# Advances in Detection and Analysis of Modern Evasive Malware




Frederick Barr-Smith

Keble College

University of Oxford

A thesis submitted for the degree of

*Doctor of Philosophy*

Trinity 2023

# Abstract

Malware authors use a variety of techniques to avoid the sophisticated detection algorithms used by Antivirus (AV) and similar technologies. This is an area that has been subject to some open-source research in the form of academic research papers. Much of the closed-source research on this topic is contained within private organisations. This thesis is composed of the systematic study and analysis of these evasion techniques.

Our research is also concerned with the discovery of novel evasion techniques and emerging attack variants. One of these techniques was research of `.dll` sideloading as an attack vector on AV. Signature manipulation and packing were also tested against AV. Additionally, we measure Living-Off-The-Land techniques (LotL) and their effectiveness at AV evasion.

We look at the implementation of these techniques by malicious actors and assess their prevalence across commodity and APT malware datasets. The purpose of this aspect of our research was to facilitate an understanding of the way modern evasive malware operates and thereby facilitate more effective detection and prevention.

We take particular care in this thesis to assess the use of evasion techniques by Advanced Persistent Threat (APT) malware. We also examine several case studies of highly evasive variants. One of these is closed-source supply chain attacks, wherein we developed several detection mechanisms to detect this highly evasive real-world attack vector.

We endeavour to contribute to the detection of these evasive malware variants. We developed and tested a detection system for closed-source supply chain attacks, making certain elements of it open-source and make other aspects of our research open source to assist with improving existing detection mechanisms.

# Contents


**1 Introduction** **2**

1.1 Contributions . . . 3
1.2 List of publications . . . 4
1.3 Other Papers Published During Dphil . . . 5
1.4 Ethics . . . 5

**2 Antivirus Systems & Evasive Malware** **6**

2.1 Definitions . . . 6
2.2 Malware Analysis . . . 7
2.2.1 AV Overview . . . 8
2.2.2 Malware Measurement . . . 8
2.2.3 Static Malware Analysis . . . 9
2.2.4 Dynamic and Sandboxed Malware Analysis . . . 9
2.2.5 APT Malware . . . 9
2.2.6 Malware Network Traffic Analysis . . . 10
2.2.7 Windows Malware . . . 11
2.2.8 Linux & IoT Malware . . . 11
2.3 AV Evasion . . . 11
2.3.1 Code Signing . . . 12
2.3.2 Attacks on AV . . . 12
2.3.3 Static Analysis Evasion Techniques . . . 13
2.3.4 Dynamic Analysis Evasion Techniques . . . 13
2.3.4.1 Sandbox Fingerprinting . . . 14
2.3.4.2 Sandbox & Anti-VM Evasion . . . 14
2.3.5 Living-Off-The-Land Techniques . . . 15
2.3.6 Evasion via Adversarial ML . . . 15
2.4 Software Supply Chain Security . . . 16

2.4.1 Open-Source Supply Chain Attacks . . . 16
2.5 Gaps in Existing Research . . . 17
2.5.1 Systematic Analysis of Evasion Techniques . . . 17
2.5.2 Closed-Source Supply Chain Attacks . . . 17
2.5.3 Summary . . . 17

**3 Overall Methodology** **19**
3.1 Introduction . . . 19
3.2 Motivation . . . 20
3.2.1 Increase in Volume and Sophistication . . . 20
3.2.2 Detection Gap . . . 21
3.2.3 Lack of Scientific and Academic Studies . . . 22
3.2.4 Evolution of Malware . . . 22
3.3 Overall Aim . . . 24
3.4 Specific Aims . . . 24
3.5 Research Questions . . . 25
3.6 Research Design . . . 26
3.6.1 Study 1: Anti-Virus Evasion Techniques . . . 26
3.6.2 Study 2: Living Off The Land Techniques . . . 27
3.6.3 Study 3: Closed-Source Software Supply Chains . . . 28

**4 Anti-Virus Evasion Techniques** **29**
4.1 Introduction . . . 30
4.1.1 Contributions . . . 31
4.1.2 Scope . . . 32
4.2 Background . . . 32
4.3 AV Attack Classes . . . 34
4.3.1 Signature Replacement . . . 34
4.3.2 Packing and Static Analysis Evasion . . . 35
4.3.3 `.dll` Sideloading . . . 35
4.3.4 Discounted Attack Vectors . . . 37
4.4 Experimental Setup for AV Evaluation . . . 37
4.4.1 APT Malware Dataset . . . 38
4.4.2 Ten Most-Used AVs . . . 39
4.4.3 Evaluating Signature Replacement . . . 39
4.4.4 PE File Modification . . . 40
4.4.5 Evaluating `.dll` Sideloading . . . 42

4.5 Results . . . 43
4.5.1 Signature Replacement . . . 44
4.5.2 PE File Modification . . . 45
4.5.3 Increase in Detection Rate . . . 46
4.5.4 Ten Most-Used AVs . . . 46
4.5.5 `.dll` Sideloading . . . 47
4.6 Discussion . . . 48
4.6.1 Permissions . . . 49
4.6.2 Distributed System Elements . . . 49
4.6.3 Authenticode . . . 50
4.6.4 Static Code Analysis . . . 50
4.7 Suggested AV Best Practice . . . 51
4.7.1 Authenticode Signature Validation . . . 51
4.7.2 Miscalibrated Heuristics . . . 51
4.7.3 Scope Creep Prevention . . . 52
4.7.4 Limitations . . . 52
4.7.5 Future Work . . . 53
4.8 Related Work . . . 53
4.8.1 Static Analysis . . . 54
4.8.2 Dynamic Analysis . . . 55
4.8.3 Evasion Techniques . . . 55
4.8.4 Evasion Technique Detection . . . 55
4.8.5 APT Malware Detection . . . 56
4.8.6 Malware Measurement Studies . . . 56
4.8.7 Packing . . . 57
4.9 Conclusion . . . 57

**5 Living Off The Land Techniques 58**
5.1 Introduction . . . 59
5.2 Background & Related Work . . . 64
5.2.1 LotL Binaries . . . 64
5.2.2 Scope of our Study . . . 66
5.2.3 Related Work . . . 66
5.3 Motivation: Anti-virus products vs. Living-off-the-Land Techniques . 68
5.3.1 Windows 10 System Preparation for AV Experimental Setup . 71
5.3.2 LotL binary based Bypass Techniques . . . 72

5.4 Measuring LotL Prevalence . . . 74
5.4.1 Dataset Composition . . . 74
5.4.2 Dataset Characterisation . . . 76
5.4.3 Dataset Quality . . . 78
5.4.4 Analysis Pipeline . . . 79
5.4.5 LotL Technique Identification . . . 80
5.4.6 Parameter Analysis to Identify Execution Purpose . . . 82
5.5 Measurement Results . . . 83
5.5.1 Prevalence of LotL Techniques in Commodity Malware . . . 84
5.5.2 Comparison of Benign and Malicious Samples . . . 88
5.5.3 Prevalence of LotL techniques in APT Malware . . . 91
5.6 Case Studies . . . 93
5.7 Main Takeaways and Discussion . . . 96
5.8 Limitations & Future Work . . . 98
5.9 Conclusions . . . 101

**6 Closed-Source Software Supply Chains 102**
6.1 Introduction . . . 103
6.2 Related Work . . . 107
6.3 Challenges . . . 108
6.3.1 System Requirements . . . 109
6.4 Exorcist Design . . . 109
6.4.1 Threat Model and Scope . . . 110
6.4.2 System Overview . . . 110
6.5 Implementation . . . 111
6.5.1 Automated Static Analysis . . . 112
6.5.2 Dynamic Analysis . . . 115
6.5.3 Weighted Heuristic . . . 116
6.6 Case Studies . . . 118
6.6.1 SolarWinds . . . 118
6.6.2 NotPetya . . . 119
6.6.3 CCleaner . . . 119
6.6.4 DoFoil . . . 120
6.6.5 Dragonfly Campaign . . . 120
6.6.6 DarkSide Nullsoft Trojan . . . 123
6.6.7 Chiacoin . . . 123

6.7 Evaluation . . . 123
6.7.1 Dynamic Analysis Indicators . . . 125
6.7.2 Static Analysis Indicators . . . 125
6.7.3 False Positive Testing . . . 128
6.8 Versions . . . 129
6.9 Entropy Analysis . . . 129
6.10 Discussion . . . 131
6.10.1 Insights . . . 131
6.10.2 Limitations . . . 134
6.10.3 Future Work . . . 135
6.11 Conclusion . . . 136

**7 Conclusions** **137**
7.1 Summary . . . 137
7.2 Fulfilment of Initial Aims . . . 138
7.2.1 Fulfilment of Overall Aims . . . 138
7.2.2 Fulfilment of Specific Aims . . . 138
7.3 Fulfilment of Research Questions . . . 140
7.4 Practical Impact of Research . . . 142
7.5 Limitations and Future Work . . . 143
7.5.1 Limitations . . . 143
7.5.2 Future Areas of Research . . . 143
7.6 Final Remarks . . . 145

**Bibliography** **147**

# 1 Introduction

There has been a significant increase in the number of malware samples from what Schneier in 2000 said was alarmist to call 6 a day [1], to what Ugarte-Pedrero et al. stated in 2019 to be an average of over 100,000 new samples daily [2].

Cohen, in his 1984 doctoral thesis on the subject of malware, widely considered to be the first serious academic work on the subject of malware, determined that identifying whether a piece of software is malicious is undecidable [3], by comparison to Turing's proof regarding the 'Halting Problem' [4]. The as yet unrefuted implication of Cohen's thesis is that malware detection cannot be completely solved.

Despite this, it is still important to investigate the nature of malicious software. Research should be performed to systematically analyse the characteristics and behaviour of malware to facilitate the detection of this software. It is also important to conduct research to assist with the development of detection algorithms, to prevent the proliferation of malicious software and mitigate its impact. This is the larger goal of the research conducted within this thesis, that we explore in more detail in Chapter 3.

Malware research is now a mature field, with $3.5 billion predicted revenue in 2024 for AV software, proving the value of AV and the study of malicious software. The actors creating this malicious software are also now more organised. Measuring recent trends

in malicious software and the ways in which malicious software evades detection is an area worth researching.

The structure of the thesis is modular and interlinked. In Chapter 2, we identify gaps in existing research and summarise the state-of-the-art in existing research. In Chapter 3, we build upon these identified gaps and provide additional motivations for our chosen area of research and the aims and methodologies of this research.

After we describe the background, related work and overall methodology of our thesis, the core of our thesis is composed of three main studies. The first of these studies, in Chapter 4, is our research on the susceptibility of AV to APT evasion techniques that acts as a motivating chapter. The second of these studies is in Chapter 5, consisting of the research on LotL techniques observes broader trends within a large dataset as well as again conducting practical evasion testing against AV systems. The third of these studies, in Chapter 6, observes the highly evasive malware variants involved in closed-source supply chain attacks, whilst simultaneously aiming to build detection methods for these attacks.

In the final chapter in the thesis, Chapter 7, we conclude and retrospectively analyse the holistic contributions of our thesis with reference to the aims, research questions and methods described in Chapter 3.

## 1.1 Contributions

The main contributions of our research are to enable the more efficient static and behavioural analysis of modern malware and how precisely it evades AV systems. We also catalogue how this software makes use of sophisticated evasion techniques.

This is to enable the development of better detection systems, which we contribute to by making elements of our research open-source. This research has also involved collaboration with security researchers from multiple different organisations, to facilitate the effective dissemination of this research.

Specifically, our contributions are the following:

- **Measured the effectiveness of several evasion techniques against real-world AV systems.** In Chapter 4, we analysed three specific evasion techniques and their effects on real-world AV. These are Authenticode signature replacement, packing and `.dll` sideloading. We measured and catalogued how effective these techniques are.

- **Analyse and catalogue the evasive potential and prevalence of LotL techniques.** In Chapter 5, we looked into one specific class of techniques, that being of LotL techniques. In this study, we measured and documented a malware evasion technique in the wild. We practically measured how effective these techniques are against locally deployed AV to motivate the large-scale experiment. We then perform a large-scale analysis of how prevalent this technique is across commodity malware datasets.
- **Propose and implement a system to detect closed-source supply chain attacks.** In Chapter 6, we document our investigation into the security of closed-source supply chains, particularly the highly evasive malware variants that conduct these attacks. We designed and developed a system to detect malicious functionality being inserted in the software supply chain. This was then tested on several case studies of note.

## 1.2 List of publications

Six papers were produced during the course of this doctorate. This thesis comprises elements from three papers. The following list details the aforementioned publications:

– Frederick Barr-Smith, Xabier Ugarte-Pedrero, Mariano Graziano, Riccardo Spolaor and Ivan Martinovic
**Survivalism: Systematic Analysis of Windows Malware Living-Off-The-Land**
IEEE Symposium on Security and Privacy. 2021

– Frederick Barr-Smith, Tim Blazytko, Richard Baker and Ivan Martinovic
**Exorcist: Automated Differential Analysis to Detect Compromises in Closed-Source Software Supply Chains**
ACM CCS Software Supply Chain Offensive Research and Ecosystem Defenses. 2022

– Frederick Barr-Smith, Sebastian Kohler, Richard Baker and Ivan Martinovic
**APTful Dodger: Measuring Susceptibility of Antivirus to APT Evasion Techniques**
USENIX WOOT Conference on Offensive Technologies. 2024
*(Under Submission)*

## 1.3 Other Papers Published During Dphil

Whilst the following paper is not included in the thesis, this was also produced during the course of the doctorate and was part of a 'Digital Crime Scene' project.

– Frederick Barr-Smith, Thomas Farrant, Benjamin Leonard-Lagarde, Danny Rigby, Sash Rigby and Frederick Sibley-Calder
**Dead Man's Switch: Forensic Autopsy of the Nintendo Switch**
DFRWS EU 2021

This is another paper published during the course of the DPhil, not eventually entered into the thesis, on the subject of Tor cybercrime.

– Frederick Barr-Smith and Joss Wright
**Phishing With A Darknet: Imitation of Onion Services**
APWG Symposium on Electronic Crime Research (eCrime) 2021

The following paper was accepted to AsiaCCS 2021. The subject of this paper was the analysis of network flows using machine learning algorithms over a dataset of 996,712,027 network flows composed of several different datasets. These algorithms were used to detect and classify the nature and characteristics of the malware samples from whose execution these flows were extracted.

– Michal Piskozub, Fabio De Gaspari, Frederick Barr-Smith, Luigi Mancini and Ivan Martinovic
**MalPhase: Fine-Grained Malware Detection Using Network Flow Data**
ACM Asia Conference on Computer and Communications Security. 2021

## 1.4 Ethics

All research conducted as part of this thesis has gone through the Computer Science Departmental Research Ethics Committee, Auxiliary plans for all research were formulated, in the event of aspects of the research not being approved.

We have also been fastidious regarding the ethical requirements of vulnerability research. During the course of the research in this thesis, we discovered a number of vulnerabilities in AV software. We actively responsibly disclosed these vulnerabilities to AV vendors. We then proceeded to work with AV vendors to resolve these issues.

# 2

# Antivirus Systems & Evasive Malware

Before conducting a more granular analysis of various aspects of malicious software and AV evasion, we acknowledge the key elements of research that preceded this thesis and summarise the relevant existing literature.

From this aggregation of all relevant existing research, we also identify where there are gaps and areas where novel and impactful research can be conducted. These gaps in research provided sufficient motivation for us to research AV evasion techniques in a systematic way in addition to developing a solution applicable to the rapidly growing area of software supply chain attacks.

This chapter is intended to give the reader sufficient information to understand the contents of the later chapters. In addition to this, it provides context for our studies of evasive malware and its understanding and detection.

The works that are included in later chapters provide a more in-depth review of related work relevant to each branch of research. We also summarise important related work.

## 2.1 Definitions

This subsection provides definitions for key acronyms used throughout this thesis. These acronyms are essential for understanding the concepts discussed in the sub-

sequent chapters. Their inclusion is intended to provide quick reference and aid in comprehending the specialised language intrinsic to the field of cybersecurity and malware analysis.

In the course of this thesis on modern evasive malware and AV technologies, various specialised acronyms are frequently used for efficiency and precision. This section lists and defines the key acronyms. Each acronym is briefly explained.

| | |
|---|---|
| **API** | Application Programming Interface: A set of protocols for building software and applications. |
| **APT** | Advanced Persistent Threat: Stealthy, targeted and well resourced cyberattacks. |
| **AV** | Antivirus: Software for malware detection and removal. |
| **DLL** | Dynamic Link Library: Shared libraries in Windows OS. |
| **EDR** | Endpoint Detection and Response: Security for monitoring cyber threats. More advanced version of AV for enterprise deployment. |
| **LotL** | Living-Off-The-Land: Using legitimate tools installed on systems for malicious activities. |
| **MITRE ATT&CK** | Framework categorising cyber attacker tactics and techniques. |
| **MSI** | Microsoft Installer: Windows software installation component. |
| **OS** | Operating System: Software managing hardware and system resources. |
| **PE** | Portable Executable: File format for Windows executables and libraries. |
| **RAT** | Remote Access Trojan: Malware for remote system control. |
| **TTP** | Tactics, Techniques, and Procedures: Methods used by cyber attackers. |
| **VM** | Virtual Machine: An emulation of a computer system for running multiple OS. |
| **VT** | VirusTotal: A cloud-based online service that analyses malicious binaries to identify their behaviour and characteristics. |
| **VPN** | Virtual Private Network: Secured network connection routed through external network. |

## 2.2 Malware Analysis

Malware research is now an established scientific field [5–7]. In this section, we provide the necessary background in malware and attempt to summarise areas of research relevant to the thesis.

### 2.2.1 AV Overview

Before analysing the ways in which malware evades AV, we define what an AV system is. An AV system is software that identifies whether a binary or system activity is malicious or benign. The definition of these systems has changed over time to arrive at what is currently being offered as a security solution by most vendors. These systems are typically composed of file ingestion, static analysis, dynamic analysis and some signature-based components.

Another aspect of AV which must be recognised is their transformation in the years since their inception. Early AV systems were solely on-premises programs, wherein the endpoint system would be configured and for the most part, just check signatures against a list of known malicious software. Later AV contained central configuration in these on-premises deployments. Now, many of these AV systems have central configuration modules that are sometimes cloud-based. This is also reflected within the consumer AV market, where there has also been a move to more hybrid systems.

Initial AV systems relied heavily on signatures, rather than a holistic approach. Polymorphism and packers [8], wherein malware binaries signatures are changed to evade these systems. This led to signature-based analysis being less important and causing dynamic analysis to be more widely adopted as an additional AV detection mechanism. The default is for modern AV systems to use both methods in combination with each other.

### 2.2.2 Malware Measurement

There are large-scale measurement studies of malware that exist, even branching out to specific types such as recently, cryptomining malware [9], not to be confused with cryptojacking malware, in-browser cryptojacking is a threat wherein malicious JavaScript on a site will mine cryptocurrencies on a client's browser [10, 11]. Other individual malware types are covered in the literature, including the landmark paper AVClass and its successor AVClass2, which enabled samples in malware datasets to be classified into different families [12]. The relatively recent trend of ransomware [13,14] and protection methods against ransomware [15] has been explored. These techniques range from Kharaz et al.'s [16, 17] deployment of behavioural analysis and file access pattern detection to Kolondeker et al.'s [18] interception of cryptographic API keys.

Research has also enumerated ways to ensure that the experimental results for malware experiments are reliable, correct and useful [19,20]. These papers are not simply abstract suggestions or complaints and their robust frameworks have been implemented in later research.

### 2.2.3 Static Malware Analysis

One of the core static analysis methods used by malware analysts is to reverse engineer the malware binaries. The trend in industry is towards the production of tools that facilitate this automated reverse engineering. This automated reverse engineering helps remove repetitive tasks within the reverse engineer's workflow. This is beneficial as the time and expertise of reverse engineers are expensive.

Deobfuscation of binaries is another area in which there has been a large amount of research [21,22]. One aspect of this is extraction of cryptographic primitives from the binary [23]. Another is that of symbolic or concolic execution. Symbolic execution motivated the obfuscation detection section of Chapter 6 as there are several novel techniques we use to identify changes in obfuscation between subsequent versions of given binaries.

### 2.2.4 Dynamic and Sandboxed Malware Analysis

Academic analysis of malware by detonation in sandboxes began with the Anubis platform. This led to a number of different papers that looked at the behaviour of malware [24–26] and latterly the establishment of a security company, LastLine. Zhu et al. looked at semantic extraction of malware behaviour from the security literature [27]. This addressed the issue of anti-VM and environment-aware malware [28].

Subsequently other platforms emerged such as Hybrid Analysis and VirusTotal(VT), that hosted Cuckoo sandbox and other such systems to detonate malware in. These enable analyses such as Graziano's [29], which analysed how APT groups leverage online malware sandbox platforms for malware development.

### 2.2.5 APT Malware

In this thesis we pay particular attention to the analysis of APT malware. APT malware is often highly evasive and many of our case studies are of APT malware samples and campaigns.

Specifically, Hassan et al. discuss the application of provenance to identify APT malware [30], focusing on mapping MITRE ATT&CK. There are a number of papers that deal with targeted attacks, conducted by nation state adversaries [31]. APT malware detection methods are also covered in the literature, with data provenance as one method [32, 33] and information flow tracking as another [34].

The spectre of APT attacks even galvanised the development of the influential intrusion killchain model [35]. This is part of the intrusion-detection model that has been universally adopted, following on from Denning's highly influential 1987 paper [36]. There have also been efforts in making APT malware detection more of a proactive rather than reactive approach, an example of this is the effort to detect APT malware at early stages by Mahmoud et al. [37].

A recent and highly successful innovative mutation of this model is what has been termed *Double Extortion* (DE). DE is an evolution in the operating model of ransomware, where after penetrating the network, attackers will exfiltrate data from the network and use this data for the purpose of extortion. Chainalysis have identified a 300% increase in ransomware attacks throughout the course of 2020, perhaps due to the increased rewards for perpetrators caused by DE [38].

### 2.2.6 Malware Network Traffic Analysis

Intrusion detection is a field that is now several decades old, from the aforementioned initial 1987 text by Denning, that proposed methods of analysis of logs of various provenance to identify malicious activity. This led to the development of Network Intrusion Detection Systems (NIDS) such as Bro (currently Zeek) and Snort that facilitate the detection of cyber attacks in computer networks.

This field has evolved from detection of intrusions to detection and classification of particular malware samples. Large-scale measurement studies have also been performed to evaluate the network behaviour of malware [39]. Other research has looked at network forensic artefacts of malware delivery mechanisms [40] and bulletproof hosting [41].

Malware network traffic analysis is one method by which malicious activity is identified and analysed. There are also a number of papers that conduct measurement studies of network traces of malware [39]. Behavioural clustering of malware is another research area in malware network traffic analysis [42, 43]. This has led to the ascension of a large industry in the analysis of network logs by security analysts,

to detect breaches, with many books and papers written on the subject of network forensics [5].

### 2.2.7 Windows Malware

The vast majority of malware is developed for the Windows platform, due to the fact that most desktop systems are Windows. This has led to a study by Galloro et al. that analyses the evasive activity of various Windows malware samples [44].

Windows malware is the most studied malware subtype in academic research and this thesis also mostly looks at malware samples targeting Windows hosts. Such a large amount of academic research targets Windows malware because it is the platform that receives the most attention from malware developers. Like all businesses, malware operations and therefore malware developers are attempting to maximise their profit and impact. As such, they target the OS with the largest market share, this being Windows.

### 2.2.8 Linux & IoT Malware

Although the amount of malware created for Linux systems is significantly less voluminous than those of its Windows counterparts. Cozzi et al. conducted a study of Linux malware and extracted several meaningful conclusions [45]. The amount of Linux malware that exists is only projected to rise.

Although IoT systems frequently use stripped-down Linux systems such as BusyBox, they are distinctly different from Linux systems. One of the most-cited papers in all top-tier security conferences analyses the Mirai botnet, a well-known IoT botnet [46]. Griffoen and Doerr measured another interesting development in Mirai, which is that of different Mirai variants and competing malware families, through a network of 7,500 honeypot systems. Herwig et al. measure the Hajime botnet, another large and notorious IoT botnet [47]. Dang et al. also looked at the possibility of deploying honeypots to analyse IoT malware [48]. There are also more generalised measurement studies on the subject of IoT malware, now that it has become an established type of malware, one such study is conducted by Alrawi et al. [49]

## 2.3 AV Evasion

As malware analysis becomes more sophisticated, there is a simultaneous increase in evasion techniques by malware authors looking to bypass these protection mecha-

nisms. This is an area of research where the state-of-the-art changes frequently due to the cat-and-mouse nature of the field. The velocity of these changes has even led to several papers exploring the proliferation of concept drift [50, 51]. Concept drift in the context of malware research is when malware detection algorithms are developed and tested on datasets which are no longer applicable as time progresses.

This is due to the constantly evolving nature of malware. These changing characteristics and evasion techniques mean that detection mechanisms become outdated. It is arguable that the basic principles of malicious software remain the same, however, implementations change over time. As such, there is still value in conducting research into malware analysis and also in how these analysis methods are thwarted.

### 2.3.1 Code Signing

Kim et al. [52] describe the abuse of the Public Key Infrastructure (PKI) offered by Microsoft as part of Authenticode, to enable highly evasive malware; Uroz et al. also investigate the forensic challenges that this abuse poses [53]. The abuse of malware that is signed by Authenticode has also been investigated by Kotzias et al. [54]. Recent research by McGuire et al. has investigated the validation of Authenticode signatures in memory forensics [55].

There is also investigation into software that is in a grey area taxonomically speaking, that is Potentially Unwanted Programs and their distribution infrastructure [40, 56]. This includes the software used in intimate partner violence [57].

### 2.3.2 Attacks on AV

It is crucial to recognise that there are several different attacks that have been performed on the underlying AV software itself. This is different from evasion as it means that the AV has been targeted, perhaps due to its high level of privileges. There is the publication of books on the subject of attacking AV itself, notably 'The Antivirus Hackers Handbook' by Koret [58]. Whilst the material contained within this book is now somewhat dated, the basic principles remain the same.

There is also the direct targeting of AV that has been recorded by industry whitepapers and blog posts. This is the attacks performed by threat groups, that leverage attacks on AV engines as part of their attack strategy. A recent example at the time of writing is that of Moshen Dragon, which used `.dll` sideloading for a Trend Micro [59] product. Similarly, Microsoft researchers uncovered a campaign that utilised

`.dll` sideloading as part of suite of tools used for enabling the evasion of detection technologies. This Microsoft research identified that there was the direct targeting of AV software itself [60].

### 2.3.3 Static Analysis Evasion Techniques

Static analysis is a key method used by AV to detect malicious software. It is therefore one of the areas which is targeted by actors attacking AV systems. We systematically conduct experiments to explore the effectiveness of various static analysis evasion techniques in Chapter 4. We also make use of several static analysis methods in Chapter 6. Various elements of nearly all modern detection systems use these static heuristics to determine whether or not a given sample is malicious.

The field of packer detection is not limited to the simplistic identification of entropy levels. Packer detection is a mature enough field to have lead to the taxonomisation of different packer subtypes as coducted by Ugare-Pedrero et al. [8]. There are also other large-scale studies conducted to measure and analyse the nature of packed malware [61].

### 2.3.4 Dynamic Analysis Evasion Techniques

Academic research regarding dynamic analysis evasion techniques has largely centred around the evasion of sandboxes. Dynamic analysis detection is largely concerned with the sandboxed execution of malware and also the recording of this behaviour. Part of the reason for the wealth of research in this area is that it is impossible to perfectly simulate a computer system without leaving some trace evidence of the virtualisation architecture underlying that system [62].

When this behaviour occurs, it manifests itself in the form of differing malware behaviour, where a malware sample may remain inert or benign if it detects it is operating in a virtualised environment. This has been defined as the 'split behaviour' of malware and is what malware detection systems seek to counteract [28].

An additional complication has been added to this, for both attackers and defenders. This complication is that there has been a rise in the amount of cloud-hosted systems. Whereas previously if a malware sample identified that it was executing in a virtualised environment, it was likely to be an analysis environment. This means that if malware authors choose to implement anti-VM functionality, may have to lose part of their target market - those that use cloud-based systems.

#### 2.3.4.1 Sandbox Fingerprinting

The fingerprinting of sandboxes is one of the most researched areas of dynamic analysis evasion. As with many elements of malware research, it is difficult to differentiate between the different elements of offensive and defensive research. Shi et al. represent this with the development of a tool that they call 'VM Cloak' [63]. This tool aims to hide the presence of a debugger by looking for the same indicators that evasive malware would. When it finds these indicators of anti-VM checks ('redpills'); it replaces these registry values and memory states with those that would be expected in a non-virtualised environment.

There has also been investigation into the use of delays in execution to cause sandboxes to not be able to reliably execute malware samples [64]. Perhaps the most detailed technical analysis was conducted by Kuchler et al., which used testing of over 100,000 samples to prove that the vast majority of samples were executed within 2-10 minutes [65]. These pauses are caused by the invocation of the Windows Sleep API or may be custom encoded routines, such as the subroutines we use for our packer in Chapter 4.

To counteract this wide range of dynamic analysis evasion techniques, various different technologies have been deployed to attempt to ensure that malware executes its behaviour as it would on a real system. One of these techniques is that of using 'bare metal', wherein malware is simply run in an environment that is for all intents and purposes a real-world system, it is the infrastructure that is not virtualised. Research also states bare-metal systems are the only way to ensure that anti-VM behaviour is not tainting results [28]. This approach is undertaken in multiple papers by Kirat et al. [66, 67]. This is not especially scalable but does allow for higher reliability with regard to malware execution.

#### 2.3.4.2 Sandbox & Anti-VM Evasion

There have been a number of papers dealing with the problem of evading malware sandboxes. *Spotless Sandboxes* and *Sandprint* [62, 68–70] gave practical suggestions for stealthy avoidance of analysis sandboxes by malware authors, via the identification of forensic artefacts created during normal system usage.

There are also papers and exploits that look at attacking the VM itself, facilitating a sandbox escape [71]. Whilst this does not mean that this is a common attack vector, as sandbox escape exploits are quite expensive and therefore not likely to be used

unless the reward to be gained by the malware author for deploying these exploits is significant [72].

Environment-sensitive malware is a common feature in malware, There has been research into this characteristic of malware by Lindorfer et al. [73]. Sandboxed environments also attempt to obfuscate indicators of virtualisation [63], to enable the dissection of evasive malware.

### 2.3.5 Living-Off-The-Land Techniques

LotL techniques are not robustly covered in the academic literature. This is the gap in novelty which inspired our research, which we document in Chapter 5. LotL techniques are the use of native system binaries by malware or human operators. Research by Pendergrass et al. [74] discusses the use of these binaries and enumerates some detection capabilities and also identification of LotL as a malware subtype that is "*often undetected*". A recent paper by Wang et al. [75] on the subject of data provenance analysis identifies LotL as an emerging prominent evasive malware subtype.

Some of the payloads that LotL malware uses have been thoroughly investigated by several papers. A number of papers deal with the delivery of malicious PowerShell payloads [17, 76]. JavaScript payloads are also studied [77], particularly those delivered as part of a drive-by download attack [78].

Dang et al. [48] and Cozzi et al. [45] provide analyses of fileless malware on Linux systems. There is a subtle distinction between fileless malware and LotL techniques, that fileless malware is resident in memory, leaving scarce or no traces on disk. This is in contrast to LotL techniques, wherein malicious payloads are loaded into memory by native Windows system binaries.

### 2.3.6 Evasion via Adversarial ML

One area of research orthogonal to that conducted in this research is worth mentioning. This is the evasion of AV systems via the use of adversarial ML. Suciu et al. perform initial forays and experiments in this area [79] as did Lucas et al. [80]. Kolosnaji et al. also perform a practical attack using adversarial machine learning [81].

Whilst not a technique that is widely used in commodity malware, there is a considerable amount of academic research. Pierazzi et al. theorise that this may be an attack vector which is more commonly adopted and experimentally demonstrated

the vulnerability of detection systems to attacks that utilised automated adversarial machine learning to facilitate evasion [82].

## 2.4 Software Supply Chain Security

Closed-source supply chains are an evasive malware variant that has been notably absent from the existing academic literature. Rectifying this absence of coverage within the current academic literature is a key aim of our thesis. In Chapter 6, we explore our analysis of the evasive malware variant of supply-chain attacks.

### 2.4.1 Open-Source Supply Chain Attacks

Open-source supply chains, in contrast, have been subject to slightly more analysis. This is due to a few reasons. Firstly, the source data for open source supply chains is more available, as they are by their very nature open. This includes the entirety of the software's version history and the requisite source code. In closed-source software supply-chains it is somewhat more difficult to extract the entire version history of the binary and significantly more complex to extract the source code of these binaries.

Due to the wide range of source data and impact of open-source supply chain attacks, there is a large amount of research on this. Another advantage of this area of research is that the work is easily replicated and implemented in real-world systems. Therefore this research can have a genuine impact in organisations, rather than simply remaining an abstract theoretical discipline.

Backstabber's knife collection is another paper that looked at the security of software supply chains, reviewing open source attacks in the field [83]. Similarly, Vu et al. also looked at what could be done to detect and prevent supply chain attacks in open-source software, in particular Python [84]. Duan et al. similarly conducted an analysis of open source repositories in PyPy, Rubygems and NPM [85].

Additionally, Boucher and Shumailov produced several works analysing a niche variant of supply chain attacks. Respective papers demonstrate attacks on machine learning and code development systems that rely on correctly parsed, but differently visualised unicode [86].

## 2.5 Gaps in Existing Research

There are gaps across several axes in the academic literature relevant to the work conducted in this thesis. In areas that are sparsely researched there is fertile ground for additional work to be conducted. It is those gaps which have motivated the research which has been conducted in this thesis. In Chapter 3, we go into more detail concerning how these research gaps motivate the research that constitutes this thesis and the methodology we use to achieve this.

### 2.5.1 Systematic Analysis of Evasion Techniques

One area in which there is a regrettable lack of research is the practical investigation of the applicability of evasion techniques against modern AV. There are studies in individual evasion techniques, such as longitudinal analysis of packers over time and other dynamic evasion techniques that we list in Section 2.3. The gap that our research in Chapter 4 fills is that of a lack of comprehensive and comparative study of the effects of AV evasion techniques against commercial AV.

### 2.5.2 Closed-Source Supply Chain Attacks

As we identify in the previous section, at the time of formulating and publishing our research into supply chains, covered in Chapter 6, there was a lack of published research into closed-source supply chain attacks.

### 2.5.3 Summary

From analysing the current related work, we can conclude that the analysis of malware and the techniques malware uses to evade this analysis, are an area that has become impactful and has been subject to insufficient prior academic research. The body of literature reviewed reveals a broad spectrum of methodologies and approaches, each contributing uniquely to the understanding and mitigation of these sophisticated malware variants. Despite the large amount of knowledge and the variety of strategies employed, gaps remain evident, particularly regarding real-world applications and data, and taxonimisation and prevention of the rapidly evolving techniques of malware authors.

The insights gleaned from the analysis of our background and related work inform the development of our research questions and methodologies. The context provided by the related work enumerates the relevant research preceding this thesis, and motivates

our aim of seeking to address the identified gaps and contribute novel perspectives and solutions to the field.

# 3

# Overall Methodology

## 3.1 Introduction

This chapter describes the systematic methods by which we explore the complexities and challenges in the field of evasive malware. Building upon the related work and background discussed in Chapter 2, this section delves deeper into the motivations behind our research. It examines the deficiencies in existing research and how emerging trends both in commercial and academic realms shape our investigative direction. Specifically, we scrutinise how the dynamics of evasive malware motivate a more nuanced and effective approach for malware detection and analysis in the commercial sector, alongside academic research. This exploration lays the foundation for our research questions and also sets the context for the methodologies we adopt to address these questions.

We discuss how the lack of existing research in certain areas motivates our research. We explore how emerging trends in commercial research of evasive malware and the academic research of evasive malware motivate our research.

In this chapter, we specify the research questions we are trying to answer in the thesis. These research questions are clearly drawn from the motivation. These research questions also emerge from the identified gaps and the emerging needs highlighted in the previous chapters. They are carefully crafted to address the areas where current

research is lacking, particularly in the realm of evasive malware detection and analysis. The questions arise not only from the academic perspective but also take into account the practical challenges and trends observed in commercial malware research. This approach ensures that our research is both theoretically sound and practically relevant.

These research questions then map to the experimental methods. We describe the design of our research and the methods by which it is achieved, but not to the level of detail that we do in the subsequent chapters. We also describe how these experiments, measurements and tools facilitate the fulfilment of our research aims. This approach links our research objectives and the methodologies we employ, ensuring that each experiment is purposefully designed to answer key questions and contribute to the broader understanding of evasive malware.

## 3.2 Motivation

There is a long history of academic research into malware and yet it remains an ongoing arms race. The reason we are conducting this programme of research is because of the lack of coverage in the academic literature of recent evasive and sophisticated techniques successfully used by malicious software campaigns.

### 3.2.1 Increase in Volume and Sophistication

We have detailed an increase in volume of commodity malware in the previous chapter, there has been simultaneously been a rise in the amount of APT malware. Whilst not as voluminous as commodity malware, APT malware is typically notable for its technical sophistication. What differentiates APT malware from commodity malware is that it is technically advanced and persistent. This advanced nature means APT malware typically uses more innovative and cutting-edge evasion techniques to remain hidden from AV and accomplish its objectives. The actors with the resources to conduct these attacks are also results-driven, so if simpler methods work then they will use them. A relevant example of this is 'Understanding the Mirai Botnet', which analyses Mirai, one of the biggest IoT botnets of all time. Mirai spread via exploiting insecure or weak credentials being used on IoT devices [46].

There has been a large amount of research to detect malware, which we describe in more detail in our literature review in Chapter 2. This is not an abstract academic

theoretical discipline, but a field that has an accompanying multi-billion dollar commercial threat intelligence and AV industry accompanying it. This means that there are large investments by commercial and government entities to match.

Initial signature-based virus detection mechanisms that are used by AV have become obsolete. AV engines have started using dynamic analysis in addition to signature-based analysis. We look at the dynamic analysis of malware and dynamic analysis evasion techniques in depth.

### 3.2.2 Detection Gap

It is clear that there is a great incentive for malware authors to evade AV. As such, malware authors have endeavoured to develop techniques to achieve this goal. The breadth and complexity of this malicious innovation can outstrip the development of detection mechanisms by AV authors. A large part of the contributions of this thesis is to rationally and scientifically document the techniques used to evade AV but also to prove that there are flaws in the detection mechanisms of AV that should be rectified.

In Chapter 4 and Chapter 5, we investigate this detection gap in far greater detail. However, this thesis does not aim to simply identify that AV engines are a defence mechanism that can be bypassed, although we robustly identify this. The identification and calculation of the precise nature of the detection gap advanced evasion techniques create with AV software are conducted at several points within these chapters. The presence of this detection gap is used to motivate other elements of our research. These other elements of research are the analysis of the utilisation of these techniques over large datasets. Similarly, we also analysed the use of these techniques by APT groups and built upon existing research in this area.

Large amounts of AV evasion and advanced malware variant research are contained within what is known as the 'grey literature'. This is knowledge is contained in forms such as forum posts, industry whitepapers or simply in the unwritten knowledge of red team operators and malware authors. We aim to codify elements of this knowledge into the academic literature.

Wagner et al. [87] discuss the value of importation of 'grey literature' into academic work the context of threat intelligence sharing. Similarly Cascavilla et al. [88] describe the importance of 'grey literature' in threat intelligence, even going as far as to taxonomise the production of 'grey literature'.

### 3.2.3 Lack of Scientific and Academic Studies

The other element of the current state of research that this thesis aimed to improve was principally related to the lack of scientific investigations into this field. An example of this is a 2014 BlackHat presentation that was given on the subject of AV evasion [89]. This presentation and associated research notably stated that it was not academic in nature.

