features

Recent studies on honeypot software show a multitude of choices for different services and needs. Despite this, almost all of these solutions do not provide a way to analyse the collected malware samples. This is especially true for low- and medium-interaction honeypots, since these are mainly used for collection, and any analysis done is very specific and limited. **Nepenthes**[28], for example, is a low-interaction honeypot that emulates vulnerabilities commonly used by worms, with the objective of capturing these worms, with an easily extensible architecture, and its successor **Dionaea** has the option to add a shellcode detector, through the use of the *libemu* library, which detects shellcode by emulation [16]. Another example of this is **HoneySpider** [16], focused on attacks involving web browsers, which provides a simple integration with Cuckoo for file analysis. Although these honeypots provide a step in the right direction, they still lack in solving the problem related to malware that has environment-dependent behaviour. Another notable solution is **Cowrie**[29], a medium-interaction honeypot simulating vulnerable *SSH* and *Telnet* servers. It logs attacker activity, such as commands and malware uploads, but lacks native malware analysis, relying on external tools. **T-Pot** [30], one of the most well-known open-source honeypot platforms, integrates multiple honeypot related technologies into a single framework. Although it provides a great interface, it still does not provide a way to perform malware analysis natively, still not resolving the proposed problem.

While High-Interaction Honeypots have the highest chance of capturing non automated adversaries and more advanced threats, most still lack in the field of analysis of malware samples uploaded. **HIHAT**[31], a toolkit used to "transform arbitrary PHP applications into web-based high-interaction Honeypots", collects and saves malware sent to these honeypots, but does not do any analysis on them. Similarly, **PwnyPot** [16] is a High-Interaction Honeypot, which supports integration with Cuckoo 1, but it has since become largely abandoned, having only been tested for Windows 7 systems, and only for a limited set of applications within. **HoneyIoT** [17], a High-Interaction Honeypot made for IoT devices collects attack traces and malware sent to these devices, but only does analysis through VirusTotal, which runs into the problem of environment

[28] https://github.com/jrwren/nepenthes
[29] https://github.com/cowrie/cowrie
[30] https://github.com/telekom-security/tpotce
[31] https://hihat.sourceforge.net/

specific malware, as mentioned above. **HoneyICS** [18], another High-Interaction honeypot network, specialized in Industrial Control Systems, enriches collected malware data with the Alienvault OTX API using MITRE ATT&CK tactics and techniques[32], but just like HoneyIoT, it uses the VirusTotal API for analysis. Naik et al. [19] have presented an approach that combines fuzzy hashing with YARA Rules for malware sample triaging, and Ahn et al. [20] developed a machine learning based malware detection algorithm along with a method for visualizing detection data using the MITRE ATT&CK framework.

Recent works have used Large Language Models (LLMs) for acquiring and analysing attacker behaviour. Setianto et al. [21] have presented a system that leverages a pre-trained language model (GPT-2) to parse logs from a Cowrie SSH honeypot, while Raut et al. [22] intend to incorporate the ChatGPT API in a Honeypot system to encourage attackers to reveal their tactics and motives, helping on gathering behavioural data. Furthermore, Wang et al. [23] have proposed a honeypot architecture based on ChatGPT, which "*transforms the conventional "request-response" message interaction (based on terminal protocols) into a "question-answer" text interaction (based on ChatGPT API)*". While these LLM based works reveal new interesting avenues for honeypot and malware analysis they are still quite premature, resource-heavy, and hard to reproduce locally compared to classical ML based approaches.

Furthermore, while the mentioned approaches underline the growing interest in enhancing honeypots with ML, none of these approaches achieves the goal of augmenting a high-interaction honeypot with integrated ML-based real-time analysis.

2.3 Machine Learning Based Malware Analysis

Machine Learning (ML) has greatly improved the automation of pattern recognition and analysis, which has proved very useful for malware detection and classification. Traditional signature-based methods alone, while useful for detecting known threats, have become increasingly inadequate in the face of evolving threats, struggling to detect new or obfuscated malware variants [24].

On the other hand, ML can identify threats like these, through the use of large datasets with extracted features from files, network traffic, and system behaviour, with malicious and benign files. These datasets are used for model development, using supervised learning (where algorithms are trained using labelled datasets), unsupervised learning (where patterns are identified without the use of labels), semi-supervised learning, (which combines labelled and unlabelled data), or reinforcement learning (where systems learn by interacting with their environment and being reinforced when making the right decision) [25, 26], and using algorithms such as Neural Networks, Decision Trees, Naive Bayes, and Support Vector Machines (SVM) [27, 28].

[32]MITRE ATT&CK, https://attack.mitre.org/

Not only this, but the rise of machine learning has greatly enhanced the efficiency of network trace analysis, as conventional human-based detection methods have proved ineffective as network traffic gets larger and more complex evasion techniques are developed. Although implementation of machine learning methods is a great advancement, it also comes with its disadvantages, especially if it is not well implemented and trained. One of the main concerns in the early phases is related to the false positive and negative rates, which can cause great problems if exposed to real threats, outside of the training environment. Furthermore, attackers who are familiar with machine learning methods can attempt to manipulate the machine learning model into making wrong decisions, and re-training it into becoming ineffective. An example of this manipulation is poisoning, where an attacker adds training data that either leaves a backdoor on the model, or negatively impacts the model's performance. Even a 10% training data poisoning is already enough to significantly hinder a machine learning based classification models performance [29].

Recent Advancements in Machine Learning Based Malware Analysis

Recent proposals of systems for malware classification based in ML have become increasingly more heterogeneous, with novel techniques and methodologies being developed and employed for feature selection, extraction, and the classification phase itself. Liu, Wang et al. [30] have proposed a ML based malware analysis system, which extracts features using import functions, opcode n-gram and gray-scale imagery, using an S-NN (Shared Nearest Neighbour) clustering algorithm to classify malware into specific families. Xu, Ray et al. [31] have introduced a hardware-assisted malware detection framework using memory access pattern classification. Other works have begun to incorporate artificial intelligence in various stages of their pipelines for malware detection and classification [32].

Vinayakumar et al. [33, 34] evaluated classical ML algorithms and deep learning algorithms with multiple datasets, both public and private, with and without dataset bias, and proposed a new image processing technique with parameters for both types of algorithms, with the best performers being deep neural networks and convolutional neural networks. Sharmeen et al. [35, 34] have proposed a semi-supervised framework used to analyse ransomware samples from VirusShare[33] and VirusTotal, with both 15972 features from a JSON report provided by Cuckoo and a reduced set of 50 features through FastICA, where the proposed TensorFlow Deep Learning model performed the best against its classical counterparts (support vector machine, random forest and multi class classifier). Khan et al. [36, 34] proposed a "Digital DNA Sequencing Engine for Ransomware Detection Using Machine Learning", using a DNA sequencing engine to detect ransomware samples through their 26 most significant features . Carlin et al. [37, 34] tested the performance of 23 different models against features extracted using feature vectors and x86 opcodes from around 46000 samples. Out of all the models tested, the best performant after tuning was random committee, while random forest performed the best with retraining with increasingly larger datasets.

[33] VirusShare, https://virusshare.com/about

2.4 Virtual Machine Monitoring as Malware Analysis

Virtualized environments have become standardized in the context of dynamic malware analysis, mainly through the form of sandboxing available in existing tools (e.g.: Cuckoo3). Beyond this, there has been a recent emergence of more advanced monitoring techniques which provide deeper visibility of a malware's execution behaviour, both from outside and inside the guest machine (where the malware is running on). This section will present two approaches for this purpose, outlining their architectures and balancing their advantages and disadvantages in the context of malware analysis.

2.4.1 Virtual Machine Introspection (VMI)

Introduced by Garfinkel and Rosenblum [38], virtual machine introspection (VMI) is a technique used to monitor the internal state of virtual machines from the hypervisor (virtual machine monitor) level and provide insight into events within the virtual machine, as a means of detecting intrusions by monitoring the internal state of virtual machines (VMs) from the outside. These events may be related to processor registers, memory, disk, network, and any other hardware-level events, and the user must specify which of these events to collect, which are then sent to the host through an API. For Linux-based systems, the Xen Project has been a general solution years, due to its well-maintained support for VMI out-of-the-box in both x86 and ARM architectures, although recently Bitdefender presented an equivalent solution to Xen, the **KVMi** subsystem[34], through the use of Kernel-based Virtual Machine (**KVM**)[35] and Quick Emulator (**QEMU**)[36]. Though the KVMi subsystem still requires manual patching and compiling, it has nevertheless found itself in a recent rise in relation to its counterparts. Another important aspect to note about VMI-based tracing is that it works out-of-band, through event triggered hyper-breakpoints in the monitored VM, which adds a layer of abstraction and stealthiness, but this comes at a significant complexity and overhead cost.

2.4.2 extended Berkeley Packet Filter (eBPF)

The extended Berkeley Packet Filter (eBPF) is a modern in-kernel execution framework that enables the execution of safe, sandboxed user code in response to kernel events, all within the kernel itself, without requiring kernel module development or recompilation. Although lesser known for malware analysis than VMI, eBPF has seen a recent surge in usage for this purpose, becoming a more competitive alternative by each passing day. Both eBPF and VMI share a similar architecture flow in regards to virtual machine monitoring (Figure 2.1) but unlike VMI, eBPF exports events through a module inside of the monitored VM. While this removes the abstraction provided by VMI, and thus hinders stealth capabilities, it allows for much faster event exporting and communication with the monitoring controller.

[34] KVMi, https://kvm-vmi.github.io/kvm-vmi/master/kvmi.html
[35] KVM, https://linux-kvm.org/page/Main_Page
[36] QEMU, https://www.qemu.org/

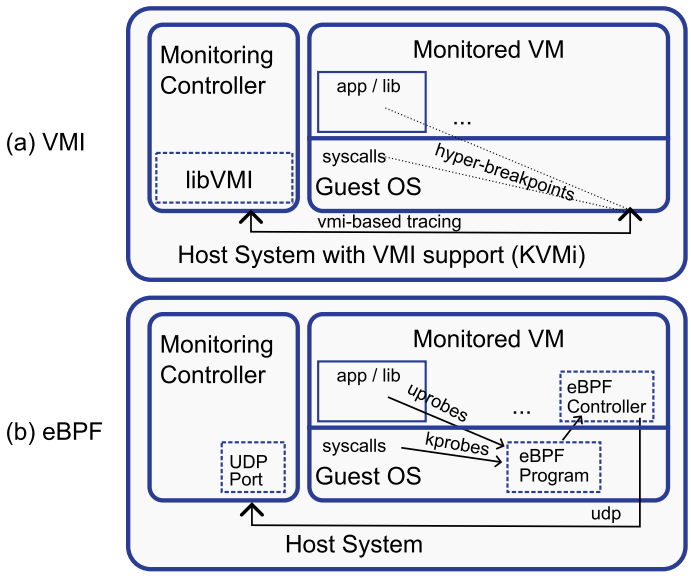

Figure 2.1: VMI and eBPF architecture comparison

The way eBPF works is by attaching itself to certain specific tracepoints, such as specific kernel-level tracepoints like system calls and process scheduling, or other hooks like kprobes and uprobes. The controlling user must specify what tracepoints to attach to and listen for events in his eBPF program, and then load this same program into the Linux kernel to start the tracing, which works by adding a call to the eBPF program in the selected tracepoints. It is also important to note that eBPF programs are verified before being loaded, to ensure they won't hang or crash the kernel. Then, as these tracepoints are reached by a process or user, the callback function in the tracepoint is called, which exports the event to the eBPF program, and subsequently to the controlling user in the user-space, which then processes these events as they see fit. For example, if we attach our eBPF program to tracepoints such as `sched_process_exec`, `sched_process_fork`, and `sched_process_exit`, the controlling user will receive, in their user level controller, events related to program execution, forking, and exiting in the target system. This makes eBPF ideal for a low-overhead solution to trace runtime behaviours of files in a system, thus making it a viable solution for malware analysis within a honeypot environment.

Chapter 3

Proposed Solution

The proposed solution introduces a novel architecture for a honeypot with embedded malware analysis and classification which relies solely on itself, without dependence from third party analysis services. When an attacker connects to the honeypot, uploads a malware file and runs it, the system automatically analyses this same file within the honeypot, capturing its behaviour and having its features extracted and sent to a previously trained machine learning classifier in order to accurately determine what type of malware family the file belongs to. The proposed solution aims to provide a way to record behaviours that may not be captured by current proposals due to the limitations of using third parties to perform the analysis. The present chapter presents the challenges encountered in the development of the proposed solution (Section 3.1), along with a general overview of the architecture and main components of our solution (Section 3.2 and 3.3).

3.1 Challenges

The main goal of our solution is to perform near real-time, accurate and stealthy sample tracing and classification in a live honeypot environment, without resorting to external services during the honeypots runtime. A task of this size comes with its share of challenges, which have to be addressed in order to provide the best results for our problems to obtain our proposed solution. These challenges are presented below.

3.1.1 Building a Representative and Balanced Training Dataset

One of the core components in our solution is a machine learning based classifier which, as explained in the introduction of this chapter, will be used to classify malware samples sent to our honeypot by their extracted features. It is thus crucial that this classifier is properly trained, and for that we need a suitable training dataset. This is not only difficult, but it requires much research and time to collect sufficient and complete data to ensure both balance and representation of the current threat landscape.

Firstly, we collected relevant malware samples from multiple different sources around the web. Although this seems like a simple task at first, as there is no shortage of malware “in the wild”, it

is important for the malware samples used to be as recent as possible, in order for our classifier to be as adapted to the currently existing threats as possibly achievable. After collecting the required samples, we gathered the features for each of these, along with finding a accurate way to label these samples based on the provided label by VirusTotal, which was a challenge within itself, as some normalization is required due to the sheer variety of existing labelling types. As we used VirusTotal to acquire these features and labels, we were further limited by the imposed limits of its API, which hindered the amount of samples we could process per day. Lastly, when all features and labels have been properly collected, we must ensure that our dataset is properly balanced, to avoid bias and over-fitting in our machine learning classifier. For this, some samples had to be removed or exchanged for different samples of other labels.

In conclusion, in order to achieve a properly trained machine learning based classifier for our honeypots malware, we collected samples from various websites and repositories, sent these samples to VirusTotal to acquire their respective features and label, processed these for optimization, and did some balancing to achieve a high-quality training dataset.

3.1.2 Achieving Consistency Between Offline Training Data and Live Honeypot Features

As explained in the previous section, our machine learning model is trained and tested using VirusTotal extracted features and labels from gathered malware samples. These features, when compared to the traces we gather from malware execution in our honeypot, can be considered quite "high-level". This is obviously a problem both for the samples classification, but also the general understanding of the malware's behaviour, and thus raises the need for real-time selection of malware related events from our honeypot, as well as some processing of these selected events to produce equivalent features to the ones used in model training.