There are more recent efforts that have been conducted to codify elements of this knowledge into academic literature, such as that by Geng et al. [90], showing that packing and code obfuscation remain the most-used PE evasion techniques. A motivating factor of our conduction of this research is the provision of a scientific element to the existing body of knowledge.

### 3.2.4 Evolution of Malware

The nature of malware has changed, it is not the case that malware takes the form of a singular binary propagating between systems, such as in the historic case of the SQL Slammer worm. Modern malware, particularly advanced variants are likely to be composed of modular components and potentially multiple stagers, taking the form of scripts, `.dll`'s and executables.

As malware detection algorithms and solutions become more and more sophisticated, the evasive techniques deployed by malware authors have also become more advanced. More long-standing evasion techniques such as the use of rootkits (to burrow deeper into the system) and process injection are well-known [91, 92]. Some of the most significant recent evasion techniques used by malware are LotL techniques, the evasion of dynamic analysis sandboxes and the use of onion services to obfuscate operations in progress.

In terms of the ability to evade AV, there has been a rapid change and metamorphosis in the evasive techniques used by different malware campaigns. These techniques have matched the migration of software systems being traditionally on-premises to cloud-based. Malware authors have taken advantage of these changes and have adopted new techniques to evade detection.

Despite these changes, some of the techniques and strategies used by malware authors remain the same. When writing malware, developers are pragmatists, if legacy techniques work but are not particularly advanced, these techniques will be used. We also explore this area in Chapter 4, where we systematically test some known and

legacy evasion techniques and prove they are still capable of evading modern analysis systems.

In this thesis, we systematically measure how evasive these techniques are versus real detection systems and also how widespread the use of these techniques is in the wild, based on analysis of large malware datasets.

There are several specific evolutions in AV evasion that have motivated the selection of the areas for investigation in this thesis:

**`DLL` Sideloading** `.dll` sideloading is a method frequently used by APT groups. The usage of this technique by threat actors has intensified in recent years. Commodity malware groups also make use of this technique and there are open-source penetration testing tools that enable scanning of target systems for vulnerabilities to this. `.dll` sideloading is an addition to the evasive capabilities of a campaign rather than a technique in isolation, by which we mean the binary itself still has to be obfuscated. We observed that there was a recent rise in the use of these techniques by threat groups. In addition to this, the use of these techniques by threat groups targets the AV directly, taking advantage of the high privilege and trust levels that AV is accorded on systems [93].

**Living-Off-The-Land** There also exists a detection gap between current detection and measurement technologies and the behaviour of malicious software. LotL techniques are mentioned in papers, referenced in the preceding Chapter, emphasising stealthiness and that it is a technique used by APT malware. We also practically test and prove the presence of this detection gap regarding LotL techniques.

**Prominence of Ransomware** The increased scale, financial motivation and therefore evasive sophistication of today's ransomware groups have led to a rapid evolution in malware and the requisite detection capabilities of AV. Similarly, as the cost to companies of not detecting these intrusions increases, there is a commercial and industrial demand for more sophisticated detection mechanisms.

As a consequence of this increased prominence of ransomware, throughout this thesis, we make use of ransomware case studies. One of our key case studies is ransomware in Chapter 5 and several of the case studies under analysis are ransomware in Chapter 6. We selected case studies on the criteria of some of the most impactful and prominent attacks, of which many of the most noteworthy were ransomware.

**Proliferation of Dynamic Analysis Sandboxes and Antivirus** Many papers that analyse the output of dynamic analysis sandboxes as their source data are covered within the literature review. Dynamic analysis sandboxes are widely used and it is important to understand their limitations.

Many commercial threat intelligence firms' first step is to '*Check VirusTotal*'. Flaws in dynamic analysis systems such as VT can then be considered to be significant and an area worthy of investigation. Zhu et al. published a meta-analysis of 15 academic papers that use VT [94]. This paper came to the conclusion that VT and its constituent AV engines may in fact misclassify the maliciousness of many malware samples. Another paper also identifies flaws in VT's classification engine [95].

## 3.3 Overall Aim

The primary objective of this thesis is to bridge the detection gap in contemporary AV technologies against modern evasive malware and to also contribute to the understanding of malware, particularly highly evasive variants. This involves a comprehensive systematic analysis of the advanced evasion techniques used by such malware, with a particular focus on those employed by APT groups and their differentiation from commodity malware.

## 3.4 Specific Aims

The overarching aim described in the previous section maps to several more granular aims. These more granular aims are designed to dissect and address the multifaceted challenges posed by modern evasive malware. By breaking down the overarching goal into these specific objectives, it is then possible to distil research questions that fulfil these specific aims. These specific aims are as follows:

**Understanding Evasive Techniques** To systematically study and categorise the various evasion techniques utilised by modern malware, particularly focusing on `.dll` sideloading, AV signature manipulation, packing, supply chain attacks and LotL techniques. This detailed examination aims to highlight the intricacies and effectiveness of these methods in evading current AV solutions.

**Evaluation of Evasion Effectiveness** To systematically measure the effectiveness of these evasion techniques against real-world AV systems. This includes a thorough analysis of the impact on AV detection rates of two specific evasion methods—Authenticode signature replacement and packing.

**Development of Detection Mechanisms** To contribute to the field by developing and testing new detection systems, particularly for closed-source supply chain attacks. This aim addresses the need for enhanced detection capabilities in light of the sophisticated evasion techniques identified and others that are known about.

**APT Malware Analysis** To conduct a focused study on APT malware, differentiating it from commodity malware in terms of technical sophistication and persistence. This tranche of research intends to provide insights into the innovative evasion methods used by APT malware and their operational characteristics.

## 3.5 Research Questions

The research questions that emerge from these overarching and specific aims are formulated to address the multifaceted nature of evasive malware and its detection. These questions are used to guide the research trajectory and to ensure that the findings are relevant, actionable, and contribute significantly to the field of malware detection and analysis. The specificity of these questions allows for a focused investigation, ensuring that each aspect of evasive malware is thoroughly explored and understood. The following research questions are the core questions that guide the research in this thesis:

> *RQ 1: How do modern real-world evasive malware techniques operate, and what are their key characteristics?*
>
> *RQ 2: What is the effectiveness of current evasion techniques as measured against real-world AV systems?*
>
> *RQ 3: What novel detection mechanisms and systems can be developed to effectively counteract the evolving landscape of evasive malware, particularly in supply chain attacks?*

## 3.6 Research Design

Each study employs a robust and tailored experimental method to address the overarching research questions and the aims specified in this chapter. In the subsequent chapters there are more granular research questions that pertain to each study. These experimental methods are designed to fulfil the overall and specific aims of the thesis. The experimental design for each is as follows:

### 3.6.1 Study 1: Anti-Virus Evasion Techniques

This study acts as a motivating chapter for the further two studies. It is a systematic exploration of the detection gap between modern commercial AV systems and the techniques used by these systems. This helps fulfil the aims and research questions pertaining to understanding the characteristics of evasive malware and the gaps in detection with regards to dealing with advanced evasion methods.

**Objective** To explore the effectiveness of AV systems against APT evasion techniques. To evaluate the effectiveness of detection of these techniques deployed upon a real-world APT malware dataset.

**Data Collection** We use the 'VX Underground' dataset of 21,477 APT malware hashes. For which we download the original binaries and subsequently programmatically alter the corresponding binaries.

**Method** We develop and execute a series of test scenarios simulating various attack vectors commonly used by APT groups, incorporating known evasion techniques such as obfuscation, polymorphism, and signature alteration.

We test cloud-based AV solutions to evaluate their detection rates against our APT malware binary dataset. We first measure the detection rates of the unaltered malware binaries. Subsequently, we then systematically alter the malware binaries to facilitate the evasion methodologies. Following on from this we measure the detection rates of the altered binaries. We then compare and contrast the detection rates.

**Vulnerability Research** In addition to the quantitative measurement and comparison of evasion techniques, as part of our research we discover and responsibly disclose `.dll` sideloading vulnerabilities to several AV vendors.

**Evaluation Metrics** Develop and apply quantitative metrics to assess the performance of AV systems against these advanced threats. Compare and contrast the different detection rates by AV and by evasion method. Measured the detection rates of each AV system against the simulated APT attacks, noting which evasion techniques were most effective in bypassing the AV defences.

### 3.6.2 Study 2: Living Off The Land Techniques

The preceding chapter illustrates different ways in which the detection gap occurs across malware datasets, in addition to a deeper exploration of several evasion techniques. Developing upon this, the second study in this thesis explores only one evasive technique, that being LotL techniques.

**Objective** To examine how malware utilises LotL techniques in Windows malware. Additionally, as a motivating factor and to fulfil the research questions and aims to assist with understanding the nature of evasive malware. It also fulfils the aim of measuring the detection gap between current AV technologies and the evasion techniques used by malware.

**Vulnerability Research** To motivate the remainder of this study, initial evasion experiments are performed. These are performed on a local VM, wherein the 10 leading AVs are tested for their susceptibility to LotL techniques as used for evasion.

**Data Collection** Sourced extensive malware datasets. For which we download and alter the corresponding binaries. This is a total of 44,606,904 samples from which we obtained 22,396,841 behavioural reports from VT.

**Methodology** Utilising a dataset of known Windows malware, conducting dynamic analysis to identify and analyse the use of native system binaries for malicious purposes. Executed the malware in a controlled, isolated environment to observe its runtime behaviour. This involved monitoring executed commands, file activities, and other behaviour to detect the execution of LotL techniques for evasion.

**Data Analysis** Analyse the data to identify common LotL techniques and their evasion strategies. Measure and analyse their prevalence at a large scale. Assess how these techniques challenge traditional detection systems.

### 3.6.3 Study 3: Closed-Source Software Supply Chains

The final constituent study of this thesis combines elements from the preceding studies into a novel domain, that of the analysis and detection of closed-source supply chain attacks. The development of a detection mechanism for supply chain attacks involves detecting attacks that make use of methods investigated in our previous studies, such as LotL, binary obfuscation and signature replacement.

**Objective** The development of a system to detect compromises in closed-source software supply chains. In the process of development and evaluation this fulfils the understanding of evasive malware variants, that of trojanised binaries inserted as part of supply chain compromise. This chapter also contains the development of a system, in comparison to the measurement-based earlier chapters.

**Data Collection** For this research, we collected 12 binaries that were deployed during closed-source supply chain attacks. We also acquired the software versions before the insertion of malicious functionality and the versions after the insertion of malicious functionality.

**Methodology** Develop and employ a tool, `Exorcist`, for automated differential analysis, comparing successive versions of closed-source software to detect anomalies indicative of compromises.

**Case Studies and Evaluation** Apply these tools to real-world software examples, identifying insertion of malicious trojanised capabilities by APT groups and exploring patterns of compromise in closed-source software supply chains.

# 4

# Anti-Virus Evasion Techniques

In this chapter, we review some of the key aspects of the evasion of AV systems. The content predominantly draws from a detailed paper on AV evasion, with a specific focus on analysing a dataset of APT malware. This analysis assesses the effectiveness of various evasion techniques employed by sophisticated actors to obscure the malicious nature of their software.

The purpose of the research on the vulnerability of modern cloud-based dynamic analysis sandbox systems to evasive malware is to systematically analyse how advanced malware can bypass these security measures using real-world samples. This involves a detailed examination of various obfuscation and evasion techniques commonly used by real-world employed by malware in cloud-hosted sandboxes, assessing the effectiveness of current sandbox detection systems. The study aims to identify weaknesses in these systems and contribute to enhancing their capability to detect and mitigate sophisticated malware threats, thereby improving the security of cloud analysis platforms.

To a certain extent, this chapter functions as a motivating chapter for the rest of the thesis. In that some of the key conclusions of the research contained within it are the overall vulnerability of AV to robust attack by skilled actors.

In this chapter, we perform two sets of experiments, to investigate various evasion techniques. We also perform vulnerability research within AV systems them-

selves.

In the first set of these experiments, we analyse the vulnerability of AV systems to malware using forged Authenticode signatures. This attack takes advantage of heuristics in AV that check but do not validate Authenticode signatures. In our second set of experiments, we analyse the effectiveness of packers in masking the malicious nature of our APT malware dataset.

We also look for exploitable flaws within the AV itself. In particular, we identified several instances of `.dll` hijacking. This is an attack that targets the underlying AV itself, rather than an attack that works by masking the maliciousness of malware binaries. This form of attack is notably prevalent in APT campaigns and provides crucial insights into the vulnerabilities inherent within AV systems.

This comprehensive exploration of AV evasion techniques and vulnerabilities not only addresses specific research queries but also significantly contributes to the thesis's overall aim. By uncovering the limitations of current AV solutions and probing the effectiveness of advanced evasion strategies, this chapter lays the groundwork for developing more robust and adaptive malware detection.

## 4.1 Introduction

AV programs are a multi-billion dollar industry. In parallel with the innovation in this industry, malware authors develop increasingly sophisticated evasion tactics. In particular, APT malware is notorious for using sophisticated and highly evasive techniques to bypass the detection mechanisms of hosts infected with these malware types.

This is not to say that evasion mechanisms purposefully use complex techniques; if simpler methods work and provide the same effectiveness, they are used. We go into greater detail regarding this in our earlier motivation in Section 3.2.1.

Since APT groups often aim to persist on systems for long periods of time, with a low-and-slow attack pattern, staying hidden is a high operational priority for these types of attackers. As such, evading sophisticated detection systems such as AV is a core goal of APT groups, regardless of the technique.

The increasing use of evasion techniques by malware led to various academic studies, that we have partially covered in Chapter 2. Kim et al. describe the growing

| APT Group | Techniques used | | | | | | Years Active |
|---|---|---|---|---|---|---|---|
| | `.dll` Sideloading | Encryption | LotL | Metadata | Network | Signature Replacement | |
| BackConfig | | ● | | ● | ● | ● | 2020 |
| Billbug | | ● | ● | | | ● | 2009-2022 |
| DEV-0569 | | | ● | | | ● | 2022 |
| Fin7 Mahalo | ● | ● | ● | | | ● | 2022 |
| Foggyweb | ● | ● | | | | | 2021 |
| HermeticWiper | | ● | ● | ● | | ● | 2022 |
| UNC3944 | | | | | | ● | 2022 |
| Winnti | ● | ● | | | | | 2011-2022 |
| ZINC | | | | | | ● | 2020-2022 |

Table 4.1: Overview of some of the recently active APT groups using evasive techniques that motivate this study, not all of which are investigated in this Chapter.

prevalence of code signing in malware but do not conduct a practical study of the effectiveness of these techniques with real-world samples against AV systems [52,54,96]. Similarly, Mantovani et al. and Aghakhani et al. identify problems with the packer detection subsystems of modern AV but do not robustly test against techniques that are used in malware deployed by modern APT groups [97, 98].

In contrast, our work provides an in-depth evaluation of these techniques. We first review APT reports to extract the current state-of-the-art techniques used by recently active APT groups. We then look at which of these techniques are used most prolifically in the current threat landscape. In Table 4.1, we illustrate the characteristics of some notable campaigns that apply the techniques we are investigating.

Proceeding from this, we enact a reasonable emulation of these modern techniques, namely changing of the Authenticode signature, `.dll` hijacking and runtime obfuscation. We then test to see how effective the major AV engines are at detecting these evasive malware variants and analyse the results.

### 4.1.1 Contributions

In this chapter, the contribution we make to the academic literature to show that by using well-known techniques, we can successfully mask the malicious nature of APT malware samples to the vast majority of AV engines deployed in-the-wild.

We show that by using these techniques, the top 10 market leaders in AV, which collectively make up 85.5% of the market, have an average reduction of 27.76% in detections using the techniques we describe in this Chapter.

We prove an architectural flaw in certificate validation by showing evasion of AV for APT samples via signature replacement. We evaluate the structural vulnerability of AV systems to changes caused by an automated runtime packer we develop. This packing also results in a reduction in detections.

We also identify `.dll` sideloading vulnerabilities within 3 major AV products. We disclosed these vulnerabilities to the affected vendors and assisted with their remediation.

### 4.1.2 Scope

In this chapter, we are only concerned with the attack surface of local AV and EDR systems. Other research has focused upon attacking the update mechanism of AV [99]. This involves remote elements of the remote components of AV and EDR systems, which are outside the scope of this research.

Instead, we principally focus on analysing classes of vulnerabilities with security implications. In particular, we are interested in vulnerabilities that are systemic to AV rather than individual evasion exploits that pertain to the architecture of only one particular AV system.

## 4.2 Background

Two different types of exploitation can be deployed against AV software. Firstly, there are attacks against the AV software itself, wherein the underlying software is exploited. Secondly, there is the evasion of the detection algorithms of the AV software, by producing malware or modifying existing malware payloads to avoid detection.

For our AV evasions we concentrate on the detections displayed by the online VT platform, which hosts different AV engines. Though these are not a complete emulation of AV engines, VT attempts to accurately emulate a locally deployed AV. Static analysis is one of the key parts of AV. Static analysis consists of the methods by which characteristics of a binary are analysed, without execution of the binary.

As it is not possible to determine maliciousness by solely extracting static characteristics from malware, AV systems also use dynamic analysis. Whilst local AV engines may not be able to fully dynamically execute malware samples, AV can identify potentially malicious behaviour. In EDR and cloud-based analysis systems, there is

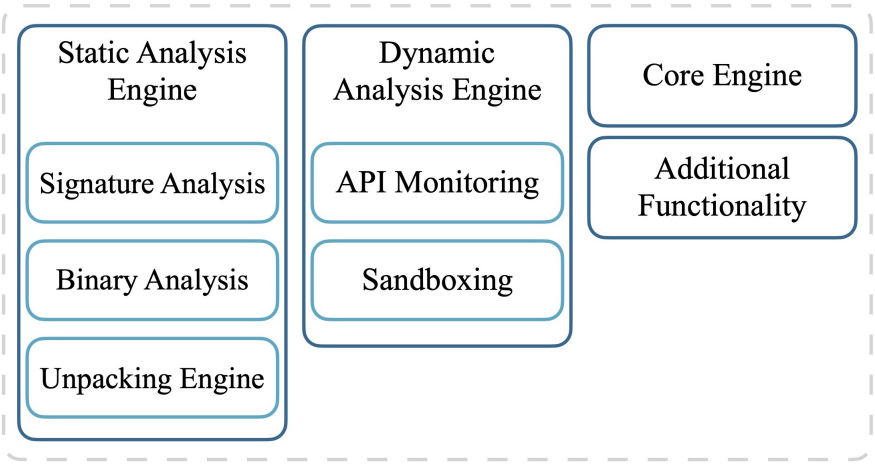



Figure 4.1: Conceptual Diagram of AV system architecture, showing the main components.

more comprehensive sandboxing. However, these dynamic analysis systems can be evaded.

**Distributed System Elements** AV and EDR systems are not a single binary that processes files on a filesystem to identify whether or not they are malicious, but are composed of several different interconnected modules. We enumerate these core elements in Figure 4.1.

These elements include the central orchestration of the AV, usually via an executable file. This central orchestration executable or service is often required to have a high level of privileges on the system to ensure the software is able to detect threats.

Whilst this high level of privileges is necessary for an AV system to function effectively, this approach also bears some risks. If the AV program itself is subverted, the malicious code is executed in the context of the AV with elevated privileges. By doing this, the malicious code gains the same level of trust and permissions that the vulnerable AV system has.

As AV companies are commercial businesses, additional components are bundled with AV packages, such as password managers, VPN network configuration managers, and other network security tools.

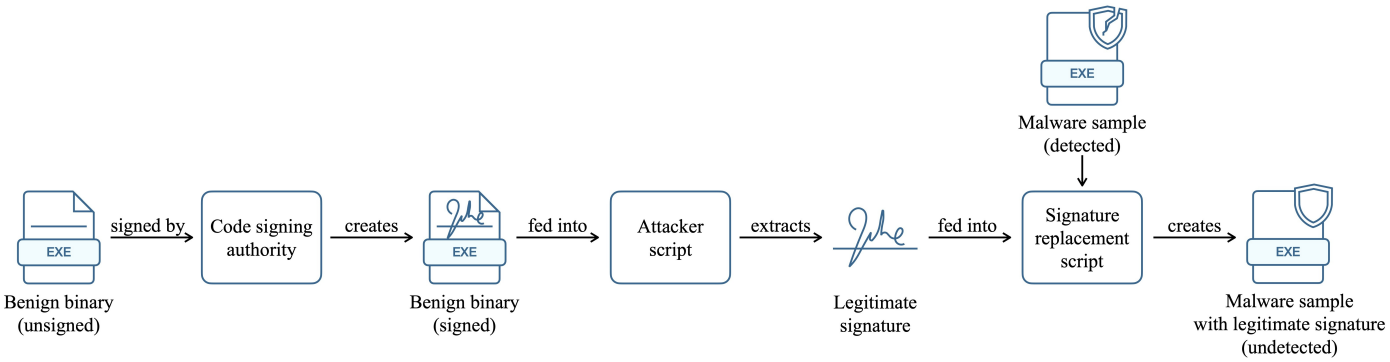



Figure 4.2: Conceptual architecture of the process by which a legitimate Authenticode signature is inserted into a malicious binary.

## 4.3 AV Attack Classes

In this section, we describe the different techniques commonly used by APT groups to evade modern AV software. An overview of these techniques and their usage by recently active APT groups is given in Table 4.1. For various reasons, we exclude several evasion techniques from our evaluation, we detail these exclusions in Section 4.3.4.

To gather a subset of attacks on AV, we conduct a manual review of techniques used by red teams and APT malware. The purpose of this review is to identify the current state-of-the-art. From this review we extract some of the most notable attack types that we are able to observe being used. This is to ensure that the evasion techniques we test accurately represent the techniques that malware operations use.

### 4.3.1 Signature Replacement

Signature replacement is another area that is investigated within the academic literature. Commercial threat intelligence organisations and industrial researchers also investigate this attack vector. There is no shortage of malicious campaigns manipulating the Authenticode signature of their payloads. In Figure 4.2, we show the process by which a malware sample can have its maliciousness obfuscated by the use of Authenticode signature replacement.

The alteration of signatures in binaries has been used by some notable campaigns. APT campaigns such as Stuxnet and Flame, conducted by the higher tier nation state groups, were able to manipulate signatures [100].

It is only recently that open source tooling and the provision of free and easily configurable certificate authorities such as 'Let's Encrypt' has made signing malicious binaries less complex for attackers [101].

Programs such as `Sigthief` [102] and `Sigpirate`, as well as documentation of common Authenticode hijack methods, has made the implementation of signature alteration far easier for the entire spectrum of offensive cyber actors. This ease of implementation has led to an increase in the use of certificate forgery, as evidenced by Kim et al. [52].

### 4.3.2 Packing and Static Analysis Evasion

Much like malicious signature replacement, evasion of static analysis and packing is a common evasive technique. This is not a novel attack mechanism; there is a large academic literature dedicated to the systematised analysis of packing [8]. This is a well-known and long-standing technique that has been used by nation states, advanced attack groups, and other such adversaries. This technique is also used by commodity malware. Our interest in this attack vector is to observe if it is still effective in enabling evasion of modern AV.

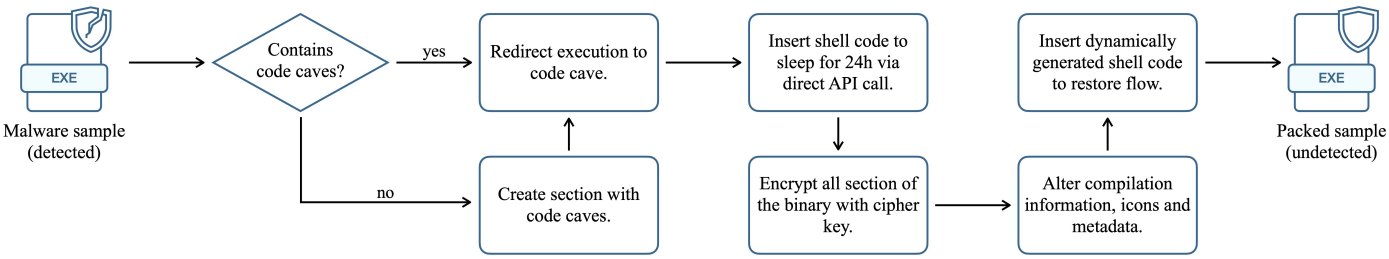



Figure 4.3: Diagrammatic representation of process undertaken to obfuscate and cloak APT malware samples for our experiments.

### 4.3.3 `.dll` Sideloading

`.dll` sideloading is an attack vector used in many modern malware campaigns. Whilst it may not result in privilege escalation in every case, it allows malicious code to be executed in the context of a trusted application. Many open-source attack tools such as `Empire` [103] and `Powersploit` [104] contain tools that facilitate `.dll` sideloading, making these attack techniques trivial to pursue for even lower tier threat actors.

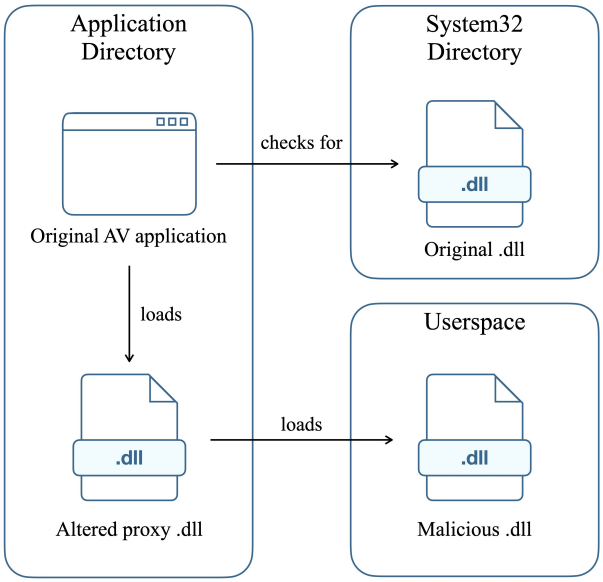



Figure 4.4: Diagram of `.dll` sideloading interrupting typical application functionality to load malicious code via an altered proxy `.dll` injected into the application directory.

The reason this is used as an attack vector is because of the ease of exploitation. It is relatively trivial to implement, allowing execution within the context of a trusted application. Whilst this may not achieve evasion of online sandboxes or similar platforms, `.dll` sideloading gives advantages to attackers with low effort investment.

Some prominent examples of this are APT41, which uses `.dll` sideloading in several of its modules. Astaroth is a banking trojan that uses `.dll` sideloading. Both of these campaigns are notable for being stealthy, using trojanised software and aiming to remain invisible to AV. Another campaign that uses this attack vector is the Moshen Dragon campaign. The adversaries used `.dll` sideloading vulnerabilities in various AV products to deploy malicious payloads [59]. Cobalt Kitty also used `.dll` sideloading attacks versus AV software [105].

`.dll` sideloading is an attack class that emerges due to a logical flaw in the architecture of many programs. This logical flaw is introduced by architectural design flaws in AV, wherein during execution, AV will attempt to load certain `.dll` files from a local directory rather than the 'System32' directory. This design flaw is not unique to AV, with many programs exhibiting similar behaviour [93].

In Figure 4.4, we show the conceptual architecture of how `.dll` sideloading functions. Our diagrammatic representation shows how this functions in a general conceptual

sense, but also functions as an exact representation of the nature of the vulnerabilities we discover in several AV suites.

### 4.3.4 Discounted Attack Vectors

To determine the eventual techniques to investigate in depth, we narrow the field of evasion techniques included within our analysis.

In selecting the techniques to investigate, we must choose methods that are feasible to test via experiments in a cloud or local sandbox. For instance, the use of multi-staged dropper functionality or fast-flux domains for evasion would be complex to realistically simulate in a cloud-based sandbox.

In addition, we avoid investigating techniques that have already been researched thoroughly in the existing academic literature. For instance, the effectiveness of process injection and process hollowing is well-known in both industry and the corpus of academic research [61, 75]. For similar reasons we do not analyse aspects of malware's network operations as this has been extensively covered by the existing literature [39].

We do not include adversarial machine learning for the use of obfuscating malware binaries. It is shown to be an effective AV evasion method in existing work and so we do not repeat existing research by testing its effectiveness [106, 107].

Finally, the other selection criteria we use is the frequency of technique use by APT campaigns. Across the reports we reviewed, we observe which techniques are being used by various different campaigns, rather than highly effective niche examples. This is to ensure the broadness of applicability of our research to both commercial and academic settings.

## 4.4 Experimental Setup for AV Evaluation

In this section, we present the different experiments that we conduct to robustly and systematically test the vulnerability of AV to different evasion techniques. These experimental setups take two forms.

The first is uploading modified payloads to the cloud-based analysis platform VT, which runs our large dataset of modified binaries in a number of different sandboxes to get a more holistic assessment of evasion capabilities with real-world datasets. These are run in external cloud-based sandboxes hosted on VT.

The second is the evasion of AV that is set up on local machines, to test the vulnerability of AV to specific payloads in a local environment. This is to discover potential `.dll` hijacking vulnerabilities in a given environment. For this second set of experiments, we set up a VM for each separate AV engine, then test if that particular AV is vulnerable to `.dll` hijacking.

### 4.4.1 APT Malware Dataset

The dataset we use for testing our evasion techniques is from the well-known malware research and development site 'VX Underground'. This dataset contains 26,774 malware hashes for modern APT malware. We discard samples that are not in an appropriate format for our evasion methods to be applied, leaving 21,477 binaries remaining for analysis.

The method we use to gather this dataset is similar to other research with slightly less contemporary datasets, wherein we automatically process malware reports to extract hashes [108]. We then download the binaries that pertain to these hashes.

This is a collection of APT malware from the period 2010-2022. Upon initial upload to the VT platform we are able to see that many engines correctly identify these samples as malware. This dataset represents an accurate cross-section of the current elite attack groups that engage in offensive cyber activity.

We choose this dataset as it is recent and timely, ensuring that we do not commit the common experimental mistake of concept drift [50, 51]. This occurs when detection mechanisms are no longer applicable due to the evolving methods and nature of malware over time. The timelines involved in conduction and publication of research mean that there is dissonance between detection mechanism development and its application to real world malware datasets.

The composition of this dataset is composed of a majority of PE32 binaries. Although desktop applications and OS have moved mostly to a 64-bit model, malware lags behind somewhat [109]. This is also reflected within the analysis environments that these programs execute in. The main OS used in the prominent VT Jujubox is a 32-bit Windows 7 system. Many online sandboxes use legacy OS as they are more likely to get malware to execute the full range of its capability. This is because legacy OS will have less inbuilt memory protection or other defensive mechanisms, that would prevent malware from being able to execute correctly. As 32-bit binaries still run on 64-bit systems, malware that runs on 32-bit systems will run on 64-bit systems.

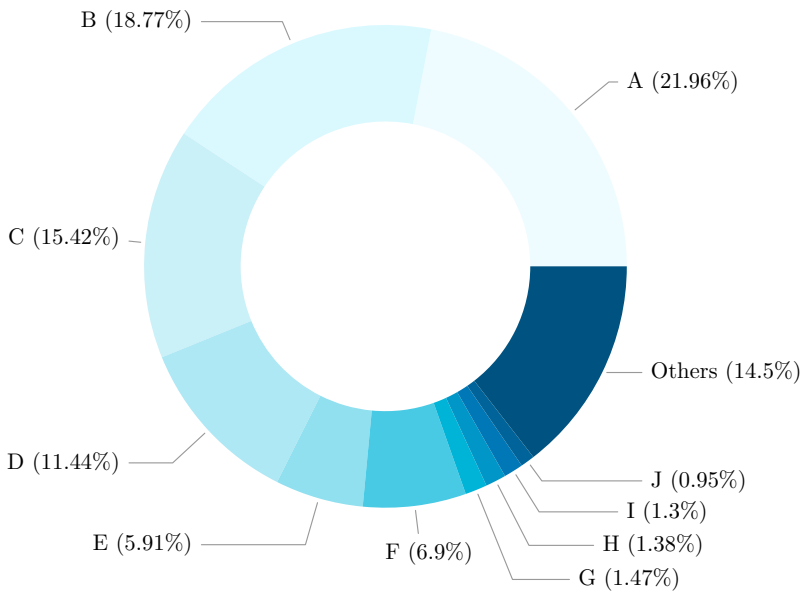



Figure 4.5: Cumulative chart showing distribution of market share by AVs.

### 4.4.2 Ten Most-Used AVs

To give an accurate representation of the actual AV market, we represent a subset of all tested AV engines with our detection results. We show this cumulative distribution in Figure 4.5. By analysing 10 out of the total 78 AVs, we nevertheless cover the 85.5% of installed AVs. Whilst we represent the larger dataset in our results, we focus on these top 10 AVs in Section 4.5.4 to examine this representative subset more precisely.

### 4.4.3 Evaluating Signature Replacement

To test for the vulnerability of AV to Authenticode signature replacement we first download a known malicious dataset. The dataset we use is the aforementioned 'VX Underground' APT dataset. Our first step is then to see the amount of AV engines on VT that detect this initial malware dataset. This establishes the ground truth baseline of detection, with the original Authenticode signature or lack thereof.

Next we perform signature replacement on all of the above samples. We automate the existing functionality of Authenticode signature replacement tools to achieve this.

We replace the Authenticode signatures in our malicious binaries with those from benign ones. By doing this, we are then able to bypass detection on different AVs. We apply signature replacement for the APT malware binaries mentioned in the previous

subsection, by using the tool `SigThief` to replace the Authenticode signatures of these binaries with one extracted from a legitimate and valid Windows binary.

In terms of the binaries we extract the legitimate Authenticode signature from, an attacker can theoretically choose those located in the System32 folder or any other binary signed by Microsoft. In particular, we replace signatures of initial binaries with the `SigCheck.exe`'s signature, a binary produced as part of the `SysInternals` [110] suite for the purpose of signature validation. We chose this binary as it is used for the purpose of validating Authenticode signatures, meaning this signature may facilitate more effective evasion when AV heuristics process malware that has had a replaced signature.

We then upload these modified versions of the malware binaries to VT to assess how the presence of an altered, but not entirely correctly replaced Authenticode signature would affect its detection rate by AVs. We conduct this experiment to determine how much the presence of a signature affects the eventual detection rate of AVs, regardless of the validity of the Authenticode signature.

Identifying the magnitude of this dissonance is how we determine the extent that AV detection is affected by the replacement of Authenticode signatures in malware binaries.

We plot the amount of detections in Figure 4.6. It is important to mention that the versions of AV engines hosted on VT do not fully reflect the enterprise or commodity detection capabilities of these software. However, these detection heuristics are a reasonable representation of the detection capabilities for given AV engines, particularly with regards to their static components.

### 4.4.4 PE File Modification

To investigate the vulnerability of AV to alteration of the binary by a malicious actor, with the properties remaining the same, we also perform modifications to the binary. We then follow the same process we undertook for Authenticode signature replacement in terms of testing and comparing versus the detection rate of the original APT malware dataset.

The objective of performing these modifications is to change the static nature of the binaries, to prevent identification of specific byte strings through mechanisms such as Yara rules. We did not alter the fundamental nature of the binary's functionality whilst performing these modifications and so all of the dynamic behaviour of the

binary would remain the same. We also alter the metadata and characteristics of the binaries to imitate and more closely resemble the metadata of a Windows system binary.

These modifications take the form of a multi-staged process by which we automatically modify binaries to have their basic functionality remain the same but alter their execution flow so that AV less easily detects them. In Figure 4.3, we diagrammatically show the process we create to automatically obfuscate the APT malware samples. To accomplish this alteration we redirect execution flow to code caves (empty areas within the binary), perform obfuscating actions and restore the original execution flow.

The precise steps we perform to achieve this are as follows:

1. Identification of code caves within the binary. Additionally, if these code caves are not found, we add an extra blank section to accommodate the malicious shellcode.

2. We redirect the execution flow to go into either the code cave or the additional section added.

3. Within this code cave we implement the shellcode which performs a simple XOR encryption of the rest of the binary. When the binary is run this is simply reversed, to facilitate runtime decryption.

4. Next, we redirect the execution flow of the program to the beginning of the binary. This ensures that the original functionality of the malicious binary remains, but with the addition of encryption which is decrypted at runtime.

After modifying these binaries, we upload them to the VT platform. We do this to assess whether the modified APT malware samples are still detected as malicious or whether they are able to successfully evade detection. Binary modification such as this is within the capabilities of most nation-state attackers. In fact, this may be towards the lower end of the spectrum of sophistication for advanced attackers.

The altered instructions at the original entry point of the binary are not encrypted, as they are necessary for the execution of the file. These unaltered bytes are known as the 'decoder stub'. This includes the additional instructions we add to the code caves to enable the obfuscation functionality.

To automatically obfuscate many malicious binaries without prior knowledge of the binary, we implement methods that automatically modify the assembly code of target binaries.

**Metadata Alteration** In performing this experiment we find one other notably effective method. This was the alteration of metadata. We list in Table 4.1 techniques that are used by several attack groups.

We also found from our experiments that altering this metadata was disproportionately effective at reducing the detection rate for malware samples. Upon first glance, this would appear to be abnormal and unexpected behaviour for AV. However, similar to Authenticode signature analysis, AV and endpoint detection systems may have heuristics that check for the presence of compilation metadata or icons.

### 4.4.5 Evaluating `.dll` Sideloading

To test the vulnerability of AV and EDR to `.dll` sideloading, we set up a testing infrastructure. In this testing infrastructure, we use Windows 10 systems on VMs. On these VMs, we install separate AV products to test whether they contain vulnerabilities. From this base image, we set up several hosts for testing.

We then scan the programs to observe which `.dll`s load at runtime but are not present in the application directory. We then copy these `.dll`s and trojanise them by injecting malicious functionality. This malicious functionality is a reverse shell.

We develop an automated Powershell script to test the vulnerability of these AV programs to `.dll` sideloading attacks. This script copies a trojanised `.dll` to each directory, runs the AV executable and then pauses execution, so we can manually check whether the reverse shell executes. Our script then deletes the `.dll`. However, in earlier iterations of our testing where we did not delete the `.dll`s, we find that malicious functionality would execute on OS boot. This evidences that it is possible to gain and maintain persistence on target systems via the use of `.dll` sideloading.

To discover these vulnerabilities we employ a specific and repeatable process. We execute AV binaries for each of the main AV providers and use Process Monitor to identify the `.dll` files that are searched for but are not present on the system. When we discover such a `.dll` we copy the `.dll` file it was looking for in the System32 folder to another directory, and inject it with malicious functionality to cause a reverse

shell to appear when the AV application is loaded. This vulnerability has security implications for several reasons:

1. A poisoned `.dll` starts with the same signed execution privileges and application 'allow list' bypasses.
2. `.dll` files executed within the context of an AV application bypass the self-defence mechanisms certain AV companies implement to prevent implantation of `.dll` files.
3. When restarting the system, certain AV programs start when the system starts. In this manner, we were able to achieve malicious persistence on the OS via inserting a malicious `.dll`.

To test for the vulnerability, a multi-stage process is pursued. We first prepare a payload to be injected on execution of a binary. We then attempt to put the payload in a specific directory and observe if it executed. The precise steps we follow are:

1. We use `Siofra` [111], a vulnerability scanner to identify binaries that attempt to load `.dll` files.
2. We compile a `.dll` that executes a reverse shell from our target machine back to our attacking machine.
3. We use `Siofra` to inject our malicious compiled `.dll` into a `.dll` copied from the Windows system directories.
4. We then copy this trojanised `.dll` into the program directory.
5. We then execute this program to test whether or not the `.dll` sideloading vulnerability exists. If a vulnerability is found then a reverse shell is generated.

At all stages of this testing we record video proof, including screenshots and other documentation, which we compiled into an eventual responsible disclosure report, sent to all different affected vendors. We also assist with remediation of vulnerabilities and testing of patches to see if the vulnerabilities remain after the AV vendors patch their software.

## 4.5 Results

In the following section, we describe the results of our evasion testing. In doing so, we aim to evidence the different ways that modern AV systems are vulnerable to attacks

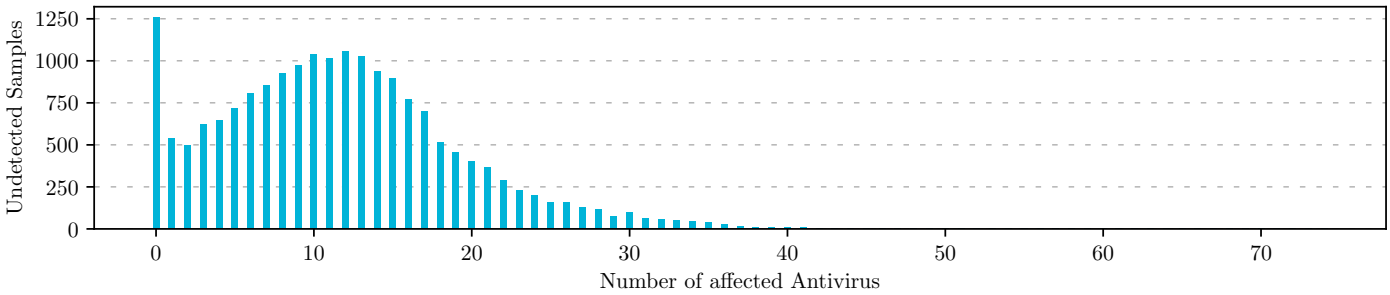



Figure 4.6: Histogram representing reduction in detections for APT malware dataset with Authenticode signature replacement.

in a systematic and scientific way. The three core areas we test are signature replacement, runtime packing and `.dll` sideloading. All of these are evasion techniques used by modern APT groups.

### 4.5.1 Signature Replacement

In Figure 4.6, we show the results of our evasion experiments wherein we replace the signatures of APT malware. What can be seen is that there is a significant reduction in aggregate detections across all of the malware samples. Notably, there is a peak of detection reductions between 6-20. This shows that for a significant majority of samples there was a slight reduction in the number of samples detected.

For all of the samples within this dataset, there is an average reduction of 10.21 engines detecting them when we replace the Authenticode signature. This shows there is a systematic flaw in the way AV systems process and validate Authenticode signatures.

In the signature replacement experiments, the structural deficiency within AV that we exploit is that of the certificate validation mechanisms. In developing AV detection mechanisms it is reasonably simple to implement an algorithm that will detect whether or not a binary has been assigned a given Authenticode signature, but harder to validate this signature. This is in part due to the complexity of chains of trust within the code signing ecosystem.

This is particularly complex if, as with our experiments, we use a legitimate digital signature, but this signature is invalid. There are mechanisms within the public key infrastructure and code signing ecosystem for key revocation if a public key is discovered to be invalid or used for a supply chain attack [96, 101].

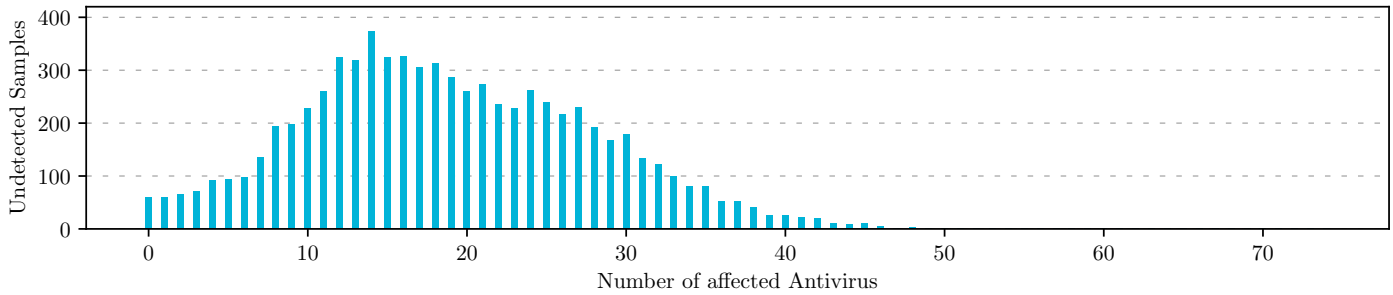



Figure 4.7: Histogram with reduction in detections for APT malware dataset with packing, obfuscation and metadata alteration.

As with other elements of the Authenticode infrastructure there are many small details that can introduce complexity here. It can be convoluted to fully validate the entire chain of trust for a given binary. As certificate characteristic changes are expected between software versions, this introduces the potential for false positive errors to emerge.

Further complications emerge when a binary has a signature assigned to it that is unable to be validated but is extracted from a legitimate binary. As such, our extraction of an Authenticode signature from the legitimate Microsoft binary, `SigCheck.exe` causes an issue for this very reason. The fact that the purpose of this binary is validating Authenticode signatures and that it is used for signature verification by many legitimate programs introduces additional complications.

The reason this presents an issue is because of the risk of false positives. As we falsify a Microsoft certificate, this further evidences the point. The effect of an AV mistakenly identifying a legitimate Microsoft executable as malicious across millions of client machines and blocking execution is significant. This may in fact cause users to uninstall their AV software and thereby have a negative business impact. The complications lie then in the fact that the impact in terms of time and effort for users of an AV program are considerable. As such, many engines use the heuristic of the presence of a signature rather than validating it.

### 4.5.2 PE File Modification

In Figure 4.7, we show the alteration in total number of detections between our initial APT malware dataset and the dataset wherein we implemented a runtime packer. We observe, as with the Authenticode forgery, that there is a reduction in the amount of AV engines detecting these samples. There is a markedly different distribution in detection reductions when comparing the packed samples to those with Authenticode

signature replacement. As such, whilst there are less samples that have a reduction in their number of detections, for those that have there is a peak reduction of 15-25 detections. There is an average reduction of 15.54 AV engines detecting each sample after we implement runtime packing routines.

We observe that there are lower overall detections when we implement our packing scheme. The impact of this detection is less marked than Authenticode signature replacement as our packing scheme may have triggered generic packing detection heuristics. Although not visible in the diagram, we find that our alteration of meta-data in terms of compilation timestamps, file icons and filenames are unexpectedly effective in reducing detections.

The editing of the binary files did not fundamentally change the nature of the binary in terms of its dynamic behaviour. In terms of the binary's static characteristics, they have been altered to obfuscate the malicious functionality. As such, our initial expectation was that there would be a reduction in the amount of detections by AV engines that use methods such as static Yara rules, signatures composed of byte sequences and similar methods to detect malware.

The evasion methods we implement are not particularly sophisticated, but do show notably effective results. As such, we can estimate that advanced threat groups would be capable of implementing far more advanced and evolved versions of the same methods.

### 4.5.3 Increase in Detection Rate

For 896 of the packed samples we observe that there is an increase in detections. For the 1172 samples that had their Authenticode signature replaced or were packed we also see an increase. These anomalous results occur due to AV engines containing heuristics that detect generic packing subroutines or signature replacement. As a result of this, with some samples our obfuscation of malicious functionality actually inserts other characteristics that appear malicious to certain AV engines.

### 4.5.4 Ten Most-Used AVs

Figure 4.8 represents results relevant to the ten AVs that we identify to have the greatest share of the market [112]. What we show is that with all these AVs, there is a significant reduction in the percentage of samples that AV detects, with each of the

evasion techniques. This means that 85.5% of AVs installed on home systems can be evaded using the techniques listed.

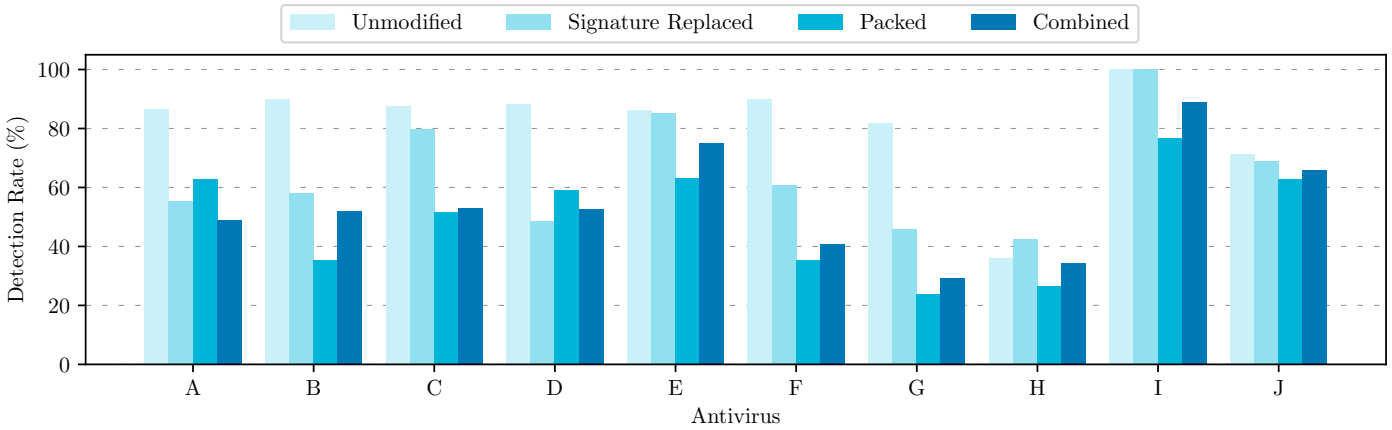



Figure 4.8: Comparative bar chart for top ten AVs, representing detected percentage of total APT malware samples.

What we can also observe is a larger amount of variance than expected between different AVs. For instance AV 'I' shows that 100% of the samples we submit are detected in the initial sample set and close to that in the replaced signature and combined datasets. We can also see that AVs 'B', 'F' and 'G' show particularly strong effectiveness at detecting samples that are packed.

An interesting conclusion to be drawn from this data is that combining the evasion methods results in more detections than these techniques in isolation. In Section 4.5.3, we state that some samples increased their detection rate due to generic heuristics deployed by AV. Similarly to this, when several techniques are used simultaneously this may trigger more generic detection heuristics. This could occur due to various obfuscation methods being analysed by AV engines to be suspicious, when used in combination.

For instance, one malware sample may be identified as potentially benign if it has an Authenticode signature removed. However if it also has a number of cryptographic and sleep subfunctions in addition to this Authenticode signature removal, this may pass the threshold required to be identified as malicious by an AV engine. This is a logical explanation for this result.

### 4.5.5 `.dll` Sideloading

In our experiments, we identified several instances of executables within AV programs being vulnerable to `.dll` sideloading. The results of our experiments are visible in

Table 4.2. These results show that we identify several instances of `.dll` sideloading vulnerabilities in AV software.

| AV Software | Executable | `.dll` |
|---|---|---|
| BitDefender Total Security | **BitDefender VPN** | |
| | agentcontroller.exe | bcrypt.dll |
| | bdreinit.exe | version.dll |
| | bdsubwiz.exe | winmm.dll |
| | bdvpnservice.exe | |
| | vbpnuiapp.exe | |
| | hydra.exe | |
| Kaspersky | **Kaspersky VPN 5.2** | |
| | kldw.exe | userenv.dll |
| Trend Micro Antivirus | **Trend Micro AirSupport** | |
| | uiAirSupport.exe | msimg32.dll |
| | Uninstall.exe | version.dll |
| | update.exe | uxtheme.dll |
| | **Trend Micro Password Manager** | winmm.dll |
| | PwmConsole.exe | |

Table 4.2: Instances of `.dll` sideloading discovered and the AV software in which it was identified. We enumerate the `.dll`s poisoned as part of the vulnerability research process.

What is notable about these vulnerabilities is that it is not the core AV software that contains these vulnerabilities, but the peripheral components. BitDefender and Kaspersky's VPN products contain the vulnerability. Trend Micro's security software contains vulnerabilities within their customer support and password management functionality. These items being peripheral to the core AV functionality does not diminish the impact of this vulnerability, as they are still signed with the same Authenticode signature that the other components are. This means that malicious code can still run within the context of the trusted process.

## 4.6 Discussion

In researching the vulnerabilities and bypass techniques in this chapter, we aim to identify structural deficiencies and detection gaps within AV as a software class. In

this section, we explore the implications of our results in more granular detail.

The results of our research showed that the evasions advanced threat groups deploy are capable of evading modern AV systems. This means that the current structural weaknesses in AV present an attack vector that can be exploited by real-world campaigns.

### 4.6.1 Permissions

One of the key structural elements in AV systems that leads to vulnerabilities emerging is that of permissions. By its nature AV requires elevated levels of permissions to conduct analysis on files and system behaviour. Malicious actors often look to escalate their privileges on a target system. As such, AV engines, particularly those with distributed components, are targeted by malware authors.

Additionally, whilst not exploited by our research, the fact that AV programs have to ingest files presents a holistic systemic flaw. Research has specifically targeted the file processing capabilities of AV, identifying some systemic flaws. Carmony et al. identify flaws related to the complexity involved in the analysis of JavaScript payloads in `.pdf` files [97]. Jana et al. identified vulnerabilities in metadata and type classification of files by parsers in AV systems [113]. Koret also identifies that AV needs to support a wide range of file formats, presenting a wide attack surface with many exploitable edge cases [114].

### 4.6.2 Distributed System Elements

The other inherent flaw is that there are many distributed parts of an AV program. These intersecting modules can cause vulnerabilities to emerge. The distributed nature of system elements in modern AV systems was shown to cause vulnerabilities in security products in our testing for `.dll` sideloading. We discover vulnerabilities in several areas of the security solutions we test that are not part of the core AV engine.

In particular, for BitDefender and Kaspersky we discover vulnerabilities within the VPN software bundled with the AV, but not the AV itself. The vulnerabilities we discover in Trend Micro were also within the TroubleShooting Tool (AirSupport) and not the core AV engine.

Existing research has shown that as the complexity of software programs increases, so does the number of potential exploitable vulnerabilities [115]. This means that

increasing complexity of an AV system or other security product in terms of added features may lead to more vulnerabilities.

### 4.6.3 Authenticode

Authenticode is used as a chain of trust. In our results, we show there is a critical flaw in the implementation of these Authenticode verification systems by AV. The validity of a particular Authenticode certificate applied to a malware sample does not matter so much as whether or not there is a certificate. This is surprising in comparison to existing research, which to an extent assumed that certificates have to be valid.

Existing research observes the impact of attacks such as Stuxnet or Flame, wherein certificate authorities have been compromised and stolen legitimate certificates were used by malware authors. These are considered as valid by the entire chain of trust until revocation. Previously, we assumed that these Authenticode signatures had to be valid, however we achieve our results with an invalid signature.

The whole concept and necessity of certificate revocation for binary signing may be less important than initially theorised and assumed in earlier work. This is because malware authors and other malicious actors have exploited the improper verification mechanisms we identify in many AV engines to deploy their malware [52]. This shows how the different threat models for binary signing for Authenticode and TLS or similar transport layer security mechanisms lead to different security assumptions and therefore implications.

This is not to say that the whole concept of public key infrastructure and chains of trust is no longer fit for purpose. But for the purpose of binary signing, it may be argued that these systems are unfit for purpose. Without access to the closed-source source code of AV software, it is not possible to identify the reasons. The solution to this is refactoring of current AV systems with regards to their processing of malware samples with Authenticode signature alterations, particularly if these certificates cannot be verified.

### 4.6.4 Static Code Analysis

What is interesting about the results of our experiments with deploying the automated PE encryption tool we developed is the amount of engines that did not detect the samples we modified. Existing research identified systematic flaws in packer detection mechanisms [97], such as their inability to detect packers that have low levels of

entropy [98]. Existing packer detection mechanisms can be simply based on packers that have been observed in the wild.

It is notable that the design of our obfuscation mechanisms are not particularly sophisticated, if compared to the cutting edge of malware development. Whilst the dataset to which this runtime packer is applied is composed of payloads deployed by APT groups, the obfuscation mechanisms are similar to those deployed by less well-resourced cybercriminal operations.

## 4.7 Suggested AV Best Practice

Whilst we identify detection gaps that emerge in AV systems, we can also leverage our findings to advise on best practices with the development of these systems. Our suggested changes are for the purpose of remediating the issues we identify with AV and EDR systems.

### 4.7.1 Authenticode Signature Validation

In programming the Authenticode signature validation mechanisms within AV engines, validation of the certificates and their validity or revocation status should be undertaken. In tandem with this, simply the presence of a certificate, regardless of its validity, should not be valued by detection algorithms of AV software. Whilst the presence of a certificate is sometimes evidence of validity, if there are inconsistencies in the chain of trust or the certificate authority is not reputable, this is likely an indication of compromise. As many APT and supply chain attacks contain corruptions and subversion of the signing process, this subversion needs to be considered more seriously as an attack vector by AV developers.

### 4.7.2 Miscalibrated Heuristics

Another element that would benefit from refactoring within AV systems is the use of certain automated heuristics. These are heuristics that cause dangerous false negatives based on mistaken assumptions.

Some of these mistaken assumptions are because of metadata. Changing malware binaries compilation metadata to imitate the compilation metadata for native Windows binaries causes a drop in AV detections. This is especially true for other metadata elements such as the file's icon.

### 4.7.3 Scope Creep Prevention

One area in which to alter best practice is to moderate the amount of scope creep for the architecture of AV. For commercial reasons, AV providers include many ancillary utilities as part of their products. The higher amount of binaries issued as part of a security solution increases the attack surface. This increased attack surface manifests itself in the form of increased complexity of managing the interaction between these services. There is also an increased complexity of permissions configuration. As such, it makes sense for the core AV engine and products installed as part of this offering to be bundled with only the necessary components.

Even considering just the core AV components, there is a static analysis engine, a dynamic analysis engine and several other heuristics. These engines, coupled with the file processing technologies, introduce enough vulnerabilities within AV. As such, greater segmentation of software elements should be undertaken when releasing AV products.

We also recommend that care is taken to manage the permissions for AV software. To function correctly and spot malicious usage on a system, AV needs to have elevated permissions. It is therefore important that trade-offs are considered and the principle of least privilege is observed. As AV is run as a trusted process with elevated privileges, the necessity of these privileges must be analysed.

### 4.7.4 Limitations

Whilst we conduct a scientific study that looks at the vulnerability of AV to these attack vectors, the effectiveness of our attacks are limited by the fact that we analyse one factor in isolation. Whilst this allows us to analyse the effectiveness of specific techniques, actual attack groups are not bound by the same limitations as our experiments. Malware produced by these groups' campaigns can leverage several evasion techniques simultaneously.

As our techniques only use one payload, this can be regarded as not a fully accurate representation of current APT malware. APT malware uses multiple stages in attack campaigns, including droppers, additional `.dll` files and binaries in memory. There may also be 'fileless' components, wherein self-deleting scripts are run as part of the intrusion killchain or persistence is achieved through modification of registry keys or similar methods [116].

We also focus our research on the high-end of offensive cyber security campaigns, with our dataset being exclusively composed of malware binaries that have been deployed by APT groups. This means that our conclusions may not be directly applicable to typical commodity malware.

As our research is a systematic approach to validating the different attack methods in use by modern attack groups, there are certain limitations to the techniques we can explore within our research. For instance, techniques such as social engineering and multi-staged droppers are impractical to replicate in cloud-based sandboxes or on local machines in a controlled manner. Similarly, script-based viruses, process hollowing, persistence through the registry and LotL techniques have been covered extensively in the academic literature, as we review in our related work in Section 4.8.

### 4.7.5 Future Work

We hope that our work will have some effect upon AV system development. In particular, that progress will be made by AV and EDR software authors towards improving the architectural flaws we identify in AV and detection systems.

Our experiments also allow more impactful red teaming and penetration testing approaches. This is due to the fact that our research highlights several attack vectors that may be considered to be 'low-hanging fruit' for adversarial simulation. These attack mechanisms may have been considered ineffective or unnecessary as part of red teaming exercises, however we show that they can be implemented without significant time investment or expertise.

We facilitate future steps by disclosing some of our findings to relevant AV companies. We also work with these companies to patch and resolve these findings.

## 4.8 Related Work

Several papers analyse the weaknesses of modern AV and EDR systems. Mantovani et al. investigate and challenge the validity of the commonly accepted wisdom that entropy is sufficient indication of packing [98].

Similarly, Aghakhani et al. [97] explore shortfalls of machine-learning based methods in the detection of malicious software using packers. Many papers look at packing, particularly noteworthy is an experimental paper that looks at evasion of AV via runtime packers [89].

There is also research concerning the proper performance of malware experiments [20]. This influenced later research and provided a framework for malware experiments to follow proper procedure. Papers look at the problem of concept drift and other similar biases that emerge in malware datasets [19]. Research identified flaws in the online analysis systems on which much of this academic research runs [68, 94].

With such widespread issues identified by the existing literature, it is not surprising that there are detection gaps within modern AV software. In our research, we aim to identify these detection gaps and the reasons for their occurrence. We characterise how widespread these potential evasions of AV software are. We also look at which of these evasions are used most commonly by modern APT groups and conduct deeper analysis on those that are most prominent.

### 4.8.1 Static Analysis

There are various different static analysis methods used by AV to detect malicious software; for instance, entropy analysis, wherein regions of high entropy are identified as potential indicators of encryption or packing in progress [117]. Static analysis also includes automated analysis of control flow graphs and API calls [21, 61].

A different type of signature analysis is also used, wherein the digital signature of the certificates used to sign binaries is analysed. With Windows binaries this is known as Authenticode. This is also subject to flaws due to the cost of false positives with this verification process. As such, many AV engines adopt the approach of establishing whether or not a signature is present rather than verifying a chain of trust. This is to establish that a binary has been signed and does not consider the threat model of an altered signature, though earlier work has measured the scale of malicious code signing [54].

This also includes signature analysis, wherein signatures of malware are compared to a database of known malware. The fallibility of this method of detection is obvious as malware authors can rename the binary, change one byte or make other minor alterations. There are also hashes taken of elements such as the import table, in a process known as 'imphash' or import hashing [118]. These hashes can be used to track similarities between malware campaigns. There is also Authentihash, used by Microsoft to determine whether or not changes have been made to a binary.

Fuzzy hashing identifies similarities between binaries. This uses tools such as *SSDeep* to generate 'context triggered piece-wise hashes'. These tools identify similar

files, with similar characteristics. They also provide resilience against rudimentary evasion techniques that slightly alter files to evade detection algorithms. There are limits to the use of these techniques, specifically those that leverage machine learning [97].

### 4.8.2 Dynamic Analysis

The analysis of malware via online sandboxes using VMs has increased exponentially over the past few decades. Academic research such as that conducted by Shi et al. [63] and Afianian et al. [119] has shown that the deployment of these techniques has become de facto. In 2012, the prevalence of evasive techniques was estimated to be 81.4%. Further research, detailed in Chapter 5.4.1 discovered that 21.15% of malicious samples in their dataset did not execute in virtualised environments, which we took the prudent step of removing from our dataset, unlike much comparable research. This was before even looking at more sophisticated anti-VM techniques [120].

### 4.8.3 Evasion Techniques

The discovery of AV evasion techniques is an exercise conducted by criminals, academic security researchers, private security consultancies and government agencies. These entities have different approaches to responsible disclosure and publication. Private security consultancies publish their research publicly at conferences such as 'Black Hat' or as independent whitepapers [99]. Koret and Bachaalany [58, 114] investigated attacks and systematically reviewed various ways to evade AV. One other area of academic research is the evasion of AV via exploiting weaknesses in the file processing capability of AV [113]. Nedim and Laskov focus on the evasion of file processing of .pdfs [121].

Sandprint [62] performed practical identification of sandboxes via identifying specific VMs and Spotless Sandboxes [68] analysed wear-and-tear artefacts of malware sandboxes. Kuchler et al. also investigated whether or not the prevalence of sleeper functions prevented accurate analysis of malware on VT [65].

### 4.8.4 Evasion Technique Detection

There are also systems proposed to detect ransomware variants. One such system is PayBreak [18], that is used to capture keys generated by ransomware. Similarly, *UNVEIL* [16] and ShieldFS [122] have been designed to intercept API calls, to detect ransomware execution in progress.

Anecdotal evidence suggests that EDR tools are considered by practitioners to be the cutting edge of malware analysis, and security companies often present their EDR suites as their most technically sophisticated products. Hassan and Wang [30, 123] describe an approach implemented within EDRs, that of creating provenance graphs from the analysis of audit logs. This reflects a tranche of recent research, which has created systems for the analysis of audit log data to identify attacks, particularly those for APTs [124–126].

Recent research also analyses the nature of the underlying PE file to identify whether or not there is malicious functionality [127, 128]. Whilst historically static analysis of binaries for detection of maliciousness was researched, in recent years this has diverged, focusing mostly on symbolic execution. The aim of our static analysis evasion experiments is to determine whether previous static analysis research prematurely declared identifying malicious binaries via static methods to be a solved problem.

### 4.8.5 APT Malware Detection

There is a large amount of research practically analysing the techniques used by more sophisticated attack groups. Hassan et al. and Han et al. use provenance-based analysis to identify APT attacks from datasets [30, 33]. Zeng et al., Milajerdi et al., and Hossain et al. have all investigated different ways of using the context of forensic and audit log data to identify APT malware attacks [34, 123, 129].

### 4.8.6 Malware Measurement Studies

The techniques used by these groups filter down into the commodity malware ecosystem. Techniques used by more sophisticated actors are repackaged and used by less technically capable actors at a wider scale. As the malware used in these attacks is more diverse and larger in volume, this led to more meaningful conclusions by data-driven analysis.

For example, Avllazagaj et al. analyse at scale the ways malware behaviour can differ depending on which host OS it is operating on [130]. Similarly, Lever and Piskozub et al. leverage network data from large malware datasets to facilitate detection and classification of malware [39, 131].

IoT malware has also risen to prominence, and several studies have investigated particular subtypes and the IoT malware ecosystem as a whole [49, 132]. The different architectures of IoT platforms and diversity of different malware on these platforms

mean it is less possible to design universal solutions or gather homogenous datasets. Some studies have analysed notorious IoT malware families such as Mirai and Hajime [46]. Pastrana and Suarez-Tangil analyse the phenomenon of crypto-mining malware, a prevalent malware subtype [9].

### 4.8.7 Packing

Additionally, packers have been subject to analysis and development work within academia. Ugarte-Pedrero et al. [8] publish an analysis of the state of the art of packers. More recently, Mantovani et al. [98] and Aghakhani et al. [97] identify weaknesses in packer analysis techniques.

## 4.9 Conclusion

During the course of this research, we find several instances of `.dll` sideloading. We also find that it is possible to reduce the amount of AV engines that detect APT malware samples by replacing signatures and performing obfuscation steps. These vulnerabilities and evasions expose systemic flaws within the architecture of AV.

From these flaws, we extract a list of best practices for the developers of AV software, to make their software less vulnerable to exploitation — whether that be evasion exploits or exploits directly targeting the software itself.

In discovering these flaws, we expose a number of different generalisable flaws to AV software. We also show that these flaws include vulnerability to legacy attack classes. We show that newer methods being used by advanced attack groups are also effective in terms of the eventual effect on detection rates or facilitation of complex attack chains.

# 5

# Living Off The Land Techniques

This chapter develops upon the motivating elements introduced in Chapter 3, directly aligning with the overarching aim of this thesis to bridge the detection gap in contemporary AV technologies against modern evasive malware. It does so by providing further evidence of this detection gap, particularly in the context of evasive techniques, thus reinforcing the importance of the detection challenges identified earlier.

In this chapter, we provide further evidence that a detection gap exists with regards to detection of evasive techniques in the motivation section. The measurement elements emphasise the importance of the detection gap identified in the earlier sections.

Focusing exclusively on LotL techniques, this chapter delves into a detailed analysis of how malware exploits the local Windows environment on which it is deployed, using native system binaries to evade detection.

In addition to this, there is a measurement of how prolific these techniques are in terms of their use by commodity malware. This includes a dataset of millions of samples, over a time period of over a decade. These datasets are a heterogeneous combination of those obtained from commercial entities and those that are well-known in the academic malware research community. The investigation into LotL techniques directly addresses one of our key research questions, namely *How do modern evasive malware techniques operate, and what are their key characteristics?*

To provide a comprehensive understanding of the prevalence and impact of LotL techniques, we collected a large dataset encompassing millions of malware samples collected over a decade. This heterogeneous dataset, sourced from both commercial entities and well-known figures in the malware research community, offers a broad perspective on the use of these techniques in commodity malware. Furthermore, we have curated specialised datasets to facilitate a focused analysis of APT malware and samples likely to employ LotL techniques. These datasets not only provide empirical evidence of the prevalence of LotL techniques but also help in measuring their effectiveness and the extent to which they contribute to the overall vulnerability of AV systems.

Through this focused exploration of LotL techniques, this chapter seeks to contribute significantly to our aim specified in our overarching methodology chapter, of understanding of evasive malware strategies and to the development of more effective detection mechanisms.

## 5.1 Introduction

As malware detection algorithms and methods become more sophisticated, malware authors adopt equally sophisticated evasion mechanisms to defeat them. Anecdotal evidence claims LotL techniques are one of the major evasion techniques used in many malware attacks. These techniques leverage binaries already present in the system to conduct malicious actions. We present the first large-scale systematic investigation of the use of these techniques by malware on Windows systems.

In this chapter, we analyse how common the use of these native system binaries is across several malware datasets, containing a total of 31,805,549 samples. We identify an average 9.41% prevalence. Our results show that the use of LotL techniques is prolific, particularly in APT malware samples where the prevalence is 26.26%, over twice that of commodity malware.

To illustrate the evasive potential of LotL techniques, we test the usage of LotL techniques against several fully patched Windows systems in a local sandboxed environment and show that there is a generalised detection gap in 10 of the most popular anti-virus products.

Malware development and detection is a cat and mouse game, in which malware authors are continuously developing new techniques to bypass detection systems.

Security products such as AV implement static and heuristic analysis technologies to detect, classify and prevent malware from effective execution [24]. In the past, many solutions heavily relied on signature-based detection, but unfortunately these approaches have become less effective due to the use of polymorphism and packers. In turn, many products started developing heuristic analysis solutions, consisting of algorithms that allow them to detect malicious behaviour. These algorithms have become a crucial component for AV engines. Over time, these algorithms have increased in sophistication and thus require more innovative evasion techniques.

Malware authors and red teams often research and discover new methods to bypass security solutions. While their underlying goals may be different in nature, both types of attackers typically leverage state-of-the-art evasion techniques to accomplish their goals. From a defender's point of view, it is crucial to understand these attacks and study their trends in order to be able to react in a timely manner. One evasive tactic that has become popular among both red teams and malware authors is the usage of LotL techniques.

LotL techniques refer to the use of binaries that are already present on systems or are easy to install (e.g., signed, legitimate administration tools) to conduct post-exploitation activity. By leveraging these tools, an attacker can achieve registry modification, gain persistence, conduct network or system reconnaissance or perform a proxied execution of other malicious code. They can even be used to reduce the event logs generated by malicious activity without needing additional files to be downloaded onto the system.

Far from being obscure techniques, many of them are publicly documented on the internet [133]. Many open source offensive security tools leverage LotL techniques and are frequently used by the entire spectrum of actors, from legitimate red teams to amateur cybercriminals, organised crime groups or nation state actors. PoshSpy [134], a Russian state-sponsored APT29 attack module, was one of the first detected APT uses of LotL techniques, specifically *Powershell* and *Windows Management Instrumentation*. Iranian threat groups [135], APT33, APT34 and others are also well known for using native Windows binaries and other signed tools, particularly *Powershell* [136]. Table 5.1 lists the LotL techniques employed by several state-sponsored actors.

Script-based payloads are often obfuscated by various techniques to achieve higher levels of evasiveness, such as those provisioned by the Invoke-Obfuscation tool [137,

| APT Group | LotL Binaries Used | Purpose Of Execution |
|---|---|---|
| APT3 | Powershell, Rundll32, Schtasks | Credential Theft, Persistence & Proxied Execution |
| APT10 | Certutil, BitsAdmin, Net, Wmic, PsExec | Data Exfiltration & Lateral Movement |
| APT29 | Powershell, Schtasks, Wmic | Data Exfiltration, Lateral Movement & Persistence |
| APT33 | Powershell, ProcDump, Schtasks, Vbscript | Credential Theft, Data Exfiltration & Lateral Movement |
| APT34 | Certutil, Mshta, Schtasks, Powershell | C+C, Data Exfiltration, Persistence & Proxied Execution |
| Astaroth | BitsAdmin, Certutil, Regsvr32, Userinit | AV Evasion, C+C, Credential Theft & Proxied Execution |
| Dexphot | MsiExec, Rundll32, Nslookup, Schtasks | Persistence, Proxied Execution |
| Gallmaker | Powershell, Winword | C+C, Data Exfiltration & Proxied Execution |
| Havex | BitsAdmin, Powershell, PsExec | Credential Theft & Lateral Movement |
| Nodersok | Mshta, Node, Powershell | AV Evasion, Command and Control & Proxied Execution |
| SoftCell | At, Net, PsExec, Reg, Wmic | Credential Theft, Data Exfiltration, Lateral Movement & Recon |
| TA505 | Msiexec, Net, Rundll32, Powershell | C+C, Data Exfiltration, Proxied Execution & Recon |
| Turla | Powershell, PsExec, Wmic, Wscript | C+C, Data Exfiltration. & Proxied Execution |

Table 5.1: APT Groups Using LotL Techniques.

138].

Despite *Living-Off-The-Land* being a relatively well-known term in the information security community, it is sometimes hard to find a precise definition for it. Moreover, to the best of our knowledge no research contains a systematic study of LotL techniques' prevalence in malware samples. Much of the documentation available regarding LotL techniques is in the form of blog posts that document anecdotal findings on certain malware families, or technical descriptions of the techniques used by malicious actors once they obtain remote access to a compromised system.

For instance, Emotet and Trickbot, two of the most common RATs at the time of writing, reportedly use chained LotL binaries for stealthiness [139]. As a countermea-

sure, Microsoft described steps taken to combat LotL techniques used by commodity RATs [140]. The highly evasive RAT Astaroth [141], some of TA505's malware arsenal, Dexphot cryptominer [142] and Nodersok all use multiple LotL binaries simultaneously [143].

In this chapter, we analyse the LotL phenomenon as it pertains to LotL binary usage by commodity malware binaries. Our first step is to describe what a LotL binary is and how it can be leveraged by malicious software to conduct its nefarious actions. We focus our research on Windows as the dominant OS in terms of popularity and the most frequently targeted by malware [144]. Many LotL based AV evasions have been documented. As a consequence, the security community has largely assumed that LotL techniques such as proxied execution of malware are in fact effective against security solutions. As a first step, we question this assumption and raise our first research question:

- *Can LotL techniques effectively evade commercial AV?*

To answer this question, we evaluate a representative set of security products and show how some of these techniques, although well-known for attackers and defenders, are still a valid method to bypass security solutions and therefore are an open challenge for the security industry. In fact, LotL binaries are quite often used by system administrators and advanced computer users to perform system administration tasks, making it extraordinarily difficult to distinguish between legitimate and malicious behavior even for a trained analyst. We responsibly disclosed our findings to the affected AV vendors and followed up with many of them as they improved their detection capabilities.

Although existing documentation provides sound evidence of the usage of these techniques, it is still unclear how prevalent the phenomenon is among malware samples. In this way, we raise our second research question:

- *How prevalent is the use of LotL binaries in malware?*

Building on this, we try to shed some light on some of the trends in the current threat landscape, to identify:

- *What purposes do malware binaries use LotL techniques for?*
- *Which malware families and types use LotL binaries most prolifically and how does their usage differ?*

We also investigate the reasons why these techniques are difficult to detect. Some of the AV firms that engaged with our responsible disclosure communications highlighted the difficulty of separating malicious attacks from totally legitimate administration tasks conducted by system administrators. This brings us to another question:

- *What are the overlaps and differences in the behavior of legitimate and malicious binaries with respect to the usage of LotL binaries? How would this affect detection by heuristic AV engines?*

While there are some clear differences in LotL binary usage prevalence between malicious and benign samples, we also noticed certain similarities for categories such as proxied execution.

In this research, we are interested in the extent to which commodity malware makes use of these techniques. Emotet and Trickbot, two of the most common RAT at the time of writing, use chained LotL binaries for stealthiness [139]. Microsoft describe steps taken to combat fileless malware amongst commodity RATs [140]. This reflects the results of our research, wherein we find RATs are the malware subtype that use LotL techniques most. Our research mostly focuses on evaluating the prevalence of LotL techniques in malware.

Finally, we focus our attention on highly evasive and APT malware to find out that it leverages these techniques twice as much as commodity malware. State-sponsored attack groups, that researchers attribute to China (APT3/APT10) [145], Iran (APT33/APT34) [146] and Russia (APT28/APT29) [147], amongst others, are examples that motivate this study. We list some of these APT groups that conduct attacks using LotL techniques in Table 5.1.

LotL techniques often use script-based payloads. These scripts are often obfuscated by various techniques to achieve higher levels of evasiveness, such as those provisioned by the Invoke-Obfuscation tool [137, 138]. Red teaming tools are also used, such as Powershell Empire[1], PowerSploit[2] and Nishang[3]. Other tools, such as PsExec from the SysInternals suite, which may not be native on Windows, but is a trusted third-party executables, can be used in malicious ways by malign actors [148]. Despite this, little published research has been conducted into identifying precisely how LotL binaries are used in the modern malware ecosystem. Use of these tools and LotL

[1]https://github.com/EmpireProject/Empire
[2]https://github.com/PowerShellMafia/PowerSploit
[3]https://github.com/samratashok/nishang

techniques leave scarce but recognisable signatures in memory and process logs that can be identified. There is a trade-off by attack groups between the ease of use and compilation, versus evading detection by the signature-based and heuristic detection mechanisms utilising memory forensics [89].

**Contributions** To the best of our knowledge, this research presents the largest scale systematic analysis of the use of LotL techniques by commodity and APT malware to date. We make the following core contributions:

- We assess the evasiveness of LotL techniques by testing a representative set of most popular AV engines against malicious payloads deployed via LotL techniques, showing how the detection complexity of LotL represents a challenge for the industry. Even nine months after disclosure, these techniques are still not successfully detected.
- We conduct a large-scale measurement study across several datasets that are representative of modern commodity malware. We determine the prevalence of this technique and the differences among different malware families and types. We also assess the impact that LotL techniques might have in the industry due to false positive risk.
- We evaluate a dataset of APT malware, which we make public to facilitate further research and ascertain that it executes LotL techniques twice as frequently as commodity malware. We also identify which APT groups use LotL techniques the most.

## 5.2 Background & Related Work

In this section, we first define LotL binaries and enumerate some of the purposes for which these binaries are used by malware. Then, we provide an overview of related papers and research on this topic.

### 5.2.1 LotL Binaries

Due to its novelty, there is significant confusion regarding the term *Living-Off-The-Land binary*, or its shorter version *LOLbin*. In recent years, this term has become a common word used to refer to a wide range of binaries used in cyberattacks. Researchers tend to put many different tools under the umbrella of *LOLbins*. Papers

published recently prior to the publication of this research describe it as an emerging, new or recent trend [119, 149–151].

The term 'Living-Off-The-Land' has been historically used to represent the concept of feeding (or living off) what the land can provide, either by farming or hunting. Transposed to malware and intrusion, an attacker might leverage what is already available to them (either already present on the system, or easily installable) to conduct a successful attack and avoid detection.

Some have provided rather restrictive definitions for the term LOLbin. This could be subject to debate, and there is no consensus within the community around this term. For instance, the well-known website LOLBAS project [152] lists a wide range of LotL binaries. Within this list, we are able to ascertain that many of the tools are administrative commands used by system administrators in their daily operation.

In this Thesis, we define a LotL binary as any binary with a recognised legitimate use, that is leveraged during an attack to directly perform a malicious action; or to assist indirectly, in a sequence of actions that have a final malicious outcome.

Other sources also state that `Mimikatz.exe`, typically used for dumping and cracking system credentials, is a LotL binary as well [153]. According to our classification criteria, we also define `Mimikatz.exe` as a LotL binary. However in this chapter we focus only on binaries that are installed by default on Windows systems, rather than extending our analysis to third party Authenticode signed software.