For example, while VirusTotal presents a feature such as "Files with Modified Attributes", the honeypot provides traces for when the `chmod` system call is executed. To convert this tracing in to a VirusTotal-like feature, we made a python script that recognizes when this systemcall is called by a malware, and add the affected files name to the "Files with Modified Attributes", like it would be presented in VirusTotal.

3.1.3 Honeypot Design and Deployment

As one of the main components of our solution is our honeypot, its implementation was by itself a challenge to consider when coming up with our solution. Our honeypot will be high-interaction, as explained in Section 2.2, and thus requires special considerations when it comes to analysis and tracing stealthiness, in order to capture malware which detects analysis environments, and malware which only executes or uses its full range of capabilities in certain environments.

This honeypot should also be easy to deploy, in case of corruption or continuous nefarious usage. Previous works in similar systems, such as Sarracenia [39] provided basis for our deployment system, but these work with an old KVMi version which has since become outdated and

unusable in recent Linux systems. This made us look for other solutions for honeypot deployment and monitoring, which lead use to using eBPF as an alternative to KVM.

Therefore, our honeypot solution is based on running a common Linux virtual machine as a high-interaction honeypot on QEMU, with embedded eBPF based tracing, and event collection on the controller VM which runs the honeypot.

3.1.4 Ensuring Secure and Reliable Honeypot System Handling During Attacks

Although we allow the attacker to perform basically any action as they would be able to in a real Linux system, some limitations are still required to avoid corruption and misuse of the honeypot machine.

For this, measures such as standard Linux firewall rules have to be implemented, as well as a proper restoration / rollback system in case of corruption. This is achieved through the use of a script which rolls the honeypot back to a QEMU checkpoint with a clean version of the honeypot.

3.2 Proposal Solution Overview

With the previously presented objectives and challenges in mind, we propose MaCHoS (Malware Classifying Honeypot System), a honeypot system with embedded near real-time, machine-learning based malware classification capabilities. Figure 3.1 represents visually the process behind the gathering and parsing of samples for training and testing a previously selected machine learning model, which produces the classifier to be used in the honeypot. While the honeypot itself mimics commonly used systems, like services / operating systems to ensure attackers cannot easily distinguish them from legitimate targets, it captures detailed attack traces, through embedded static analysis and system call based dynamic analysis through eBPF instrumentation point attachment, which includes in-memory artefacts and multi-stage payloads. These captured malware samples and attack traces are analysed in real-time (while the attacker is still interacting with the system), through the aforementioned static and dynamic analysis techniques. Finally, a ML model trained offline using normalized VirusTotal [1] labels as ground truth, provides classifications of observed malware samples, through these captured attack traces.

The following sections will explain the three main solution components, which provide further details to support the system architecture diagram (Figure 3.1): Dataset (in Red), Machine Learning Based Classifier (in Yellow), and Honeypot and Controller (in Blue)

3.3 Main Solution Components

In order to better explain the flow of the architecture, the following subsections explain the role and functionality of its major components, also emphasising how these work together.

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

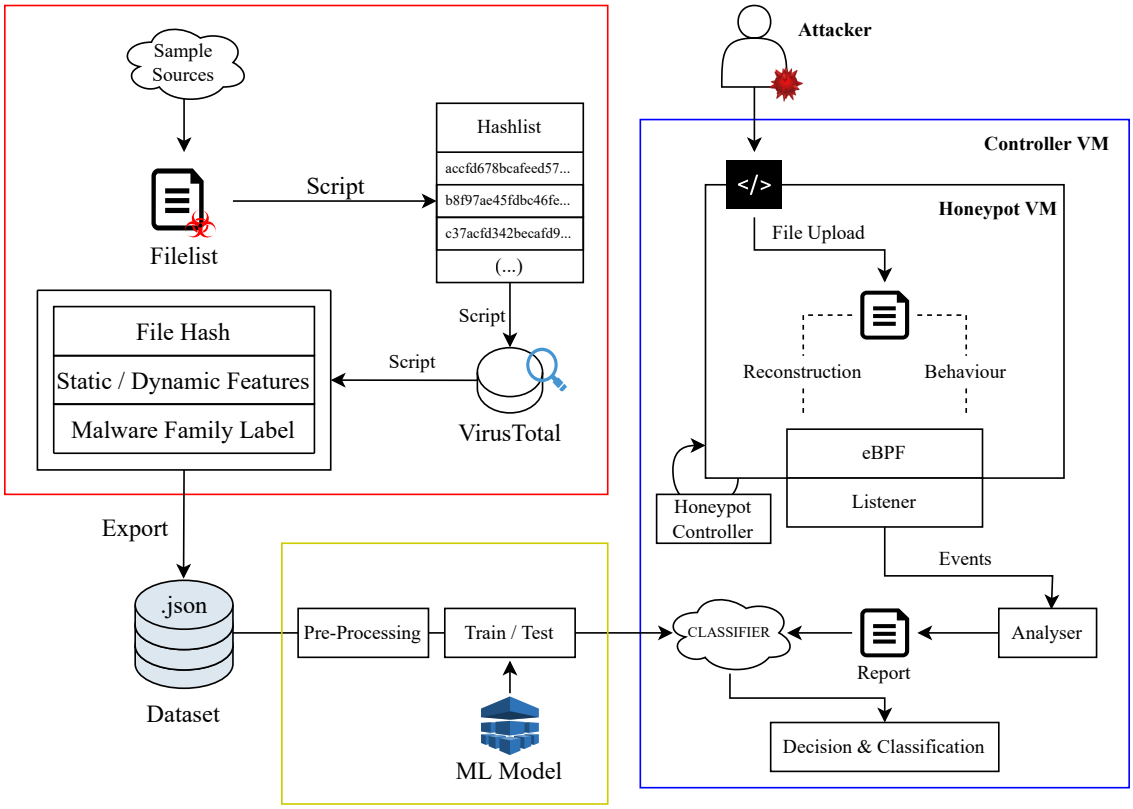

Figure 3.1: Model Training and System Architecture

3.3.1 Dataset

The left side of Figure 3.1 represents the acquisition and aggregation of sample hashes from existing malware in repositories from various different sources, combining both older and newer samples for balance and equitable representation. These hashes were then individually used to acquire the respective samples VirusTotal analysis result, through the use of VirusTotals API. We then used selected features from these same results to create our own multiclass labelled malware dataset, to use as training and testing for the machine learning based classifier. The process behind the sample aggregation and creation of the dataset is presented in Red in Figure 3.1.

3.3.2 Machine Learning Based Classifier

As mentioned when describing the dataset above, the machine learning based classifier is trained offline using VirusTotal analysis report features and their respective labels as ground truth, and its objective is to be used in the honeypot controller to classify malware samples sent to the honeypot in (near) real-time. This requires some pre-processing of the columns present in our dataset, along with some optimization to reduce the size of the resulting dataset. Although previous studies already suggested some models as being the best for malware classification tasks (as explained in the background section), we decided to conduct our own experiment using various models, training them over our dataset, to draw our own conclusions regarding what is the best model for

our use case. The process of using the dataset for pre-processing and the training / testing of the ML Model is presented in Yellow in Figure 3.1.

3.3.3 Honeypot and Controller

Arguably the most important component of the solution, the honeypot VM was designed with the intent of passing as a real system with weak credentials, to allow attackers to easily connect to and perform their techniques on it. A previously installed eBPF module is constantly listening to kernel-level events on this VM, and forwarding these same events to the controller VM through a listener. Two core functionalities of this system are the ability to reconstruct any file uploaded by the attacker on the controller side, and the live observation and recording of all behavioural traces of these same uploaded files.

The analyser present on the controller VM processes and aggregates any events generated by a uploaded file and its respective children into a feature report similar to the ones present in the dataset. When the file's execution is stopped or the honeypot VM is reset, this report is sent to the machine learning based classifier, which provides a classification for this file, and thus fulfilling its purpose in the systems architecture. The honeypot and controller system is presented in Blue in Figure 3.1.

Chapter 4

MaCHoS Design & Implementation

This chapter aims to strengthen and provide further explanation of the main components and other methodologies present in our solution, and how these were thought of and implemented. Section 4.1 starts by looking at the construction of the dataset used to train our ML model, looking at sources, feature selection, and the dataset development pipeline from start to finish. Next, Section 4.2 provides insight on the ML model trained with the previously mentioned dataset, going through the training and testing process and discussing the results it produced. Subsequently, Section 4.3 describes the honeypot, explaining how it works, from events tracing to analysis, explaining what behaviours are captured through this honeypot and how they are processed, and discussing the challenges faced along the way.

4.1 Dataset Composition

The following sections explain how we acquired our dataset used for training and testing our ML based classifier. We will approach the methodology behind acquiring the required features from the samples corresponding to the hashes in our malware hashlist (explained above and in Figure 3.1), along with the sources used to acquire these samples and any challenges encountered in these processes.

4.1.1 Dataset Sample Sources

Our dataset contains samples from diverse sources, which will be presented and explained below. All of these sources were carefully selected and researched to confirm their trustworthiness.

VirusShare Repositories

The first source for samples present in the dataset were publicly available collections provided by the VirusShare community[1]. Four collections were selected, three of them being solely comprised of ELFs, which ended up dictating the focus of the dataset, and thus the entire project. Another

[1] http://tracker.virusshare.com:6969/

extra file (VirusShare_Linux_20160715) was eventually added, which had a few interesting samples that helped filling gaps in sample diversity which the dataset suffered from at the time. The selected collections are as follows:

- **VirusShare_ELF_20140617.zip** (2,778 samples)
- **VirusShare_ELF_20190212.zip** (10,426 samples)
- **VirusShare_ELF_20200405.zip** (43,553 samples)
- **VirusShare_Linux_20160715.zip** (9,482 samples)

Although all these datasets combined add up to 66,239 samples, most of these samples ended up not being used for the final version of the dataset. Although this was partly due to the rate limits imposed by the VirusTotal API (which only allowed 500 daily requests)[2], a lot of these results ended up not being used due to what ended up being the biggest doubt / challenge faced when building this dataset.

The Mirai / Gafgyt Problem

Around late 2014, a vulnerability in the Bash command shell was found and publicised (named *ShellShock* [CVE-2014-6271][3])[4], which started a large mass of IoT device targeting botnets, one of them being BASHLITE (also known as *Gafgyt*)[5]. This, along with the source code leaks of a few of the original botnets that exploited this vulnerability led to the emergence of dozens of more variants, with an estimate of 1 million infected devices by 2016. One of these variants, named *Mirai*[6], specifically made for targeting video-game servers and its providers, along with its original BASHLITE counterpart, was extremely prevalent online at the same time as the VirusShare datasets were compiled, causing for a very large amount of samples to belong to one of these botnets. This is especially true for the 2020 dataset (which is coincidentally the largest out of the four), where the large majority of hashes belong to malware in the *Mirai* family. After a certain time of collecting VirusTotal analysis results, and in an effort to reach a balanced dataset, samples belonging to *Mirai* and later *Gafgyt* stopped being collected. In hindsight, we believe this ended up being a mistake, since a dataset made for malware classification should be representative of the current malware landscape, even if it is dominated by a specific type or family of malware.

MalwareBazaar Queries

The second source used to acquire samples for our dataset was *MalwareBazaar*, a malware exchange forum from abuse.ch[7]. Acquiring the hashes needed for the dataset was done through a script for *MalwareBazaars* API, to search for more samples of different categories. Multiple

[2] https://docs.virustotal.com/reference/public-vs-premium-api
[3] https://nvd.nist.gov/vuln/detail/CVE-2014-6271
[4] https://www.cyber.nj.gov/threat-landscape/malware/botnets/bashlite
[5] https://github.com/ifding/iot-malware
[6] https://github.com/jgamblin/Mirai-Source-Code
[7] https://bazaar.abuse.ch/browse/

queries were made, of particular tags and signatures, in order to achieve balance between distinct malware families.

Threat Insights Portal

The third and final source for samples used in the dataset were kindly provided by the administrative team at Threat Insights Portal [8], with the intent of including samples from recent threats, and to further expand the amount of unique threats in the dataset.

4.1.2 Feature Selection

The following step in building our malware dataset was comprised of selecting features from VirusTotal analysis reports. This is important since, as explained above, the machine learning model uses VirusTotal analysis results as ground truth for its training and testing. Therefore, a selection of important features from a complete VirusTotal report is necessary to achieve the best possible results and efficiency when building the dataset and, later, the model itself. These reports are comprised of two different categories, each present in their own tab in the web GUI, and each with their own unique information. The *Details* tab provides features related to static analysis, while the *Behaviour* tab provides dynamic analysis related features.

Table 4.1: VirusTotal Details tab Example (Static Features)

Metadata		**Shared Libraries**	
Build ID	bfc741433da42051ea6eaa8e946ab79463608ed2	libpthread.so.0	
Interpreter	/lib64/ld-linux-x86-64.so.2	libdl.so.2	
Header		libstdc++.so.6	
Class	ELF64	libm.so.6	
Data	2's complement, little endian	libgcc_s.so.1	
Header Version	1 (current)	libc.so.6	
OS ABI	UNIX - System V	**Imported Symbols**	
Object File Type	EXEC (Executable file)	tcsetattr	FUNC
Required Architecture	Advanced Micro Devices X86-64	mprotect	FUNC
Object File Version	0x1	_ZNSsaSEPKc	FUNC
Program Headers	8	fileno	FUNC
Section Headers	32	_ZNSsC1Ev	FUNC
Contained Segments	**Contained Sections**	...	
PHDR	undefined	**Exported Symbols**	
INTERP	.interp	X509_ATTRIBUTE_it	OBJECT
LOAD	.note.ABI-tag	ASN1_IA5STRING_it	OBJECT
LOAD	.note.gnu.build-id	EXTENDED_KEY_USAGE_it	OBJECT
DYNAMIC	.gnu.hash	Camellia_decrypt	FUNC
NOTE	...	_shadow_DES_check_key	OBJECT
GNU_EH_FRAME		...	
GNU_STACK			

To better understand what these features are and how they are distributed among their distinctive categories, Tables 4.1 and 4.2 represent the VirusTotal analysis results of a malware sample (SHA256: b57e5f0c857e807a03770feb4d3aa254d2c4c8c8d9e08687796be30e2093286c)[9], labelled by VirusTotal as *ransomware.royal/royalransom*, showing the chosen features as they ap-

[8] https://tip.neiki.dev/

[9] https://www.virustotal.com/gui/file/b57e5f0c857e807a03770feb4d3aa254d2c4c8c8d9e08687796be30e2093286c

Table 4.2: VirusTotal Behaviour tab Example (Dynamic Features)

Network Communication	**MITRE ATT&CK**	**Process and service actions**
IP Traffic	**Tactics and Techniques**	**Shell Commands**
TCP 185.125.188.59:443	Execution TA0002 (T1059, T1064)	/bin/mount /var/lib/snapd/snaps/lxd_24061.snap...
TCP 185.125.190.26:443	Persistence TA0003 (T1543, T1543.002)	/bin/mount /var/lib/snapd/snaps/powershell_226.snap...
TCP 185.125.190.27:443	Privilege Escalation TA0004 (T1543, T1543.002)	/bin/sh -c "/usr/sbin/ethtool -i $1 —/usr/bin/sed -n...
TCP 185.125.188.55:443	Defense Evasion TA0005 (T1036, T1064, T1070, T1979.004)	/lib/udev/bcache-export-cached /dev/loop0
TCP 91.189.91.42:443	Discovery TA0007 (T1082, T1083, T1518, T1518.001)	(...)
TCP 54.217.10.153:443	Command and Control TA0011 (T1071, T1090, T1573)	
TCP 91.189.91.43:443		

File System Actions	
Files Opened	**Files Deleted**
/etc/ld.so.cache	/etc/udev/rules.d/70-snap.core18.rules
/lib64/libc.so.6	/etc/udev/rules.d/70-snap.core20.rules
/lib64/libdl.so.2	/etc/udev/rules.d/70-snap.lxd.rules
/lib64/libgcc_s.so.1	/etc/udev/rules.d/70-snap.powershell.rules
(...)	(...)
Files Written	**Files With Modified Attributes**
/etc/udev/rules.d/70-snap.snapd.rules.PpxBv5lyYvwj~	/run/mount/utab.1LjDBw
/etc/udev/rules.d/70-snap.snapd.rules.bt4c1kjrHTk5~	/run/mount/utab.jvYjmZ
/etc/udev/rules.d/70-snap.snapd.rules.vKNwhtvDGcbN~	/run/mount/utab.lock
/etc/udev/rules.d/70-snap.snapd.rules.xxvXhMfYsQgG~	/run/mount/utab.riX3oe

pear in the analysis report. Out of all the features shown in both tables, the ones that were selected for use in our dataset entries have been colour coded appropriately according to their type (Orange for Static Analysis features, Purple for Dynamic Analysis features, Green for Network Analysis features). Other features present in the exported analysis report but not seen in the web GUI are: the files size and its entrypoint address.