Binaries installed by default on Windows systems like `Reg.exe`, `Sc.exe` and `Wmic.exe` are the most frequently executed by malware. Most binaries installed by default are signed by Microsoft Authenticode [154]. An Authenticode signature proves that a binary has not been tampered or modified from its compilation. This may have a high impact in the confidence that security tools have on it; as discussed by Kim et al. [52], these binaries may even be explicitly allowlisted. Malware making use of trusted LotL binaries could thereby be more capable of evading AVs. The use of system binaries on Windows systems can be incorporated as part of the operation of malware, and more importantly, many LotL techniques make use of system binaries for the purpose for which these binaries were intended.

External signed binaries can also be used, such as `PsExec.exe` or other SysInternals binaries. We also included these in our analysis, but they are less commonly used by malware. Examples of nation state malware using external signed binaries, amongst many, are SoftCell [155] and Havex [156]. These campaigns both used `PsExec.exe` to

stealthily run remote commands for the purpose of lateral movement within compromised networks. In some rarer cases, vulnerable (but signed) drivers have been used to escalate privileges on the system. This is a technique used by the RobbinHood ransomware [157] and various APT wiper malware samples targeting Saudi Arabian systems including Dustman [158], Shamoon and Zerocleare [159].

**Traceability** Certain LotL binaries may leave more system logs than others, that can be leveraged by security tools or forensic analysts in order to detect a malicious action. For instance, Powershell can be configured to have comprehensive logging. Microsoft also recommend a number of these native binaries to be blocked from executing on systems, unless there is good reason to do otherwise [160].

### 5.2.2 Scope of our Study

In this chapter, we focus on Windows malware that executes system binaries for various purposes. These purposes typically involve progression along the intrusion kill chain [35] or avoiding AV detection. All of these techniques are deployed within the user space of the system.

Process hollowing and injection are also not within the scope of our analysis, despite being common techniques deployed by fileless malware. We do not include these process manipulation techniques as they are not LotL techniques according to our earlier definition.

### 5.2.3 Related Work

LotL malware and its occasional aliases, 'advanced volatile threat' or 'fileless' malware are mentioned sparsely in the current academic literature. This is mostly limited to introductory analyses or described as an emerging highly evasive malware variant. Li et al. [161] present an analysis of malicious Powershell scripts that has a subsection specifically describing LotL attacks and fileless attacks as "*the trend of cyber attacks in recent years*". A recent paper by Wang et al. [75] on the subject of data provenance analysis identifies LotL as an emerging prominent evasive malware subtype. Prior work includes introductory analyses [162], however LotL malware has not yet been subject to robust academic analysis. Symantec [153, 163] and Cisco Talos' [164] white papers introduce the subject with an analysis of prevalence across a number of datasets. Currently, no papers conduct a systematic analysis of the use of LotL techniques by Windows malware at scale, comprising multiple datasets. Dang et

al. [48] analyse the behaviour of fileless IoT linux malware in honeypots; this research is concerned with malware samples targeted at Windows systems.

LotL techniques are mentioned in some papers, emphasising stealthiness and their use by APT malware. Pendergrass et al. [74] state LotL is a malware subtype that is "*often undetected*". These descriptions provide evidence of the need to quantify the evasive capabilities of these techniques against specific AV engines.

In a paper on the malware analysis tool Yara, Cohen [165] describes LotL as a "*trend that has been recently observed in the tactics used by elite threat actors*"; this claim is reinforced by the results of our analysis. Research by Hassan et al. [30] states that APT malware uses "*LotL attack strategies*" to enable persistent campaigns and analyses two campaigns, compared to our less granular analysis of 16,232 samples. Their work also leverages MITRE's ATT&CK [166] by which MITRE defines an excellent taxonomy that describes and classifies most well-known attacks. Numerous LotL techniques are indexed within the MITRE ATT&CK framework. MITRE corporation and their Common Vulnerabilities and Exposures (CVE) are an established authority within the security field. Their inclusion and description of many LotL techniques are an indication that this is a subject worthy of further analysis.

Orthogonal to our research is the analysis and deobfuscation of script-based malware. Malware making use of LotL techniques often uses malicious scripts as payloads. Mitigations against malicious scripts invoked by the `Powershell.exe` binary have been tested by the identification of suspicious behavioural patterns by Ugarte et al. [76]. Hendler et al. [167] apply machine learning to the detection of Powershell malware. Curtsinger et al. [77] propose detection mechanisms for malicious Javascript, analysing the prevalence of malware using other LotL techniques to execute VBscript, Javascript, .HTA and other filetypes. While these papers propose effective detection methods for narrow subsets of malicious payloads delivered, they do not analyse the wider malware ecosystem and how these payloads are triggered by LotL binaries.

Recent papers describe the use of and mitigations against Powershell malware [138, 161], including Li's, that includes a subsection stating the increasing prevalence of LotL techniques. Saad [168] proposes detection mechanisms for malicious Javascript. A paper on data provenance analysis also identifies LotL as a recent prominent evasive malware subtype [75], as does [74] Pendergrass et al, that states it is a malware subtype that is `often undetected`, evidencing the need for practical testing of the evasive capabilities of these techniques versus specific AV engines.

Previous work deals with Windows malware and the analysis of this malware through dynamic behavioural analysis in sandboxed environments. A clear evolution is shown from established early research, that describes dynamic and behavioural analysis on the Anubis platform [24, 25, 28, 169]. As dynamic behavioural analysis is a mature field, more recent analyses focus on particular areas of sandboxed malware's dynamic behaviour, such as discovering APT malware development using cloud sandboxes [29], spearphishing [170] or cryptomining malware [9]. The analysis and possible countermeasures for the evasive capabilities of malware and evasive behaviour within sandboxes has also been studied in depth, including robust assessment of the limitations of sandboxed dynamic analysis. Notable examples being `Sandprint` and `Spotless Sandboxes` that analysed forensic indicators of virtualisation in sandboxes [67, 68, 171].

## 5.3 Motivation: Anti-virus products vs. Living-off-the-Land Techniques

Security researchers have documented many cases where LotL techniques can be used to successfully evade security products [172]. In many of these cases, these LotL binaries are leveraged to proxy the execution of a malicious payload, having it execute under the context of a legitimate looking process, or spawning a new process as a child of a legitimate system process. In some cases, these payloads are executed as a side effect of the LotL binary invocation, while in others it is just the result of its main documented behaviour. Many AV products reportedly fail to properly detect these techniques.

To answer our first research question, we first analysed whether current AV products flag LotL techniques as indicators of malicious behavior. To this end, we first selected a representative set of 10 end-user AV products and simulated attacks by leveraging a reverse shell executed by common well-known LotL based proxied execution techniques.

The AV engines we installed on each host were taken from an industry compiled list of the ten most popular AV engines [173], all of which advertise behavioural analysis capabilities. Moreover, the intention of this study is not to test the detection capabilities of any particular AV product or to compare them against each other, but to determine if a generalised detection gap exists. Although we endeavoured to configure every product to apply its full detection capabilities, these results shall not

be used for comparison of AVs as behavior may vary depending on the environment, targeted end users, and configuration settings.

We installed free trial versions of these AV engines to establish the evasive potential of LotL techniques within the Windows OSs, and paid for Cylance Smart Antivirus is as there was no free version available. These software were only selected where the AV engine fully reflected the capabilities of the product. As in, the full dynamic and static analysis capabilities. Evasion of these AV engines illustrates why malware authors may make use of these techniques, for the purpose of surreptitiously conducting post-exploitation activity on systems. This motivates our analysis of the prevalence of these techniques.

We performed these experiments within networked Windows 10 VMs, with up-to-date local AV products connected to their cloud components. We omit the precise setup details here, but describe them in Section 5.3.1. We leveraged a reverse shell to assess how vulnerable AV systems are to evasion by malware deploying LotL techniques. We deem a reverse shell, capable of allowing remote execution of commands, to be sufficient proof of successful code execution. A remote network connection executing commands is identical to many RAT's functionality. We conducted our experiments by running this reverse shell from different LotL binaries to test whether AV products detect these techniques as malicious. We obfuscated reverse-shell payloads when necessary and used various payload types to test AV's capability to detect the delivery mechanism itself, not the specific payload being delivered through static signatures. The LotL binaries that we leveraged for testing and the specific techniques by which we achieved evasion are described in detail in Section 5.3.2.

Table 5.2 shows the results for this experiment. We can observe that most popular AV engines allowed us to establish a reverse shell and execute commands, using various LotL techniques that are common in commodity and APT malware.

**Responsible Disclosure and Response** We issued a document to all AV vendors concerned, containing in-depth technical results of our evasion tests with above 90 days notice. To assist with remediation, the documents we provided included evidence of how each evasion was conducted. Some vendors patched their detection algorithms and actively engaged with our research; others denied or ignored the responsible disclosure process. Moreover, some AV vendors reported difficulties with detecting some of the documented cases as detection rules are prone to false positives.

| AV | Ftp.exe | Mshta.exe | Wmic.exe | Rundll32.exe | Regsvr32.exe | Bitsadmin.exe |
|---|---|---|---|---|---|---|
| **Avast Premium Security** | | | | | | |
| **Bitdefender Internet Security** | | | | | | |
| **Cylance Smart AV** | | | | | | |
| **Eset Internet Security** | | | | | | |
| **Kaspersky AV** | | | | ✓ | ✓ | |
| **Malwarebytes for Windows Premium** | | | | | | |
| **McAfee Total Protection** | | | | | | |
| **Sophos Home Premium** | | | | | | |
| **Webroot SecureAnywhere AV** | | | | | | |
| **Windows Defender / AMSI** | | | | | ✓ | ✓ |

Table 5.2: AV detections for our initial experiments on Windows 10.

**Experiment Repetition** We repeated our tests on Windows 10 machines, nine months after our initial evasion testing. This allowed us to test if AV vendors had included new heuristic rules in their products to detect LotL binary usage. The detection capabilities of AVs in this second experiment are shown in Table 5.3. We can observe that a number of vendors updated their detection capability. 25 out of 60 of the exact same payloads were detected.

In some of the tests where our reverse shell was detected, we modified the payload (applied a different obfuscation method or ran a different payload) while maintaining the exact same command line arguments for the LotL binaries, in an attempt to establish whether the detection was implemented to match the delivery mechanism or the payload. By leveraging these obfuscated and modified payloads, we successfully executed a reverse shell in 19 of these 25 blocked instances.

These results show that LotL techniques are still a significant challenge for AV vendors. These tools are commonly leveraged by legitimate users in unpredictable ways, and security companies struggle to deploy effective detection strategies without false positives. The following sections will show how these techniques are a prevalent phenomenon in commodity malware, and how this problem should not be overlooked by the security community.

| AV | Ftp.exe | Mshta.exe | Wmic.exe | Rundll32.exe | Regsvr32.exe | Bitsadmin.exe |
|---|---|---|---|---|---|---|
| **Avast Premium Security** | | | | | ✓ | |
| **Bitdefender Internet Security** | | | | | ✓ | |
| **Cylance Smart AV** | | | | | | |
| **Eset Internet Security** | ✓ | | | ✓ | ✓ | |
| **Kaspersky AV** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Malwarebytes for Windows Premium** | | | | | | |
| **McAfee Total Protection** | | | | | | ✓ |
| **Sophos Home Premium** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Webroot SecureAnywhere AV** | | | ✓ | | | |
| **Windows Defender / AMSI** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Table 5.3: AV detections for our repeated experiments (9 months after initial testing).

### 5.3.1 Windows 10 System Preparation for AV Experimental Setup

In this subsection, we describe in great detail the methodology we undertook to answer the research question of '*Can LotL techniques effectively evade commercial AV?*'. We also justify our design decisions.

We took two specific steps to facilitate the testing of evasive the potential of malware on Windows 10 systems.

**Bypassing AMSI** Windows underlying Anti-Malware Scan Interface (AMSI) is a part of the Windows OS. Before installing an AV on each virtual system, we disabled AMSI. While various bypasses that have been historically available for Windows 10 systems have been patched afterwards, an AMSI bypass exploit is available at the time of writing [174].

The purpose of using this approach to disable AMSI is to isolate the detection and prevention capabilities of various engines against LotL binaries. We disable AMSI in all cases but those where we were testing Windows Defender and AMSI specifically. AMSI comes installed by default on all Windows systems, but we wanted to test the detection capabilities vs. LotL of all different commercial engines, not just Microsoft's offering.

**Enabling Unauthenticated Share Access** By default in Windows 10, guest access to shares is disabled. To perform these bypasses this feature was enabled [175]. We accomplished this by running a command as a user and editing group policy to enable insecure guest logons to an SMB share. Alternatively, SMB Server login details can be cached to access them in a similar manner without the editing of registry keys.

Whilst unauthenticated access to an SMB share is not necessary for all tests, it was necessary for those that involve staging from an SMB server. As the LotL payloads we developed involved multiple stages, some of which involved delivery via an SMB share, we enabled unauthenticated guest access to these shares to ensure that our payloads worked.

### 5.3.2 LotL binary based Bypass Techniques

In this subsection, we continue the explanation of how we set up our evasion experiments. The following binaries were selected from the full list of the LOLBAS project, discussed in Section 5.7, to which we contributed several novel attack methods. The binaries that we selected were selected as they the capability for proxied execution via several methods.

As the aim of this section is evaluating LotL techniques evasive capability for commercial AV, we test payloads leveraging these binaries these against the deployed AVs in our VMs. Proxied execution is important as it simulates an attacker achieving remote code execution.

**Ftp.exe** `Ftp.exe` can be used to execute commands from a stored text file containing instructions (*-s:file.txt* parameter). When the file is executed a simple Powershell reverse shell can be spawned.

For some of the AV engines that are more resilient to evasion techniques, the tool Powercat [176] was invoked to achieve a UDP reverse shell. Additionally for 2 engines, the Invoke-Obfuscation [177] toolkit was invoked to obfuscate the contents of Powercat for evasion.

**Mshta.exe** `Mshta.exe` allows the execution of scripts contained within `.hta` files. We found it sufficient to use Mshta to load the `.hta` file from a network share, which would call a Powershell script from VBScript and then execute a reverse shell to evade

AV. Many AV programs were able to identify this payload file as malicious, but when obfuscation was applied (Spookflare [178]), detection was avoided.

**Wmic.exe** `Wmic.exe` is a native binary on Windows systems that can, amongst other malicious uses, execute other binaries. For most of the AVs, evasion is achieved by invoking a simple Powershell reverse shell from `Wmic.exe`. For other AVs, evasion was achieved by executing a shell over UDP using Powercat, instead of TCP. Further obfuscation was necessary to bypass AMSI and Windows Defender, for which Invoke-Obfuscation [177] was also used.

**Rundll32.exe** We use Koadic [179], a tool for post-exploitation, for testing the viability and detectability of `Rundll32.exe`. Koadic provides several command-line parameters to provide to the `Rundll32.exe` binary. Koadic's evasive exploitation leverages the fact that `Rundll32.exe`, when provided with the '*-javascript*' parameter, allows the execution of JavaScript code. Koadic is sufficient to achieve a bypass for most AV engines, for others JS-Rat-Py [180], a similar remote access tool, suffices.

Some AV engines were able to identify the execution of JavaScript by `Rundll32.exe` as malicious, spawning a network connection. To bypass these, we generated a malicious `.dll` with MSFvenom, or in particularly evasive cases, we generated and compiled a reverse shell `.dll` from C++ code.

**Regsvr32.exe** `Regsvr32.exe` binary can be used to execute remote `.sct` files in the *squibblydoo* [181] attack. This attack enables the execution of remotely hosted `.sct` files, containing Javascript code that can execute arbitrary commands such as creating ActiveXObjects. For our repeated experiments, as the `squibblydoo` attack has been patched by Microsoft, we used a bypass that still allows this attack to be conducted, described by Oddvar Moe [182].

For a number of the bypasses, Koadic was used. This tool allows a reverse shell to be executed in the form of JavaScript code. Where Koadic was detected, JS-Rat-Py's Regsvr32 component was not. For others, we used custom code within an `.sct` file.

**Bitsadmin.exe** `Bitsadmin.exe` was used to create Background Intelligent Transfer Service jobs. It is typically used for the transfer of files between different machines. However it has undocumented functionality allowing the execution of binaries or

arbitrary commands. It can execute malicious files and bypass application whitelisting.

`Bitsadmin.exe` was used to create a Powershell reverse shell, by creating a job and resuming it. This Powershell reverse shell was spawned by either calling Powershell directly, or performing this via several layers of obfuscation using an `.hta` file. This `.hta` file calls Powershell via a Visual Basic script within that `.hta` file called by `Mshta.exe`.

## 5.4 Measuring LotL Prevalence

In this section we measure the prevalence of LotL techniques in malware and try to answer the research questions raised, particularly '*How prevalent is the use of LotL binaries in malware?*'. This measurement effort was conducted over 9 separate sub-datasets. In total, we collected 31,805,549 samples from which we obtained 16,048,202 behavioural reports from VT.

Public datasets may vary in distribution, depending on how they were collected. Initially we used a dataset of 44,606,904 samples from which we obtained 22,396,841 behavioural reports from VT. However we omitted 12.8 million of these samples to ensure that we only included samples where provenance could be guaranteed.

### 5.4.1 Dataset Composition

To be as comprehensive as possible, we obtained public and private datasets from different sources. We chose varied datasets to represent possible malware distributions. We included public malware datasets leveraged by the research community throughout the years to ensure completeness and reproducibility. To complement these data sources, we created three additional datasets following three different collection strategies. These datasets are listed in Table 5.4.

**Public Datasets** We included several public and well-known malware datasets to enable reproducibility, conducting pre-processing where necessary to isolate PE binaries. We included these larger scale datasets to represent commodity malware used in previous research efforts [9, 97]. We included a collection from the VirusShare [184] corpus of hashes of unpacked binaries, in addition to hashes identified as PE hashes by existing pre-processing of files. We included the Ember dataset [183] of Windows malware PE files, released for training machine learning models. We also leveraged a

| Type | Dataset Name | No. Of Hashes | No. Of Behavioural Reports | No. Of Crash Reports | No. Of Blank Reports |
|---|---|---|---|---|---|
| Public | Ember [183] | 1,235,190 | 612,400 | 56,339 | 113,021 |
| | Ember Benign | 740,679 | 158,763 | 10,364 | 76,320 |
| | VirusShare [184] | 18,176,364 | 8,639,474 | 234,416 | 2,027,913 |
| | Vx Underground [185] | 394,383 | 102,541 | 6,550 | 28,952 |
| | Georgia Tech [186] | 8,070,223 | 5,095,615 | 285,134 | 281,451 |
| | MalShare [187] | 2,903,350 | 1,277,507 | 66,319 | 176,701 |
| Private | APT | 16,232 | 7,668 | 336 | 2,081 |
| | Yara | 31,840 | 31,834 | 436 | 50 |
| | VirusTotal Balanced | 237,288 | 122,400 | 10,270 | 16,513 |

Table 5.4: Constituent datasets and their behavioural reports.

feed of malware executables, collected and distributed by Georgia Tech [186]. Finally, we also included samples from VX-Underground [185] and Malshare [187], both well known public datasets.

**VirusTotal Balanced Dataset** We gathered 237,288 hashes from VT. We attempted to balance the presence of families and variants to avoid over-representing certain prolific families. Malware feeds like VT usually distribute large numbers of variants of the same polymorphic families [2]. The prevalence of a given malware family in such a feed is not necessarily related to its freshness, impact, or prevalence in the wild. For this reason, we used a custom approach to measure AV label similarity. This approach is based on the min-hash algorithm applied to normalised AV labels for samples with at least 20 positives. In order to normalise labels, we leveraged the pre-processing code in the AVClass [12] project. AVClass is a tool that facilitates extraction of the family name that a sample belongs to, according to the labels provided by the different AV vendors.

By leveraging this sample selection method, we avoided the over-representation of prominent polymorphic families (such as worms or file infectors like Virut or Allaple) or identical variants of the same family. The removal of these variants balanced the representation of every family in the dataset.

We collected a total of 1.5 million unique PE executable files from VT in the course of 6 days (December 10th to December 16th, 2019), and finally selected a subset of 237,288 diverse samples. We make these hashes available to enable further research.

**APT Malware** We gathered a dataset of APT malware following a method similar to the dataset paper *dAPTaset* [188]. We processed HTML pages and pdf files and extracted all malware hashes contained within these files. We collected APT reports from various threat intelligence companies, government agencies and other entities that release threat intelligence data online, mostly from the APTnotes [189] repository. We then processed these reports, creating an aggregated collection of APT malware hashes, which we make public.

**Yara Rule Match Malware** We collected an additional dataset by leveraging the live hunting service from VT. To this end, we deployed three Yara [165] rules we wrote to detect the use of LotL binaries based on the behavioural footprint of the samples. We make these rules public in our repository to facilitate further research. Livehunt ran the provided Yara rules against all malware binaries uploaded to the VT platform. We used Livehunt to identify new malware hashes uploaded to VT that matched the behavioural characteristics of malware using LotL techniques. We subsequently inserted these malware hashes into our database for augmentation and analysis.

Our Yara rules import an extension of the default Yara libraries that integrates the use of the Cuckoo module [190]. The Cuckoo module in Yara allows rules to match artefacts of the behavioural report. To this end, we leveraged the *file_access* parameter. Whenever a process creates another process, the report shows an entry indicating that the image (PE file) of the created process has been accessed. This artefact allowed us to preselect candidate samples by finding occurrences of the file system locations of LotL binaries being accessed.

### 5.4.2 Dataset Characterisation

To expand upon the nature of the datasets that we analysed, we illustrate their characteristics and identify the differences between them.

**Number of Positives** Within VT, every sample is scanned by a number of different AV engines. We define the 'Number of Positives' variable as the number of engines

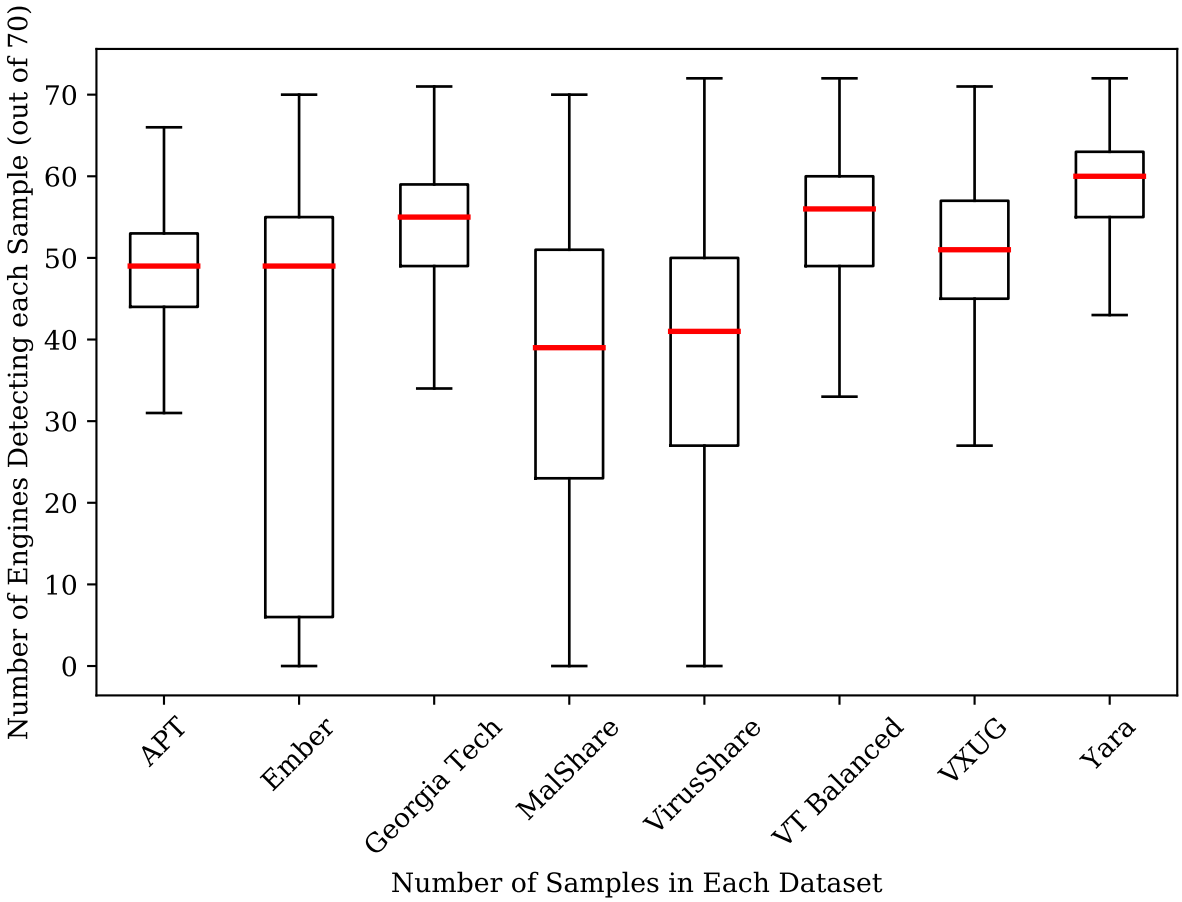



Figure 5.1: Number of Positive Detections Distribution

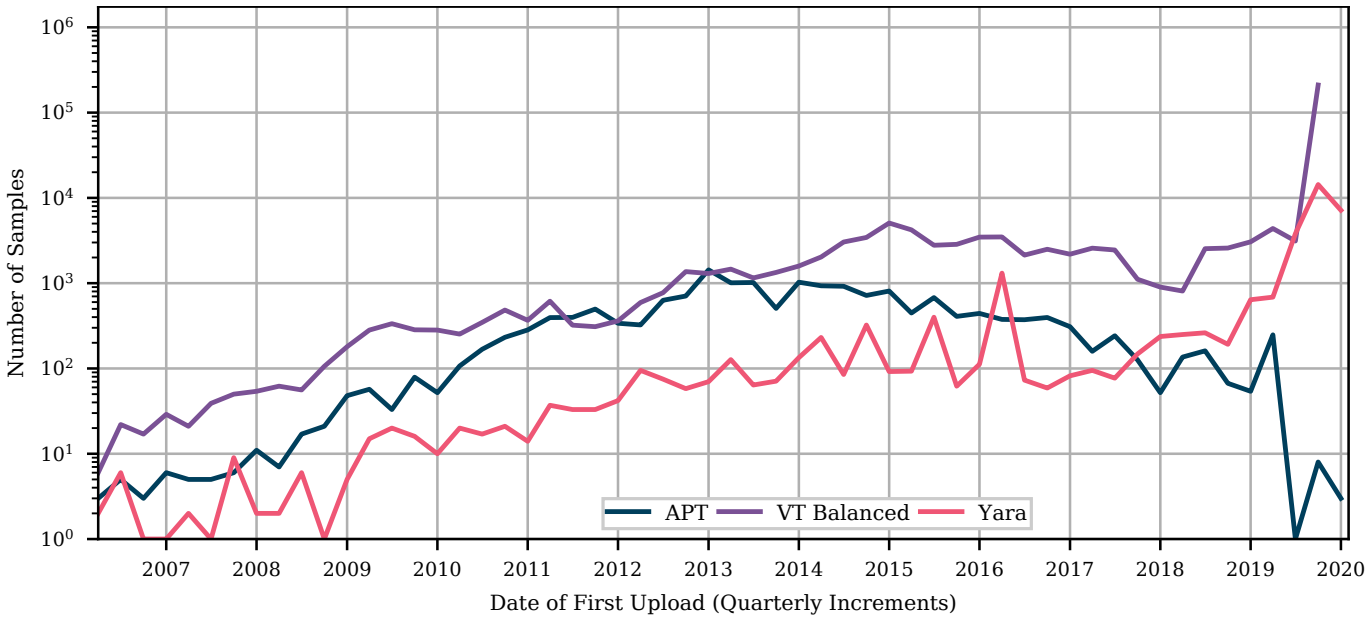



Figure 5.2: First Seen Distribution for Private Datasets

that identify a sample as malware. Figure 5.1 plots the number of positive matches found by various AV engines to characterise the dataset. A low amount of detections may be due to either evasive capabilities for that sample or because a binary is benign.

**First Seen Date** The *first_seen* field is a value within VT reports which contains the date that this malware was first uploaded to the VT platform. We plot the *first_seen* date for each dataset in Figure 5.2 and Figure 5.3, for which the sample range is mostly within the period from 2015 to 2020.

### 5.4.3 Dataset Quality

Rossow et al. [20] describe flaws and limitations in existing malware analysis papers and prudent steps to take in order to avoid biases. We adhere to the three critical assessment criteria (correctness, transparency and realism) as far as is possible in our research.

**Correctness** We removed goodware as recommended by Pendlebury et al. [19], to ensure reliability. We accomplished this by separating elements of the Ember dataset that are labelled as known goodware. We did not remove samples with zero detections, as this may be evasive malware and malware uses LotL techniques partially for the purpose of achieving evasiveness.

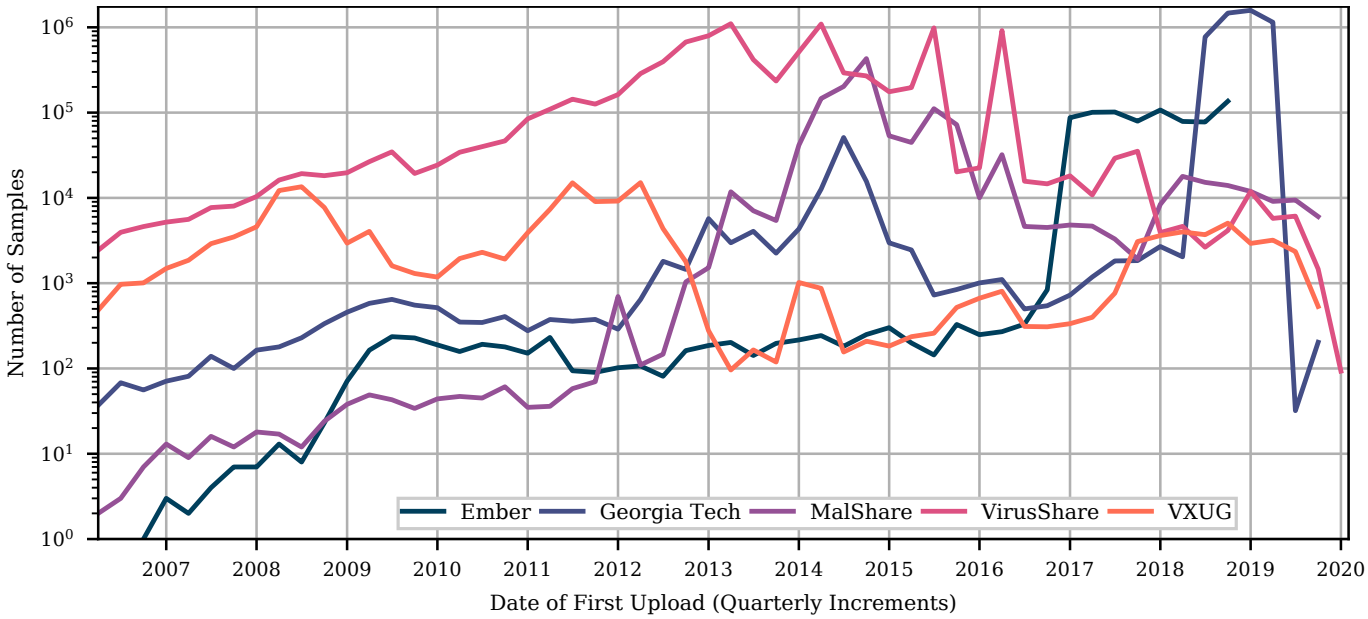



Figure 5.3: First Seen Distribution for Public Datasets

We removed hashes from our dataset if there were duplicates in sub-datasets or if there was incomplete data available for a particular hash. We also removed several datasets, reducing our initial count of over 45,000,000 malware hashes and associated reports to a distilled selection of datasets.

**Transparency** We describe all datasets and their provenance and characteristics as thoroughly as possible The virtual environment in which these samples were detonated is VT's fork of Cuckoo sandbox and similar derivatives.

**Realism** The aim of dataset selection is to analyse samples that are a representative cross-section of modern malware. We included several well-known public malware datasets as a realistic and typical sample. We also included and focused our analysis on several datasets that we collected, to show the malware landscape from different representative perspectives.

### 5.4.4 Analysis Pipeline

Once we had gathered the different datasets composed of Windows PE binaries, we analysed the behaviour of the samples. To facilitate this we implemented an analysis pipeline consisting of three phases: data collection (described in the previous section), data augmentation and analysis.

We augmented the data for the collected malware hashes by querying VT's public API and adding the returned additional data to our database. We added this data to provide contextual information about the malware under analysis as well as enabling the filtering of malware by characteristics. This additional data includes detection by different AV products, the *first_seen* date and behavioural reports from dynamic analysis of the binary. Each behavioural analysis report contains a list of shell commands and processes executed or created. We additionally ran AVClass [12] against each sample to add a family classification. Finally, we processed all the behavioural reports to identify the samples that apply any LotL technique among all the candidates. We provide a detailed description of the process followed in Section 5.4.5.

### 5.4.5 LotL Technique Identification

We processed all the collected behavioural reports by using pattern matching to identify invocations of LotL binaries within the execution of the malware. For this, we parsed the following report indicators:

**Shell Commands** Shell commands are commands that a malicious binary executes in the shell of the host OS. Shell command logs can show executions of a system binary by a reference to its absolute path. The command prompt for Windows also includes a number of aliases such as *reg* for `Reg.exe`, by which system binaries can also be executed.

**Processes** Process logs specify system binaries executed by a malware sample. Parameters provided to the executed binary are also contained within process logs in behavioural reports.

In our analysis, we assessed that a sample uses LotL techniques if its behavioural report contained at least one execution of a LotL binary. We recorded the details of every LotL binary execution with its parameters and inserted them into the database. We then analysed the parameters of these malware samples to determine the most frequent parameter types and execution purposes in every dataset. More specifically, we identify these two separate types of binaries:

**Default System Binaries** This set of binaries comprises those native to the system according to the LOLBAS project, together with those present in the System32 folder of a Windows 10 system. To identify the execution of these binaries, we conducted string matching over the process logs.

**Installed Signed Binaries** We extended our selection of LotL binaries with those not installed by default, mentioned in the 2017 Symantec LotL report and the SysInternals tool suite [191], in addition to those listed in the LOLBAS project.

**Pattern Matching Refinement** We iteratively validated a cross-section of each result set, identified misclassified cases and removed them from our results database, refining our pattern matching approach. We conducted this until all identified LotL commands were properly categorised and mapped to an execution purpose. Additionally, as shown in Table 5.4, we identified anti-VM malware artefacts or erroneous reports and do not include them within our measurements.

We also found a number of LotL binary executions where no parameters were provided. These occur in the case of either an execution of a binary without parameters, or process hollowing, replacement or injection, all of which are out of scope for our analysis. To justify removal, we manually validated a small number of these samples.

Additionally, we identified several system binaries being executed in the reports that were artefacts of the sandbox and are not directly related to LotL technique usage. One such forensic artefact we removed is the execution of `Explorer.exe`, that is typically a remnant of the execution of malware samples by the sandboxed environment. We removed executions of `Explorer.exe` with the parameter of the absolute address of *C:* and a suffix of the SHA256 hash of the malware sample in question. We did not remove forensically relevant executions of `Explorer.exe` to open webpages or other directories.

We also removed instances of `Verclsid.exe`, after manually verifying a subset of the 64,385 executions in our dataset. We removed these instances as they were simply forensic artefacts showing verification of COM objects before they were instantiated by Windows Explorer. Whilst the malicious usage of this tool to spawn processes [192] is known, we did not observe any occurrences of this behaviour in our dataset.

Finally, we found that many legitimate binaries from the Ember benign dataset were spawning `Msiexec.exe`. We manually inspected these cases and determined them to be legitimate installer files packaged with the MSI installer tool. We discarded these cases as they do not represent LotL binary usage.

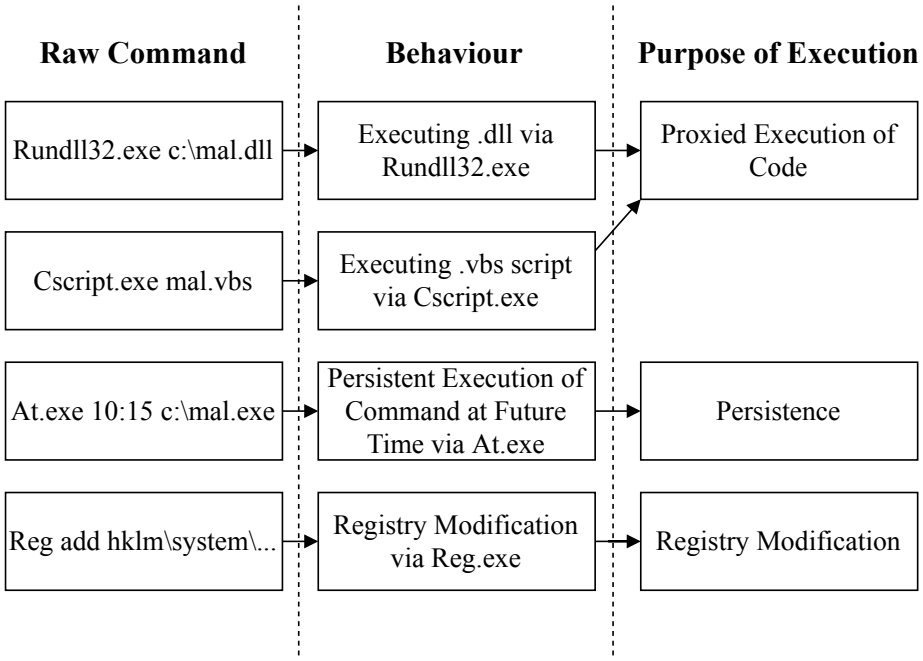



Figure 5.4: Examples of raw commands being mapped to behaviours, which are in turn mapped to execution purposes. This illustrates the classification heuristic we applied to identify an execution purpose.

### 5.4.6 Parameter Analysis to Identify Execution Purpose

To identify the purpose of LotL technique executions, we observed the parameters provided by malware samples. Figure 5.4 illustrates the mapping of four process executions. This mapping was conducted across all datasets by identifying individual execution purposes, such as executions of `Net.exe` with the *stop* parameter as task stopping. After individual commands had been mapped to an execution purpose, we then selected all matching executions for that binary. We repeated this across all system binary executions until every execution was classified as belonging to a specific execution purpose or removed as a misclassified case (refining our pattern matching rules). Following this approach, we grouped the parameters by purpose into nine separate categories. Three of these categories are related to execution:

- **Proxied Execution.** Malware using a LotL binary to abstract execution of other code *e.g.,* `Mshta.exe` executing *.hta* files and `Rundll32.exe` executing *.dll* files.
- **Persistence.** Malicious code achieves persistence if it configures or modifies the system to execute a command or stored job at a future point in time. This is typically resultant from a system configuration change by a malware sam-

ple. *e.g.,* `Sc.exe` with the *create* parameter, `Bitsadmin.exe` with the */create* parameter or `Schtasks.exe`/`At.exe` with a datetime parameter.

- **Delayed Execution.** Using a LotL binary to delay execution *e.g.,* `Ping.exe` executing *-n*, followed with a number, and an `&` to trigger the execution of another binary.

The following three categories, instead, relate to the modification of underlying system components. Malware typically engages in this behaviour in order to enable further propagation or action on objectives on a machine.

- **Firewall Modification.** Modifying firewall configuration *e.g.,* `Netsh.exe` with the *Firewall* parameter and successive commands.
- **Registry Modification.** Modifying registry settings *e.g.,* `Reg.exe` with an *Add* or *Delete* parameter and a registry location.
- **Permissions Modification.** Modifying permissions of a file *e.g.,* `Cacls.exe` with a parameter that includes an absolute file location.

Finally, we distinguish the following three categories that are not related to execution or system modification:

- **File Opening.** Opening a file, via binary execution *e.g.,* `Explorer.exe` followed by a relative or absolute file location.
- **Reconnaissance.** Elicitation of local or remote configuration for lateral movement, leveraging a LotL binary *e.g.,* `Net.exe` with *user localgroup administrators* or execution of `Ipconfig.exe`, outputting to a file.
- **Task Stopping.** Surreptitious stopping of another process or service using a LotL binary *e.g.,* `Taskkill.exe` executed with another process name as parameter. In many cases, the goal of these executions is to disrupt system security services.

## 5.5 Measurement Results

We followed the methodology described in Section 5.4 to evaluate the prevalence and nature of LotL techniques, and to answer the research questions raised for this study.

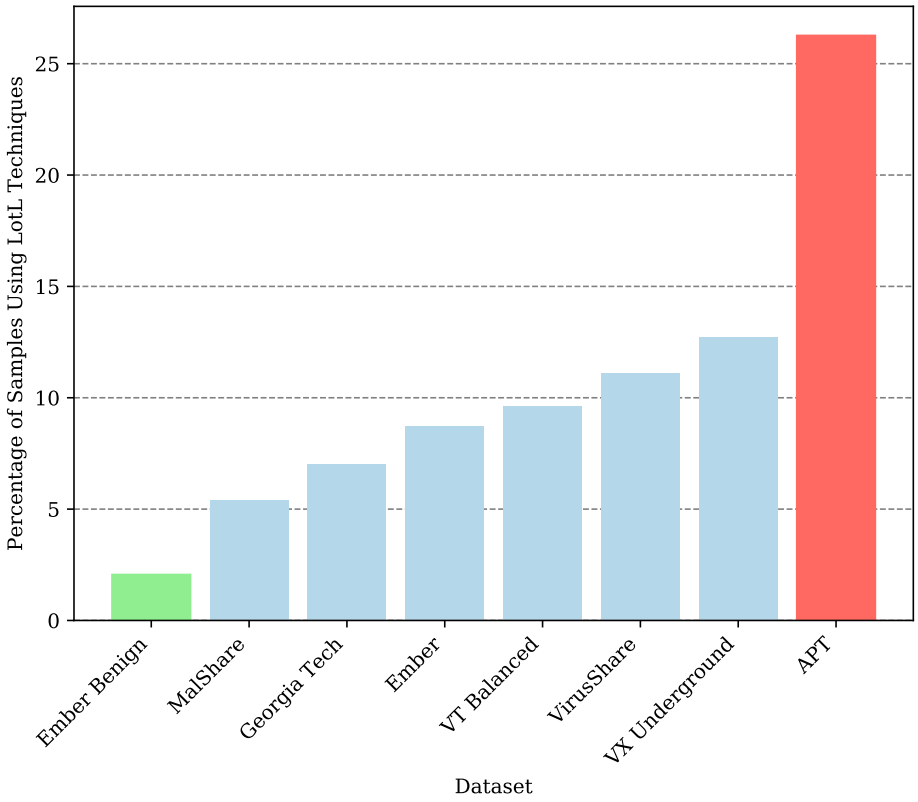



Figure 5.5: Relative prevalence of samples using LotL techniques per dataset.

### 5.5.1 Prevalence of LotL Techniques in Commodity Malware

The largest datasets we used were taken from public datasets and those provided by industrial collaborators. As these datasets show a wide cross-section of modern commodity malware, we are able to analyse them to extract meaningful conclusions.

**Relative Prevalence Between Datasets** Figure 5.5 shows the percentage of malware samples that use LotL techniques in every dataset. If we omit Ember's benign dataset and the APT malware dataset, we can observe that between 5.42% and 12.72% of commodity malware samples used a LotL technique at least once. Within our VT balanced dataset, we can observe that 9.6% of samples leveraged this type of technique. These numbers show that LotL techniques are a relatively widespread phenomenon and not an anecdotal occurrence in specific malware samples or families.

Interestingly, APT datasets employed LotL techniques significantly more frequently than commodity malware. 26.26% of the APT dataset made use of LotL techniques,

more than twice that of comparable commodity malware datasets. Our findings confirm the assertions of many reports that state that APT campaigns use LotL techniques due to the increased sophistication and evasive requirements of state-sponsored malware authors. Section 5.5.3 presents deeper findings about the APT dataset, while the rest of this section focuses on commodity malware. We excluded the Yara dataset from this figure given its different nature. In this particular case, 47% of the files retrieved by our Yara rules were later assessed to have used some kind of LotL technique. This result is also interesting, as it means that half of the times that a malicious sample accessed a system binary, it did so to leverage it as a LotL technique, instead of for other purposes.

Another notable finding is that in the Ember benign dataset, 2.05% of the samples show executions of LotL techniques. This number is significantly lower than the 9.41% average prevalence of LotL techniques across regular commodity malware datasets. We provide additional evidence for this observation in Section 5.5.2.

**Most Frequently Used LotL Binaries** Table 5.5 shows the most commonly executed LotL binaries in our commodity malware datasets. We can observe that commodity malware executes some system binaries more frequently than others. These numbers also show a significant variability depending on the specific dataset distribution, as each of the datasets might have been collected in a different way and at a different time. If we look at the VT balanced dataset, the most commonly used binaries were `Reg.exe`, `Nslookup.exe`, `Regasm.exe`, `Runas.exe`, `Schtasks.exe`, and `Sc.exe`. Some of these binaries were used for system administration tasks such as editing the registry or creating scheduled tasks. Others were used to change or elevate privileges, or to enable network activities. It is notable that certain binaries show different distributions across datasets. This means that although LotL techniques are a widespread approach among malware writers, they are used in an heterogeneous way among malware families or malware types.

In our results, there are well-known use cases for which malware uses these binaries. For instance, `Reg.exe` and `Regedit.exe` are executed for the purpose of modifying the registry; `Schtasks.exe`, `Sc.exe` and `At.exe` are typically used for persistence; `Rundll32.exe`, `Mshta.exe` and `Wscript.exe` are used for proxied execution of code.

| **System Binary** | **Frequency of LotL Binaries By Dataset** VTB | Ember | GT | MS | VS | VXU |
|---|---|---|---|---|---|---|
| Reg | 15.49 | 7.07 | 42.16 | 4.26 | 11.10 | 5.88 |
| Nslookup | 15.14 | 4.49 | 0.58 | 4.55 | 0.00 | 0.70 |
| Regasm | 9.93 | 1.25 | 0.00 | 0.44 | 0.00 | 1.60 |
| Runas | 7.84 | 0.39 | 6.51 | 0.00 | 0.00 | 0.00 |
| Schtasks | 7.50 | 3.78 | 3.66 | 0.26 | 0.00 | 4.27 |
| Sc | 5.87 | 6.08 | 1.44 | 1.17 | 14.08 | 10.00 |
| Wscript | 3.31 | 1.95 | 1.59 | 0.61 | 2.36 | 5.50 |
| Rundll32 | 3.16 | 4.70 | 2.63 | 14.00 | 5.25 | 8.62 |
| Regsvr32 | 2.99 | 3.62 | 2.99 | 8.58 | 4.47 | 5.87 |
| Attrib | 2.83 | 4.63 | 15.59 | 0.32 | 1.18 | 3.63 |
| Net | 2.52 | 8.45 | 4.14 | 1.89 | 9.85 | 9.48 |
| Ping | 2.14 | 27.19 | 5.61 | 1.31 | 5.06 | 4.81 |
| Taskkill | 1.49 | 2.39 | 0.67 | 4.04 | 3.40 | 6.30 |
| Netsh | 1.40 | 3.37 | 0.62 | 2.49 | 5.19 | 6.54 |
| Timeout | 1.36 | 0.74 | 0.56 | 0.33 | 0.00 | 1.13 |
| Wmic | 1.27 | 0.62 | 0.50 | 36.14 | 9.63 | 0.55 |
| Mshta | 1.09 | 4.68 | 0.76 | 10.72 | 0.74 | 0.60 |
| Cacls | 0.89 | 0.00 | 0.48 | 0.23 | 0.80 | 3.11 |
| Regedit | 0.52 | 1.55 | 0.00 | 0.00 | 6.97 | 2.79 |
| Tasklist | 0.00 | 0.00 | 0.00 | 0.00 | 2.60 | 0.85 |
| Cscript | 0.00 | 0.88 | 3.96 | 0.00 | 1.52 | 0.00 |
| Explorer | 0.69 | 0.69 | 0.41 | 0.00 | 1.91 | 6.0 |
| Msiexec | 0.55 | 1.78 | 1.78 | 0.57 | 0.58 | 0.00 |
| Vssadmin | 0.00 | 0.81 | 0.00 | 0.58 | 0.00 | 0.00 |

Table 5.5: Most frequently used binaries for each dataset (percentage of the number of distinct sample-binary executions observed). 5 most common highlighted in grey.

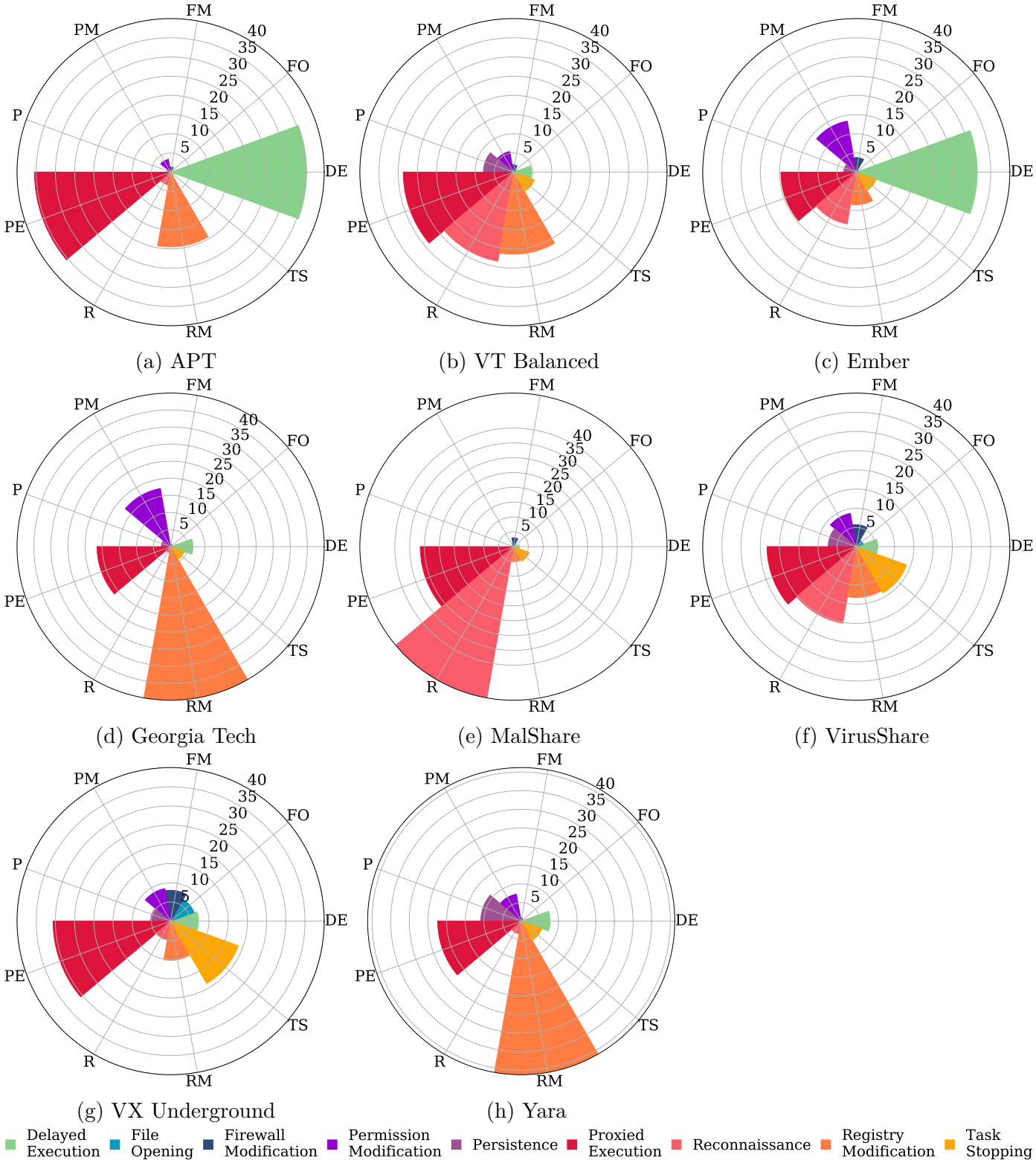



Figure 5.6: Comparative execution purposes for each dataset by percentage.

**Parameters and Execution Purpose** We also identified the execution purpose for the occurrences of the most commonly used LotL binaries by parsing their parameters. Figure 5.6 shows the resulting distribution of execution purposes. The most frequent purposes are proxied execution, reconnaissance, task stopping and modification of the registry. We can also observe that there is a significant variability between datasets, as expected from the results shown in Table 5.5. Nevertheless, these results show that malware uses LotL binaries not only to surreptitiously execute other code, but also to modify the underlying OS via registry modification, to enable lateral movement via reconnaissance, or to avoid other software running in the system (e.g., many malware families try to stop security software applications when they start running on a system).

**Malware Families** To answer our research question of '*Which malware families and types use LotL binaries most prolifically and how does their usage differ?* ' we analyse how different families use LotL binaries . We identified a number of malware families that are notably prolific (i.e., those for which nearly all of their samples used one or more LotL techniques), while others had a very low LotL binary adoption. We limited this classification to families that contained at least 100 samples, to prevent bias caused by potentially under-represented families with a small number of samples. Figure 5.7 shows the number of families, as reported by AVClass, for which a given percentage of their samples leveraged a LotL technique. We can observe that a high number of families did not use LotL binaries, while a small number of families presented high prevalence of these techniques. This means that while there is a significant adoption of LotL overall, the usage of these techniques is concentrated in a limited number of families.

### 5.5.2 Comparison of Benign and Malicious Samples

We have observed that LotL techniques are a widespread phenomenon both in commodity malware and APT campaigns, and although there might be some differences across malware types or families, it is clear that the security industry should not overlook this threat. With this in mind, we raise another question: *Why do some security products have difficulties detecting these techniques?*

At the beginning of this chapter in Section 5.3 we confirm that many AV products cannot effectively detect the usage of these binaries, and some responses to our responsible disclosure messages suggest that detection mechanisms could be very prone

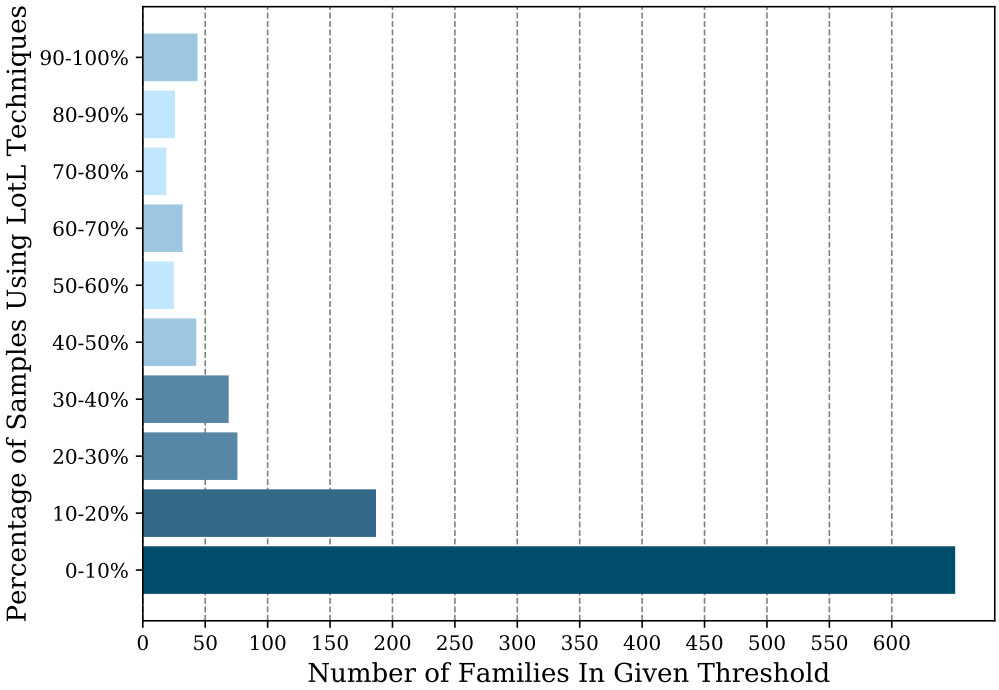



Figure 5.7: Distribution of families using LotL techniques.

to false positives. In this section, we compare the usage of LotL binaries in legitimate software. To this end, we leveraged the benign software dataset from Ember described in more detail in Section 5.4.3, and our VT balanced dataset.

Table 5.6 shows the most commonly executed binaries in the benign dataset. `Regsvr32.exe`, `Sc.exe` and `Rundll32.exe` are among the most commonly executed binaries, all of which are used for the purpose of executing code. We can notice that the most commonly executed binaries for malware (see Table 5.5) and benign software differ significantly.

To answer our research question of '*What are the overlaps and differences in the behavior of legitimate and malicious binaries - How would this affect detection by heuristic AV engines?*' Figure 5.8 shows samples that leverage a LotL technique, grouped by execution purpose. We observe that while some execution purposes are more prevalent in malicious binaries than in benign binaries (e.g. reconnaissance, persistence and registry modification), there is an opposite trend for others. For instance, we can notice that the most frequent purpose in both malware and benign software is proxied execution. This finding correlates with our observations in Section 5.3, where we tested several AV solutions to find out if they detected the proxied execution of a reverse shell.

| System Binary | Samples | Percentage |
|---|---|---|
| Explorer | 230 | 12.62% |
| Regsvr32 | 190 | 10.43% |
| Sc | 148 | 8.12% |
| Rundll32 | 128 | 7.03% |
| Taskkill | 99 | 5.43% |
| Ping | 71 | 3.90% |
| Net | 66 | 3.62% |
| Mstsc | 58 | 3.18% |
| Attrib | 58 | 3.18% |
| Regedit | 40 | 2.20% |

Table 5.6: Top 10 most frequently used binaries in benign dataset (number of samples and percentage of the number of distinct sample-binary executions observed).

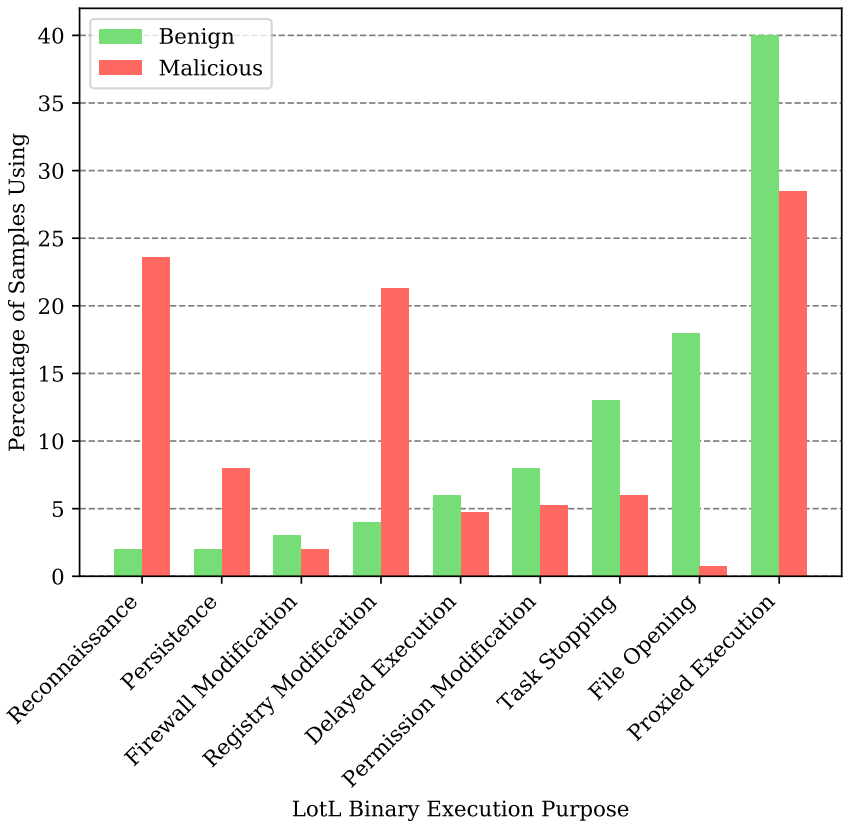



Figure 5.8: Execution purpose of benign and malicious samples from Ember and Ember Benign datasets.

| System Binary | Samples | Percentage |
|---|---|---|
| Ping | 493 | 25.96% |
| Rundll32 | 228 | 12.01% |
| Reg | 212 | 11.16% |
| Wscript | 194 | 10.22% |
| Xcopy | 122 | 6.42% |
| Net | 77 | 4.05% |
| Tasklist | 48 | 2.53% |
| Ipconfig | 47 | 2.47% |
| Expand | 44 | 2.32% |
| Systeminfo | 41 | 2.16% |

Table 5.7: Top 10 most frequently used binaries in APT dataset (number of samples and percentage of the number of distinct sample-binary executions observed).

The results show that these behavioural patterns pose a significant challenge for creators of heuristic AV algorithms. This is also reflected by the response of some vendors to our responsible disclosure process, where one of the claims suggested that the false positive risk was significantly high. The finding also simultaneously illustrates that not all LotL binaries or purposes are equally prevalent in benign and malicious software, and therefore AV vendors still have the opportunity to create heuristic rules to identify the types of techniques that are not as prevalent in benign software. This is exemplified by one vendor's statement that *"The risk of false positive is too high"*, when attempting to detect malicious behaviour undertaken using LotL techniques and differentiate it from legitimate behaviour.

Distinct differences also exist regarding LotL binary executions of malicious software and benign software in the EMBER dataset. These differences may aid AV vendors in the development of heuristic detection algorithms. Delayed execution, reconnaissance, persistence, permissions modification and registry modification are all more common in the malicious dataset.

### 5.5.3 Prevalence of LotL techniques in APT Malware

In the previous section, we observed how LotL techniques are significantly more frequent in APT malware than commodity malware (see Figure 5.5). For this reason, in this section we focus on APT malware and measure several aspects. First, we show the particular APT campaigns with the highest percentage of samples that use LotL binaries, and we group them based on threat intelligence reports. We also enumerate

| **APT Campaign** | **Percentage** | **LotL Binaries** |
|---|---|---|
| Havex | 100.00% | Rundll32 |
| Hurricane Panda | 100.00% | Msiexec |
| El Machete | 100.00% | Schtasks |
| Regin | 100.00% | Dllhost |
| Lazarus | 100.00% | Net, Netstat, Ping, Reg |
| Keyboy | 100.00% | Net, Rundll32 |
| Keyboy | 100.00% | Rundll32 |
| Black Vine | 94.39% | Net, Ping, Reg, Regsvr32, Rundll32 |
| Roaming Tiger | 89.47% | Expand, Powershell |
| WIRTE Group | 80.00% | Rundll32, Schtasks, Wmic |
| APT28 Zebrocy | 77.78% | Reg, Tasklist |
| Magic Hound | 75.00% | Attrib, Taskkill, Wscript |
| Hangover | 71.86% | Attrib, Cscript, Findstr, Net, Reg, Wscript |
| APT28 Zebrocy | 66.67% | Mshta, Wscript |
| Lotus Blossom | 66.67% | Net, Rundll32 |
| APT27 | 63.64% | Msiexec |
| Subaat | 60.42% | Attrib, Ping, Reg, Regasm |
| Dimnie | 54.36% | Ping, Rundll2 |

Table 5.8: APT campaigns (from threat intelligence campaign reports) with over 50% of samples using LotL binaries.

the LotL binaries that APT malware leverages most often.

Table 5.7 lists the most common LotL binaries that we observed in the APT dataset. We can observe that that APT malware uses many of the same binaries as commodity malware, with `Ping.exe` as the most frequent one. This binary is often used for subtly delaying execution, with '*localhost*' as the most frequent destination. `Rundll32.exe` and `Wscript.exe`, tools for proxying the execution of malicious code, are used by many samples; substantially more than they are within commodity malware datasets. The reconnaissance tools `Ipconfig.exe`, `Net.exe` and `Tasklist.exe` are also used by a number of different samples. `Reg.exe` is commonly executed by APT malware, but more infrequently than in commodity malware. We observe that APT malware mainly uses LotL techniques for delayed and proxied execution, as well as reconnaissance. This may in part reflect that APT malware aims to enable stealthy lateral movement and exfiltrate data.

**APT Campaigns** Table 5.8 displays some of the APT campaigns we identified as using LotL techniques most prolifically, based on threat intelligence reports where behavioural data was available. It is notable that several APT groups, such as APT28 and Keyboy, have multiple campaigns and samples. This indicates that APT groups that use LotL techniques use them across multiple campaigns.

## 5.6 Case Studies

In this section we investigate and describe two recent ransomware families from our dataset: Gandcrab and Cerber. Ransomware is one of the most prevalent threats nowadays due to how profitable it is for malicious actors [193]. We chose these families because they heavily rely on LotL techniques of different types and execution purposes and illustrate how a malware sample can chain different LotL techniques to achieve its goals. We executed samples from these families to observe and document their behaviour at different phases of the intrusion killchain. We also describe two APT malware groups, Turla and GreyEnergy. APT group Turla shows an evolution in the adoption of LotL techniques over time, while GreyEnergy [194] was involved in highly disruptive attacks to the Ukranian power grid. We manually analysed the use of LotL binaries in these samples.

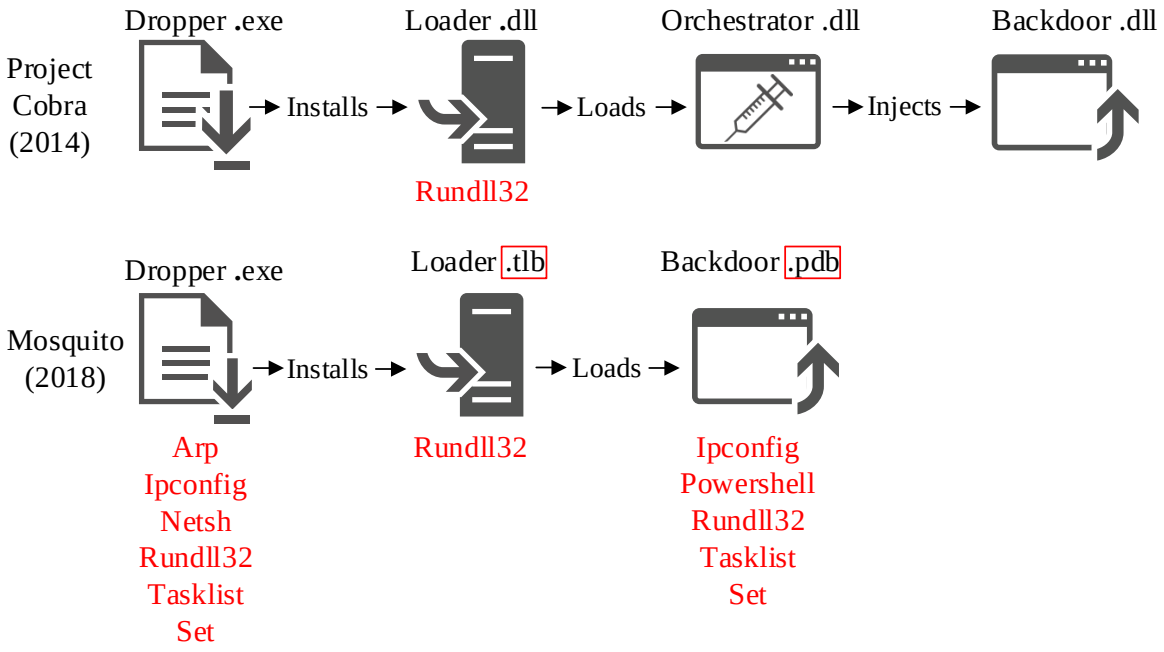



Figure 5.9: Increasing adoption of LotL techniques by APT group Turla. Shown in two successive modular highly evasive trojans, Cobra (2014) [195] and Mosquito (2018) [196].

**Gandcrab** Gandcrab is a ransomware family that operated from late 2018 to early 2019, in several different variants. We executed Gandcrab ransomware samples in a local sandboxed environment and recorded the executed commands. We observed the following LotL binaries being used by Gandcrab samples:

```
C:\Windows\System32\cmd.exe /c vssadmin delete shadows /all /
    quiet
```

```
C:\Windows\System32\cmd.exe /c wmic shadowcopy delete
```

Many other ransomware families, such as *Petya* use the LotL binary `Vssadmin.exe`, with the parameters *delete shadows /all /quiet*, to silently delete backups. Functionally *shadowcopy delete* provided as parameters to `Wmic.exe` accomplishes the same effect.

We observed other samples in the family using the `Nslookup.exe` binary, passing a parameter of '*gandcrab.bit*' or a similar domain, with no further functionality. This subroutine is repetitively executed in attempts to contact command and control servers for Gandcrab [197] that were no longer online and responding to the query.

**Cerber** Cerber, like Gandcrab, is a notably successful ransomware that has several variants. Apart from proxied execution via LotL binaries, we observed other evasive

techniques in use by Cerber, such as process injection.

Unlike other ransomware families, Cerber does not delete shadow copies. However, we noticed an especially complex proxied execution chain. We observed that `Wscript.exe` and `Rundll32.exe` execute a malicious `.vbs` and `.dll` file, sequentially:

```
C:\Windows\System32\WScript.exe C:\Users\admin\enYXval36C\38
    oDr5.vbs

C:\Windows\System32\Rundll32.exe 8ivq.dll arzy949
```

We also observed that Cerber delays the execution via `Ping.exe` and modifies the registry to achieve persistence. The addition of a malicious `.vbs` file to the `HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce` registry key means this malicious script is run every time a user logs on.

```
C:\Windows\System32\cmd.exe /c ping 127.0.0.1 && reg add
    HKEY_CURRENT_USER\Software\Microsoft\Windows\
    CurrentVersion\RunOnce /v enYXval36C /t REG_SZ /d C:\
    enYXval36CenYXval36C\enYXval36C.vbs /f
```

We also note that some Cerber samples invoke `Mshta.exe` to open the ransom note.

```
C:\Windows\System32\mshta.exe ransom.hta
```

**Turla and GreyEnergy** We executed the GreyEnergy [194] dropper malware locally (used for attack campaigns on the Ukranian power grid) and recorded the following shell commands executed using Process Monitor [198]:

```
C:\Windows\SysWOW64\Rundll32.exe {64F97CDC...FAB40CA\}.db #1
    #1}

C:\Windows\System32\cmd.exe /c (ping localhost >> nul & del
    $path\grey3.exe >> nul)
```

The execution of `Rundll32.exe` by GreyEnergy with the parameter of a `.db` file causes the proxied execution of a malicious `.dll` file, given a `.db` suffix to appear more innocuous. This technique is also used by the Mosquito malware visible in Figure 5.9, however Mosquito masks malicious `.dll`s with the use of `.tlb` and `.pdb` extensions to the files. Execution of `Ping.exe` is used to check internet connectivity or to act as a stealthy alternative to the sleep command in many programming languages.

Figure 5.9 also illustrates how the Turla APT group has adopted more LotL techniques in its recent malware campaigns, and exemplifies how some malicious actors are evolving to conceal their actions.

`Rundll32.exe` execution by GreyEnergy is also made persistent by having its location stored in a startup folder that executes every time the system starts. The execution purpose of `Ping.exe` by the second command of the GreyEnergy dropper is to delay the deletion of the initial dropper file. The dropper deletes itself to remove forensic evidence.

## 5.7 Main Takeaways and Discussion

Our results confirmed that LotL techniques are not a negligible phenomenon. While several technical articles cover this topic, our systematic measurement went one step further in facilitating the understanding of the adoption of this technique by commodity malware. In this chapter, we examined several different malware datasets and conducted an AV evaluation experiment, wherein we confirmed low detection rates for several documented LotL techniques. Given these results, we can distill the following conclusions and takeaways:

- Almost every popular AV product we tested had difficulties detecting malicious usage of LotL binaries. Even after responsibly disclosing these issues to each vendor, only a fraction of them implemented successful detection mechanisms. Some of these vendors implemented detection mechanisms for the specific malicious payloads we implemented, but not the delivery mechanism itself. Others reported that it is challenging to implement such countermeasures due to a prohibitive false positive risk.

- To correct this detection gap, we worked with AV vendors directly to implement detection capabilities. For instance, Kaspersky implemented detections resultant from our responsible disclosure process, such as `Trojan.Win32.Lolbas.btad.exec`, `Trojan-Spy.Win32.Agent.ftps` and others.

  We also release Yara rules to aid in the detection of LotL techniques. Furthermore, our study shows that there are differences in execution purposes between benign and malicious samples, providing a vector for development of detection algorithms. In fact, recent papers [30, 75] explore this promising line of research to overcome the limitation of existing security products.

- In some descriptions, LotL techniques exclusively refer to the subset of approaches that can achieve AV evasion. Nevertheless, in this chapter we adopted a broader scope, and observed that within a balanced dataset of malware, 9.6% of malware used a native system binary to perform a malicious action.
- With regards to execution purpose, we observe that LotL binaries were not only leveraged for proxied execution or evasion, but also to implement common malicious routines such as delaying execution, modifying system configuration, ensuring persistence or stopping security services.
- There was a large variability of prevalence of LotL techniques among different families as reported by the AVClass tool. Nevertheless, we can observe that a majority of families showed a low prevalence, while a minority of families used these techniques with a much higher prevalence. This means that these techniques are common enough to be adopted by certain malware authors, but are not the only (or most prevalent) way of implementing malicious functionality in malware.
- Legitimate software used LotL binaries less than malware, and although these binaries were used for different purposes, the prevalence is significant enough to make the accurate detection of malicious usage a challenge for security vendors. Conversely, APT malware leveraged LotL binaries at twice the rate of commodity malware.

Given this evidence, we can conclude that LotL techniques have a significant adoption in current, state-of-the-art malware and that they represent a challenge for the security industry from a detection perspective. With this research, we tried to shed light onto this phenomenon and raise the awareness within the research community about this open problem.

We show that the use of LotL techniques by malware for the purpose of evasion is highly effective. This enables dynamic analysis evasion and also presents an obstacle for authors of heuristic AV algorithms. AV evasion research conducted several months after our initial experiments shows that many AV products are not able to patch against malware that is using LotL techniques [199, 200].

The reason for LotL's evasive capability is the complexity of differentiating between benign and malicious executables usage of proxied execution via LotL binaries. The LotL binaries executed and parameters provided may be identical, between benign and malicious functionality. A malicious payload or suspicious larger sequence of

actions may be the only differentiating factor. As malware can frequently obfuscate these payloads and it is impractical to track all system binary process executions, this explains the challenge facing heuristic AV when attempting to detect LotL use by malware.

We identify a potential strategy for the development of detection algorithms in our research. This is the lower prevalence of LotL techniques in our EMBER benign dataset, shown in Figure 5.5. Resultant from this, heuristic AV should focus on the analysis of parameters of LotL binary execution, as LotL techniques are less commonly used by legitimate software.

**LOLBAS Project Contributions** As an exemplar of some of the techniques in use by modern malware, we identified and documented two new Windows system binaries and added them to the LOLBAS project repository [152].

`At.exe` executes scheduled tasks at a future time, which is used to stealthily maintain persistence on infected systems. An example of this is SoftCell [155], a threat group that uses `At.exe` for the purpose of persistence. Many other threat groups use `Schtasks.exe` to execute scheduled tasks, for persistence.

`Netsh.exe` is a Windows binary that is typically used to alter firewall configurations. This is the only purpose we observe it being used for by malware. However, we document a use case, where `Netsh.exe` can be used to load malicious `.dll`s.

Identical to `Schtasks.exe`, the primary use is to execute commands at a future time, which is used to stealthily maintain persistence on infected systems.

`Netsh.exe` is a Windows binary that is used to alter firewall configurations. This utility enables malign usage via what is termed a 'helper' `.dll`. As a prerequisite to execution of this `.dll`, an 'InitHelperDll' entry point must be present within the binary. Consequently, when crafting a malicious dynamically linked library to be used by `Netsh.exe`, an attacker must ensure the presence of this entry point. A `.dll` payload generated by MSFvenom sufficed in testing.

## 5.8 Limitations & Future Work

**Intended or Unexpected Functionality** Our measurement results do not differentiate between the standard functionality of binaries versus non-standard uses that leverage side effects to achieve some result. Some of these cases are documented in the

LOLBAS project [152]. An example of intended or standard usage is `Netsh.exe` being used to modify firewall rules, while an unintended usage would be to use `Netsh.exe` to run `.dll`s.

We do measure whether parameters from executed samples match these usage patterns. Malware binaries in our dataset do employ these techniques, but make up less than 2% of total measured LotL binary usage. For this reason, we do not include a comparison of intent in our measurement results.

**Anti-VM Malware** Due to data originating from dynamic analysis sandboxes hosted in the cloud, anti-VM evasion techniques used by malware may affect data quality. We mitigate this through the exclusion of malware samples that have minimal execution or crash during execution in the sandbox. Malware has been making use of environment detection and similar methodologies to halt execution [68] for a number of years. This may affect data quality, as evasive malware making use of anti-VM techniques does not demonstrate its actual behaviour. As such, we may underestimate the number of samples that potentially use LotL binaries. Nevertheless, our numbers are a lower-bound of the prevalence of this technique and prove that it is a significant phenomenon that should not be ignored.

An obscure but by no means abnormally complex case is that of GravityRAT, wherein a WMI request to the CPU in order to establish temperature was used to determine whether or not the malware was executing in a virtualised environment [201].

**Human Operators** Many threat actors that use LotL techniques are human operators executing them from remote shells, offensive security tools, Powershell, Visual Basic or Batch scripts [155]. While there is anecdotal evidence of cases like these, it is harder to generalise and to conduct a measurement study over a representative dataset as we did. Instead, we narrowed the scope of our research and focused on the usage of LotL binaries by malware. We show that this phenomenon is not negligible and that malware authors also leverage these techniques in their binaries and not solely in post-exploitation scripts.

A Powershell malware corpus was included in our preliminary initial dataset under analysis but removed due to scarcity of behavioural reports. Effective sandboxed detonation of Powershell and script-based malware identifies an area for future research.

**Linux LotL** A future line of research is to explore the use of these techniques on Linux systems (titled GTFObins [202]). In a similar manner to LotL techniques on Windows, these binaries can be used to achieve malicious functionality. Whilst Linux malware are not as numerous as their Windows counterparts, it is a subject worthy of analysis due to the rise of IoT botnets running lightweight Linux systems [48].

**Mac LotL** There has also been a further development in 2023. This development is known as LOObins [203]. This is binaries that are on Mac systems that can be used to achieve malicious functionality.

**Detection** Another future direction for research in this field is the deployment of detection technologies that attempt to accurately capture the identified patterns of use for LotL techniques such as EDR systems. Future research should leverage recent work [30, 75] on data provenance analysis for process execution chains to enable the modelling of legitimate process relationships and identification of suspicious behavioural patterns.

Testing the evasive potential of these techniques against EDR solutions and custom detection scripts could be beneficial for understanding and improving the current detection capabilities of these solutions.

The difficulty involved in AV correctly identifying malicious use of LotL techniques for things such as as execution of reverse shells is that benign executables need to execute commands using these same Windows binaries. Detection for this behaviour may have an unacceptable amount of false positives. Detection of obfuscated parameters and deployment of native Windows binaries for malign usage needs to be more actively detected by AV systems, of which there has been some research regarding Powershell [116].

Forensic indicators of persistence via Startup Folders are ample. There is forensic evidence left of this as the method by which the system executes this is a symbolic link to an actual executable. Scheduled tasks also leave forensic evidence on a system.

If logging of Windows events is instituted via a security incident event management system this can also assist with retrospective detection, if perhaps not proactive prevention [204,205]. Network log collection using methods such as intrusion detection based on data collected from port mirroring can also indicate that stealthy post-exploitation behaviour and lateral movement is occurring. LotL techniques have

quite a small network footprint and may be difficult to isolate from normal network activity on a Windows domain.