Table 4.3 presents the selected features, which have been combined with the files VirusTotal classification verdict to build a multi-class labelled dataset, for training and testing of the machine learning based malware classifier.

Table 4.3: List of features that compose the malware dataset

Category	**Feature**
Static (Details)	Selected Header Fields
	Import List
	Export List
	Section List
	Shared Libraries
Dynamic (Behaviour)	Files Opened
	Files Deleted
	Files Written
	Files with Changed Attributes
	Shell Commands
Network	IP Traffic
Class	Malware Class Label

Why *these* features?

One question that may arise while considering the selected features for our dataset is "Why select these specific features?". To answer this, we must first look at what samples we plan on acquiring.

From early in our solutions development, we decided that we would limit ourselves to solely

gathering Executable and Linkable Format (ELF) samples. While this may be detrimental for the honeypots capabilities, as some of the files uploaded by attackers may not be able to be classified, it was a necessary step, not only due to the availability in existing malware repositories, but also the standardization of possible features in our dataset. Future work on this solution would definitely consider adding other types of files to our dataset and model, but this would require a feature overhaul and acquiring of many more samples to use in our model training dataset.

MITRE ATT&CK Tactics and Techniques

Another decision faced while selecting what features to use for our dataset was the usage of MITRE ATT&CK Tactics and Techniques. While this feature would be of great help in more accurately presenting what behavioural techniques the analysed malware samples would employ while in the target system, acquiring a reliable way of extracting these solely from an observation of the malwares execution without relying on third party sandbox software ended up being abandoned and left up for consideration in future work.

4.1.3 Hashlist to Dataset Pipeline

For building our dataset with previously selected features from VirusTotal analysis results, a few scripts were written, each with different but complementing objectives. These scripts are in charge of reading from a text file with hashes from files in the selected malware repositories, and doing preliminary checking for each of these hashes to see if their (static) analysis report has not been previously fetched and is complete. If these conditions are met, the hashes corresponding behavioural (dynamic) analysis report is also fetched from VirusTotal, and both are written locally to two distinct files (in different folders), each with the hash as a name, for correspondence.

Algorithm 1 shows the first of these scripts, which checks if a report already exists for each hash in our input file. If the previous condition is passed successfully, the corresponding “Details” (static) report is fetched from VirusTotal. If this fetch is also successful, the report is sent to the “analyse” method, explained below (the “analyse” method is included in the code for Algorithm 1, but they have been separated for readability) .

Algorithm 2 is responsible for the aforementioned analysis method, and receives the static analysis “Details” report fetched from VirusTotal, and performs further checks. We need to have this preliminary fetch to check if the corresponding file is fit to be used in the dataset, but both this report and the corresponding “Behaviour” (dynamic) report will be fetched once again. It starts by verifying if the file is an ELF, and then checks if the file has been classified, which interestingly is not always true in VirusTotal. If these conditions are passed, then two additional scripts are run, taking the files hash as input, which re-fetch and write the files “Details” and “Behaviour” reports to local files, respectively. Lastly, when all of the hashes have been processed and analysed, a small mismatching script is run to check if all file analysis reports have a corresponding behavioural report, with any mismatches being removed, and the suggested labels in the generated reports are normalised. This last point is crucial in ensuring the datasets quality, and will be ex-

Algorithm 1 Script that verifies if a file exists and is valid for analysis

Input Hash File
Output (Calls 'analyse' method)

for $hash \in hashfile$ **do**
  $present \leftarrow 0$
  if $hash \in static\_report\_folder$ **then**
    $present \leftarrow 1$ ▷ File has already been analysed
    $continue$
  end if

  if $present == 0$ **then**
    $static\_analysis\_report \leftarrow getStaticReport(hash)$ ▷ Gets the files static report through the VirusTotal API
    if $static\_analysis\_report.status\_code == 200$ **then**
      $analyse(static\_analysis\_report)$
    else if $static\_analysis\_report.status\_code == 404$ **then**
      $continue$ ▷ Analysis does not exist
    else if $static\_analysis\_report.status\_code == 429$ **then**
      $exit()$ ▷ API Rate exceeded
    end if
  end if
end for

$mismatchChecking()$
$normalizeLabels()$

plained in detail later.

After this process concluded, we end up with two folders with every files respective "Details" and "Behaviour" report, ready to be compiled into one singular dataset. The previously mentioned mismatch checking script ensures that there is a corresponding "Behaviour" (dynamic) and "Details" (static) report for each file, ensuring a complete analysis.

After the mismatched have been removed and the labels have been normalised, our final script is in charge of iterating through both report folders, extracting the necessary features from each corresponding pair (static and dynamic) of reports, and assembling a final JSON file with these features for each malware sample. A few of these features are subject to some preprocessing prior to their addition to the dataset:

- Import, Export, Section, and Shared Library lists are sanitized to ensure there are no duplicates within them.
- Files Opened, Written, Deleted, and Modified Attribute list items are reduced to their main subfolder (e.g.: "/etc/passwd" ⇒ "/etc/"), in order to reduce size
- IP Traffic items are reduced to their IP only (e.g.: "127.0.0.1:22" ⇒ "127.0.0.1")
- The Data item (corresponding to a file's endianness) is reduced to only show if the file is big or little endian (e.g.: "2's complement, little endian" ⇒ "little")

Algorithm 2 VirusTotal Report "analyse" Method

Input Item Static Analysis Report (as JSON)
Output (Writes results to local files [In other scripts])

$hash \leftarrow static\_analysis\_report[sha256\_hash]$
if $static\_analysis\_report[item\_type] == ELF$ **then**
    if $static\_analysis\_report[file\_classification] != (null)$ **then** ▷ Check if file has been labelled
        None
    else
        $writeStaticReportToFile(hash)$ ▷ Re-Fetches static report
        $writeDynamicReportToFile(hash)$ ▷ Fetches dynamic report
    end if
end if

Suggested Label Normalization

One of the challenges encountered when building our dataset was related to the way we process malware labelling from VirusTotal. With this being a 'labelled' dataset, the VirusTotal analysis verdict for the chosen samples is also extracted along with the selected features. This verdict label is formatted as (e.g. trojan.gafgyt/ddos):

[Type].[Family]/[SubFamily (or) AlternativeFamilyName]

This sort of labelling allows for an excessive amount of possible combinations, and thus a normalization process was required to ensure a good balance between number of labels and their respective accuracy to their original label. Our first approach to tackle this problem was through limiting VirusTotals verdict label to only the first appearing malware family (e.g.: trojan.gafgyt/ddos $\Rightarrow$ gafgyt). This solution ended up being discarded, as it caused cases where files with the same features ended up with different labels (trojan.gafgyt/ddos and trojan.ddos/gafgyt). In order to account for these issues, ClarAVy [40] was used to more accurately summarize the detection labels from all sources that VirusTotal provides (from the industry and the community), along with its features, and reach a more accurate label. This approach ended up resolving our previously mentioned issues, being used to provide a simpler yet accurate label to all files present in our dataset.

4.1.4 Dataset Versions and Representation

As the dataset building process spanned through multiple months of development, multiple versions of this dataset were made, and this subsection explains changes made and process behind these changes.

Table 4.4 shows the multiple iterations of the dataset along its development, by their composing labels and respective sample size. A few important aspects to note are the noticeable labelling normalization as explained in Section 4.1.3, which caused the large label scheme change from iterations 2 to 4. Another important aspect to consider is that the dataset only includes sample

Table 4.4: Dataset iterations and respective labels / values

Dataset	Iteration 1		Iteration 2		Iteration 3		Iteration 4	
Labels	virus	508	virus	560	mirai	612	gafgyt	1931
	trojan.mirai	303	trojan.mirai	372	gafgyt	1875	SINGLETON	940
	trojan.gafgyt	755	trojan.gafgyt	501	rootkit	341	mirai	777
	trojan.rootkit	228	trojan.rootkit	435	tsunami	453	prometei	565
	trojan.tsunami	217	trojan.tsunami	403	generica	407	tsunami	465
	trojan.generica	303	trojan.generica	389	prometei	565	berbew	285
	hacktool	98	hacktool	148	kaiji	295	kaiji	229
	worm	118	worm	132	berbew	221	dofloo	194
			trojan.prometei	501	dofloo	185	xmrig	191
			trojan.kaiji	273	flooder	158	mayday	141
			trojan.berbew	263	xorddos	117	setag	138
			miner	219	(other	3343)	local	135
			trojan.dofloo	183			ares	109
			trojan.flooder	128			multiverze	100
Total		2530		4507		8572		6342

labels which included 100 or more samples available from our sources. This rule was implemented in an effort to reduce the number of labels in our dataset while still including the most prominent malware families, since there were quite a lot of labels with insufficient samples, some even have just one single sample. In iteration 3, all other samples belonging to the remaining malware families were bundled in an "other" label, although we did not consider this as the most optimal solution. This issue ended up being fixed for Iteration 4 using ClarAVy, by the inclusion of the "SINGLETON" label, which includes samples that do not seem to fit in any of the available families.

4.2 Machine Learning Based Classifier

With the finalisation of the process of compiling our dataset, we moved on to the processing, training and testing of our machine learning based classifier. A preliminary study of previous work on machine learning based threat classification was conducted, with Random Forest models being the clear favourite for classification based on a multi-class labelled dataset [41, 42]. Despite this, a study was conducted using four different machine learning models: Random Forest, Gradient Boost, Naïve-Bayes, and Support Vector Classifier (SVC), over the finalised version of our dataset. The dataset was imported and preprocessed with StandardScaler being applied to number based columns, OneHotEncoding for text based columns, and MultiLabelBinarizer to list based columns (Import, Export, Section Lists, Shared Libraries, all file related columns, Shell Commands and IP Traffic). After this, all models were fit to a training set (80%) and submit to testing (20%) of the processed dataset, with the aim of extracting the top 100 most relevant features present in list based columns. This is needed because preprocessing the full dataset leads to the generation of 25587 columns, which severely affects performance and accuracy. Extracting the most relevant features reduces the dataset to around 900 columns, which is much more manageable. As only Random Forest, Gradient Boost and SVC

provided satisfactory results on this first part of testing, these were the chosen candidates to proceed in the testing, each with the following parameters (scikit-learn v1.7.0): RandomForestClassifier with a balanced class weight (`class_weight='balanced'`), with a 25 maximum depth over 600 estimators (`max_depth=25, n_estimators=600`), and 0.75 maximum features (`max_features=0.75`), GradientBoostingClassifier with a 25 maximum depth over 600 estimators (`max_depth=25, n_estimators=600`), and SVC with rbf kernel (`kernel='rbf'`)

With this reduced dataset, the models were then put through a 5-fold cross-validation process, to pick which is the best performing model. Table 4.5 shows the result of this cross-validation over the three remaining models.

Table 4.5: RandomForest vs GradientBoost performance (reduced feature size)

Metric / Model	RandomForest	GradientBoost	SVClassifier
Fit time	34.39s	295.13s	2.67s
Accuracy	0.932	0.936	0.901
Precision	0.932	0.934	0.895
Recall	0.932	0.936	0.901
F1 - Score	0.933	0.935	0.903

4.2.1 Model Performance Results

From Table 4.5 we can conclude that the best performing model is Gradient Boost, but this model has extremely long fitting times, which is detrimental for future implementations which should take into account the retraining of the model with samples acquired from the honeypot. On the other hand, the fastest model was SVC, but this model also ended up being the worst performer out of the three. With this, the chosen model for our project ended up being Random Forest, as it provides the best results within an acceptable fitting time which makes it viable for future implementation and extensions. Additionally, Figure 4.1 presents the 20 most relevant features used for classification of samples in our chosen machine learning based classifier, while Figure 4.2 shows the full classification report for this same model.