Whilst more common post-exploitation behaviour such as running cryptomining algorithms and encrypting files on a system may be more easily detectable, this behaviour is also frequently obfuscated.

## 5.9 Conclusions

In this work, we have demonstrated a comprehensive evaluation of LotL techniques as executed by malware. Results from this research verified the presence of LotL techniques being used by modern malware, including both commodity and APT malware.

We observed that these techniques were used extensively by APT malware, for which we have released a dataset of hashes. We have shown that APT malware uses LotL techniques twice as often as commodity malware.

We also demonstrated that we these techniques could be used to successfully evade the detection algorithms used by the 10 most popular AV engines. This was coupled with responsible disclosure of these issues to vendors and discussion of why countermeasures to these techniques are complex to implement. AV Vendor's response to responsible disclosure reports and our analysis and comparison of benign to malicious samples, have assisted with identification of why heuristic detection algorithms may fail to comprehensively detect malicious activity achieved using LotL techniques.

Our results show that malware, in various forms on Windows systems, makes use of LotL binaries to perform malicious actions and evade detection. We also showed how malware leverages both intended and unintended functionality for these binaries.

# 6

# Closed-Source Software Supply Chains

Closed-source software supply chains make up a significant proportion of the entirety of software supply chains in the world today. They remain an attractive target for malfeasors because once an attacker has managed to infiltrate a closed supply chain, the impact of their intrusion is magnified. This is because attackers can propagate their attack to all downstream clients of the affected organisation.

APT groups frequently target these software supply chains for malware deployment, leveraging the high-impact nature of such attacks. Given the substantial resources and effort invested by these groups, the malware payloads used are meticulously crafted to be highly evasive, thus posing a significant challenge to existing detection systems. This chapter is dedicated to addressing one of the core research questions of this thesis: *'What novel detection mechanisms and systems can be developed to effectively counteract the evolving landscape of evasive malware, particularly in supply chain attacks?'*

In this chapter, we present an approach that we implemented to detect this highly evasive malware variant. This approach uses the technique of differential analysis to detect the insertion of malicious functionality.

We develop a detection system for the purpose of detecting closed-source supply chain attacks, employing differential analysis as its core method.

We rigorously test the efficacy of this system using a dataset comprised of APT malware samples that were used in closed-source supply chain attacks.

This chapter, therefore, serves as a key component in fulfilling the overarching aim of this thesis, to enhance the detection and analysis of modern evasive malware.

## 6.1 Introduction

The insertion of trojanised binaries into supply chains are a particularly subtle form of cyber-attack that require a multi-staged and complex deployment methodology to implement and execute. In the years preceding this research there has been a spike in closed-source software supply chain attacks used to attack downstream clients or users of a company.

To detect this attack type, we present an approach to detecting the insertion of malicious functionality in supply chains via differential analysis of binaries. This approach determines whether malicious functionality has been inserted in a particular build by looking for indicators of maliciousness. We accomplish this via automated comparison of a known benign build to successive potentially malicious versions.

To substantiate this approach we present a system, `Exorcist`, that we have designed, developed and evaluated as capable of detecting trojanised binaries in Windows software supply chains. In evaluating this system, we analyse 12 samples from high-profile APT attacks conducted via the software supply chain.

Trojanised software is not a new concept or attack vector. However, as the impact and revenue of cyber-espionage and cybercrime enterprises increases, so does the evasive sophistication of these cyber-attacks. A recently published quantitative analysis of supply chain attacks, [206] stated that the volume of this attack type has been increasing sharply over previous years. Enck and Williams conducted summits for government agencies and industry to ascertain the most difficult challenges in detecting supply chain attacks, due to the rapid increase of detected supply chain attacks of 430% in 2020 and 650% in 2021 [207].

APT groups use this attack vector to conduct campaigns, due to its high impact and stealth. SolarWinds was a notable APT supply chain attack, used to penetrate many different organisations. This compromise occurred in the software build chain and was used to attack downstream clients of SolarWinds. This exploited the fact that SolarWinds Orion is network orchestration software, used to monitor and control the

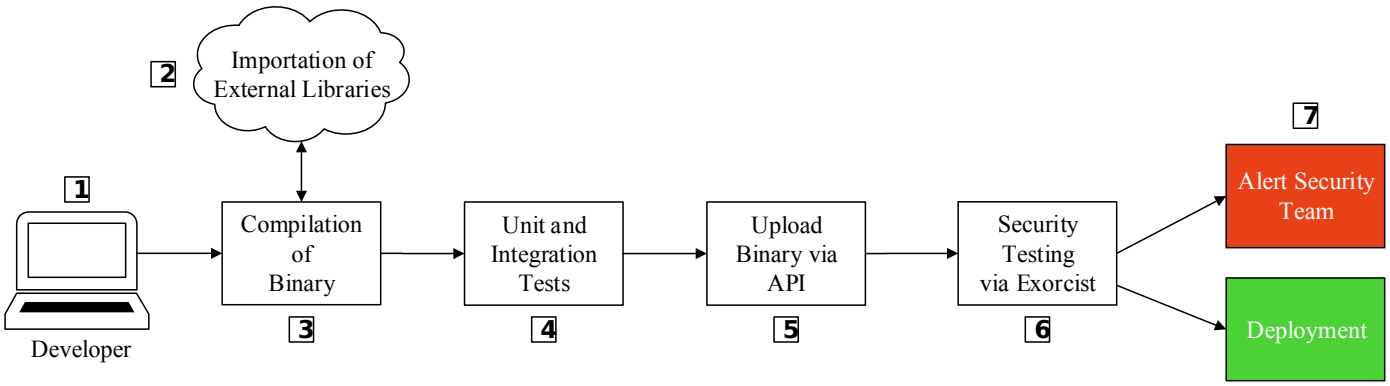



Figure 6.1: Positioning of `Exorcist`'s security testing within a typical continuous integration and deployment pipeline (stage 6).

networks in which it is deployed, allowing access to a far greater number of target systems for the malfeasors.

NotPetya, an infamous recent cyber attack, estimated to have caused $10 billion in total damage to victim organisations, had its initial payload delivered via a closed-source supply chain [208]. Dragonfly campaign was another attack on the energy sector that targeted a closed-source supply chain. Throughout their intrusion killchain, the malfeasors made use of various trojanised binaries, that used off-the-shelf tools to create trojanised versions of common Windows software. These intrusions are particularly devastating towards a victim organisation as they damage all clients of a given company that use a given piece of software and typically have a long dwell time in a network before the intrusion is detected [29, 209].

The trojanised binaries use evasive mechanisms such as LotL techniques and are highly sophisticated operations [30, 116]. Significant challenges involved in the detection of this attack class, that we explore in more depth in Section 6.3.These attacks, amongst others, such as Trojan Source [210] attacks where payloads are hidden in an especially stealthy manner, motivate our research. The complexities of identifying compromises in closed source supply chains are part of the reason that this problem space has not been explored in the academic literature.

To detect closed-source supply chain attacks we propose the approach of differential analysis of binaries between build versions. To demonstrate the effectiveness of this approach we design, implement and evaluate a system, `Exorcist`, that compares a known benign binary in a previous build to a new build. In this comparative differential analysis, `Exorcist` automatically deploys different detection heuristics.

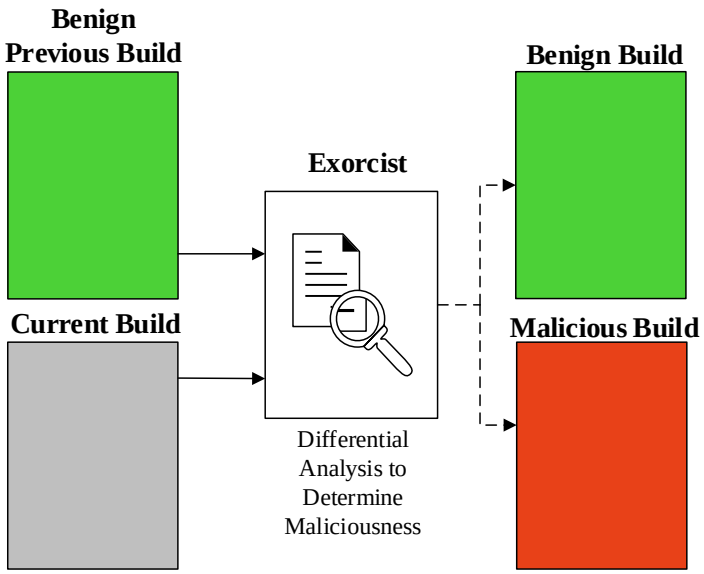



Figure 6.2: Core functionality of system - determining whether an updated version of a binary in the build pipeline has additional malicious functionality added between build versions.

`Exorcist` then determines whether this new binary is malicious through weighting and aggregating these heuristics. Figure 6.3 shows the architecture of the system we develop to compare suspected malicious binaries with previous build versions.

In our research, we aim to ascertain whether differential analysis can identify maliciousness injected between build versions. We can determine the effectiveness of the approach by testing if `Exorcist` detects trojanised binaries during the software build process.

We develop detection algorithms to analyse changes in the static characteristics of binaries and their dynamic behaviour. `Exorcist` also aggregates various detection heuristics and weights their results to determine whether or not each build is malicious.

We also measure certain characteristics of the closed-source supply chain attacks we analyse. We ascertain the prevalence and distribution of indicators of maliciousness in the real-world closed-source supply chain attack campaigns we use to evaluate `Exorcist`. In Section 6.7, we tabulate and characterise the distribution of these indicators. From developing and testing the detection capabilities of our differential analysis approach implemented in `Exorcist`, the resultant key contributions of our work are:

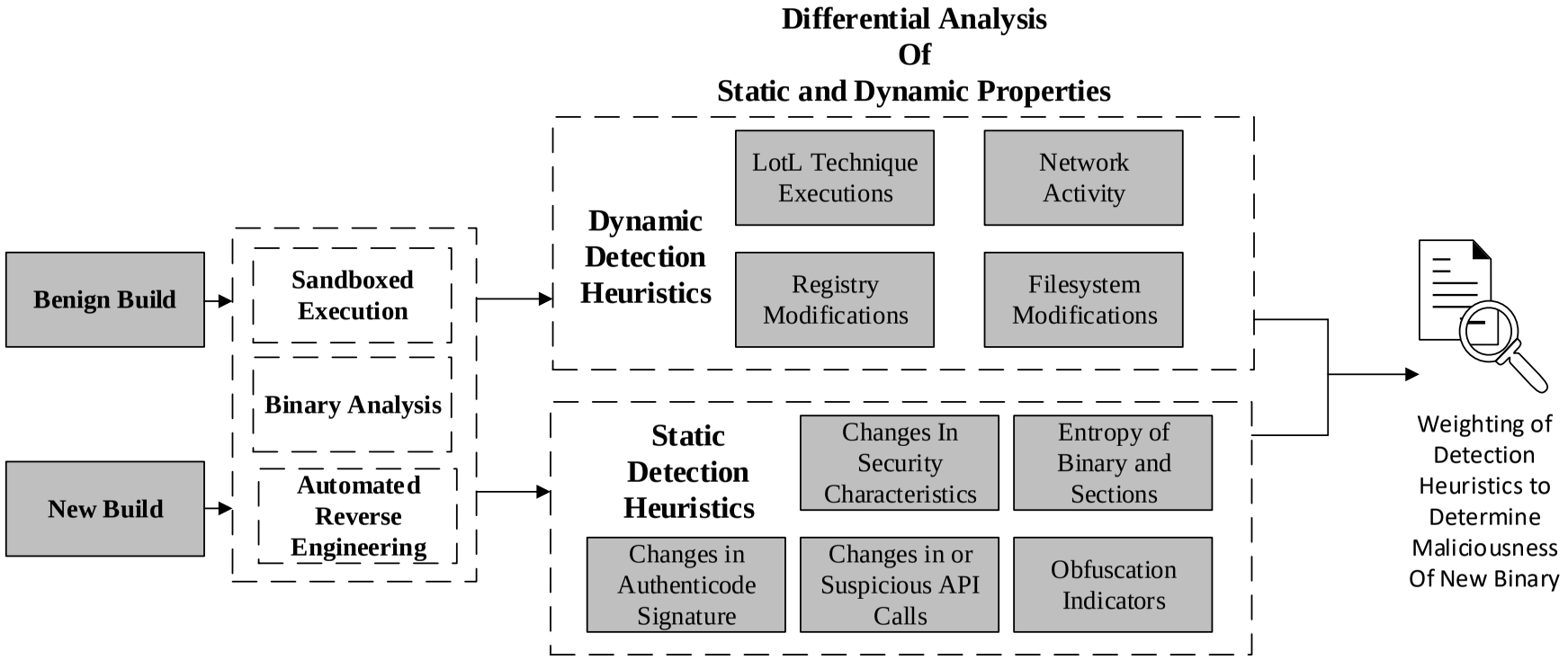



Figure 6.3: Core functionality of system - determining whether an updated version of a binary in the build pipeline has additional malicious functionality added via the differential analysis of dynamic behaviour and static properties of binaries.

- We introduce an automated method to detect insertions of trojanised binaries in software supply chains, evaluated in experiments analysing 12 evasive real-world APT campaigns. Further evaluations show a notably low false positive rate.
- We implement several detection heuristics that operate via differential analysis of two binaries. We evidence advantages of this approach over the analysis of binaries in isolation.
- We implement novel obfuscation detection heuristics that detect malicious activity even when other indicators have been removed. We develop other detection heuristics that implement our approach of differential analysis.

## 6.2 Related Work

There has been recent research into open-source supply chain security [83, 211–213], but little published research into the security of closed-source supply chains and how to detect compromised binaries within them, leaving a gap in the literature. Thomas et al. looked at backdoors from a more abstract theoretical perspective [213].

There has been work directly related to the security of open-source software supply chains. Of note are measurement studies conducted by Duan, Ohm and Zimmerman et al. [83,85,214], however these papers investigate the injection of malicious functionality into the supply chain of open-source software via interpreted languages, rather than compiled binaries.

Recent academic papers analysed the incidents where Debian OpenSSL and Juniper's Dual EC had purposefully inserted cryptographic flaws [215, 216]. Detection and analysis of hardware and firmware backdoors are studied in the academic literature [217–221]. These papers highlight the growing awareness of this attack type in the academic community.

**Static Analysis** Existing works investigate similar areas to the static analysis functionality that we include within our system. Kim et al. measured abuses of binary signing via Authenticode across a large malware dataset [52]. Our identification of malicious functionality via Windows API calls builds upon the ransomware detection developed by Kolodenker et al. [18].

Other work has similarly recognised the limitations of static analysis for detecting maliciousness in purposely obfuscated binaries [97] and detection of malware in general [222].

**Binary Similarity** Binary analysis has been covered extensively in the security literature. Egele et al. dynamically executed binaries to determine their similarity [223]. BinDiff is used for binary similarity detection, with DeepBinDiff augmenting its capabilities using machine learning [224]. Ming et al. [225] implement a system using static and dynamic analysis to check system call equivalence.

**Dynamic Analysis** Academic analysis of malware by detonation in sandboxes began on the Anubis platform, with papers that observed the behaviour of malware [24–26]. VirusTotal has since become the default analysis platform, with sufficient material for Zhu et al. to analyse 15 academic papers using the platform [94].

## 6.3 Challenges

In this section, we explain the key challenges in systematic detection of closed-source supply chain attacks via differential analysis. These challenges all go some way towards explaining the lack of current research into closed-source supply chain attacks and illustrate why it is a worthwhile problem to attempt to solve with the development of a detection system.

**Highly Evasive** Supply chain attacks are designed to be hard to detect. They are frequently conducted by APT groups [226], that Han et al. show using 'low-and-slow' techniques, wherein malware implants slowly and cautiously undertake their actions upon objectives [33]. APT attacks using evasive mechanisms such as LotL techniques have been investigated [30, 75], with systems for analysis of malware samples after their entry and deployment into a network. By contrast, `Exorcist` detects malware at the point of insertion in the build process, rather than a retrospective post-hoc analysis.

**Proprietary Code** To prevent loss of intellectual property, closed-source supply chains are proprietary. This is in contrast to existing solutions for open-source software supply chain verification, wherein the code is assumed to be publicly accessible and to have no data privacy concerns with regards to submitting compiled code to

online analysis systems. The closed-source nature of the software leads to additional problems. One such problem is that the implementation of a system such as `Exorcist` must not submit code to any online verification service during its execution to prevent the leakage of confidential information.

**Lack of Data** While closed-source supply chain attacks are a substantial and growing threat, there is not yet a large corpus of existing attacks from which to develop a system. Limited data are available for system evaluation. To correctly test these systems requires the acquisition of a released piece of software subsequently identified as malicious. In addition to a trojanised version of the binary, a known benign binary is needed to conduct differential analysis. This adds complexity as this software is also proprietary and may be subject to limited distribution. Such pairs of binaries are still comparatively rare. The rarity of these events and the required data may explain the scarcity of previous published research and present an obstacle to researchers attempting to further progress in this field.

### 6.3.1 System Requirements

The requirements and eventual design of `Exorcist` are based on the challenges enumerated in the previous section and aim to practically evaluate the effectiveness of our approach of differential analysis. We identified the requirements on which `Exorcist` is based:

1. Capability of automatic implementation and deployment as part of a development pipeline, between the continuous integration and continuous deployment stages.
2. The ability to identify novel and potentially malicious dynamic behaviours and static characteristics from previous build versions through a process of differential analysis.
3. `Exorcist`'s analysis should execute locally, rather than using cloud-based services for analysis of maliciousness, as `Exorcist` will be processing proprietary code.

## 6.4 Exorcist Design

To evidence the viability of using an approach of differential analysis of binaries to detect supply chain attacks, we design and implement a detection system. This

system, Exorcist, is designed as part of a build pipeline. It compares a potentially trojanised binary with a previously known benign version to identify whether the latter version of the binary is malicious.

In this section, we describe the architecture and components of Exorcist. Exorcist analyses the difference between the dynamic behaviour and static properties of two successive versions of a binary. In this comparison, differential analysis is performed to ascertain whether changes cross the threshold of maliciousness.

This differential analysis combines several detection heuristics, as whilst one malicious indicator may be suspicious, several co-existing indicators are more reliable and accurate in determining overall maliciousness. For instance, indicators of obfuscation or packing in isolation may potentially indicate maliciousness, if identified simultaneously this is more likely to indicate malfeasance.

### 6.4.1 Threat Model and Scope

Our threat model is that software developers in a build pipeline may have infected machines or be malicious actors themselves. We assume that upstream components excluding the compiler before the binary is ingested into Exorcist may be compromised. The specific threat we detect is malicious functionality injected into a software supply chain before a compiled executable is released.

In developing Exorcist, we narrow the focus and scope to Windows PE binaries [127]. This includes dlls and Microsoft Installer MSI files.

### 6.4.2 System Overview

To fulfil the aforementioned requirements in Section 6.3.1 and evidence the practical implementation of a differential analysis approach to detect supply chain attacks, we developed Exorcist. This system is deployed within a build pipeline, as shown in Figure 6.1. Exorcist will alert the security team if the build has indicators of potential compromise.

Exorcist has two main components that analyse differences between successive binary versions to look for indicators of maliciousness. An automated static analysis component, which uses novel methods such as obfuscation detection algorithms to detect indicators of injection of malicious components in combination with existing automated reverse engineering methods. Exorcist detect abnormal dynamic activity in the execution of a given executable that is not present in the initial binary.

| Category | Malicious Indicator | Weight ($w_i$) |
|---|---|---|
| Packing | High Entropy Section<br>High Entropy Binary<br>Section Removal<br>Permissions Changes<br>Vsize/Psize Ratio<br>OEP Change | 1.0 |
| Obfuscation | Cyclomatic Complexity<br>Control-Flow Flattening<br>Overlapping Instructions<br>Function Complexity<br>Large Basic Blocks<br>Average Instructions | 5.0 |
| API Calls & Imports | Import Changes<br>API Changes | 1.0 |
| Binary Signing | Signature Altered<br>Signature Removed | 3.0 |
| DLL Characteristics | Characteristics Altered<br>Characteristics Removed | 3.0 |

Table 6.1: Detection heuristics implemented for differential analysis of the static properties of successive binary versions.

A weighted heuristic system is applied to all detection heuristics that trigger when suspicious differences between builds are identified. If the weights in this system exceed a predetermined threshold, `Exorcist` classifies the latest build version as malicious. The indicators detected by comparing build versions are classified as major or minor, depending on how severe the detection heuristic determines it to be. This facilitates the identification of malicious aberrations between builds, considering both dynamic behaviour and static characteristics.

## 6.5 Implementation

`Exorcist` is a combination of several modular components written in Python, orchestrated through a central processing component that weights the outcome of detection heuristics. These detection heuristics implement differential analysis between successive software versions to detect maliciousness. In this section we provide more detail on our implementation of these modules and systematised them via our weighted

heuristic system. We also qualify what differentiates minor and major indicators and why these thresholds were set.

**Selection of Heuristics** In selecting the heuristics deployed within our system, we closely examined historical data and APT closed-source supply chain attacks. This analysis allowed us to identify common traits and techniques used in these attacks, which informed our choice of heuristics. We then developed and tested programmatic prototypes of these analysis techniques to see if they were appropriate for inclusion within `Exorcist`.

By basing our selection on real-world attacks, we ensured that the heuristics are not only theoretically sound but also practically relevant, directly addressing the types of threats seen in recent high-profile closed-source supply chain attacks. This process on identifying characteristics of real-world APT attacks led to a more focused and effective method than any alternative approach.

### 6.5.1 Automated Static Analysis

To determine the legitimacy of a given binary we deploy different static analysis methods, that we list in Table 6.1. We use a subset of static analysis techniques to detect injected maliciousness during the build process. We select this subset of detection heuristics based on known techniques used in prior supply chain attacks.

To create the static analysis portion of `Exorcist` we write detection algorithms that analyse certain binary characteristics, whilst leveraging existing static analysis research. We combine these methods to ascertain if the differences between the two binaries pass the threshold of maliciousness.

**Windows API Calls and Imports** `Exorcist` extracts a list of API calls from both builds of the binary using the Python library *pefile* to extract the Import Address Table (IAT) and resolving all API calls. To detect suspicious system calls we compare those extracted to a list of the most frequently occurring API calls by malware [227].

We compare the frequency of these calls between builds, to see if anomalous decreases or increases occur. We classify more than ten novel additional invocations of API calls within our suspicious API list as indicating the addition of malicious functionality. We identify a reduction of over 25% of calls between builds as indicative of IAT obfuscation. We determine these thresholds to prevent heuristics triggering on the

normal range of expected changes between software versions. These are both classified as major indicators of maliciousness in our weighted heuristic system.

There are also imports that are seldom called by benign programs. Malware analysts use import hashing to determine the maliciousness of samples [118]. Some malware reconstructs the IAT dynamically at runtime, so an absence or reduction of imports between builds can indicate maliciousness. Cheng et al. researched malware API obfuscation and binary deobfuscation [61,228]. `Exorcist` does not deobfuscate binaries, but identifies the presence of obfuscation.

Legitimate binaries do not typically obfuscate API calls aside from the purpose of preventing reverse engineering through toolkits such as Themida and VMProtect [229]. If differential analysis reveals the insertion of obfuscation between versions, the use of copyright protection tools will be known to the security team and can therefore be disregarded.

**Binary Signing** Many software supply chains rely on signing compiled binaries, to attempt to ensure their provenance. Kim et al. measure a wide range of malware abusing the signing process [52], such as Stuxnet and Flame. We explore abuse of Authenticode in far greater detail in Chapter 4 and the abuse of binaries legitimately signed by Authenticode in Chapter 5.

Most known supply chain attacks contain signature manipulation. Research states AV detection algorithms evaluate malware's maliciousness due to it being signed, without assessing signature revocation or the trustworthiness of the certificate authority [53].

We extract Authenticode signatures from a binary and analyse the difference between this extracted signature and the one extracted from previous builds. Changes in signature characteristics are considered important, not just certificate validity. Changes such as the certificate authority changing between builds or alteration of the chain of trust can indicate an attack using signature corruption in progress, with scant legitimate reasons for this to occur and so is weighted heavily within our system. `Exorcist`'s weighted heuristic system classifies changes of these signatures as a minor indicator and removal as a major indicator of maliciousness.

**Packer Detection** Packing schemes frequently include some distortion in the rate of entropy within a given executable [98]. This has typically been perceived to be an increase in entropy, though recent research has shown some packing schemes may

result in a decrease in entropy. Any alteration in the final entropy of these binaries is likely to indicate potential malicious alterations in progress.

If one or two of the following properties emerge between builds, `Exorcist` categorises this as a minor indicator. If more than two emerge, it is instead categorised as a major indicator. We observe the following properties to identify use of packers [8, 97]:

`Exorcist` calculates the Shannon entropy [230], for the executable as a whole and for subsections of the binary. Entropy can be a reliable indicator of compression or encryption used by packers [117]. An entropy increase across an entire binary or a section between software versions may indicate compression or cryptographic operations. `Exorcist` adopts Mantovani et al. [98]'s threshold of entropy of 7.0 as the difference between high and low entropy in a binary.

Whilst some changes in the nature of the sections of the executable file can be expected between software versions, `Exorcist` recognises several changes that may indicate maliciousness. `Exorcist` extracts these characteristics using the Python library *pefile* and some additional processing: (i) Changes in the Original Entry Point (OEP) of the binary between compilations. (ii) Any changes in section names or permissions to read, write or execute. (iii) Whether the ratio of code to data changes. (iv) Whether there is an abnormal difference between section Vsize (virtual size) and Psize (physical size), this being the size of the executable in memory and size on disk. If these values differ there may be decompression routines. (v) Whether there has been addition or removal of sections.

**Obfuscation Detection** Obfuscation is used to impede analysis and hide malicious functionality [231–234]. Code obfuscation artificially increases the program's complexity so it can be detected with code complexity metrics, such as a graph's cyclomatic complexity or the average instructions per basic block [235]. We use such metrics as maliciousness indicators by identifying code complexity increases between different program versions. We analyse two binaries using a disassembler and compare how many complexity metrics show a complexity increase in the newer program version.

We count the overall number of detected instructions and functions in our binary comparison. To detect obfuscation techniques such as instruction and data encodings [236, 237] which significantly increase code size, we compute average instructions per basic block. For control-flow obfuscations such as opaque predicates and range

| Category | Malicious Indicator | Weight ($w_i$) |
|---|---|---|
| Registry Modification | Query Registry<br>Registry Addition<br>Registry Deletion<br>Registry Alteration | 1.0 |
| Filesystem Modification | File Creation<br>File Deletion | 1.0 |
| Network Activity | Change in Volume<br>Suspicious IP/DNS | 1.0 |
| LotL Techniques | Proxied Execution<br>Reconnaissance<br>Script Execution | 1.0 |

Table 6.2: Detection heuristics implemented for differential analysis of dynamic behaviour of successive binary versions.

dividers [232], we calculate the function's control-flow graphs average cyclomatic complexity. We also look for complex state machines and control-flow flattening by counting the functions our *control-flow flattening detection heuristic* [238] finds.

While comparing the average values of the different complexity metrics works fine for smaller programs, it doesn't work that efficient for larger program, especially if only a minority of functions has been modified in a supply chain attack. Comparing average values of different complexity metrics works for smaller programs but is less efficient for larger programs, especially if a minority of functions are modified. For these, we use a more fine-grained analysis: For average instructions per basic block and average cyclomatic complexity heuristics, we compute the metrics on a function level, take the top 10% and calculate an average score for each metric. This way, we can identify large complexity increases, even if only applied locally to specific functions.

### 6.5.2 Dynamic Analysis

`Exorcist` deploys the executable being built to a local automated instance of Cuckoo sandbox. `Exorcist` then records the behaviour of this executable within the sandbox and compares the behaviour between versions to see if behaviour has been added or removed. We list the new dynamic behaviour that `Exorcist` classifies as malicious in Table 6.2.

As legitimate software has almost no reason to terminate during execution, A severe

reduction in functionality between builds is a maliciousness indicator, potentially evidencing anti-VM techniques in use and therefore trigger `Exorcist`'s detection algorithms.

**Network Activity** We record and identify any deviation in network activity between build versions of a binary resultant from execution of a binary. This network footprint is the list of ports, domains and IPs extracted from the packet capture recorded during each binary's sandboxed execution. If there is any network activity to new IPs, domains or ports this is classified as a major indicator.

**Processes** Another indicator of maliciousness is the process activity that is recorded during the execution of the binary. If there is any alteration of process executions between builds, this may be indicative of malicious functionality being added. Recent research has analysed the detection capabilities of evasive malware by analysis of provenance graphs and process execution chains [75].

Evasive malware in supply chains frequently uses a wide range of LotL techniques to achieve their goals. We identify LotL technique executions in these process execution logs by analysing which processes execute using applications installed by default on Windows systems. Similarly, the use of scripting languages by LotL binaries is suspicious. If process logs show the addition of two or more LotL commands between builds, this is classified as a major indicator.

**Filesystem Modification** Any indication of filesystem or registry modification can indicate surreptitious modification of the OS by a malicious binary, if it deviates from that conducted from previous builds. We define the threshold between minor and major indications of maliciousness with this detection heuristic by the number of modifications in comparison to the previous version.

### 6.5.3 Weighted Heuristic

Once the results of all our differential analyses are assembled, they are combined to produce a final determination of whether a binary is determined as malicious or not. This combination is accomplished via a weighted score, defined as follows:

$$S = \sum w_i \cdot s_i$$

Where $s_i$ is a score assigned to the result of analysis category $i$ and $w_i$ is a weight applied for that category. Scores are assigned as:

| | Campaign | Filetype | Static | | | | | Dynamic | | | | Score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | API Calls | Packing | Binary Signing | Obfuscation | Security (Dll) Characteristics | Registry Modification | Network Activity | Filesystem Modification | LotL | |
| | CCleaner | EXE | ○ | ● | ○ | ◐ | ● | ● | ● | ● | ○ | 9.5 |
| | Chiacoin | EXE | ● | ● | ● | ◐ | ● | ● | ○ | ● | ○ | 12.5 |
| | DarkSide | MSI | ◐ | ◐ | ○ | ● | ● | ○ | ○ | ○ | ○ | 9.0 |
| | Dofoil | EXE | ○ | ◐ | ● | ◐ | ○ | ○ | ○ | ○ | ○ | 6.0 |
| | NotPetya | DLL | ○ | ◐ | ○ | ◐ | ○ | ○ | ○ | ○ | ○ | 3.0 |
| Dragonfly | Ewon 1 | EXE | ● | ● | ● | ● | ○ | ○ | ○ | ○ | ● | 11.0 |
| Dragonfly | Ewon 2 | EXE | ● | ● | ● | ◐ | ● | ◐ | ○ | ● | ● | 13.0 |
| Dragonfly | Mbconnect 1 | EXE | ● | ● | ● | ◐ | ● | ● | ○ | ● | ● | 13.5 |
| Dragonfly | Mbconnect 2 | EXE | ● | ● | ○ | ◐ | ○ | ○ | ○ | ○ | ○ | 4.5 |
| Dragonfly | Mesa Imaging | MSI | ● | ● | ○ | ● | ● | ○ | ○ | ○ | ● | 11.0 |
| Solar | Sunburst | DLL | ○ | ● | ○ | ● | ○ | ◐ | ○ | ◐ | ○ | 7.0 |
| Solar | Supernova | DLL | ○ | ● | ○ | ● | ○ | ○ | ○ | ○ | ○ | 6.0 |

Table 6.3: Overview of detection results for the APT closed-source supply chain attacks. The malicious properties are those that differential analysis identified in the trojanised binary version but not the initial benign version. ● is a major indicator of maliciousness. ◐ is a minor malicious indicator. The severity of maliciousness is determined by the weighted heuristic system.

$$s_i = \begin{cases} 1.0 & \text{Major indication} \\ 0.5 & \text{Minor indication} \\ 0.0 & \text{otherwise} \end{cases}$$

Weights are then applied at a per-category level, as we list in Table 6.1 and 6.2. We select these weights based upon the amount of false positives for each category type. Due to the likelihood of false positives occurring in relative comparison to static indicators, dynamic indicators are assigned a weighting of 1.0.

For instance, Authenticode signatures [96] are not typically changed or removed between builds, so these categories are assigned higher weightings. Multiple coexisting obfuscation indicators are rare between versions of benign binaries, so are assigned a higher weighting.

A final result is obtained by comparing the score $S$ to a threshold $\alpha$. If $S \geq \alpha$ then the second iteration of the binary is considered malicious, otherwise it is deemed benign. The value of $\alpha$ allows the weighted heuristic system to be configured to trade off the risk of false positives against false negatives. In our case studies, we set a maliciousness threshold of 2.0 to accommodate a small number of false positive indicators.

## 6.6 Case Studies

In this section, we demonstrate the practical effectiveness of our approach. To do this, we evaluate the detection capabilities of our analysis system `Exorcist`. We describe the results of automated differential analysis upon 12 recent APT software supply chain attacks, that we list in Table 6.3, with the exact versions and timelines in Section 6.8.

### 6.6.1 SolarWinds

Multiple trojans were inserted in various SolarWinds products in 2020 in a closed-source supply chain attack [239].

**Sunburst** Multiple threat intelligence reports recognise this campaign as evasive [240]. Despite the sophistication of this attack, this `.dll` file contained the most indicators of potential obfuscation of the samples we analysed (six out of eight possible indicators). This is coupled with several changes of static characteristics between versions,

such as section removal, a marginal global increase in entropy and a change in OEP of 48324 bytes. This shows that by attempting code obfuscation, the malicious actors enabled `Exorcist` to detect their trojanised software.

**Supernova** A further compromised `.dll` was used in this campaign - `app_web_logoimagehandler.ashx.b6031896.dll`. Both of these *.dll*s show indicators of the addition of packing and obfuscation between versions, as shown in Table 6.6.

`Exorcist` identified more static indicators of maliciousness, although slightly fewer indicators of obfuscation in this `.dll`. In particular, the '.rsrc' and '.reloc' sections of the binary were removed. New suspicious API calls were added, including calls used for persistence such as 'RegCreateKeyEx'and 'SetEnvironmentVariableW'.

### 6.6.2 NotPetya

Similar to SolarWinds, NotPetya relied on a trojanised update mechanism to deliver a malicious `.dll`.

The malicious version of this binary contains a similar amount of code to the benign binary but with marginally higher overall complexity. `Exorcist` identified an entropy increase between versions on a global level and a large shift of 36352 bytes in the OEP address. These obfuscation and packing indicators mean `Exorcist` classifies the binary as malicious. Despite these static properties, NotPetya had the least indicators of all samples analysed.

### 6.6.3 CCleaner

CCleaner was a targeted attack, deployed as a trojanised binary with minor changes from the original binary. The trojan in CCleaner was subtle, only executing in a specific target environment; in a manner similar to Stuxnet [241]. `Exorcist` detected this from the large amount of new registry operations added between versions, shown in Table 6.4. These were principally composed of registry read operations that identified the environment the binary was executing in. The trojanised binary also adds 2 write operations that disabled future updates and changed network settings.

The malicious binary had the memory protection of ASLR removed via the removal of the 'DLLCharacteristic' 'HighEntropyVA'. `Exorcist`'s packer detection heuristics also trigger upon the addition of sections, such as the 'pdata' section highlighted

| Campaign | | Registry | Filesystem | Network | Signature Matches | LotL |
|---|---|---|---|---|---|---|
| CCleaner | | 3444 | 370 | 1 | 0 | 0 |
| Chiacoin | | 2767 | 186 | 0 | 4 | 13 |
| DarkSide | | 0 | 0 | 0 | 4 | 0 |
| DoFoil | | 0 | 0 | 0 | 0 | 0 |
| NotPetya | | 0 | 0 | 0 | 0 | 0 |
| Dragonfly | Ewon 1 | 0 | 0 | 0 | 0 | 5 |
| | Ewon 2 | 5 | 0 | 0 | 37 | 9 |
| | MbConnect 1 | 2504 | 38 | 0 | 66 | 1 |
| | MbConnect 2 | 0 | 4 | 0 | 42 | 1 |
| | Mesa Imaging | 0 | 0 | 0 | 17 | 2 |
| Solar | Sunburst | 0 | 0 | 0 | 0 | 0 |
| | Supernova | 0 | 0 | 0 | 0 | 0 |

Table 6.4: Overview of dynamic analysis results. The numbers shown here represent amount of separate events in a malicious binary not present in the previous benign version.

in Rascagnere's analysis of the incident [242]and a 0.25 increase in global file entropy.

It is the only campaign where there is additional network activity not present in the initial file. Researchers identified this ICMP request to '224.0.0.0' [243] as anti-VM functionality.

### 6.6.4 DoFoil

DoFoil miner, also known as SmokeLoader, deployed a trojanised binary in place of the benign torrent application, 'MediaGet'. The static indicators for this binary are sparse, with binary signature removal and the obfuscation indicator of more code being added to the trojanised binary. `Exorcist`'s heuristics classify binary signature removal and detection of obfuscation indicators as a major indicator. There are also minor indications of packing - a section '.rdata' being added, with entropy of 7.45, and an OEP shift.

### 6.6.5 Dragonfly Campaign

Three ICS vendors' software downloads were compromised. The companies were eWON, MB Connect Line and Mesa Imaging and the hacks occurred in June-July

2013 and in January 2014. These three campaigns and their constituent malicious samples had different indicators across a variety of axes.

**Dragonfly Ewon** There were two different binaries trojanised during Dragonfly group's Ewon campaign.

`Exorcist` identified additional LotL techniques implemented in the trojanised binary not present in the benign version. The `Find.exe` binary was used for reconnaissance, to identify certain characteristics of a target environment:

```
C:\Windows\System32\find.exe find  /i "no matching devices
    found" tap__log.txt
```

`Sc.exe` and `Taskkill.exe` are leveraged to terminate the existing 'Talk2mVpnService' service, ensuring only the malicious program version was running. This deviates from the usual purpose of task stopping by malware, which is to stop local security applications, to prevent detection and removal.

```
C:\Windows\System32\cmd.exe /c sc stop Talk2mVpnService
```

```
C:\Windows\System32\cmd.exe /c taskkill /F /T /IM
    Talk2MVpnService.exe
```

`Exorcist` detected these samples using native Windows system binaries to obfuscate and proxy execution. This is a common technique used by malware to proxy execution and thereby enable AV evasion. In the three aforementioned examples, there were several native system binaries executed in a proxied execution chain. The first command listed below uses the `Rundll32.exe` binary to run additional malicious functionality through a `.dll`:

```
C:\Windows\System32\rundll32.exe C:\Users\ADMINI~1\AppData\
    Local\Temp\TmProvider.dll RunDllEntry
```

The second uses the `Cmd.exe` binary to proxy execution of further malicious functionality through a `.bat` script:

```
C:\Windows\System32\cmd.exe /c C:\Program Files(x86)\eCatcher
    -Talk2M\Talk2mVpnService\Drivers\install.bat
```

`Exorcist` detects another incidence of proxied execution via the `Cscript.exe` LotL binary:

```
C:\Windows\system32\cscript.exe nologo renametap.vbs
```

**Mbconnect** Dragonfly trojanised an installer file and a tool that checked the network environment produced by the ICS vendor `MbConnect` to facilitate their campaign. Figure 6.5 shows 'MbConnect 1' to exhibit dynamic activity in the form of filesystem read operations and registry modifications not present in the initial benign binary.

`Exorcist` identified static indicators in'MbConnect 1'. There is an increase in global binary entropy of 1.4, 4 sections removed from the binary, a high entropy (7.56) section '.rsrc' added and Authenticode signature removal. There were also 313 less overall API calls, indicating potential IAT obfuscation.

'Mbconnect 2' contained static indicators of the addition of malicious functionality into the benign initial binary. These indicators are an entropy increase in the '.reloc' section from 0.2 to 5.09 and the removal of 5 sections.

For both binaries, Cuckoo signatures matched on the trojanised binary but not the initial benign sample. The trojanised binary triggered anti-sandbox, anti-debugger and anti-VM signatures, through addition of suspicious API calls. 'GetTickCount' detects if malware samples are executing in a virtualised environment [63]. Both files add 'ShellExecuteA', 'CreateProcessA' and 'CreateThread' API calls, to proxy execution. `Exorcist` detects other suspicious calls added to both 'MbConnect 1' and 'MbConnect 2' to facilitate persistence and registry manipulation, such as 'RegSetValueExA' and 'CreateFileA'.

**MesaImaging** `Exorcist` assessed this trojanised binary to contain potentially malicious changes not in the benign binary. Memory corruption prevention mechanisms present in the benign binary were removed, the 'DLLCharacteristics' 'NX_COMPAT' (Data Execution Prevention) and 'NO_SEH' (Structured Exception Handling). There was the addition of 32 suspicious API calls, a reduction in total API calls of 3 and removal of sections from the file. 5 out of 8 indicators of obfuscation were present.

MesaImaging showed LotL activity to proxy execution via the `Rundll32.exe` program, similar to the Ewon campaign:

```
C:\Windows\System32\rundll32.exe C:\\Program Files (x86)\\
    MesaImaging\\Swissranger\\sys\\libusb0.dll
    usb_install_driver_np_rundll C:\\Program Files (x86)\\
    MesaImaging\\Swissranger\\sys\\libusbSR.inf
```

As with the Ewon campaign, `Exorcist` identified invocations of LotL techniques through `Cmd.exe` and `Rundll32.exe` to proxy execution of further malicious functionality:

```
C:\Windows\System32\cmd.exe /c C:\Users\ADMINI~1\AppData\
    Local\Temp\setup.exe & c:\windows\system32\rundll32.exe C
    :\Users\ADMINI~1\AppData\Local\Temp\tmp687.dll ,
    RunDllEntry
```

### 6.6.6 DarkSide Nullsoft Trojan

The ransomware group 'DarkSide' trojanised a version of the 'NullSoft Installer' for 'SmartPSS' security camera monitoring software after compromising a network.

Cuckoo signatures matches identified anti-VM techniques querying the virtualised environment. The remaining indicators of maliciousness identified by `Exorcist` in 'MbConnect 1' are static - showing 5 additional suspicious API calls, notably 'AdjustTokenPrivileges' and 'SetFileSecurityA', that show permissions modification. Obfuscation indicators show additional code and increased code complexity. These static properties reveal attempts to pack and obscure elements of program execution, including a marginal entropy increase in several sections.

### 6.6.7 Chiacoin

In May 2021, the ChiaCoin wiki on GitHub was edited to replace a link to the legitimate installer with a trojanised binary. Less sophisticated than other attacks, with the binary added and removed less than 48 hours later, `Exorcist` detected several malicious indicators. This binary deployed evasion techniques to prevent detection but did not attempt to masquerade as the benign installation binary. As shown in Table 6.4, the trojanised Chiacoin binary displayed different post-execution behaviour by reading 184 files and deleting 2.

## 6.7 Evaluation

In this section, we examine the APT case studies `Exorcist` analysed, to assess the prevalence of indicators. We also describe the setup and results of false positive experiments conducted to ensure the reliability and accuracy of `Exorcist`.

| | Campaign | Filetype | Static | | | | | Dynamic | | | | Score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | API Calls | Packing | Binary Signing | Obfuscation | Security (Dll) Characteristics | LotL | Registry Modification | Network Activity | Filesystem Modification | |
| Solar | CCleaner | EXE | ○ | ○ | ○ | ○ | ○ | ◐ | ◐ | ○ | ○ | 1 |
| | Chiacoin | EXE | ○ | ◐ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | 0.5 |
| | DarkSide | MSI | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | 0 |
| | Sunburst | DLL | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | 0 |
| | Supernova | DLL | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | 0 |

Table 6.5: Overview of detection results for false positives. The properties identified are those that are in the second benign binary version but not the initial benign version. ● is a major indicator of maliciousness. ◐ is a minor malicious indicator.




Figure 6.4: Frequency of dll characteristic alteration in samples between builds in APT malware dataset.

### 6.7.1 Dynamic Analysis Indicators

The different indicators extracted from `Exorcist`'s dynamic execution of the binary varied in their frequency within the APT supply chain attack dataset. In Table 6.4, we tabulate the different indicators `Exorcist`'s heuristics detected.

We visualise the frequency of indicators of maliciousness within our APT dataset in Figure 6.5. We can observe that there is a large variance between different indicator types. For instance, network activity is scarce, where only CCleaner was identified to have malicious network indicators during sandboxed execution.

### 6.7.2 Static Analysis Indicators

In Table 6.6, we enumerate the static properties indicating maliciousness identified by `Exorcist`. We observe that all campaigns contain multiple static indicators of maliciousness, particularly entropy increases and shifts in the OEP for given binaries.

We report our obfuscation analysis results in Table 6.7. All campaigns we analysed revealed indicators of obfuscation. This is in contrast to our false positive testing, wherein there were no indicators of obfuscation, as we report in Table 6.5.

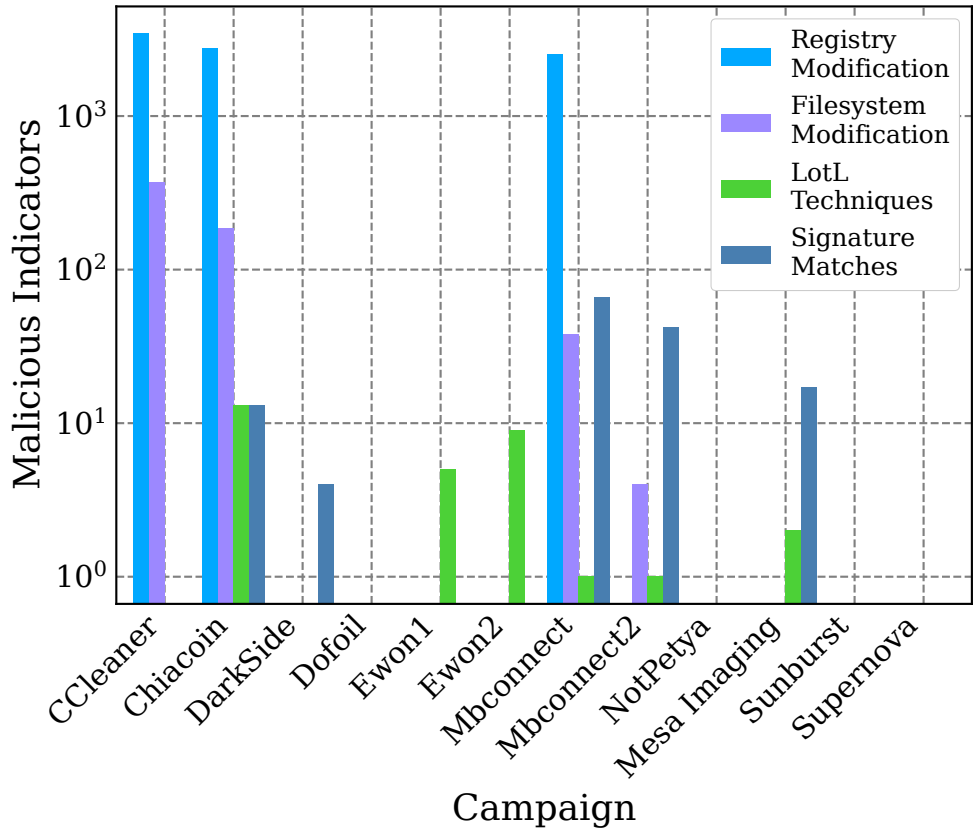



Figure 6.5: Comparative frequency of dynamic analysis indicators. The Y axis represents total number of maliciousness indicators in the trojanised binary and not the initial file.

| | Campaign | API Calls | Sections | Entropy Increase | OEP Shift | Binary Signing | Security Characteristics Altered |
|---|---|---|---|---|---|---|---|
| | CCleaner | -3 | + − | ◐ | ✓ | | 1 |
| | Chiacoin | +57 | + − | ● | ✓ | ✓ | 1 |
| | DarkSide | +5 | + − | ◐ | | | 3 |
| | Dofoil | 0 | + − | ● | ✓ | ✓ | |
| | NotPetya | 0 | | ● | ✓ | | |
| Dragonfly | Ewon 1 | +136 | + | ● | ✓ | ✓ | |
| Dragonfly | Ewon 2 | +68 | + − | ● | ✓ | ✓ | 1 |
| Dragonfly | Mbconnect 1 | -313 | + − | ● | ✓ | ✓ | 1 |
| Dragonfly | Mbconnect 2 | +74 | + − | ● | ✓ | | |
| Dragonfly | Mesa Imaging | -3 | − | ● | ✓ | | 2 |
| Solar | Sunburst | 0 | − | ● | ✓ | | |
| Solar | Supernova | 0 | − | ● | ✓ | | |

Table 6.6: Overview of static differential analysis results. In the sections column, + denotes sections added to a PE file and − denotes section removal. In the entropy column ● signifies entropy increase across the whole file and sections; whilst ◐ signifies an entropy increase in a section or sections. In 'OEP Shift' and 'Binary Signature' columns, a tick denotes an OEP shift or binary signature removal or modification.

| | | Instructions | | | | Functions | | Cyclomatic Compexity | |
|---|---|---|---|---|---|---|---|---|---|
| Campaign | **Total** | Total | Overlap | Avg. per Block | Avg. per Block Top | Total | Flat | Avg. | Avg. Top |
| CCleaner | **4/8** | | ✓ | ✓ | ✓ | ✓ | | | |
| Chiacoin | **2/8** | | | | | | | ✓ | ✓ |
| DarkSide | **5/8** | ✓ | | ✓ | ✓ | ✓ | ✓ | | |
| DoFoil | **3/8** | ✓ | ✓ | | | ✓ | | | |
| NotPetya | **3/8** | ✓ | | ✓ | ✓ | ✓ | ✓ | | |
| Dragonfly Ewon 1 | **5/8** | ✓ | ✓ | | | ✓ | ✓ | | ✓ |
| Dragonfly Ewon 2 | **3/8** | | | ✓ | | | | ✓ | ✓ |
| Dragonfly Mbconnect 1 | **2/8** | ✓ | ✓ | | | ✓ | | | |
| Dragonfly Mbconnect 2 | **3/8** | ✓ | ✓ | | ✓ | ✓ | ✓ | | |
| Dragonfly Mesa Imaging | **5/8** | | ✓ | | | | | ✓ | ✓ |
| Solar Sunburst | **6/8** | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ |
| Solar Supernova | **4/8** | | ✓ | ✓ | ✓ | ✓ | | | |

Table 6.7: Overview of obfuscation analysis results. A ✓ represents an obfuscation indicator that passes the threshold of suspiciousness within a given binary.

Figure 6.4 shows the frequency of 'DLLCharacteristics' removed between build versions in our case studies. 'DLLCharacteristics' are a misnomer, as they don't concern solely dynamically linked libraries, instead they refer to security properties of a given PE file. Apart from 'Terminal Server Aware', all 'DLLCharacteristics' altered by the campaigns we analysed have direct security implications.

### 6.7.3 False Positive Testing

`Exorcist` is designed to be implemented as part of a build pipeline, where malicious trojans will be exceptionally rare event. It is therefore important that `Exorcist` has a low false positive rate.

We evaluate whether `Exorcist` is susceptible to false positive errors via downloading successive released versions of software that are known to be benign and processing them in `Exorcist`. We show the results in Table 6.5. Our false positive testing revealed `Exorcist` to have a notably low false positive rate.

In this analysis, we evaluate and test 10 different samples from 5 campaigns. We could not obtain binaries for the other campaigns, due to the aforementioned scarcity

we describe in Section 6.3.

The only static indicators are those of packing, in the second benign Chiacoin binary. These packing indicators are due to the removal of 4 sections and the addition of another section. This additional section has high entropy (7.999), the same as the global binary entropy. Notably, there were no suspicious indicators detected by obfuscation detection algorithms.

There are two incidences of dynamic maliciousness within the false positive dataset, both of which occur with the legitimate CCleaner binaries. The CCleaner binary has added a use of LotL binaries between its benign versions. `Exorcist` detects the usage of the `Ie4uinit.exe` binary:

```
C:\Windows\SysWOW64\ie4uinit.exe -ShowQLIcon
```

The preceding invocation of `Ie4uinit.exe` is benign, with the code for Cuckoo Sandbox stating that invocation of this LotL binary with this parameter should be ignored [244].

There is a true false positive of 4 registry operations conducted in the latter version of CCleaner not present in the initial version. `Exorcist`'s heuristics identify these as a minor amount of additional registry accesses, although networking registry elements and error reporting are registry keys which malware may access.

## 6.8 Versions

In Table 6.8 we list the versions and release dates of the APT case studies we analyse. In Table 6.9 we list the versions and release dates of the false positive case studies we analyse.

It is worth noting that the release dates within these tables are extracted from the compilation date of the software. It is possible for these dates to be falsified by malware authors. Indeed, as we discover in our experiments in Chapter 4, obfuscation and falsification of compilation metadata is an effective technique for AV evasion.

## 6.9 Entropy Analysis

In Figure 6.6 and Figure 6.7 we show the entropy distribution across several of the binaries in our dataset. It can be expected that between successive build versions there is some alteration of entropy, binary size and section sizes. High entropy in

| Campaign | | Benign Version | Infected Version | Benign Version Release Date | Infected Version Release Date |
|---|---|---|---|---|---|
| CCleaner | | v5.32 | v5.33 | 2017-06-29 | 2017-08-03 |
| Chiacoin | | ChiaSetup-1.1.4.exe | ChiaSetup-1.1.4.exe | 2020-09-27 | 2020-08-01 |
| DarkSide | | V2.002.0000009.0.R.190426.exe | V2.002.0000007.0.R.181023-General-v1.exe | 2009-12-05 | 2020-08-01 |
| DoFoil | | 2, 1, 0, 0 | 2, 1, 0, 0 | 2017-10-27 | 2018-02-06 |
| NotPetya | | ZvitPublishedObjects188 | ZvitPublishedObjects189 | 2017-06-21 | 2017-06-12 |
| Dragonfly | Ewon 1 | 3.1.0.85 | N/A | 1999-04-08 | 2007-03-31 |
| | Ewon 2 | 4.0.0.13073 | 4.0.0.13073 | 1992-06-19 | 2007-03-31 |
| | MbConnect 1 | 1.1.1.0 | 1.1.1.0 | 2014-06-24 | 2013-07-14 |
| | MbConnect 2 | setup_1.0.1.exe | setup_1.0.1.exe | 1992-06-19 | 2013-07-14 |
| | Mesa Imaging | SwissrangerSetup1.0.14.706.exe | SwissrangerSetup1.0.14.747.exe | 2009-12-05 | 2011-05-28 |
| Solar | Sunburst | 2020.2.15300.12766 | 2020.2.5300.12432 | 2020-08-11 | 2020-05-11 |
| | Supernova | OrionWebv2020.2.15300.12766 | OrionWebv2019.4.5200.9083 | 2020-08-11 | 2020-03-24 |

Table 6.8: Versions and Release Dates of software in APT dataset

| Campaign | | Benign Version | Second Benign Version | Benign Version Release Date | Second Benign Version Release Date |
|---|---|---|---|---|---|
| CCleaner | | v5.34 | v5.35 | 2015-12-29 | 2015-12-29 |
| Chiacoin | | ChiaSetup-1.1.4.exe | ChiaSetup-1.1.4.exe | 2020-09-27 | 2020-09-27 |
| DarkSide | | V2.002.0000008.0.R.190225.exe | V2.002.0000009.0.R.190426.exe | 2009-12-05 | 2009-12-05 |
| Solar | Sunburst | 2020.2.15300.12766 | 2020.2.15300.12901 | 2020-08-11 | 2020-12-10 |
| | Supernova | v2016.2.5300.959 | v2017.3.5300.1974 | 2016-09-16 | 2018-03-06 |

Table 6.9: Versions and Release Dates of software in false positive dataset.

an additional section, such as for 'Solarwinds (Sunburst)' is suspicious. A marked increase in sections present in the original benign binary such as with 'Dragonfly Mbconnect 2' or 'Solarwinds (supernova)' is also suspicious.

## 6.10 Discussion

In the evaluation of our detection mechanisms, we have meticulously quantified their performance in identifying and countering evasive malware strategies, particularly those employed in sophisticated APT scenarios and closed-source supply chain attacks. This section not only delineates the efficacy of our approaches but also contextualises their significance within the broader framework of our research objectives.

The results from our evaluation should be interpreted as more than mere performance metrics; they are a testament to the feasibility and effectiveness of our proposed solutions in the real-world context of cybersecurity. Key takeaways from this evaluation include:

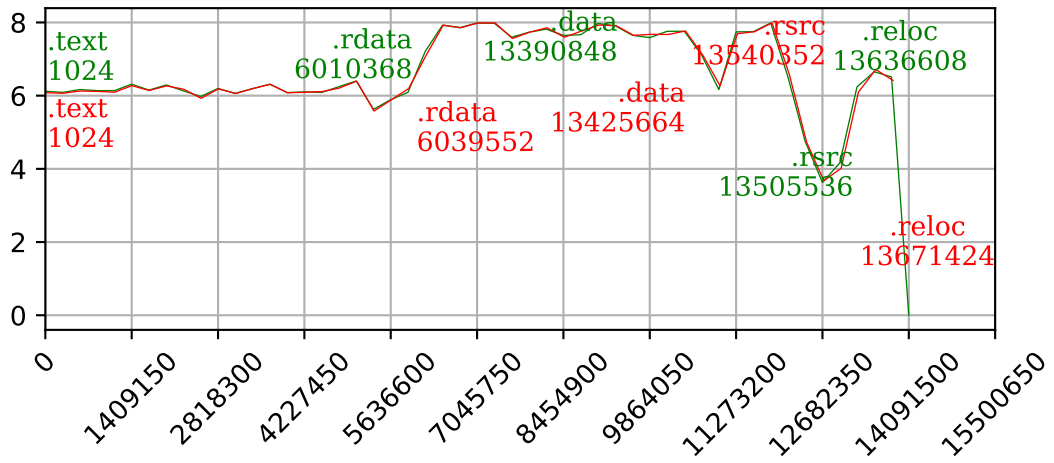
**Effectiveness Against Evasive Techniques** The success rate in detecting various evasive methods offers insights into the robustness of our heuristics and algorithms. This directly addresses our aim to improve detection capabilities in the face of advanced evasion tactics, that we discuss in Chapter 3.

**Response to Real-World Threats** By applying our detection mechanisms to scenarios modelled on real-world APT attacks and supply chain compromises, we validate their practical applicability, aligning with our goal to contribute tangible solutions to current cybersecurity challenges.

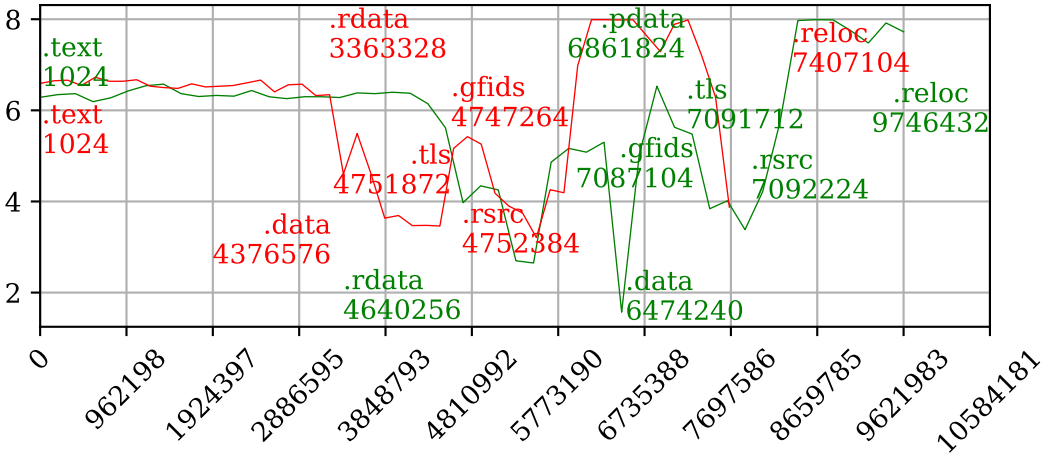
The evaluation also highlights areas where our approaches could be further refined, providing a clear direction for future research and development.

### 6.10.1 Insights

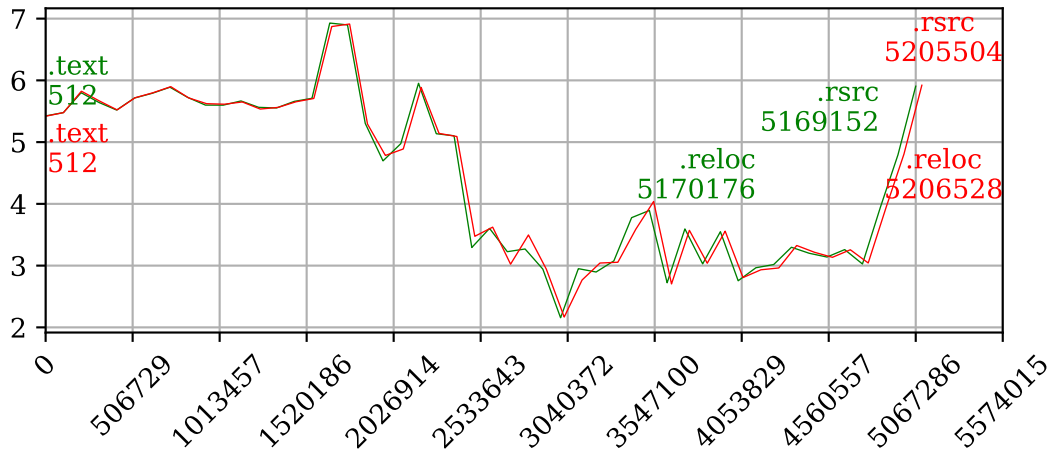
In this section, we discuss what insights can be extracted and consider `Exorcist`'s limitations and potential future research directions. Our research shows that it is possible to accurately detect trojanised binaries in the software supply chain via deployment of the `Exorcist` system.




(a) DoFoil

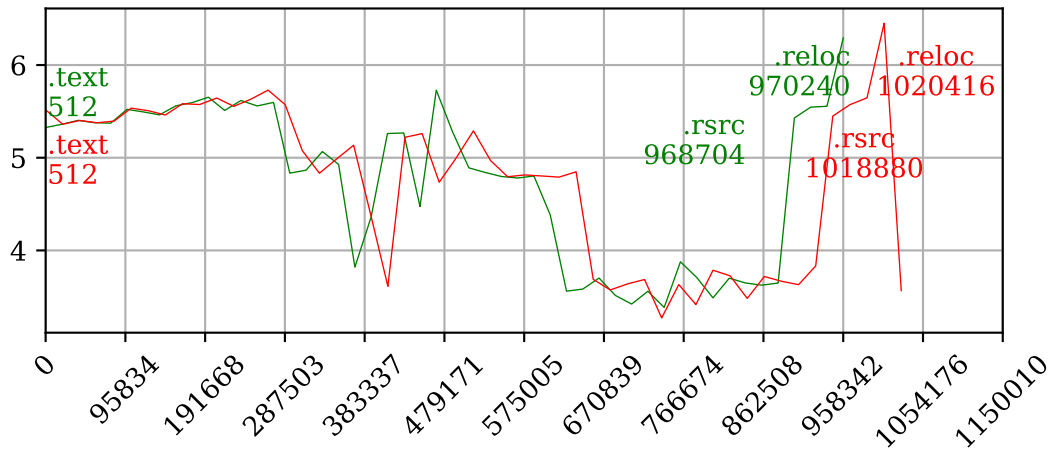



(b) CCleaner

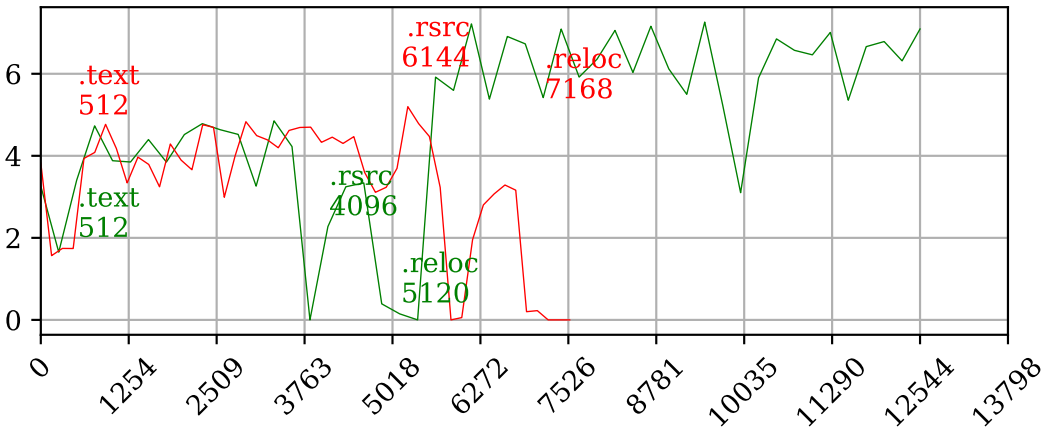



(c) NotPetya (M.E. Doc)

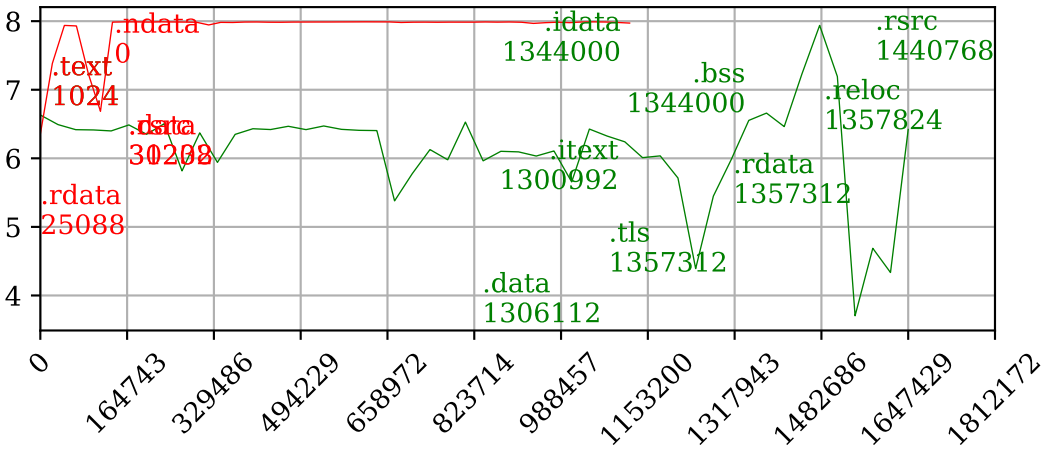
Figure 6.6: Representation of entropy, with entropy on the y axis - on a scale from 0.0 to 8.0. The x axis shows the offset. The green line maps the entropy of different parts of the benign binary and the red line maps the entropy of different parts of the malicious binary. The annotations in green and red delineate the different sections of the binary.




(a) Solarwinds (Sunburst)




(b) Solarwinds (Supernova)




(c) Dragonfly Mbconnect 2

Figure 6.7: Representation of entropy, with entropy on the y axis - on a scale from 0.0 to 8.0. The x axis shows the offset. The green line maps the entropy of different parts of the benign binary and the red line maps the entropy of different parts of the malicious binary. The annotations in green and red delineate the different sections of the binary.

Whilst this chapter is concerned with the design, implementation and testing of a detection system, our use of real-world attacks offers some insight into the nature and characteristics of modern supply chain attacks. We can see that some commonalities emerge with regards to how often each indicator of maliciousness occurs.

One insight from the results of our testing is the prevalence of indicators of obfuscation. We discovered that these obfuscation and packing indicators were present in either minor or major form in all of the malicious samples analysed. This differed from our initial expectation of a spectrum of sophistication for these campaigns, with more sophisticated attacks leaving scarce indicators of obfuscation.

Another surprising result was the prevalence of a subset of indicators. All samples analysed showed the addition of static characteristics indicating maliciousness. These included shifts in OEP, section addition and subtraction or most commonly entropy increases.

### 6.10.2 Limitations

The principle drawback of the system we have designed and implemented is that in the act of publication, we have revealed `Exorcist`'s architecture. As both Kerchoff [245] and Dolev-Yao [246] stated, if the security of a crypto-system relies on its obscurity, this system is not secure. The same applies to all published malware and intrusion detection research.

An attacker who is aware of the architecture of `Exorcist` is able to subvert it. This is a drawback common to all intrusion detection research. The aim of our research is to investigate the feasibility of using differential analysis of software builds to reduce the attack surface of closed source software supply chains.

We do not extend our analysis to Linux Executable Linker Format files or Mac's Mach-O file format as many of `Exorcist`'s analysis components only operate upon PE files on the Windows OS.

Furthermore, due to our scope being limited to binaries that have been trojanised during the supply chain we exclude notable real-world supply chain attacks. We did not include the highly impactful Nobelium [247], Log4J and Hafnium [248] attacks for this reason.

Despite thorough simulation with a range of parameters, we could not extract dynamic behaviour from `.dll`'s. The explanation for this is that `.dll`s are imported as part

of a larger software suite.

There are many commercial and academic research projects that audit and perform static code analysis, however direct analysis of the code preceding its compilation into an executable is not in scope for our research [249–253].

The purposeful insertion of code with vulnerabilities enabling remote code execution or exfiltration of data, rather than directly inserting malicious functionality, will not be detected by `Exorcist` [216]. Similarly, `Exorcist` will not be able to detect the insertion of backdoors. However, this attack vector increases the visibility of an attacker to other detection mechanisms.

### 6.10.3 Future Work

We hope that our research has highlighted that closed-source supply chain attacks can be detected via differential analysis of binary versions and leads to further academic research on this topic.

Advanced threat groups are aware of the latest developments in detection technology and work to evade these detection technologies. We would not recommend that organisations implement `Exorcist` as a 'plug-and-play' system, but instead use the principles of our differential analysis approach.

Engaging with industry and implementing this detection system in actual networks could broaden the research scope and ensure that the findings of this research can be implemented in organisations. Collaborations may offer access to diverse datasets and insights.

The current detection mechanisms can be further refined to improve accuracy and reduce false positives. Future work could explore the integration of machine learning and artificial intelligence to create more adaptive and sophisticated detection algorithms. Efforts to minimise false positives will be critical in ensuring the reliability and operational viability of `Exorcist`, particularly in complex and fast-paced detection environments.

Conducting comprehensive testing of `Exorcist` against malware deployed during open-source supply chain attacks would provide a deeper understanding of its strengths and limitations in this domain. Although we did not cover this during our research, as open-source supply chain security attacks have received sufficient attention in the academic literature, it is possible that our detection mechanisms work equally well on

closed and open source supply chain attacks. At the very least, there are transferable concepts.

## 6.11 Conclusion

In this chapter, we have presented `Exorcist`, a system substantiating a differential analysis approach to detecting trojanised binaries in closed-source supply chains. We took a first step in the academic study of detecting trojanised binaries in software supply chains. We tested this system against 12 real-world supply chain attacks to evaluate its fulfilment of our earlier stated system requirements and found it had a high degree of accuracy and low false positive rate.

# 7 Conclusions

## 7.1 Summary

The key conclusion of this research is that AV contains flaws and that modern malware exploits these flaws to remain undetected. We have evidenced these flaws through systematic exploration of the flaws in practical AV deployments. We have also explored the effectiveness of evasion techniques and their use by malware through large-scale measurement studies and the development of detection solutions.

In the works constituting this thesis we investigate the flaws present in these systems in more depth. We have also identified and measured the exact nature of malware's use of these techniques. We also investigated several evasion techniques that modern malware uses to exploit these weaknesses and robustly tested them against a wide range of commercially successful AV systems. We worked with AV companies to resolve these issues.

In addition to identifying errors in existing systems, we assist with correcting these architectural flaws. Along with the aforementioned responsible disclosure and patch testing, we also assist with the open-sourcing and sharing of detection algorithms to facilitate more effective detection. In Chapter 6, we develop a detection system focusing on a particular evasive malware variant, closed-source supply chain attacks.

## 7.2 Fulfilment of Initial Aims

With reference to Chapter 3, it is important to analyse to what extent we have fulfilled the aims and research questions enumerated by the methodology chapter of our thesis.

### 7.2.1 Fulfilment of Overall Aims

We have comprehensively addressed the primary objective of bridging the detection gap in contemporary AV technologies against modern evasive malware, while contributing significantly to the understanding of malware, especially those variants that are highly evasive. Our detailed analysis of advanced evasion techniques, focusing predominantly on those employed by APT groups. This included several focused case studies.

This thesis also resulted in numerous contributions towards the open-source community. Firstly, Yara rules that we have contributed to the open-source community. Secondly, we have added certain exploitation methods to the `LOLBAS` project. We also disclosed and resolved evasion techniques to AV vendors and an additional 3 zero-day vulnerabilities to AV vendors. This shows a material impact of our research on improving the detection capabilities of AV, particularly versus evasive malware.

The thesis has also contributed towards understanding of evasive malware behaviour. By meticulously cataloguing and analysing a range of evasion techniques, this work not only enhances the existing knowledge base but also offers practical applications for cybersecurity professionals. These applications range from improving heuristic detection algorithms to developing more robust defence strategies against these evolving threats. While the research has made substantial strides in closing the detection gap, it also highlights areas for further exploration, which we explore in greater detail later in this chapter.

### 7.2.2 Fulfilment of Specific Aims

**Understanding Evasive Techniques** The thesis has successfully achieved the aim of systematically studying and categorising various evasion techniques used by modern malware. The detailed examination of methods such as `.dll` sideloading, AV signature manipulation, packing, supply chain attacks, and LotL techniques has been comprehensive. This research has not only highlighted the intricacies of these methods but also their effectiveness in circumventing current AV solutions. The thorough

categorisation and analysis have provided a clear understanding of the operational mechanisms behind these evasion techniques, significantly contributing to academic malware research in this area.

**Evaluation of Evasion Effectiveness** In this area, the thesis has had noteworthy results. The in-depth analysis focused on specific evasion methods: Authenticode signature replacement, packing, `.dll` sideloading and LotL techniques. Providing valuable insights into their impact on AV detection rates. This includes extensive measurement studies, that both characterise and analyse the nature of evasive malware.

This evaluation has been crucial in understanding the practical challenges faced by current AV systems and has highlighted areas where these systems need to evolve. The findings from this evaluation serve as a vital resource for developing more resilient AV technologies that can better cope with the evolving landscape of malware evasion tactics.

**Development of Detection Mechanisms** To contribute to the field by developing and testing new detection systems, particularly for closed-source supply chain attacks. This aim addresses the need for enhanced detection capabilities in light of the sophisticated evasion techniques identified.

The research constituting this thesis has notably advanced the goal of developing new detection systems, particularly for closed-source supply chain attacks. The research undertaken in this area addresses the urgent need for enhanced detection capabilities in response to the sophisticated evasion techniques identified earlier.

These developments are particularly crucial in the context of increasing threats to software supply chains and provide a foundation for future innovations in malware detection.

We worked with vendors to help minimise the detection gap for particular malware variants. In addition to this, we disclosed three zero-day `.dll` hijacking vulnerabilities to AV vendors.

**APT Malware Analysis** We use many case studies to explore this area of our research. Within this research, nation-state groups are at the cutting-edge of malware evasion technique research. One of the novel conclusions that our research has drawn, particularly in Chapter 5, is that these advanced evasion techniques filter

down the food chain. This means commodity malware adopts the techniques of more sophisticated variants.

The focused studies on APT malware conducted in this thesis have effectively differentiated it from commodity malware, particularly in terms of technical sophistication and persistence and the methods used. This research segment has offered significant insights into the innovative evasion methods used by APT groups and their unique operational characteristics.

The findings from this analysis have deepened the understanding of APT malwares' complexities, providing a nuanced view of state-level cyber threats. This analysis is invaluable in shaping future strategies for detecting and mitigating such advanced and persistent threats.

In addition to this targeted research on APT malware, we have also conducted extensive analysis on commodity malware. This helps with understanding the broader nature of evasive malware and how evasive techniques used by advanced groups are propagated to less sophisticated actors.

In the works constituting this thesis, where possible we draw conclusions applicable to the greater majority of malware samples. In Chapter 5, we measure the prevalence of a particular evasion technique over large datasets to facilitate this goal.

## 7.3 Fulfilment of Research Questions

It is important to also ascertain the extent to which the research questions we set out in Section 3.5 have been addressed in our thesis. In this section, we enumerate the various research questions and evaluate to what extent we have answered them.

> *RQ 1: How do modern real-world evasive malware techniques operate, and what are their key characteristics?*

A comprehensive analysis of modern evasive malware techniques has been conducted, elucidating their operational modalities and key characteristics. The study delved into various evasion techniques and helped to provide a detailed understanding of their intricacies.

We have fulfilled this research question by offering a deep and systematic exploration of contemporary evasive malware tactics. This understanding was particularly con-

tributed to by Chapter 4 and Chapter 5, which facilitated this understanding through extensive and thorough measurement studies.

> *RQ 2: What is the effectiveness of current evasion techniques as measured against real-world AV systems?*

The effectiveness of evasion techniques against real-world AV systems has been a central focus of this thesis. Through empirical analysis and testing, the research provided clear insights into the performance of various evasion methods, such as Authenticode signature replacement, packing, LotL techniques and other evasion techniques against contemporary AV systems.

This evaluation revealed significant insights into the limitations of current AV technologies, effectively addressing this research question by quantifying and exemplifying the effectiveness of evasion techniques in real-world scenarios.

Research in this thesis has effectively identified and analysed the existing gaps in AV technologies concerning advanced evasion methods. By systematically and quantitatively evaluating the current limitations and vulnerabilities of AV systems in detecting and countering evasive malware techniques, the research has clearly demonstrated these gaps.

Furthermore, this thesis proposed strategies and improvements to address these shortcomings, thereby directly responding to this research question. The suggestions and findings offer a pathway for enhancing AV technologies, making them more robust against sophisticated evasion strategies.

> *RQ 3: What novel detection mechanisms and systems can be developed to effectively counteract the evolving landscape of evasive malware, particularly in supply chain attacks?*

The research within this thesis has successfully contributed novel detection mechanisms and systems, especially the `Exorcist` system, which we designed to detect supply chain attacks. By developing and testing new detection methodologies, the research has taken significant steps in providing practical tools and algorithms to facilitate countering the evolving threat posed by evasive malware.

This includes the development of systems to assist with their detection. In achieving this aim, we prioritise the creation of real-world systems and make them available via open-source. — In Chapter 6, we develop a detection system for closed-source supply chain attacks. Further to this, we also release open-source 'Yara' rules as part

of Chapter 5, to aid malware researchers and practitioners with detection of LotL techniques. In all experiments conducted as part of this thesis, we have attempted to make our results reproducible and use open-source datasets, or make the datasets we have used available.

## 7.4 Practical Impact of Research

Identifying that AV systems are a flawed model does not suffice as enough of a scientific contribution to merit a doctoral thesis. As such, our research was highly practical and concerned with the analysis and deployment of real-world systems and real-world data. This involved collaboration with academia, industry and government and ensured that the conclusions and tools produced from this research did not simply end up as a theoretical exercise. This was coupled with the fact that throughout the doctorate, talks on this research were given to security practitioners and other organisations.

Existing research attempting to bridge the detection gap is largely contained within corporate whitepapers, blog posts or informal knowledge-sharing networks for malware research. The exploration and measurement of malware development, AV evasion techniques and methods for the detection of closed-source supply chain attacks have largely not been present in the academic literature thus far. One of the principal contributions of this thesis is the insertion of a largely informal and scattered body of knowledge regarding evasive malware into measured and rational scientific papers.

As mentioned repeatedly in this thesis, malware development and detection is a field that is constantly in flux. Therefore there is likely to continually be an evolution of evasion techniques and detection methods. The research conducted in this thesis leads to further encoding of these evasion techniques into the academic corpus, instead of leaving this as an activity to be exclusively pursued by private industry, state-sponsored offensive cyber actors and criminal actors.