4.2.2 Classification Report Considerations

Although our chosen model presents quite positive global metrics, there are still a few considerations to keep in mind in regards to specific malware class performance. The first of these classes to consider is multiverze, which presents the worst performance results out of all the different classes. While one could simply attribute this to a lack of samples in the datasets testing split, this alone cannot explain the mediocrity of the score, since there are other classes, such as mayday and ares, that have a similar number of support samples but yielded much better results. Microsoft Security Intelligence describes this malware family as "a generic detection of various ELF-based malware families, such as CoinMiner, Kaiji, Mirai, and Ngioweb" [10], and this description allows

[10] https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Linux/Multiverze

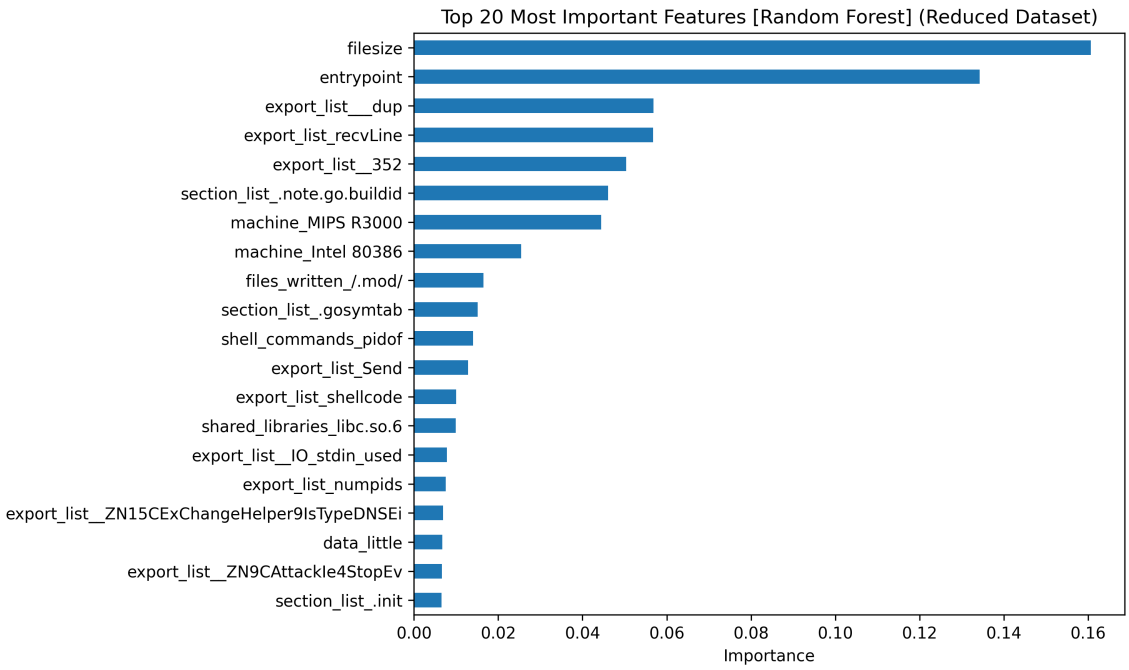

Figure 4.1: Top 20 Most Relevant Features (RandomForest) [Reduced Feature Dataset]

us to get further insight on why this specific family is struggling. Earlier iterations of the dataset were used to generate earlier versions of our model, and these had classes which suffered from similar issues (such as the previously mentioned "other" class and the "*generica*" class). While ideally generic classes should be broken down into smaller, more specific classes, it is important to consider these generalizations regardless, since they may account for other types of malware which can be overlooked if not used.

4.3 MaCHoS Honeypot and Analysis Solution

To wrap up the implementation process of MaCHoS, this section describes the process behind the implementation and challenges faced when designing, running and robustness of our honeypot solution.

4.3.1 Architecture

As explained in the previous Section 3.3.3, the honeypot uses eBPF, and works with two main components: the tracers working inside the honeypots VM, and an external back-end for monitoring and analysis. These tracers are in a eBPF component in the honeypot VM, while a loader deals with loading this same eBPF component in the VMs kernel and forwarding events to the external monitor. The monitor receives raw events, maintains internal state, reconstructs files, tracks process hierarchies and sessions, and emits structured JSON events via UDP broadcast. Figure 4.3.1 shows the interaction that happens between the honeypot VM and the monitor/controller VM (as

Malware Class	Precision	Recall	F1-Score	Support
SINGLETON	0.86	0.84	0.85	189
ares	0.9	0.95	0.92	19
berbew	0.98	0.97	0.97	59
dofloo	0.97	0.9	0.94	41
gafgyt	0.99	0.98	0.98	375
kaiji	0.95	0.95	0.95	40
local	0.69	0.67	0.68	27
mayday	0.88	0.92	0.9	24
mirai	0.89	0.96	0.92	158
multiverze	0.32	0.32	0.32	19
prometei	0.99	1	1	110
setag	0.91	0.97	0.94	30
tsunami	0.97	0.89	0.93	109
xmrig	0.89	0.97	0.93	40
xorddos	0.96	1	0.98	27
Global Metrics				
accuracy			0.93	1267
macro avg	0.88	0.88	0.88	1267
weighted avg	0.93	0.93	0.93	1267

Figure 4.2: Classification Report (RandomForest) [Reduced Feature Dataset]

shown on the left side of Figure 3.1, which handles the events received from the honeypot and resets the honeypot VM to a earlier "safe" snapshot if required (due to excessive abusive usage or corruption).

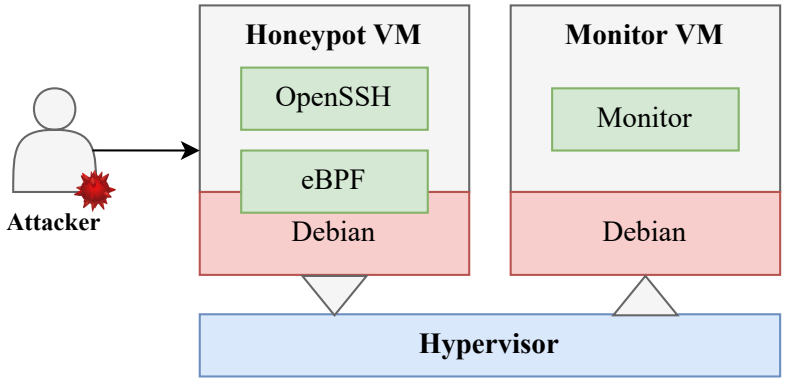

Figure 4.3: eBPF deployment architecture

These broadcast events are received by an analysis script (Algorithm 3), that tracks files sent by attackers and their respective events, structuring these runtime events and static analysis made on the file itself. Files uploaded are tainted with a *TRACKED* flag upon finishing their upload, and their behaviour starts being saved upon execution until exit or a VM restart. Any child process behaviours are also saved. Furthermore, upon execution, a unique identifier for the file is saved along with the behaviour, with its corresponding value being the folder of the reconstructed file on the controller machine, used to add static analysis.

Algorithm 3 Analysis script for processing received UDP events from the honeypot

Input JSON Events from eBPF
Output Static and Dynamic File Features

$bindSocketToMulticast()$
$tracing \leftarrow \{\}$

while $True$ **do**
    $event \leftarrow receiveAndLoad(socketData)$
    $session\_id = event['session']$ ▷ Session Id
    $pid = event['pid']$ ▷ Process Id
    $type = event['type']$ ▷ Event type (exec, write, open, etc.)
    $parent = event['subsession']$ ▷ Parent Process Id (if it exists else null)
    $kind = event['kind']$ ▷ Checks if file has TRACKED flag (if uploaded by attacker)

    if $type == exec \;\&\&\; kind == TRACKED$ **then** ▷ Malware Execution
        $controllerPath \leftarrow getControllerFilePath()$
        $tracing[session\_id][pid] \leftarrow generateTracerSkel(controllerPath)$ ▷ Generates skeleton of report and performs static analysis on the reconstructed file (in the controller)
    else if $parent\ != \ null$ **then** ▷ Malware Activity
        $updateTracing[session\_id][parent][type](eventArgs)$
    end if
end while

4.3.2 Captured Behaviours

For accurately capturing the necessary features to classify malware samples sent to our honeypot, we trace kernel functions and system calls such as `open`, `write`, `close`, `chmod`, and `unlink` to detect file uploads and modifications. With these functions alone, we can capture permission changes, file / record deletions, as well as reconstruct the files themselves from their write streams. Furthermore, our eBPF instrumentation also includes network related tracing, which allows for linking this network activity to processes while allowing filtering of benign traffic.

Table 4.6 lists the tracing points used in our honeypot to track malware behaviours. These trace points allow for a behavioural report composed of all features necessary for the machine learning classifier to accurately classify the malware sample sent to the honeypot. Along with this, static analysis is also performed on the sample, using the python package elftools[11], since it is reconstructed on the controller upon finishing its uploading process (triggered by a file_close event). When an exit event happens for the malware sample, the analysis is given as finalised and its product is sent to the machine learning based classifier for classification of the sample.

4.3.3 Issues with Implementation

Although our honeypot solution does fulfil all tracing requirements to gather the features needed by the machine learning model, there were still a few issues that required changes in implementation and slight compromises to achieve the best possible result.

[11] https://github.com/eliben/pyelftools

Table 4.6: File related eBPF tracing points in honeypot implementation

Type	Trace Point	Purpose
syscall	`sys_enter_connect`	Trace outgoing TCP connections
fexit	`do_sys_openat2`	Trace file openings
fentry	`filp_close`	Trace file closures and writes; triggers file reconstruction and analysis
fentry	`chmod_common`	Detect file permission changes
LSM hook	`path_unlink`	Trace file deletions
LSM hook	`bprm_committed_creds`	Trace execution of new binaries (covers all `execv*` variants)
tracepoint	`sched_process_fork`	Track process creation and parent-child relationships
syscall	`sys_enter_write`	Monitor file modifications and session activity
tracepoint	`sched_process_exit`	Detect process termination and clean up state

- Since directly tracing all `execve` system calls requires iterating over a variably sized argument list, we instead opted to trace the `bprm_committed_creds` LSM hook. This provided several advantages, such as only needing to trace a single point rather than multiple system calls, enabling correlation with previously uploaded files (by accessing the inode of the executed binary), and making it easier to obtain the arguments for the executed command.

- The `bpf_d_path` helper function, which resolves full absolute path names, is only available in certain contexts. We were able to use it successfully when tracing program execution and file closures, but not in other cases such as `unlink` or `chmod`, where access to absolute file paths would also have been useful.

- Capturing data from `write` system calls in order to reconstruct file contents posed another challenge. eBPF ring buffer allocations must use a fixed size at compile time, whereas `write` operations produce variably sized payloads. As a workaround, we split the data into fixed-size chunks, each fitting within the buffer size limit. A variable number of such chunks can be emitted per write, as long as an upper bound is enforced. In practice, this method worked reliably for uploaded files via SFTP, downloads using tools like `wget` or `curl`, and streaming uploads over SSH (e.g., using the session piped into `tar`).

Having detailed the design and implementation of MaCHoS and its main components, the following chapter will now turn to its evaluation.

Chapter 5

Evaluation

Chapters 3 and 4 presented the architecture design and implementation of MaCHoS, tackling the proposed research questions RQ1, RQ2, and RQ3, as explained in the introduction chapter. In the present chapter though, we will evaluate this same solution, ultimately addressing the remaining RQ4 and RQ5. In order to evaluate our solution, we will compare it with a baseline honeypot and analysis system, which is representative of commonly used approaches. This evaluation serves to provide answers to research questions RQ4 and RQ5:

> **RQ4:** *Can a machine-learning based malware classification system integrated in a high-interaction honeypot provide more effective and insightful malware analysis than classical approaches that separate collection and subsequent analysis with third-party services?*
>
> **RQ5:** *Can system call (syscall) tracing based dynamic analysis provide additional insight on malware / attacker behaviour and help with its respective classification?*

and thus, all evaluation techniques must stem from these same questions. In order to facilitate this, we will divide these research questions into multiple, easier to tackle evaluation criteria, and we will test our solution against each of these individually. The criteria are as follows:

EC1: What will be the baseline honeypot and analysis (representative of classical approaches) used to compare against our MaCHoS solution?

EC2: To what extent does the proposed honeypot system attract attackers more effectively than existing solutions?

EC3: How accurate is the integrated malware classification component compared to classical third-party classification services?

EC4: In a direct case study with identical malware samples, what differences emerge between the proposed system and classical third-party-based analysis?

EC5: Can integration of ML-based classification into a high-interaction honeypot outperform classical approaches that separate collection and third-party analysis when faced with complex attacks?

EC6: What are the differences between eBPF based honeypot tracing and its (more traditional and commonly used) VMI based counterpart?

Each of the following sections will tackle each corresponding evaluation criteria.

5.1 Baseline (Classical) Honeypot and Analysis Solution

In order to properly assess the previously described evaluation criteria, a honeypot and analysis system representative of classical approaches was composed, to serve as a baseline for comparison against our solution, MaCHoS. This section will explain our process behind setting up this classical system and its components.

5.1.1 Static/Dynamic Analysis Tools Study

To evaluate our approach against classical honeypot and analysis methodologies, a study for currently available, open-source static / dynamic analysis tools was conducted. These tools have been previously explained in the Section 2.1.1, and this section provides insight on our testing performed on these tools. All tools were installed and configured in a host machine running Ubuntu 22.04, with 8 CPU cores and 16GiB Memory, with varying levels of success which will be discussed below.

- **Cuckoo 3:** As mentioned in Section 2.1.1, Cuckoo has only recently been ported to Python 3, under the name Cuckoo 3. Although promising, this newest unfortunately only supports Windows guest VMs, making it out of scope for this work, since the focus is specifically Linux malware.

- **CAPEv2:** Although CAPEv2 has recently implemented the use of Linux as a guest VM, and thus enabling Linux malware analysis, this tool ended up not being the chosen option for our baseline due to its complexity of installing and running a Linux guest VM, mostly due to its recency and complexity, along with an insufficient exaplanation of this process in the tools provided documentation.

- **LiSa:** The final (and eventually chosen) option for our baselines analysis tool was LiSa, which was developed with the intention of being used specifically to analyse Linux binaries. While it was by far the easiest tool to set up and use, it does have its limitations in terms of dynamic analysis, since it only tracks IP Traffic and files opened by the analysed sample.

5.1.2 Baseline Honeypot Architecture

Figure 5.1 illustrates a representation of the setup of this classical honeypot. For our representation of a classical honeypot and analysis component, an open machine was set up with an instance of Cowrie, a medium interaction SSH honeypot[1] for attack trace and sample capturing, and another separate machine was set up with LiSa, to perform analysis on these same samples. Additionally, a web-hosted visualization dashboard has been setup for easy access to general metrics. Attackers who connect to the honeypot do so through SSH in port 22 or 2222, and all their connection events and traces (commands executed, files uploaded, IP address and credentials used) are logged. These same logs are also parsed for display in the web dashboard.

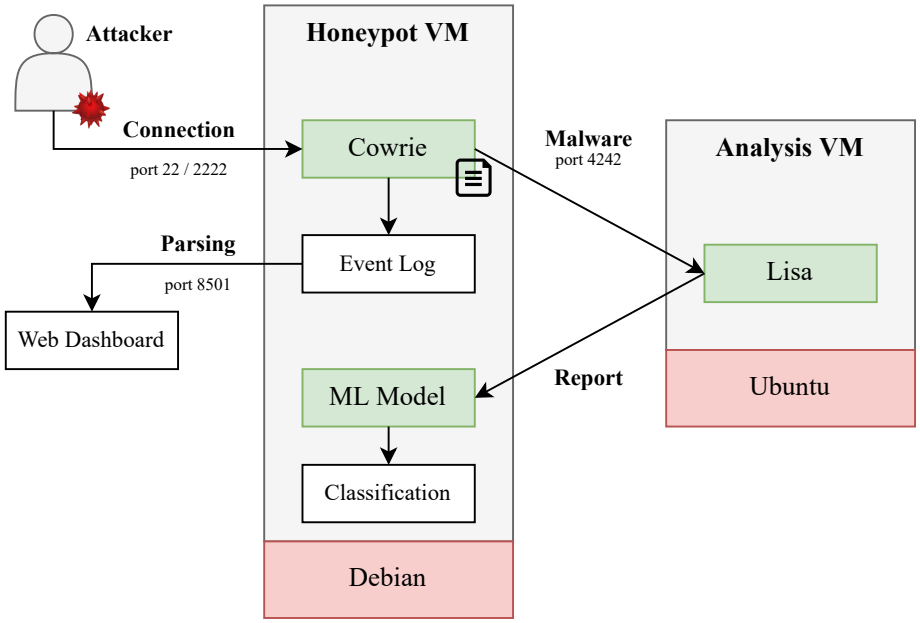

Figure 5.1: Classical Approach Architecture

Any malware sent to the honeypot machine is saved in the controller filesystem, and is automatically uploaded to the analysis machine (port 4242), through the use of a custom made script, as seen in Algorithm 4.

The analysis machine then processes the file, returning a report with the static and dynamic analysis traces it captures. The report is returned to the honeypot machine, which converts and parses the report information into a readable format for the machine learning model, which then receives this parsed report and returns a classification for the respective malware sample.

With this system presented and ready, we can now tackle the remaining evaluation criteria.

5.2 Attacker Capturing Effectiveness

The first step when evaluating a honeypot solution should be in regards to how capable it is to attract and fool attackers into performing their techniques in the honeypot machine. For this, it is

[1] cowrie, https://github.com/cowrie/cowrie

Algorithm 4 Script for sending honeypot malware for analysis

Input (Listens for changes in received files folder)
Output File Classification by Machine Learning Based Classifier

$uploadedFilesFolder \leftarrow os.listdir('downloads')$ ▷ Save original uploaded files list before listening for new files
while True **do**
  for $file \in os.listdir('downloads')$ **do**
    if $file \notin uploadedFilesFolder$ **then** ▷ Detect new files
      $postConfirmation \leftarrow sendFileForAnalysis(file)$
      $time.sleep(120)$ ▷ Sleep through the analysis duration
      $tasklist \leftarrow getTasksFromAnalysisTool$
      for $analysisTask \in tasklist$ **do**
        if $postConfirmation['task\_id'] == analysisTask['task\_id']$ **then**
          if $analysisTask['result'] != FAILURE$ **then**
            $classifySample(analysisTask)$
          end if
        end if
      end for
    end if
  end for
end while

important to not only compare the amount of traffic that is generated daily in the target machine, but to also look at how many attackers return to the honeypot in multiple different occasions. The more convincing the honeypot is to a given attacker, the more likely this same attacker is to come back and further attempt to corrupt or use the machine in a malicious way. On the other hand, if this same attacker can easily conclude that the machine is in fact a honeypot in a preliminary probing stage, it is far less likely to even attempt to perform any malicious behaviour on this same machine. In order to evaluate our eBPF based approach in this regard, we will compare it against the previously set-up classical baseline approach, in order to understand if our solution is able to outperform classical solutions in attracting and fooling any attackers.

For this, we let the machines run fully exposed for the same time frame (12 days), and gathered a few metrics for each machine, which we will now discuss.

5.2.1 Daily Connections

On average, our eBPF based approach had 11,49 times more daily connections than the baseline (24117,75 to 2097,92). Even though these values include any type of connections, even those without a login attempt, the difference is quite considerable. Figure 5.2 shows the daily trends for the number of connections between both of the approaches.

It is worth noting that both of the honeypots were running intermittently before the time frame displayed by the graph. In these prior runs, both of the approaches were very similar in terms of daily connections. We speculate that some time between those runs and the time frame presented by the graph, our eBPF honeypot was added to a list of known vulnerable machines, which would

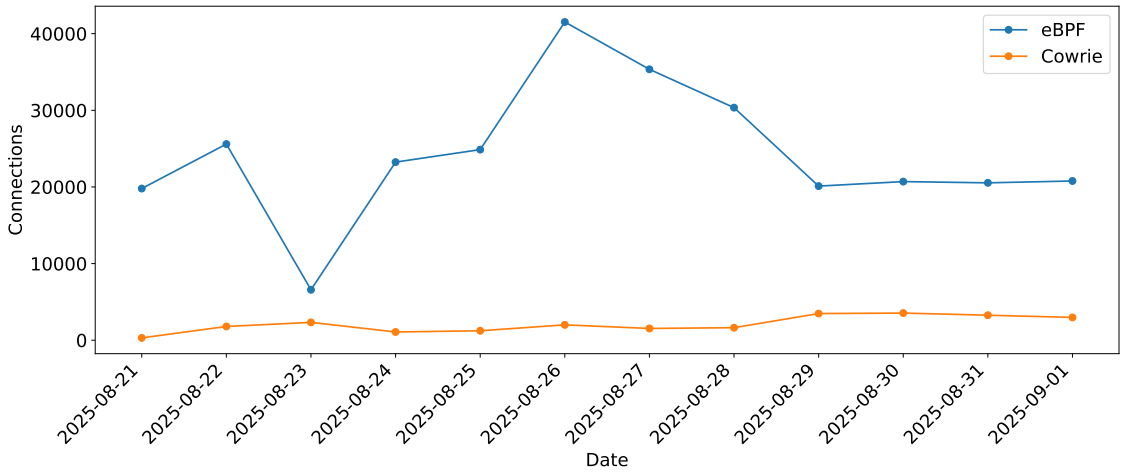

Figure 5.2: Daily Connections Comparison

explain the difference in traffic volume between the approaches in the time frame shown in the graph.

5.2.2 Daily Different IP Addresses

Similarly to the previous metric, our eBPF honeypot presented a higher number of different IPs per day than the baseline approach, although only 7,12 fold (1244,67 to 174,67). This number by itself may seem inconclusive, but it becomes relevant when we determine the average number of connections for each IP address. This is achievable through the following equation:

$$Avg_{ConnsPerIP} = \frac{\frac{\sum_{i=1}^{n} C_i}{n}}{\frac{\sum_{i=1}^{n} I_i}{n}} = \frac{\sum_{i=1}^{n} C_i}{\sum_{i=1}^{n} I_i}$$

Where,

- C_i = Number of connections on a given day i,
- I_i = Number of different IPs on a given day i,
- n = Total number of days.

When applying this equation to both solutions (MaCHoS and the baseline), we observe that each different IP that connects to the eBPF honeypot does so on average 19,37 times per day, while this number is only 12,01 for the baseline approach (about a 38% decrease). Figure 5.3 shows the daily trends for the number of different IP addresses that connected to each of the honeypot approaches.

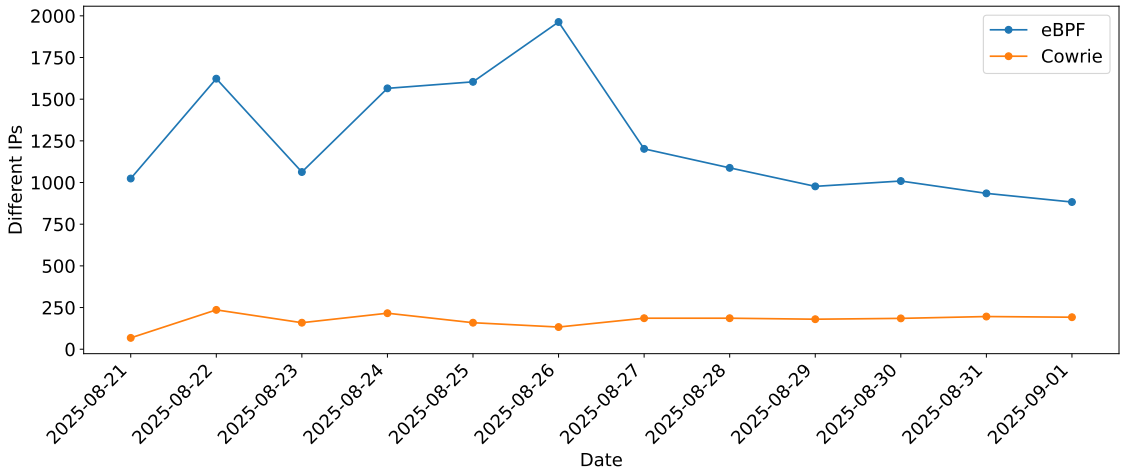

Figure 5.3: Unique Remote IPs Comparison

Furthermore, Figures 5.4 and 5.5 show the number of y different IP addresses that connected to our approach and to the baseline approach, respectively, a given x amount of times. For example, from Figure 5.4, we can conclude that around 5 different IP addresses connected to our MaCHoS system's honeypot 6 times. Looking at the trend line also present in the graph, we can see that most IPs tend to do a relatively small amount of connections (1 to 20), although as seen previously, our proposed eBPF honeypot shows higher signs of activity than its baseline counterpart, also having an interesting cluster of activity around the 3000 to 5000 connections range.

It is also worth mentioning that, on average, while only 3,4% of IPs that connected to our eBPF honeypot also connected to the baseline approach, 22,3% of the IPs that connected to the baseline approach also connected to our proposed eBPF approach, used in MaCHoS. The figures for this metric in each of the approaches through the 12 days of this evaluation can be seen in Figure 5.6.

Figure 5.7 shows, for each day, how many of the total unique IPs were seen only in our proposed approach eBPF honeypot, how many were seen by both of the approaches in the same day, and how many were seen by both approaches on different days. This may be indicative that the attackers who connected to both honeypots may be more interested in mass scanning, while those who connected solely to our honeypot may be focused on more specific targets, through Shodan[2] or a database of known vulnerable machines, which further solidifies the explanation for the disparity between the approaches.

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

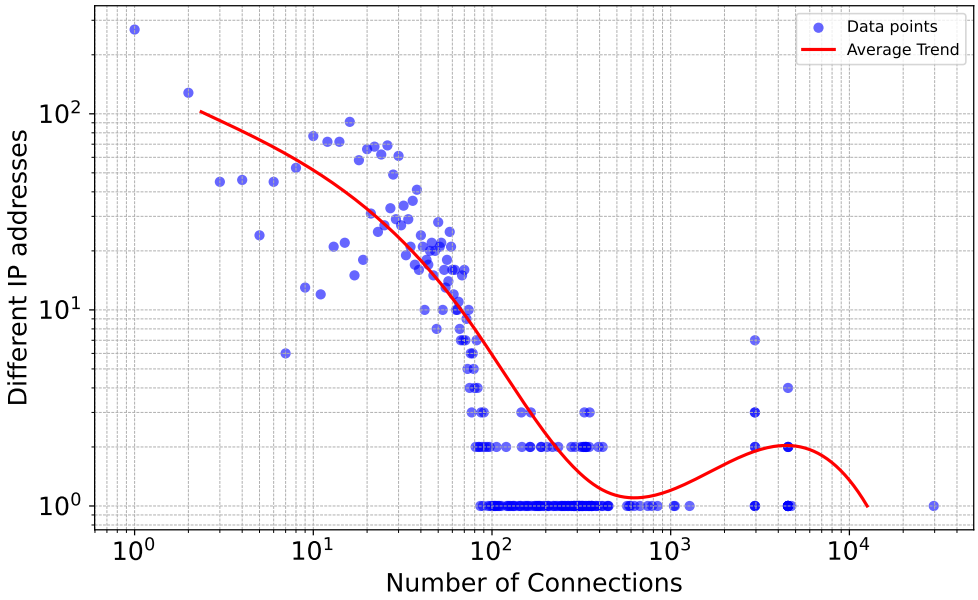

Figure 5.4: Unique IPs by Number of Connections (MaCHoS Solution)

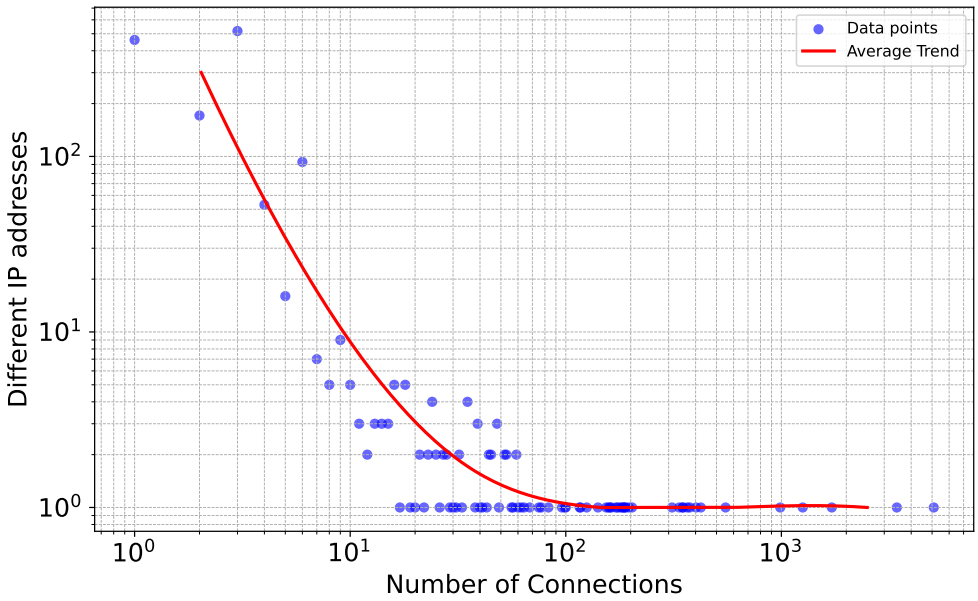

Figure 5.5: Unique IPs by Number of Connections (Baseline Approach)

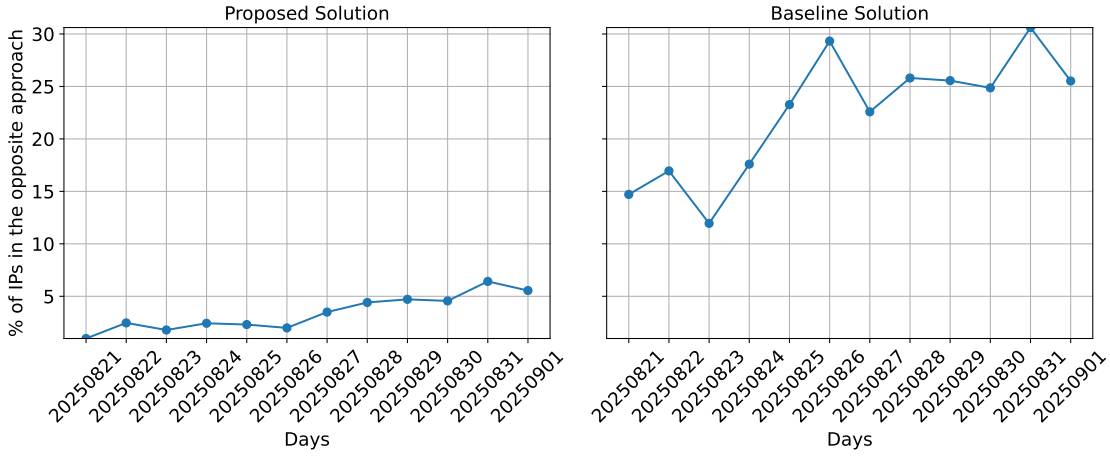

Figure 5.6: Percentage of total different IP addresses seen in opposite approach

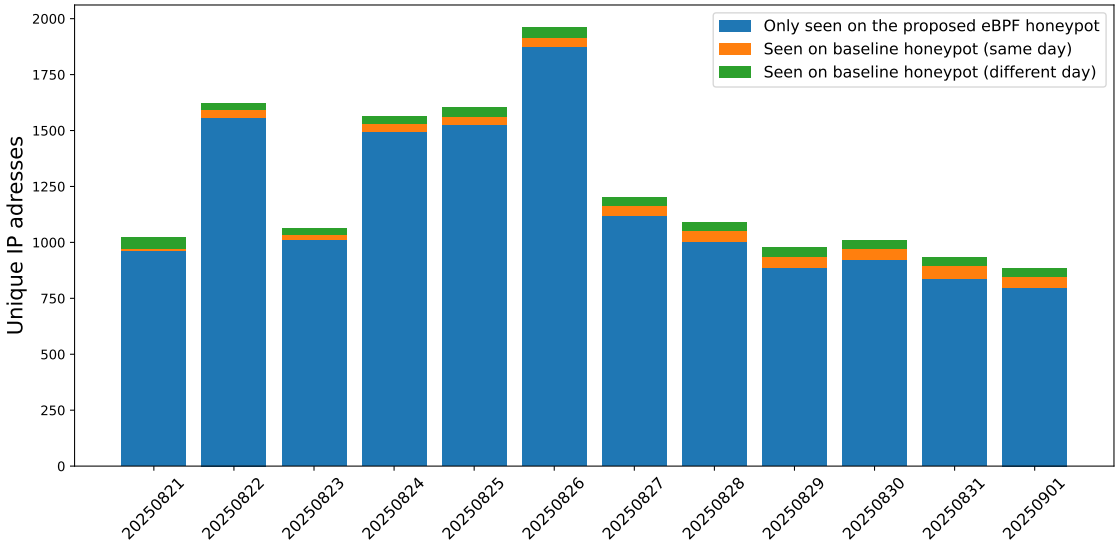

Figure 5.7: Unique IP overlap between approaches per day

5.2.3 Daily Successful Login Attempts

The final comparison metric between the two approaches is the successful login attempts. Unsurprisingly, our eBPF approach saw about 12,59 times more successful logins per day than the baseline (18683,75 to 1484,00). As before this number may seem inconclusive, but it becomes quite impressive when we compare the average percentage of successful logins out of all daily connections. This is achievable through the following equation:

$$Avg_{successfulLogin\%} = \frac{\frac{\sum_{i=1}^{n} L_i}{n}}{\frac{\sum_{i=1}^{n} C_i}{n}} \times 100 = \frac{\sum_{i=1}^{n} L_i}{\sum_{i=1}^{n} C_i} \times 100$$

Where,

- L_i = Number of successful logins on a given day i,
- C_i = Number of connections on a given day i,
- n = Total number of days.

When applying this equation to both the approaches, we determined that 77,4% of all eBPF honeypot connections manage to login successfully, while this is only true for 70% of all baseline honeypot connections. This is extremely impressive, especially when considering that the baseline approach runs a honeypot with no login credential restrictions (any credentials are able to log in), compared to our eBPF approach, which only allows for a handful of credentials when logging in. Figure 5.8 presents the daily trends for the number of successful logins in both of the evaluated approaches.

Table 5.1 compares the total values for the three discussed metrics between both the approaches, in the chosen time frame. It is worth noting that the total of different IPs may seem low compared to the values seen in Section 5.2.2, this is due to the fact that this value considers the time frame as a whole, but for the previous calculations, we imposed that the number of different IPs would be reset each day.

Table 5.1: Timeframe totals for each approach (all metrics)

Averages	Connections	Different IPs	Successful Logins
eBPF	289413	2632	224205
Cowrie	25175	1455	17808

With all of these metrics compared, it is safe to say that our eBPF based approach surpasses the baseline approach by quite a margin, making it much more attractive to the average attacker. This opening evaluation is a large step into confirming the premise of EC2 (and therefore RQ4 and RQ5), as we prove that the our MaCHoS solution honeypot attracts attackers more effectively than existing solutions (represented by the established baseline).

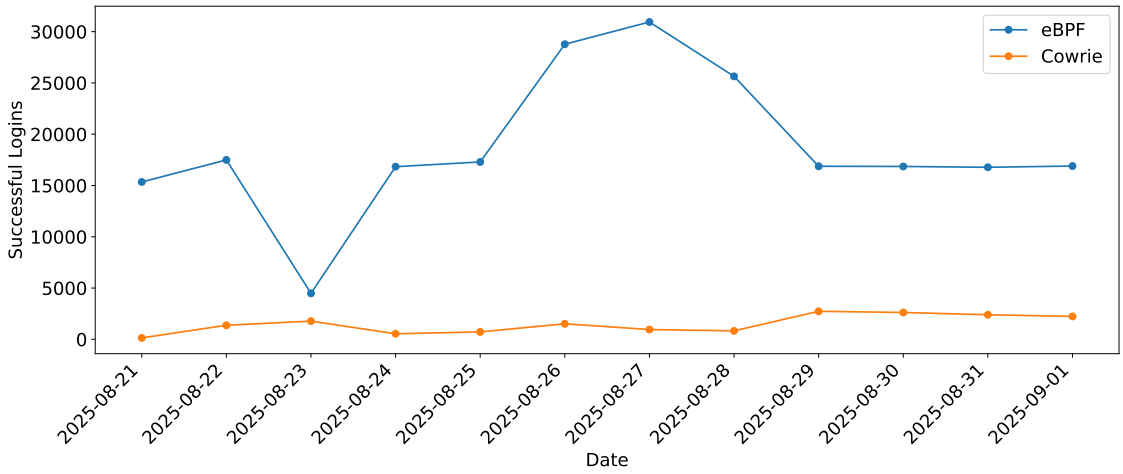

Figure 5.8: Daily Successful Logins Comparison

5.3 Model Classification Accuracy

Although MaCHoS and baseline approach use an identically trained machine learning model to classify samples collected by their respective honeypots, this same model may present different levels of accuracy depending on the quality and completeness of the static and dynamic analysis traces it receives. In this section we will compare our approach, which performs integrated analysis with dynamic traces collected from the malware's execution on our eBPF honeypot, with the baseline approach, which relies on a third-party tool to perform its analysis, since this approaches honeypot (cowrie), simply collects received malware samples. This allows us to evaluate both how the model performs with traces from each of the approaches honeypots and analysis components, and also to get an idea of what type of samples each of the approaches capture.

To establish a fair comparison, VirusTotal will be used as the metric for the accuracy evaluation, since both approaches rely on a model trained using VirusTotal reports as ground truth. All samples used were collected by the honeypots and classified by the model in the exact same time frame (July 15th, 2025 - September 9th, 2025).

5.3.1 MaCHoS Classification Effectiveness

Our solution captured a total of 52 different malware samples during the aforementioned time frame. Interestingly, of these 52 different samples sent by attackers, 12 (around 23.0%) were benign. This is quite impressive and unexpected, as one would initially assume that any file sent to a honeypot by an external entity would be malware, but there is a possibility that an attacker may preliminarily send a benign file to the target system on purpose, to understand if this target system allows for file sending and execution. Unfortunately, as we did not prepare for this eventuality, and since our machine learning classifier is only trained to distinguish malware between its different families, these 12 samples will be inaccurately classified, and therefore false positives in our

solution.

Furthermore, out of the 52 total samples, 8 (around 15.0%) returned labels that the machine learning model is not trained to recognise. Although all of these labels had samples in the original dataset we compiled from the various repositories (as explained in Section 4.1), these labels did not fulfil the previously set 100 sample threshold requirement to be present in the final dataset used for training and testing the machine learning model, and will thus yield incorrect classifications when evaluated by the trained model). Therefore, our model misclassified these cases in their correct malware category.

Lastly, out of the remaining 32 samples, the machine learning model incorrectly classified 1 of these samples, in which their correct label was in the dataset. This brings our final total down to 31 out of 52 accurate classifications, or 59,6% accuracy. Considering how many of these samples were out of the scope of the machine learning model, this result is far from negative. Nevertheless, future work should address these gaps, in order to achieve a better result that accounts for these cases.

5.3.2 Baseline Approach Classification Effectiveness

The baseline approach, representative of a classic honeypot and analysis system, captured 41 different samples in the aforementioned time frame. Out of these 41 samples sent by attackers, 3 were benign, which interestingly is a far smaller figure than in our solution. As mentioned in the previous section, as the machine learning model is not trained to distinguish benign samples from malicious ones, these 3 samples were incorrectly classified by the model, thus considered false positives.

Contrary to our approach, none of the remaining 38 samples belonged to labels which were not present in our training dataset. This is indicative that the malware received by the baseline honeypot is much more commonly seen and represented in public databases than the one received by our eBPF honeypot in MaCHoS, since our model was trained on a dataset comprised of samples from some of these known public databases.

Lastly, out of the remaining 38 samples, the machine learning model incorrectly classified 12 of these samples.

This leaves the final classification accuracy for the baseline approaches model at 26 out of 41, or 63,4%.

5.3.3 Conclusion

Although the baseline approach slightly surpassed our proposed solution in the final model accuracy, there are a few key considerations which should be addressed:

The higher proportion of samples which turned out to be benign or belonging to labels not present in the finalised model training dataset significantly impacted the proposed solutions model final accuracy score. Although this is a sign that the dataset could have been better prepared to include these scenarios, it also suggests that the proposed solutions eBPF honeypot is able to attract

and capture a more diverse set of interactions and samples than those seen in current approaches and datasets. Table 5.2 displays the direct comparison of the models accuracy between the two approaches, divided by each label.

Table 5.2: Model accuracy comparison by label

	Proposed Approach				Baseline Approach		
Labels	Total	Correct	Incorrect		Total	Correct	Incorrect
SINGLETON	13	12	1		16	8	8
gafgyt	13	13	0		0	0	0
mirai	1	1	0		2	1	1
prometei	0	0	0		0	0	0
tsunami	0	0	0		1	1	0
berbew	0	0	0		0	0	0
kaiji	0	0	0		0	0	0
dofloo	0	0	0		0	0	0
xmrig	5	5	0		15	14	1
xorddos	0	0	0		1	1	0
mayday	0	0	0		0	0	0
setag	0	0	0		0	0	0
local	0	0	0		0	0	0
ares	0	0	0		0	0	0
multiverze	0	0	0		3	1	2
(Not-In-Dataset)	8	0	8		0	0	0
(Benign)	12	0	12		3	0	3
Total	52	31	21		41	26	15
Accuracy	$(31/52) \times 100 = 59,6\%$				$(26/41) \times 100 = 63,4\%$		

Ideally, more samples are required to provide more accurate conclusions, but this batch of classifications gave us a good idea on how both of the approaches compare against each other in regards to classification, against real attackers.

Although our MaCHoS solutions model was marginally less effective at classifying captured samples than the baseline, it still showed promising results for future iterations. Even though its effectiveness did not surpass the baseline, the amount of unconventional samples captured in our MaCHoS solution definitely provided a greater deal of insight into lesser known threats in current systems, which greatly favours it when approaching and answering RQ4.

5.4 Manual Testing

Besides considering how both of the approaches compared in regards to the malware received by their respective honeypots, manual testing was performed to understand how these systems behave when we are the ones sending the malware, in order to compare how each of the approaches may differ in terms of feature gathering. Two samples were manually tested on both of the approaches, the first being picked from the samples received in our MaCHoS eBPF honeypot component, and one sample directly re-used from our model training dataset.

Folders	Sockets
/proc/	/socket:[54612]/
/sys/	/socket:[53825]/
/tmp/	/socket:[53831]/
/run/	
/dev/	
/-2/	
//	
/home/	
/-6/	

Table 5.3: Open and Write trace endpoints only seen in MaCHoS (eBPF honeypot sample)

5.4.1 eBPF Honeypot Sample

The first tested sample was a sample previously sent to our eBPF honeypot. This sample was sent to the baseline approach and analysed, and the results from both approaches were compared. This samples sha256 hash is *94f2e4d8d4436874785cd14e6e6d403507b8750852f7f2040352069a75da4c00*, and it was labelled by both of the approaches and VirusTotal (after label normalization) as a *SINGLETON*. As far as the capturing differences between both of the approaches, these were as follows:

In the static analysis fields, both approaches did equally well, both capturing 70 imported symbols, 77 exported symbols, 35 sections, and 3 shared libraries.

In the dynamic analysis fields though, we saw a much better performance by our proposed approach, managing to capture 9 more folders where the malware sample opened files in, as well as 3 sockets (Table 5.3). Furthermore, out of these folders and sockets, we also captured write traces in 4 of these folder and all 3 sockets (in the aforementioned table in Orange).

MaCHoS also captured 2 shell commands run (“dash” and “lsof”) and **6109** contacted IPs, while the baseline approach was only able to capture one contacted IP.

From this test we conclude that, for the chosen sample, MaCHoS was far superior at capturing dynamic traces than the baseline approach. Both approaches did equally well at capturing static traces.

5.4.2 Reused Dataset Sample

The second sample was randomly picked from our machine learning model training dataset. The selected samples sha256 hash is *3e1fcc69ff604cf01cf90b5eb69bfadce00274ea910d5e9df95edb5bea341cc9*, and this sample was labelled by both of the approaches and VirusTotal (after label normalization) as a *xmrig*. The capturing differences from both of the approaches are as follows:

Interestingly, this malware’s static features are basically empty. It does not have any sections, nor does it have a symbol table, only strictly necessary header information, which we manually confirmed with *objdump*. As far as dynamic features go, our approach surpassed the performance

Folders	Pipes
/home/ /sys/ /-2/ /usr/ /dev/ /-12/	pipe:[82335]/

Table 5.4: Open and Write trace endpoints only seen in MaCHoS (reused dataset sample)

of the baseline.

Our approach captured 4 more folders where the malware sample opened files in, as well as 1 pipe (Table 5.4). Furthermore, out of these folders and pipes, we also captured write traces in 3 of these folders and the 1 pipe (in the aforementioned table in Cyan).

It is also worth noting that the baseline approach captured 2 folders that the sample opened files in that our approach did not manage to capture: "/root/" and "/bin/".

MaCHoS also captured 2 shell commands run ("dash" and "xtables-nft-multi") as well as **13** IPs contacted, while the baseline approach did not capture a single IP.

Similarly to the previous test, we conclude that for the chosen sample, MaCHoS was marginally superior at capturing dynamic traces than the baseline approach. Once again, both approaches did equally well at capturing static traces.

5.4.3 Conclusions

Overall, both of the approaches did equally well as far as static analysis is concerned, but our approach was overall better at capturing dynamic analysis based traces. The process for manual testing was also critical in finding and accounting for a lot of edge cases which we originally did not consider when building our system, both in the capturing of events and traces, and the analysing of samples. Additionally, this section helped addressing RQ4 and RQ5, especially in regards to how the different approaches provide insight on malware and attacker behaviour.

5.5 Complex Malware Considerations

While the previous sections compared both approaches in regards to their luring effectiveness, sample classification and behavioural tracing, this section discusses how MaCHoS handles complex attacker behaviours, particularly second stage payloads, analysis evasion, and fileless malware.

5.5.1 Second Stage Payloads

One of the challenges that most common low and medium interaction honeypots struggle with capturing is second stage payloads, which cannot be captured by a classical honeypot approach as they simply capture malware received for later analysis without executing it. On the other hand,

our solution is able to capture this type of attack, as seen by the example below. This was a real attack, captured by our honeypot, and the attack description will be accompanied by the actual events as saved on our controllers log file.

An unknown attacker connected to our eBPF honeypot on July 16th, 2025 (address 196.251.**.***) and performed the following actions:

The attacker started by probing the system in an ssh session (sessionID: 3881), running commands such as “bash -c uname -r | awk ’printf $1”’.

```
{``session'': ``3881-sshd-196.251.**.***-2254'', ``subsession'':
    ``none'', ``pid'': 3949, ``type'': ``exec'', ``filename'':
    ``/usr/bin/bash'', ``fid'': ``8.1:438'', ``args'': [``bash'',
    ``-c'', ``uname -r | awk '{printf $1}'''], ``kind'': ``NORMAL''}}
```

When it deemed the system as fit for deploying their payload, the attacker started a new session (sessionID: 3955) to upload their main payload into the honeypot using “scp -qt /var/tmp/SgQbTKWW”

```
{``session'': ``3955-sshd-196.251.**.***-140'', ``subsession'':
    ``none'', ``pid'': 3965, ``type'': ``exec'', ``filename'':
    ``/usr/bin/scp'', ``fid'': ``8.1:9068'', ``args'': [``scp'',
    ``-qt'', ``/var/tmp/SgQbTKWW''], ``kind'': ``NORMAL''}
```

Which wrote the payload contents to the file “/var/tmp/SgQbTKWW” on the honeypot (seen below).

```
{``session'': ``3955-sshd-196.251.**.***-140'', ``subsession'':
    ``none'', ``pid'': 3965, ``type'': ``file_close'', ``filename'':
    ``/var/tmp/SgQbTKWW'', ``fid'': ``8.1:12973'', ``fd'': 3,
    ``written'': true, ``ondisk'':
    ``3955-sshd-196.251.**.***-140/3965-scp/file-_var_tmp_SgQbTKWW''}
```

The attacker then exited session 3955 and went back to session 3881, running a long list of commands which included “chmod 777 SgQbTKWW ; ./SgQbTKWW” (which provided permissions for executing and executed the previously uploaded payload).

```
{``session'': ``3881-sshd-196.251.**.***-2254'', ``subsession'':
    ``none'', ``pid'': 3993, ``type'': ``exec'', ``filename'':
    ``/usr/bin/chmod'', ``fid'': ``8.1:571'', ``args'': [``chmod'',
    ``777'', ``SgQbTKWW''], ``kind'': ``NORMAL''}

{``session'': ``3881-sshd-196.251.**.***-2254'', ``subsession'':
    ``none'', ``pid'': 3993, ``type'': ``chmod'', ``filename'':
    ``SgQbTKWW'', ``fid'': ``8.1:12973'', ``mode'': 511}
```

```
{'session': '3881-sshd-196.251.**.***-2254', 'subsession': 'mal-3994',
    'pid': 3994, 'type': 'exec', 'filename': '/var/tmp/SgQbTKWW',
    'fid': '8.1:12973', 'args': ['./SgQbTKWW'], 'kind': 'TRACKED'}
```

As seen by the “exec” event above, the file was flagged as “TRACKED” since it was uploaded by an attacker. Furthermore, from this point onwards, every event generated by the process id attached to this file or its children will have the “’subsession’: ’mal-3994”’ flag

Later in the main payload execution, this malware writes a payload to another file, "/dev/shm/retea", provides it with execution permissions and executes it, thus completing the second stage payload uploading and execution.

```
{``session'': ``3881-sshd-196.251.**.***-2254'', ``subsession'':
    ``mal-3994'', ``pid'': 3994, ``type'': ``file_close'',
    ``filename'': ``/dev/shm/retea'', ``fid'': ``0.23:2'', ``fd'': 3,
    ``written'': true, ``ondisk'':
    ``3881-sshd-196.251.**.***-2254/3994-SgQbTKWW/file-_dev_shm_retea''}
```

```
{``session'': ``3881-sshd-196.251.**.***--2254'', ``subsession'':
    ``mal-3994'', ``pid'': 4132, ``type'': ``exec'', ``filename'':
    ``/usr/bin/chmod'', ``fid'': ``8.1:571'', ``args'': [``chmod'',
    ``+x'', ``retea''], ``kind'': ``NORMAL''}

{``session'': ``3881-sshd-196.251.**.***--2254'', ``subsession'':
    ``mal-3994'', ``pid'': 4132, ``type'': ``chmod'', ``filename'':
    ``retea'', ``fid'': ``0.23:2'', ``mode'': 493}
```

```
{'session': '3881-sshd-196.251.**.***-2254', 'subsession': 'mal-3994',
    'pid': 4133, 'type': 'exec', 'filename': '/dev/shm/retea', 'fid':
    '0.23:2', 'args': ['./retea',
    'KOFVwMxV7k7XjP7fwXPY6Cmp16vf8EnL54650LjYb6WYBtuSs3Zd1Ncr3SrpvnAU',
    'Haceru'], 'kind': 'TRACKED'}
```

This is proof that our honeypot solution can capture second stage payload execution attacks, which classical approaches are not able to capture. It should also be greatly underlined that, at the time of this evaluation, the file "/var/tmp/SgQbTKWW" has never been seen by VirusTotal, which is extremely relevant since it highlights that our eBPF based approach uncovers samples which have not been previously seen in large public databases. This is a testament to the systems capability of capturing genuinely novel samples, which in turn enables early identification and study of these same novel samples.

5.5.2 Analysis Evading Threats

Conveniently enough, the same attacker who performed the previously shown second stage payload attack has also connected to the baseline honeypot on multiple occasions. This is good because it provides a direct comparison on how a complex attacker behaves in each of the approaches.

Similarly to the sessions started by this address in our eBPF solution, all sessions start with a "uname" command, presumably for probing the system for a debug or analysis environment. However, the subsequent behaviour diverged substantially between the two approaches. Contrary to what we saw in the eBPF solution, where the attacker subsequently loaded and executed a payload that loads and executes further payloads with persistence capabilities, the attacker terminated the connection immediately after receiving the output from the supposed probing command in the baseline approach.

This suggests not only that the information disclosed by this commands output did not meet the attackers expectation or (correctly) raised suspicions regarding the authenticity of the baseline

environment, but it also further solidifies our eBPF based approach as being more capable than currently used approaches in capturing and keeping complex attackers interested.

Furthermore, this comparison raises the importance of realism when creating a honeypot system. Attackers that perform verification techniques against debug and analysis environment prior to deploying payloads or performing malicious behaviours will halt this process if the honeypot in question is not deceptive enough to pass as a real system for these attackers and their verification processes.

5.5.3 Fileless Malware

Along the runtime of our eBPF honeypot, there were 30 occasions where attackers loaded and ran fileless malware. Such was the case for this session, by an attacker 134.199.***.** on July 29th, 2025:

Similarly to the second stage payload attack, the attacker started by probing the system in an ssh session (sessionID: 4362), running the command “uname -snrvm”


```
{''session'': ''4362-sshd-134.199.***.**-2755'', ''subsession'':
    ''none'', ''pid'': 4367, ''type'': ''exec'', ''filename'':
    ''/usr/bin/uname'', ''fid'': ''8.1:690'', ''args'': [''uname'',
    ''-snrvm''], ''kind'': ''NORMAL''}
```


Deeming the system as fit for payload deployment, the attacker uploaded their main payload into the honeypot using “scp -qt /var/tmp/CbCoXHIqCQSZahFW”


```
{''session'': ''4362-sshd-134.199.***.**-2755'', ''subsession'':
    ''none'', ''pid'': 4370, ''type'': ''exec'', ''filename'':
    ''/usr/bin/scp'', ''fid'': ''8.1:9068'', ''args'': [''scp'',
    ''-qt'', ''/var/tmp/CbCoXHIqCQSZahFW''], ''kind'': ''NORMAL''}
```


The attacker then started a new session (sessionID: 4378), and executed a chmod command and ran the previously uploaded payload (through “bash -c cd /var/tmp ; chmod +x CbCoXHIqCQSZahFW ; ./CbCoXHIqCQSZahFW &>/dev/null & ”).


```
{''session'': ''4378-sshd-134.199.***.**-48849'', ''subsession'':
    ''none'', ''pid'': 4387, ''type'': ''exec'', ''filename'':
    ''/usr/bin/chmod'', ''fid'': ''8.1:571'', ''args'': [''chmod'',
    ''+x'', ''CbCoXHIqCQSZahFW''], ''kind'': ''NORMAL''}

{''session'': ''4378-sshd-134.199.***.**-48849'', ''subsession'':
    ''mal-4388'', ''pid'': 4388, ''type'': ''exec'', ''filename'':
    ''/var/tmp/CbCoXHIqCQSZahFW'', ''fid'': ''8.1:2389'', ''args'':
    [''./CbCoXHIqCQSZahFW''], ''kind'': ''TRACKED''}
```


Posteriorly, during the main payload execution, the malware writes and executes a separate payload in memory.


```
{''session'': ''4378-sshd-134.199.***.**-48849'', ''subsession'':
    ''mal-4388'', ''pid'': 4388, ''type'': ''file_close'',
    ''filename'': ''/memfd:oMkJkwMTFysYUGxy (deleted)'', ''fid'':
    ''0.1:1043'', ''fd'': 8, ''written'': true, ''ondisk'':
```

```
    ''4378-sshd-134.199.***.**-48849/4388-CbCoXHIqCQSZahF/
    file-_memfd:oMkJkwMTFysYUGxy (deleted)''}

{''session'': ''4378-sshd-134.199.***.**-48849'', ''subsession'':
    ''mal-4388'', ''pid'': 4399, ''type'': ''exec'', ''filename'':
    ''/memfd:oMkJkwMTFysYUGxy (deleted)'', ''fid'': ''0.1:1043'',
    ''args'': [''/proc/4388/fd/7''], ''kind'': ''TRACKED''}
```


All of these events were captured by the honeypot, and the payloads static and dynamic analysis traces were sent to our machine learning classifier, which classified this as a cryptocurrency mining malware (multiverze label), though VirusTotal labels this as benign. Even more interesting though, is the fact that out of all 30 instances of this same malware being uploaded and run, all of them had the same hash, even though they came from different IPs in completely different dates.

From the analysis performed in this section, it is clear that our solution, MaCHoS, has the capacity to capture multiple kinds of complex attacks which are not seen in commonly used approaches. This section addresses RQ5, showing that our solution provides additional insight on malware and attacker behaviour, in comparison with classic honeypot and analysis systems.

5.6 eBPF vs VMI for MaCHoS Honeypot

Lastly, we aim to evaluate if our eBPF approach was the correct choice to integrate our high-interaction honeypot. For such, we compared eBPF with VMI, which has been traditionally used for VM tracing and analysis. A question that commonly arises when using eBPF for tracing in a VM is "why not just use VMI instead?". To evaluate this, we ran a performance benchmark using VMs in two different hardware platforms, a low-end user PC and a server-class PC, focusing on comparing the overhead generated by monitoring system calls (`sys_getpid`, `sys_clone`, `sys_execve`, `sys_write`) with both eBPF and VMI, using a configuration without monitoring as a baseline. While, for the system calls that generate less events, the performance was pretty similar, when testing performance with system calls that generate a larger number of events, the eBPF approach performed 20x to 100x better than its VMI counterpart. This is because while VMI works synchronously with the target system's execution, pausing the system for each event, eBPF works synchronously with the kernel, emitting events to a queue or buffer. However, this comes at a cost, since eBPF based approaches can suffer from event loss if the kernel generates too many events in a short period of time, which was also experienced in this benchmark when running more complex system calls.

Another problem to keep in mind is the loss of isolation between host and target when using eBPF, which can result in a degradation in stealthiness. There are ways to work around this problem though, hiding the monitoring module from commands such as `ls` and `ps aux`.

Considering the previously mentioned tests, we conclude that despite both of the approaches differences, eBPF is a viable alternative to VMI for honeypot tracing, therefore justifying its use in MaCHoS.

Chapter 6

Conclusion

This dissertation introduced a novel honeypot and analysis architecture capable of classifying received malware samples. Through static analysis of the reconstructed samples on the controller, dynamic analysis of traces captured by eBPF from sample executions on the honeypot, and a machine learning model trained with VirusTotal samples, the honeypot processes and classifies the received malware.

This architecture was implemented in the MaCHoS system, and subsequently compared against a baseline approach, representative of currently existing honeypot and analysis solutions. The proposed architecture far surpassed the baseline in attacker activity and behavioural capturing, as well as consideration for complex attacker techniques such as second stage payloads and fileless malware. The classification of received malware samples on the proposed approach was on par with the baseline approach.

The usage of eBPF in the honeypot system provided a reliable way to listen for and record attacker behaviour on a live target, which in turn allowed for a deep and insightful analysis of their tactics and tools used. This component was a big difference maker, as it created a bridge for communication from the honeypot VM to its controller, without compromising stealth.

6.1 Limitations

6.1.1 Dataset Representation

The current ML model is trained to classify 14 types of malware families: gafgyt, mirai, prometei, tsunami, berbew, kaiji, dofloo, xmrig, xorddos, mayday, setag, local, ares, and multiverze. Even though these were the most seen samples in our source datasets, during the evaluation phase we noticed that a large number of samples captured by the honeypot did not belong to these labels, which significantly hindered the models effectiveness. A good way that could solve this problem is to include the newly acquired samples in our own dataset, and retrain the machine learning classifier with these new samples. Although this field of research is ever-evolving, and thus there is not one simple solution that can classify all threats old and new, using new samples in our model allows us to stay much closer to the current threat landscape and better prepare for future attacks.

6.1.2 Controller Disk Space

One of the main limitations our honeypot system faces is regarding the space taken up by its logs. On average, every 20 minutes of honeypot run-time currently generates about 50MB worth of logs, which stacks up quite a lot over time. Furthermore, since the proposed solution reconstructs any files sent by the attacker on the controller, the size of the logs for that particular session can go up considerably. A possible solution for this limitation would be to reduce the verbosity of the logs saved for each session, while still allowing for the sessions to be reconstructed if needed. Another way to tackle this problem would be to implement some sort of filter based log rotation, where sessions without any interesting attacker behaviour would be deleted or archived after a certain predefined amount of time.

6.2 Future Work

Considering the previously described capabilities and limitations, the following points will present a few areas for future enhancements to the proposed system and its components:

- **Feature Review and Model Retraining:** Review currently used features to highlight redundancies or underutilised features to optimise model training, and use samples received by the honeypot to add more supported labels to the model and adapt to evolving attacker behaviour.
- **Scaling:** Deploying more instances of the proposed honeypot solution, and developing a centralised network to control, add and remove instances, ensuring a scalable solution without increasing complexity.
- **Honeypot Enhancement:** Make honeypot logging more lightweight to optimise disk usage and system overhead, and enhance session reconstruction to facilitate attacker behaviour analysis.
- **Network Tracing:** Add network tracing to the proposed honeypot solution to further characterize attackers and their behaviour, and to capture complex network targeting attacks.

6.3 Concluding Remarks

We believe this dissertation provides valuable contributions to the field of honeypots, particularly in regards to embedded analysis, as well as the field of threat intelligence, providing new ways to capture attacker behaviour and payloads associated with these attacks. Furthermore, we believe this dissertation presents a good argument for the more generalised usage of integrated ML based analysis components in high interaction honeypot systems for classifying malware received by the honeypots, instead of relying on third party services for this purpose.

List of Acronyms

API Application Programming Interface.

C&C Command-And-Control.

CPU Central Processing Unit.

CVE Common Vulnerabilities and Exposures.

DDoS Distributed Denial of Service.

eBPF extended Berkeley Packet Filter.

ELF Executable and Linkable Format.

GUI Graphical User Interface.

IoT Internet of Things.

IP Internet Protocol.

IRC Internet Relay Chat.

JSON JavaScript Object Notation.

KVM Kernel-based Virtual Machine.

LLM Large Language Model.

ML Machine Learning.

P2P Peer To Peer.

QEMU Quick Emulator.

RAT Remote Access Trojan.

S-NN Shared Nearest Neighbour.

SHA Secure Hash Algorithm.

SOC Security Operations Center.

SSH Secure Shell.

SVC Support Vector Classifier.

SVM Support vector Machine.

URL Uniform Resource Locator.

VM Virtual Machine.

VMI Virtual Machine Introspection.

Bibliography

[1] National Intelligence Council. Future of the battlefield, 2021. Part of the Global Trends 2040: Deeper Looks series.

[2] Niels Provos. A virtual honeypot framework. In *USENIX Security Symposium*, volume 173, pages 1–14, 2004.

[3] Adel Alshamrani, Sowmya Myneni, Ankur Chowdhary, and Dijiang Huang. A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities. *IEEE Communications Surveys & Tutorials*, 21(2):1851–1877, 2019.

[4] Mohammadhadi Alaeiyan, Saeed Parsa, and Mauro Conti. Analysis and classification of context-based malware behavior. *Computer Communications*, 136:76–90, 2019.

[5] Miguel Faísco, Ibéria Medeiros, and Hans P. Reiser. Towards real-time malware classification through honeypot analysis. In *2025 20th European Dependable Computing Conference Companion Proceedings (EDCC-C)*, pages 56–59, 2025.

[6] Niku Waltteri Saulinpoika Nuutinen, Miguel Faísco, Milan Petrusic, Ibéria Medeiros, and Hans P. Reiser. Evaluating eBPF as an alternative to virtual machine introspection for high-interaction honeypot implementation. In *Proceedings of the 14th Latin-American Symposium on Dependable and Secure Computing (LADC)*, 2025. accepted for publication.

[7] Imtithal Saeed, Ali Selamat, and Ali Abdelrahman. A survey on malwares and malware detection systems. *International Journal of Computer Applications (IJCA)*, Vol. 67:pp. 25–31, 04 2013.

[8] Jay G Heiser. Understanding today's malware. *Information Security Technical Report*, 9(2):47–64, 2004.

[9] J Carrillo-Mondéjar, José Luis Martínez, and Guillermo Suarez-Tangil. Characterizing linux-based malware: Findings and recent trends. *Future Generation Computer Systems*, 110:267–281, 2020.

[10] Samuel Kim. PE header analysis for malware detection. *Master's Projects*, (624), 2018.

[11] BooJoong Kang, Suleiman Y Yerima, Sakir Sezer, and Kieran McLaughlin. N-gram opcode analysis for android malware detection. *arXiv preprint arXiv:1612.01445*, 2016.

[12] Ömer Özcan, Muhammed Karaaltun, and Mehmet Hacibeyoğlu. Opcode and n-gram based malware classification. In *2024 8th International Artificial Intelligence and Data Processing Symposium (IDAP)*, pages 1–8. IEEE, 2024.

[13] Vikas Sihag, Anita Mitharwal, Manu Vardhan, and Pradeep Singh. Opcode n-gram based malware classification in android. In *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pages 645–650. IEEE, 2020.

[14] Niclas Ilg, Paul Duplys, Dominik Sisejkovic, and Michael Menth. A survey of contemporary open-source honeypots, frameworks, and tools. *Journal of Network and Computer Applications*, 220:103737, 2023.

[15] Amir Javadpour, Forough Ja'fari, Tarik Taleb, Mohammad Shojafar, and Chafika Benzaïd. A comprehensive survey on cyber deception techniques to improve honeypot performance. *Computers & Security*, 140:103792, 2024.

[16] Marcin Nawrocki, Matthias Wählisch, Thomas C Schmidt, Christian Keil, and Jochen Schönfelder. A survey on honeypot software and data analysis. *arXiv preprint arXiv:1608.06249*, 2016.

[17] Chongqi Guan, Heting Liu, Guohong Cao, Sencun Zhu, and Thomas La Porta. Honeyiot: Adaptive high-interaction honeypot for iot devices through reinforcement learning. In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 49–59, 2023.

[18] Marco Lucchese, Francesco Lupia, Massimo Merro, Federica Paci, Nicola Zannone, and Angelo Furfaro. Honeyics: A high-interaction physics-aware honeynet for industrial control systems. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, pages 1–10, 2023.

[19] Nitin Naik, Paul Jenkins, Nick Savage, Longzhi Yang, Kshirasagar Naik, Jingping Song, Tossapon Boongoen, and Natthakan Iam-On. Fuzzy hashing aided enhanced yara rules for malware triaging. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1138–1145, 2020.

[20] Gwanghyun Ahn, Kookjin Kim, Wonhyung Park, and Dongkyoo Shin. Malicious file detection method using machine learning and interworking with mitre att&ck framework. *Applied Sciences*, 12(21):10761, 2022.

[21] Febrian Setianto, Erion Tsani, Fatima Sadiq, Georgios Domalis, Dimitris Tsakalidis, and Panos Kostakos. Gpt-2c: a parser for honeypot logs using large pre-trained language models. In *Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ASONAM '21, page 649–653, New York, NY, USA, 2022. Association for Computing Machinery.

[22] Umesh Raut, Ajinkya Nagarkar, Chinmay Talnikar, Mustafa Mokashi, and Rishabh Sharma. Engaging attackers with a highly interactive honeypot system using chatgpt. In *2023 7th International Conference On Computing, Communication, Control And Automation (ICCUBEA)*, pages 1–5, 2023.

[23] Ziyang Wang, Jianzhou You, Haining Wang, Tianwei Yuan, Shichao Lv, Yang Wang, and Limin Sun. Honeygpt: Breaking the trilemma in terminal honeypots with large language model, 2024.

[24] Adit Kumar Chakravarty, Aditya Raj, Subham Paul, and S Apoorva. A study of signature-based and behaviour-based malware detection approaches. *Int. J. Adv. Res. Ideas Innov. Technol*, 5(3):1509–1511, 2019.

[25] Taiwo Oladipupo Ayodele. Types of machine learning algorithms. *New advances in machine learning*, 3(19-48):5–1, 2010.

[26] Cangshuai Wu, Jiangyong Shi, Yuexiang Yang, and Wenhua Li. Enhancing machine learning based malware detection model by reinforcement learning. In *Proceedings of the 8th International Conference on Communication and Network Security*, ICCNS '18, page 74–78, New York, NY, USA, 2018. Association for Computing Machinery.

[27] Robert Moskovitch, Dima Stopel, Clint Feher, Nir Nissim, and Yuval Elovici. Unknown malcode detection via text categorization and the imbalance problem. In *2008 IEEE International Conference on Intelligence and Security Informatics*, pages 156–161, 2008.

[28] Mohit Sewak, Sanjay K Sahay, and Hemant Rathore. Comparison of deep learning and the classical machine learning algorithm for the malware detection. In *2018 19th IEEE/ACIS international conference on software engineering, artificial intelligence, networking and parallel/distributed computing (SNPD)*, pages 293–296. IEEE, 2018.

[29] Kshitiz Aryal, Maanak Gupta, and Mahmoud Abdelsalam. Analysis of label-flip poisoning attack on machine learning based malware detector. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 4236–4245. IEEE, 2022.

[30] Liu Liu, Bao sheng Wang, Bo Yu, and Qiu xi Zhong. Automatic malware classification and new malware detection using machine learning. *Frontiers of Information Technology & Electronic Engineering*, 18(9):1336–1347, 2017.

[31] Zhixing Xu, Sayak Ray, Pramod Subramanyan, and Sharad Malik. Malware detection using machine learning based analysis of virtual memory access patterns. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 169–174, 2017.

[32] Amir Djenna, Ahmed Bouridane, Saddaf Rubab, and Ibrahim Moussa Marou. Artificial intelligence-based malware detection, analysis, and mitigation. *Symmetry*, 15(3), 2023.

[33] R Vinayakumar, Mamoun Alazab, KP Soman, Prabaharan Poornachandran, and Sitalakshmi Venkatraman. Robust intelligent malware detection using deep learning. *IEEE access*, 7:46717–46738, 2019.

[34] Matthew G Gaber, Mohiuddin Ahmed, and Helge Janicke. Malware detection with artificial intelligence: A systematic literature review. *ACM Computing Surveys*, 56(6):1–33, 2024.

[35] Shaila Sharmeen, Yahye Abukar Ahmed, Shamsul Huda, Bari Ş Koçer, and Mohammad Mehedi Hassan. Avoiding future digital extortion through robust protection against ransomware threats using deep learning based adaptive approaches. *IEEE Access*, 8:24522–24534, 2020.

[36] Firoz Khan, Cornelius Ncube, Lakshmana Kumar Ramasamy, Seifedine Kadry, and Yunyoung Nam. A digital dna sequencing engine for ransomware detection using machine learning. *IEEE Access*, 8:119710–119719, 2020.

[37] Domhnall Carlin, Philip O'Kane, and Sakir Sezer. A cost analysis of machine learning using dynamic runtime opcodes for malware detection. *Computers & Security*, 85:138–155, 2019.

[38] Tal Garfinkel and Mendel Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2003.

[39] Stewart Sentanoe, Benjamin Taubmann, and Hans P. Reiser. Sarracenia: Enhancing the performance and stealthiness of SSH honeypots using virtual machine introspection. In *Proc. of the 23rd Nordic Conference on Secure IT Systems*, 2018.

[40] Robert J. Joyce, Derek Everett, Maya Fuchs, Edward Raff, and James Holt. Claravy: A tool for scalable and accurate malware family labeling. In *Companion of the 16th ACM/SPEC International Conference on Performance Engineering (WWW Companion '25)*, 2025.

[41] Domhnall Carlin, Philip O'Kane, and Sakir Sezer. A cost analysis of machine learning using dynamic runtime opcodes for malware detection. *Computers & Security*, 85:138–155, 2019.

[42] Shaila Sharmeen, Yahye Abukar Ahmed, Shamsul Huda, Bari Ş Koçer, and Mohammad Mehedi Hassan. Avoiding future digital extortion through robust protection against ransomware threats using deep learning based adaptive approaches. *IEEE Access*, 8:24522–24534, 2020.

[43] Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. Understanding linux malware. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 161–175, 2018.

[44] Sudhakar and Sushil Kumar. An emerging threat: Fileless malware - a survey and research challenges. *Cybersecurity*, 3(1):1–12, 2020.

[45] Vasilis G. Tasiopoulos and Sokratis K. Katsikas. Bypassing antivirus detection with encryption. In *Proceedings of the 18th Panhellenic Conference on Informatics (PCI '14)*, pages 1–6, Athens, Greece, 2014. ACM.

[46] Masaya Sato and Toshihiro Yamauchi. Vmm-based log-tampering and loss detection scheme. *Journal of Information Technology*, 13(4):1–15, 2011.

[47] Andrew Case, Austin Sellers, David McDonald, Gustavo Moreira, and Golden G. Richard III. Defeating edr evading malware with memory forensics. In *DefCon 24*, 2024.

[48] Akira Yokoyama, Kou Ishii, Rui Tanabe, Yinmin Papa, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, Daisuke Inoue, Michael Brengel, Michael Backes, and Christian Rossow. Sandprint: Fingerprinting malware sandboxes to provide intelligence for sandbox evasion. In Fabian Monrose, Marc Dacier, Gregory Blanc, and Joaquin Garcia-Alfaro, editors, *Research in Attacks, Intrusions, and Defenses*, pages 165–187, Cham, 2016. Springer International Publishing.

[49] R. McGrew. Experiences with honeypot systems: Development, deployment, and analysis. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, volume 9, pages 220a–220a, 2006.

[50] Nagababu Pachhala, S Jothilakshmi, and Bhanu Prakash Battula. A comprehensive survey on identification of malware types and malware classification using machine learning techniques. In *2021 2nd international conference on smart electronics and communication (ICOSEC)*, pages 1207–1214. IEEE, 2021.

[51] Seyhmus Yilmaz and Sultan Zavrak. Adware: a review. *International Journal of Computer Science and Information Technologies*, 6(6):5599–5604, 2015.

[52] Wenjun Fan, Zhihui Du, David Fernández, and Victor A Villagra. Enabling an anatomic view to investigate honeypot systems: A survey. *IEEE Systems Journal*, 12(4):3906–3919, 2017.

[53] Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti. Detecting environment-sensitive malware. In *Recent Advances in Intrusion Detection: 14th International Symposium, RAID 2011, Menlo Park, CA, USA, September 20-21, 2011. Proceedings 14*, pages 338–357. Springer, 2011.

[54] Vatsal Vasani, Amit Kumar Bairwa, Sandeep Joshi, Anton Pljonkin, Manjit Kaur, and Mohammed Amoon. Comprehensive analysis of advanced techniques and vital tools for detecting malware intrusion. *Electronics*, 12(20), 2023.

[55] Ege Tekiner, Abbas Acar, A Selcuk Uluagac, Engin Kirda, and Ali Aydin Selcuk. Sok: cryptojacking malware. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 120–139. IEEE, 2021.

[56] Osama Khalid, Subhan Ullah, Tahir Ahmad, Saqib Saeed, Dina A. Alabbad, Mudassar Aslam, Attaullah Buriro, and Rizwan Ahmad. An insight into the machine-learning-based fileless malware detection. *Sensors*, 23(2), 2023.

[57] Pengbin Feng, Jianhua Sun, Songsong Liu, and Kun Sun. Uber: Combating sandbox evasion via user behavior emulators. In *Information and Communications Security: 21st International Conference, ICICS 2019, Beijing, China, December 15–17, 2019, Revised Selected Papers 21*, pages 34–50. Springer, 2020.