We chose to be overly transparent in our communication with AV firms. We hope that this is a model that is emulated for future academic research.

## 7.5 Limitations and Future Work

This section outlines the limitations of our research approach. Whilst our thesis provides important insights into malware detection and analysis, it's important to recognise the constraints under which this research was conducted. Following this acknowledgement, we identify areas for future research that can build on the foundation established by this thesis, addressing these limitations and expanding understanding of evolving malware threats.

### 7.5.1 Limitations

Whilst this research is extensive and acknowledges the dynamic and continually evolving landscape of modern malware, new variants of malware emerge constantly, often equipped with sophisticated evasion techniques. This rapid evolution presents a challenge to maintaining the relevancy and effectiveness of our detection and analysis methods. As such, the findings of this thesis, though comprehensive at the time of writing, must be contextualised within a threat landscape that is in constant flux.

### 7.5.2 Future Areas of Research

We have fulfilled the initial aims of this thesis and contributed extensively to the field of research in evasive malware. The completion of this research does leave several avenues for productive future research in this field. In this section, we will outline several of the potential areas in which research can be conducted.

With regards to the specific evasion techniques and phenomena investigated during this research, in the time since publication of the research constituting this thesis, they have been shown to be used prolifically by threat actors. We also discuss the potential open problems in evasive malware and where future research is required.

**LotL Techniques** In the case of LotL, we have seen LotL techniques become almost universal amongst threat groups, after the timely publication of our paper. Therefore we expect to see further developments regarding academic research in these subfields, reflecting the increased prominence of this technique in the malware ecosystem.

**Cross-Platform LotL Techniques** As we state at the conclusion of Chapter 5, in Section 5.8, we address LotL techniques that can be used on other platforms. These

other OS are Linux and Mac systems. It seems that this would be a fruitful area for further research. As such, a systematic exploration on the use of LotL techniques on these OS by malware could be a valuable area of research. Future work could focus on exploring unique vulnerabilities and defences for each platform.

Whilst we acknowledge the existence of research in Linux and IoT devices, in Section2.2.8, there are other platforms that it may be worth investigating evasive malware upon. Smart devices such as smartwatches have been shown to contain malware [254]. There was a published advisory by the U.S. Army Criminal Investigation Division, warning about the threat of APT malware distribution via smartwatches [255].

**`.dll` hijacking** We have also observed during the course of our research, that `.dll` hijacking has become a popular attack technique for APT groups to use. We, therefore, expect to see academic and industrial research building off the conclusions of our research and further analysing `.dll` hijacking. A systematic study of the use of these attack methods or the vulnerability of a wider sample of software to these attack techniques would be scientifically valuable. Future research could explore the evolution of `.dll` hijacking techniques, their effectiveness against various defences, and methods to systematically detect and prevent such attacks.

**Closed-Source Supply Chains** The amount of attacks making use of the software supply chain as an attack vector will continue to increase over time. This will be the case for both closed-source and open-source supply chains. As such, there is an open problem in the field regarding both measurements of these attacks and research involving the development of systems to detect and prevent these attacks. We hope that both industrial and academic research can progress further in understanding and protecting these supply chains.

**Measurement of AV Evasion by Malware** Galloro et al. recently published a longitudinal analysis of the evasion behaviour of Windows malware [44]. Other future measurement studies in this vein that assist in understanding the nature of the current malware threat through a longitudinal lens would also be valuable. This should also enable the development of more effective detection technologies. Future research could continue to analyse the evasion behaviour of malware through longitudinal studies. These studies could provide critical insights into trends and patterns in malware evolution, informing the development of more effective detection technologies.

**Advanced Evasion Methods** As mentioned in the limitations subsection immediately prior to this, there are continually evolving evasion techniques. The use of direct API access such as Direct Win32 API Calls by malware is a way to bypass detection heuristics and is an area that may be worth further investigation. It is not an evasion technique that is explored within this thesis, though we do employ API call analysis as a method of static analysis in Section 6.5.1 in Chapter 6.

An example of this is the SockDetour backdoor [256], which accessed native Windows libraries to create sockets. Whilst recent research has analysed API usage patterns to determine maliciousness, it should be noted that the paper pursuing this is titled 'Advanced Methods', as such it cannot be assumed that this is implemented by default in detection systems [257].

## 7.6 Final Remarks

We have shown that there is a real threat posed to the security guarantees offered by AV, caused by the use of evasion techniques by malicious actors. Our research shows that these tactics, far from being theoretical or edge cases, are actively employed by both commodity and APT malware. This reality challenges the effectiveness of existing AV solutions and underscores the need for more robust and sophisticated defenses.

AV developers should make their systems more accurate and secure. Whilst the techniques we discovered to be effective in evading AV were in some cases entirely novel, others have been known to the security community for some time. As such, AV developers should endeavour to fix these issues.

In this thesis, we have systematically dissected a range of evasion techniques, revealing their complexity and the deftness with which they circumvent traditional AV defenses. For example, the replacement of Authenticode signatures can be solved with more effective validation mechanisms, rather than the rudimentary heuristics simply checking for the presence of an Authenticode signature. It is also the case that the changing of metadata and addition of packing should not result in dramatic reductions in detection rates and these issues should be rectified by AV developers.

The solutions to these problems are not trivial, but greater attention should be given to these flaws by AV developers. A further instance of these problems is the LotL techniques discussed in this thesis, in Chapter 5. Whilst this is a relatively novel

attack class, OS and AV developers can detect and mitigate these problems. The body of work contained within this thesis contributes towards solving these problems. We contribute towards measurement and understanding of these attack methods. This thesis has responded to these challenges by developing new detection mechanisms, some of which have been open-sourced, contributing to the broader efforts to combat evasive malware. By exploring the complex dynamics of malware evasion and offering practical solutions to enhance detection capabilities, this work establishes an effective basis for future research and development.

# Bibliography


[1] B. Schneier, *Secrets & Lies: Digital Security in a Networked World*. USA: John Wiley & Sons, Inc., 1st ed., 2000.

[2] X. Ugarte-Pedrero, M. Graziano, and D. Balzarotti, "A Close Look at a Daily Dataset of Malware Samples," *ACM Transactions on Privacy and Security*, vol. 22, no. 1, pp. 1–30, 2019.

[3] F. Cohen, "Computer Viruses," 1985.

[4] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *J. of Math*, vol. 58, no. 345-363, p. 5, 1936.

[5] R. Messier, *Network Forensics*. Wiley Publishing, 1st ed., 2017.

[6] M. Sikorski and A. Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. USA: No Starch Press, 1st ed., 2012.

[7] M. Ligh, S. Adair, B. Hartstein, and M. Richard, *Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*. Wiley Publishing, 2010.

[8] X. Ugarte-Pedrero, D. Balzarotti, I. Santos, and P. Bringas, "SoK: Deep Packer Inspection: A Longitudinal Study of the Complexity of Run-Time Packers," 2015.

[9] S. Pastrana and G. Suarez-Tangil, "A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth," *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, no. ii, pp. 73–86, 2019.

[10] T. Holz, C. Gorecki, F. C. Freiling, K. Rieck, and F.-f. S. Networks, "Measuring and Detecting Fast-Flux Service Networks," *Network & Distributed System Security Symposium (NDSS)*, 2008.

[11] H. L. Bijmans, T. M. Booij, and C. Doerr, "Inadvertently making cyber criminals rich: A comprehensive study of cryptojacking campaigns at internet scale," *Proceedings of the 28th USENIX Security Symposium*, pp. 1627–1644, 2019.

[12] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "AVclass: A Tool for Massive Malware Labeling," in *Research in Attacks, Intrusions, and Defenses* (F. Monrose, M. Dacier, G. Blanc, and J. Garcia-Alfaro, eds.), (Cham), pp. 230–253, Springer International Publishing, 2016.

[13] K. Liao, Z. Zhao, A. Doupe, and G. Ahn, "Behind closed doors: measurement and analysis of CryptoLocker ransoms in Bitcoin," in *2016 APWG Symposium on Electronic Crime Research (eCrime)*, pp. 1–13, jun 2016.

[14] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, "Cutting the gordian knot: A look under the hood of ransomware attacks," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9148, pp. 3–24, 2015.

[15] Z. A. Genç, G. Lenzini, and P. Y. Ryan, "No random, no ransom: A key to stop cryptographic ransomware," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10885 LNCS, pp. 234–255, 2018.

[16] A. Kharraz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware," in *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, (USA), pp. 757–772, USENIX Association, 2016.

[17] A. Kharraz and E. Kirda, "Redemption: Real-Time Protection Against Ransomware at End-Hosts," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10453 LNCS, pp. 98–119, 2017.

[18] E. Kolodenker, W. Koch, G. Stringhini, and M. Egele, "PayBreak : Defense against cryptographic ransomware," *ASIA CCS 2017 - Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security*, pp. 599–611, 2017.

[19] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "Tesseract: Eliminating experimental bias in malware classification across space and time," *Proceedings of the 28th USENIX Security Symposium*, pp. 729–746, 2019.

[20] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. Van Steen, "Prudent practices for designing malware experiments: Status quo and outlook," *Proceedings - IEEE Symposium on Security and Privacy*, pp. 65–79, 2012.

[21] B. Yadegari, B. Johannesmeyer, B. Whitely, and S. Debray, "A Generic Approach to Automatic Deobfuscation of Executable Code," in *2015 IEEE Symposium on Security and Privacy*, pp. 674–691, 2015.

[22] T. Blazytko, M. Contag, C. Aschermann, T. Holz, R.-u. Bochum, T. Blazytko, M. Contag, C. Aschermann, and T. Holz, "Syntia : Synthesizing the Semantics of Obfuscated Code This paper is included in the Proceedings of the Syntia : Synthesizing the Semantics of Obfuscated Code," *USENIX Security*, 2017.

[23] J. Calvet, J. M. Fernandez, and J. Y. Marion, "Aligot: Cryptographic function identification in obfuscated binary programs," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 169–182, 2012.

[24] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A View on Current Malware Behaviors," 2009.

[25] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. F. Wang, "Effective and efficient malware detection at the end host," *Proceedings of the 18th USENIX Security Symposium*, pp. 351–366, 2009.

[26] U. Bayer, P. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering," *Network and Distributed System Security Symposium (NDSS)*, 2009.

[27] Z. Zhu and T. Dumitraş, "FeatureSmith: Automatically Engineering Features for Malware Detection by Mining the Security Literature," pp. 767–778, 2016.

[28] D. Balzarotti, M. Cova, C. Karlberger, C. Kruegel, E. Kirda, and G. Vigna, "Efficient Detection of Split Personalities in Malware," *Network and Distributed System Security*, 2010.

[29] M. Graziano, D. Canali, L. Bilge, A. Lanzi, and D. Balzarotti, "Needles in a Haystack: Mining Information from Public Dynamic Analysis Sandboxes for Malware Intelligence," *24th USENIX Security Symposium*, pp. 1057–1072, 2015.

[30] W. U. Hassan, A. Bates, and D. Marino, "Tactical Provenance Analysis for Endpoint Detection and Response Systems," *IEEE Symposium on Security and Privacy*, 2020.

[31] S. Hardy, M. Crete-Nishihata, K. Kleemola, A. Senft, B. Sonne, G. Wiseman, P. Gill, and R. J. Deibert, "Targeted threat index: Characterizing and quantifying politically-motivated targeted malware," *Proceedings of the 23rd USENIX Security Symposium*, pp. 527–541, 2014.

[32] M. Barre, A. Gehani, and V. Yegneswaran, "Mining Data Provenance to Detect Advanced Persistent Threats," *11th International Workshop on Theory and Practice of Provenance (TaPP 2019)*, 2019.

[33] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats," *NDSS 2020, Network and Distributed Systems Security Symposium*, no. February, 2020.

[34] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. N. Venkatakrishnan, "HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1137–1152, may 2019.

[35] E. Hutchins, M. Cloppert, and R. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *6th International Conference on Information Warfare and Security, ICIW 2011*, no. July 2005, pp. 113–125, 2011.

[36] D. E. Denning, "An Intrusion-Detection Model," *IEEE Transactions on Software Engineering*, vol. SE-13, pp. 222–232, feb 1987.

[37] M. Mahmoud, M. Mannan, and A. Youssef, "APTHunter: Detecting Advanced Persistent Threats in Early Stages," *Digital Threats: Research and Practice*, 2022.

[38] Chainalysis, "The 2021 crypto crime report," 2021.

[39] C. Lever, P. Kotzias, D. Balzarotti, J. Caballero, and M. Antonakakis, "A Lustrum of Malware Network Communication: Evolution and Insights," in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 788–804, 2017.

[40] C. C. Ife, Y. Shen, S. J. Murdoch, and G. Stringhini, "Waves of malice: A longitudinal measurement of the malicious file delivery ecosystem on the web,"

*AsiaCCS 2019 - Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pp. 168–180, 2019.

[41] S. Alrwais, X. Liao, X. Mi, P. Wang, X. Wang, F. Qian, R. Beyah, and D. McCoy, "Under the Shadow of Sunshine: Understanding and Detecting Bulletproof Hosting on Legitimate Service Provider Networks," in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 805–823, may 2017.

[42] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of HTTP-based malware and signature generation using malicious network traces," *Proceedings of NSDI 2010: 7th USENIX Symposium on Networked Systems Design and Implementation*, pp. 391–404, 2010.

[43] B. Alahmadi, E. Mariconti, R. Spolaor, G. Stringhini, and I. Martinovic, "BOTection: Bot Detection by Building Markov Chain Models of Bots Network Behavior," *Acm Asiaccs*, 2020.

[44] N. Galloro, M. Polino, M. Carminati, A. Continella, and S. Zanero, "A Systematical and longitudinal study of evasive behaviors in windows malware," *Computers and Security*, vol. 113, 2022.

[45] E. Cozzi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Understanding Linux Malware," *Proceedings - IEEE Symposium on Security and Privacy*, vol. 2018-May, pp. 161–175, 2018.

[46] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in *Proceedings of the 26th USENIX Conference on Security Symposium*, SEC'17, (USA), pp. 1093–1110, USENIX Association, 2017.

[47] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, "Measurement and Analysis of Hajime, a Peer-to-peer IoT Botnet," *Network and Distributed Systems Security*, no. February, 2019.

[48] F. Dang, Z. Li, Y. Liu, E. Zhai, Q. A. Chen, T. Xu, Y. Chen, and J. Yang, "Understanding fileless attacks on linux-based IoT devices with HoneyCloud," *MobiSys 2019 - Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 482–493, 2019.

[49] O. Alrawi, C. Lever, K. Valakuzhy, R. Court, K. Snow, F. Monrose, and M. Antonakakis, "The circle of life: A large-scale study of the IoT malware lifecycle," *Proceedings of the 30th USENIX Security Symposium*, pp. 3505–3522, 2021.

[50] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Transcending transcend: Revisiting malware classification in the presence of concept drift," in *IEEE Symposium on Security and Privacy (IEEE SP)*, pp. 1165–1183, 2022.

[51] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "TranscEnd: Detecting concept drift in malware classification models," *Proceedings of the 26th USENIX Security Symposium*, pp. 625–642, 2017.

[52] D. Kim, B. J. Kwon, and T. Dumitras, "Certified Malware: Measuring breaches of trust in the windows code-signing PKI," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 1435–1448, 2017.

[53] D. Uroz and R. J. Rodríguez, "On Challenges in Verifying Trusted Executable Files in Memory Forensics," *Forensic Science International: Digital Investigation*, vol. 32, p. 300917, 2020.

[54] P. Kotzias, S. Matic, J. Caballero, and R. Rivera, "Certified PUP: Abuse in authenticode code signing," *Proceedings of the ACM Conference on Computer and Communications Security*, vol. 2015-Octob, pp. 465–478, 2015.

[55] M. McGuire and S. Dowling, *Cyber crime: A review of the evidence*. No. October, 2013.

[56] P. Kotzias, L. Bilge, and J. Caballero, "Measuring PUP Prevalence and PUP Distribution through Pay-Per-Install Services This paper is included in the Proceedings of the," *Proceedings of the 25th USENIX Security Symposium*, pp. 739–756, 2016.

[57] R. Chatterjee, P. Doerfler, H. Orgad, S. Havron, J. Palmer, D. Freed, K. Levy, N. Dell, D. McCoy, and T. Ristenpart, "The Spyware Used in Intimate Partner Violence," in *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 441–458, may 2018.

[58] J. Koret and E. Bachaalany, *The Antivirus Hacker's Handbook*. Wiley Publishing, 1st ed., 2015.

[59] J. Chen and A. Ehrlich, "Moshen Dragon's Triad-and-Error Approach — Abusing Security Software to Sideload PlugX and ShadowPad," 2022.

[60] Microsoft Defender Threat Intelligence, "Threat actor leverages coin miner techniques to stay under the radar – here's how to spot them," 2020.

[61] B. Cheng, J. Ming, J. Fu, G. Peng, T. Chen, X. Zhang, and J.-Y. Marion, "Towards Paving the Way for Large-Scale Windows Malware Analysis: Generic Binary Unpacking with Orders-of-Magnitude Performance Boost," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, (New York, NY, USA), pp. 395–411, Association for Computing Machinery, 2018.

[62] A. Yokoyama, K. Ishii, R. Tanabe, Y. Papa, K. Yoshioka, T. Matsumoto, T. Kasama, D. Inoue, M. Brengel, M. Backes, and C. Rossow, "Sandprint: Fingerprinting malware sandboxes to provide intelligence for sandbox evasion," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9854 LNCS, pp. 165–187, 2016.

[63] H. Shi, J. Mirkovic, and A. Alwabel, "Handling anti-virtual machine techniques in malicious software," *ACM Transactions on Privacy and Security*, vol. 21, no. 1, pp. 1–31, 2017.

[64] Y. Oyama, "Investigation of the diverse sleep behavior of malware," *Journal of Information Processing*, vol. 26, no. June, pp. 461–476, 2018.

[65] A. Küchler, A. Mantovani, Y. Han, L. Bilge, and D. Balzarotti, "Does every second count? Time-based evolution of malware behavior in sandboxes," in *NDSS 2021, Network and Distributed Systems Security Symposium, 21-24 February 2021, Virtual Conference* (ISOC, ed.), 2021.

[66] D. Kirat, G. Vigna, and C. Kruegel, "BareBox: Efficient malware analysis on bare-metal," *ACM International Conference Proceeding Series*, pp. 403–412, 2011.

[67] D. Kirat, G. Vigna, C. Kruegel, G. Vigna, and C. Kruegel, "BareCloud: Bare-metal Analysis-based Evasive Malware Detection," *Usenix Security Symposium*, 2014.

[68] N. Miramirkhani, M. P. Appini, N. Nikiforakis, and M. Polychronakis, "Spotless Sandboxes: Evading Malware Analysis Systems Using Wear-and-Tear Artifacts," *Proceedings - IEEE Symposium on Security and Privacy*, pp. 1009–1024, 2017.

[69] R. Paleari, L. Martignoni, G. F. Roglia, and D. Bruschi, "A fistful of redpills: How to automatically generate procedures to detect CPU emulators," *3rd USENIX Workshop on Offensive Technologies, WOOT 2009*, 2009.

[70] R. Tanabe, W. Ueno, K. Ishii, K. Yoshioka, T. Matsumoto, T. Kasama, D. Inoue, and C. Rossow, "Evasive Malware via Identifier Implanting," in *Detection of Intrusions and Malware, and Vulnerability Assessment* (C. Giuffrida, S. Bardin, and G. Blanc, eds.), (Cham), pp. 162–184, Springer International Publishing, 2018.

[71] H. Zhao, Y. Zhang, K. Yang, and T. Kim, "Breaking turtles all the way down: An exploitation chain to break out of Vmware ESXi," *13th USENIX Workshop on Offensive Technologies, WOOT 2019, co-located with USENIX Security 2019*, 2019.

[72] T. Ormandy, "An Empirical Study into the Security Exposure to Hosts of Hostile Virtualized Environments," *Test*, pp. 1–10, 2007.

[73] M. Lindorfer, C. Kolbitsch, and P. Milani Comparetti, "Detecting environment-sensitive malware," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6961 LNCS, pp. 338–357, 2011.

[74] J. A. Pendergrass, N. Hull, J. Clemens, S. Helble, M. Thober, K. McGill, M. Gregory, and P. Loscocco, "Technical Report: A Toolkit for Runtime Detection of Userspace Implants," 2019.

[75] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter, and H. Chen, "You Are What You Do : Hunting Stealthy Malware via Data Provenance Analysis," *Network and Distributed System Security Symposium*, no. February, 2020.

[76] D. Ugarte, D. Maiorca, F. Cara, and G. Giacinto, "PowerDrive: Accurate deobfuscation and analysis of powershell malware," *16th Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, vol. 11543 LNCS, pp. 240–259, 2019.

[77] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert, "ZOZZLE: Fast and Precise in-Browser JavaScript Malware Detection," in *Proceedings of the 20th USENIX Conference on Security*, SEC'11, (USA), p. 3, USENIX Association, 2011.

[78] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna, "Revolver: An automated approach to the detection of evasive web-based malware," *Proceedings of the 22nd USENIX Security Symposium*, pp. 637–651, 2013.

[79] O. Suciu, S. E. Coull, and J. Johns, "Exploring Adversarial Examples in Malware Detection," *2019 IEEE Security and Privacy Workshops (SPW)*, pp. 8–14, 2018.

[80] K. Lucas, M. Sharif, L. Bauer, M. K. Reiter, and S. Shintre, *Malware Makeover: Breaking ML-based Static Analysis by Modifying Executable Bytes*, vol. 1. Association for Computing Machinery, 2021.

[81] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *2018 26th European signal processing conference (EUSIPCO)*, pp. 533–537, IEEE, 2018.

[82] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing Properties of Adversarial ML Attacks in the Problem Space," *2020 IEEE Symposium on Security and Privacy*, pp. 1332–1349, may 2020.

[83] M. Ohm, H. Plate, A. Sykosch, and M. Meier, *Backstabber's Knife Collection : A Review of Supply Chain Attacks*, vol. 1. Springer International Publishing, 2020.

[84] D. L. Vu, I. Pashchenko, F. Massacci, H. Plate, and A. Sabetta, "Towards Using Source Code Repositories to Identify Software Supply Chain Attacks," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, (New York, NY, USA), pp. 2093–2095, Association for Computing Machinery, 2020.

[85] R. Duan, O. Alrawi, R. P. Kasturi, R. Elder, B. Saltaformaggio, and W. Lee, "Towards measuring supply chain attacks on package managers for interpreted languages," in *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*, The Internet Society, 2021.

[86] N. Boucher, I. Shumailov, R. Anderson, and N. Papernot, "Bad Characters: Imperceptible NLP Attacks," *Proceedings - IEEE Symposium on Security and Privacy*, vol. 2022-May, pp. 1987–2004, 2022.

[87] T. D. Wagner, K. Mahbub, E. Palomar, and A. E. Abdallah, "Cyber threat intelligence sharing: Survey and research directions," vol. 87, p. 101589, 2019.

[88] G. Cascavilla, D. A. Tamburri, and W.-J. Van Den Heuvel, "Cybercrime threat intelligence: A systematic multi-vocal literature review," vol. 105, p. 102258, 2021.

[89] A. Swinnen and A. Mesbahi, "One Packer to Rule them All: Empirical Identification, Comparison and Circumvention of Current Antivirus Detection Techniques," *BlackHat USA*, p. 32, 2014.

[90] J. Geng, J. Wang, Z. Fang, Y. Zhou, D. Wu, and W. Ge, "A survey of strategy-driven evasion methods for pe malware: Transformation, concealment, and attack," p. 103595, 2023.

[91] G. Hoglund and J. Butler, *Rootkits: Subverting the Windows Kernel*. Addison-Wesley Software Security Series, Addison-Wesley, 2006.

[92] A. Klein and I. Kotler, "Windows Process Injection in 2019," *Black hat 2019*, 2019.

[93] A. Stewart, "Dll Side-Loading: A Thorn in the Side of the Anti-Virus Industry," 2014.

[94] S. Zhu, J. Shi, L. Yang, B. Qin, Z. Zhang, L. Song, and G. Wang, "Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines," *Usenix*, 2020.

[95] P. Peng, L. Yang, L. Song, and G. Wang, "Opening the blackbox of virustotal: Analyzing online phishing scan engines," *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, pp. 478–485, 2019.

[96] D. Kim, B. J. Kwon, K. Kozák, C. Gates, and T. Dumitraş, "The broken Shield: Measuring revocation effectiveness in the windows code-signing PKI," *Proceedings of the 27th USENIX Security Symposium*, pp. 851–868, 2018.

[97] C. Carmony, M. Zhang, X. Hu, A. Bhaskar, and H. Yin, "Extract Me If You Can: Abusing PDF Parsers in Malware Detectors," in *NDSS 2016, Network and Distributed Systems Security Symposium*, 2016.

[98] A. Mantovani, S. Aonzo, X. Ugarte-Pedrero, A. Merlo, and D. Balzarotti, "Prevalence and Impact of Low-Entropy Packing Schemes in the Malware Ecosystem," in *Network and Distributed System Security Symposium*, 2020.

[99] T. Ormandy, "Sophail : Applied attacks against Sophos Antivirus," pp. 1–30, 2012.

[100] N. Virvilis and D. Gritzalis, "The Big Four - What We Did Wrong in Advanced Persistent Threat Detection?," in *2013 International Conference on Availability, Reliability and Security*, pp. 248–254, 2013.

[101] T. Smith, L. Dickinson, and K. Seamons, "Let's Revoke: Scalable Global Certificate Revocation," *Network and Distributed Systems Security (NDSS) Symposium*, jan 2020.

[102] secretsquirrel, "Sigthief." `https://github.com/secretsquirrel/SigThief`, 2020.

[103] EmpireProject, "Empire." `https://github.com/EmpireProject/Empire`, 2020.

[104] PowerShellMafia, "Powersploit." `https://github.com/PowerShellMafia/PowerSploit`, 2018.

[105] A. Dahan, "Operation Cobalt Kitty: A large-scale APT in Asia carried out by the OceanLotus Group," *Cybereason*, 2017.

[106] G. Severi, J. Meyer, S. Coull, and A. Oprea, "Explanation-guided backdoor poisoning attacks against malware classifiers," *Proceedings of the 30th USENIX Security Symposium*, pp. 1487–1504, 2021.

[107] L. Demetrio, S. E. Coull, B. Biggio, G. Lagorio, A. Armando, and F. Roli, "Adversarial EXEmples: A Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection," *ACM Trans. Priv. Secur.*, vol. 24, sep 2021.

[108] G. Laurenza and R. Lazzeretti, *dAPTaset: A Comprehensive Mapping of APT-Related Data*, vol. 11981 LNCS. Springer International Publishing, 2020.

[109] DeepInstinct, "Beware of the 64-bit Malware," 2017.

[110] M. Russinovich, "Sigcheck."

[111] Github, "Siofra," 2022.

[112] Slintel, "Market Share for Top Anti-Virus Technologies."

[113] S. Jana and V. Shmatikov, "Abusing File Processing in Malware Detectors for Fun and Profit," in *2012 IEEE Symposium on Security and Privacy*, pp. 80–94, 2012.

[114] J. Koret, "Breaking Antivirus Software," *Syscan*, 2014.

[115] T. Dullien, "Weird Machines, Exploitability, and Provable Unexploitability," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 2, pp. 391–403, 2020.

[116] D. Hendler, S. Kels, and A. Rubin, "Detecting malicious powershell commands using deep neural networks," *ASIACCS 2018 - Proceedings of the 2018 ACM Asia Conference on Computer and Communications Security*, pp. 187–197, 2018.

[117] R. Lyda and J. Hamrock, "Using Entropy Analysis to Find Encrypted and Packed Malware," 2007.

[118] Mandiant, "Tracking Malware with Import Hashing," 2014.

[119] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste, "Malware dynamic analysis evasion techniques: A survey," *ACM Computing Surveys*, vol. 52, no. 6, pp. 1–33, 2019.

[120] R. Rubira Branco, G. N. Barbosa, and P. D. Neto, "Scientific but Not Academical Overview of Malware Anti-Debugging, Anti-Disassembly and Anti- VM Technologies - White Paper," *Black Hat*, 2012.

[121] N. Rndic and P. Laskov, "Practical Evasion of a Learning-Based Classifier: A Case Study," in *2014 IEEE Symposium on Security and Privacy*, pp. 197–211, 2014.

[122] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi, "Shieldfs: A self-healing, ransomware-aware filesystem," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ACSAC '16, (New York, NY, USA), p. 336–347, Association for Computing Machinery, 2016.

[123] M. N. Hossain, S. Sheikhi, and R. Sekar, "Combating Dependence Explosion in Forensic Analysis Using Alternative Tag Propagation Semantics," in *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 1139–1155, 2020.

[124] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. D. Stoller, and V. N. Venkatakrishnan, "Sleuth: Real-time attack scenario reconstruction from COTS audit data," *Proceedings of the 26th USENIX Security Symposium*, pp. 487–504, 2017.

[125] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, "ATLAS: A sequence-based learning approach for attack investigation," *Proceedings of the 30th USENIX Security Symposium*, pp. 3005–3022, 2021.

[126] S. M. Milajerdi, R. Gjomemo, B. Eshete, and V. N. Venkatakrishnan, "Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 1795–1812, 2019.

[127] D. Nisi, M. Graziano, Y. Fratantonio, and D. Balzarotti, "Lost in the Loader : The Many Faces of the Windows PE File Format," *24th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '21)*, 2021.

[128] O. Alrawi, M. Ike, M. Pruett, R. P. Kasturi, S. Barua, T. Hirani, B. Hill, and B. Saltaformaggio, "Forecasting malware capabilities from cyber attack memory images," *Proceedings of the 30th USENIX Security Symposium*, pp. 3523–3540, 2021.

[129] J. Zeng, Z. L. Chua, Y. Chen, K. Ji, Z. Liang, and J. Mao, "WATSON: Abstracting Behaviors from Audit Logs via Aggregation of Contextual Semantics," *Proceedings 2021 Network and Distributed System Security Symposium (NDSS)*, no. February, 2021.

[130] E. Avllazagaj, Z. Zhu, L. Bilge, D. Balzarotti, and T. Dumitras, "When malware changed its mind an empirical study of variable program behaviors in the real world," *Proceedings of the 30th USENIX Security Symposium*, pp. 3487–3504, 2021.

[131] M. Piskozub, F. De Gaspari, F. Barr-Smith, L. Mancini, and I. Martinovic, "MalPhase: Fine-Grained Malware Detection Using Network Flow Data," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '21, (New York, NY, USA), pp. 774–786, Association for Computing Machinery, 2021.

[132] H. L. Bijmans, T. M. Booij, and C. Doerr, "Just the tip of the iceberg: Internet-scale exploitation of routers for cryptojacking," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 449–464, 2019.

[133] LOLBAS-Project, "LOLBAS-Project."

[134] FireEye Threat Research, "Dissecting One of APT29's Fileless WMI and PowerShell Backdoors (POSHSPY)," 2017.

[135] "Threat Brief: Iranian-Linked Cyber Operations," 2020.

[136] ClearSky Cyber Security, "MuddyWater Operations in Lebanon and Oman Using an Israeli compromised domain for a two-stage campaign," pp. 1–16, 2018.

[137] B. Rivera and R. Inocencio, "Doing More with Less: A Study of Fileless Infection Attacks," no. September, pp. 71–89, 2015.

[138] A. Rousseau, "Hijacking .NET to Defend PowerShell," *arXiv.org*, p. 2, 2017.

[139] N. Pinkas, L. Rochberger, and M. Zatz, "Triple Threat: Emotet Deploys Trickbot To Steal Data and Spread Ryuk," 2019.

[140] Microsoft, "Out Of Sight But Not Invisible: Defeating Fileless Malware with Behavior Monitoring, AMSI, and Next-Gen AV," tech. rep., Microsoft, 2018.

[141] Microsoft Defender ATP Research Team, "Dismantling a Fileless Campaign: Microsoft Defender ATP's Antivirus Exposes Astaroth Attack," 2019.

[142] Microsoft, "Insights from one year of tracking a polymorphic threat," 2019.

[143] Microsoft Defender ATP Research Team, "Bring your own LOLBin: Multi-stage, fileless Nodersok campaign delivers rare Node.js-based malware," 2019.

[144] StatCounter, "Desktop Operating System Market Share Worldwide."

[145] National Cyber Security Centre, "Advisory: Turla Group Exploits Iranian APT to Expand Coverage of Victims."

[146] National Cyber Security Centre, "APT10 Continuing to Target UK Organisations."

[147] National Cyber Security Centre, "Reckless Campaign of Cyber Attacks by Russian Military Intelligence Service Exposed."

[148] M. Russinovich, "PsExec," 2016.

[149] S. Saad, W. Briguglio, and H. Elmiligi, "The curious case of machine learning in malware detection," *ICISSP 2019 - Proceedings of the 5th International Conference on Information Systems Security and Privacy*, pp. 528–535, 2019.

[150] G. Hăjmăşan, A. Mondoc, and O. Creţ, "Dynamic behavior evaluation for malware detection," *2017 5th International Symposium on Digital Forensic and Security, ISDFS 2017*, pp. 1–6, 2017.

[151] K. Oosthoek and C. Doerr, "SoK : ATT & CK Techniques and Trends in Windows Malware,"

[152] O. Moe, "Living Off The Land Binaries and Scripts."

[153] C. Wueest and H. Anand, "Living off the land and fileless attack techniques," tech. rep., Symantec, 2017.

[154] Microsoft, "Windows Authenticode Portable Executable Signature Format," tech. rep., Microsoft, 2008.

[155] M. Levi, A. Serper, and A. Dahan, "Operation Soft Cell: A Worldwide Campaign Against Telecommunications Providers," *Virus Bulletin*, no. October, pp. 1–13, 2019.

[156] Department of Homeland Security, "Russian Government Cyber Activity Targeting Energy and Other Critical Infrastructure Sectors (TA18-074A)."

[157] A. Brandt and M. Loman, "Living Off Another Land: Ransomware Borrows Vulnerable Driver to Remove Security Software," 2020.

[158] National Cybersecurity Authority, "Destructive Attack " DUSTMAN " Technical Report," 2019.

[159] IBM X-Force Incident Response, "New Destructive Wiper " ZeroCleare " Targets Energy Sector in the Middle East," no. January, 2020.

[160] Microsoft, "Microsoft Recommended Block Rules," 2019.

[161] Z. Li, Y. Chen, Q. A. Chen, T. Zhu, C. Xiong, and H. Yang, "Effective and light-weight deobfuscation and semantic-aware attack detection for powershell scripts," *Proceedings of the ACM Conference on Computer and Communications Security*, p. 3, 2019.

[162] Sudhakar and S. Kumar, "An emerging threat Fileless malware: a survey and research challenges," *Cybersecurity*, vol. 3, no. 1, pp. 1–12, 2020.

[163] Symantec, "Living off the Land: Turning Your Infrastructure Against You," 2019.

[164] V. Svajcer, "Hunting for LoLBins," 2019.

[165] M. Cohen, "Scanning memory with Yara," *Digital Investigation*, vol. 20, pp. 34–43, 2017.

[166] MITRE, "MITRE ATT&CK."

[167] D. Hendler, S. Kels, and A. Rubin, "AMSI-Based Detection of Malicious PowerShell Code Using Contextual Embeddings," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '20, (New York, NY, USA), pp. 679–693, Association for Computing Machinery, 2020.

[168] S. Saad, F. Mahmood, W. Briguglio, and H. Elmiligi, "JSLess: A Tale of a Fileless Javascript Memory-Resident Malware," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11879 LNCS, pp. 113–131, 2019.

[169] U. Bayer, C. Kruegel, and E. Kirda, "TTAnalyze: A tool for analyzing malware," *15th Ann. Conf. European Inst. for Computer Antivirus Research*, pp. 180–192, 2006.

[170] Y. Baychev and L. Bilge, *Spearphishing malware: Do we really know the unknown?*, vol. 10885 LNCS. Springer International Publishing, 2018.

[171] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys*, vol. 44, no. 2, 2012.

[172] Microsoft Defender ATP Research Team, "Latest Astaroth Living-Off-The-Land Attacks Are Even More Invisible But Not Less Observable."

[173] Opswat Metadefender-Cloud, "Windows Anti-malware Market Share Report," 2019.

[174] Context Information Security, "AMSI Bypass," 2019.

[175] Microsoft, "AllowInsecureGuestAuth."

[176] Github, "Powercat," 2019.

[177] D. Bohannon, “Invoke-Obfuscation,” 2019.

[178] Halil Dalabasmaz, “SpookFlare,” 2020.

[179] Github, “Koadic.”

[180] Hood3dRob1n, “JSRat-Py.”

[181] R. Nolen, S. Miller, and R. Valdez, “Threat Advisory: “Squiblydoo” Continues Trend of Attackers Using Native OS Tools to “Live off the Land”.”

[182] O. Moe, “Discovering The Anti-Virus Signature and Bypassing It,” 2020.

[183] H. S. Anderson and P. Roth, “EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models,” 2018.

[184] VirusShare, “VirusShare.com.”

[185] VX Underground, “VX Underground Samples,” 2020.

[186] Impact Cyber Trust, “GT Malware Netflow Daily Feed,” 2020.

[187] MalShare, “MalShare,” 2020.

[188] M. Rezaeirad, B. Farinholt, H. Dharmdasani, P. Pearce, K. Levchenko, and D. McCoy, “Schrödinger’s RAT: Profiling the stakeholders in the remote access Trojan ecosystem,” *Proceedings of the 27th USENIX Security Symposium*, pp. 1043–1060, 2018.

[189] Aptnotes, “aptnotes.”

[190] Yara, “Yara: Cuckoo Module,” 2019.

[191] Microsoft, “Sysinternals Utilities Index,” 2017.

[192] M. Levan and K. Haag, “Old Phishing Attacks Deploy a New Methodology: Verclsid.exe,” 2017.

[193] D. Y. Huang, M. M. Aliapoulios, V. G. Li, L. Invernizzi, E. Bursztein, K. McRoberts, J. Levin, K. Levchenko, A. C. Snoeren, and D. McCoy, “Tracking Ransomware End-to-end,” *Proceedings - IEEE Symposium on Security and Privacy*, vol. 2018-May, no. 1, pp. 618–631, 2018.

[194] Nozomi Networks, “GreyEnergy : Dissecting the Malware from Maldoc to Backdoor Comprehensive Reverse Engineering Analysis,” no. February, 2019.

[195] G Data, “Analysis of Project Cobra,” 2015.

[196] ESET Research, "Diplomats in Eastern Europe Bitten by a Turla Mosquito," no. January, 2018.

[197] Y. Lemmou and E. M. Souidi, "Inside GandCrab Ransomware," in *Cryptology and Network Security* (J. Camenisch and P. Papadimitratos, eds.), (Cham), pp. 154–174, Springer International Publishing, 2018.

[198] Microsoft, "Process Monitor," 2019.

[199] F. Fauzi, "AV Comparison on Fileless Attack," 2020.

[200] Slaeryan, "How I evaded "next-generation" Cylance Smart AntiVirus in less than 15 minutes," 2020.

[201] P. Rascagneres and W. Rascagneres, "GravityRAT - The Two-Year Evolution Of An APT Targeting India," 2019.

[202] GTFOBins, "GTFOBins," 2020.

[203] infosecB, "aptnotes."

[204] K. Berlin, D. Slater, and J. Saxe, "Malicious behavior detection using windows audit logs," *AISec 2015 - Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security, co-located with CCS 2015*, pp. 35–44, 2015.

[205] JPCERT Coordination Center, "Detecting Lateral Movement through Tracking Event Logs," pp. 0–80, 2017.

[206] D. Geer, B. Tozer, and J. S. Meyers, "For good measure: Counting broken links: A quant's view of software supply chain security," *login Usenix Mag.*, vol. 45, no. 4, 2020.

[207] W. Enck and L. Williams, "Top Five Challenges in Software Supply Chain Security: Observations From 30 Industry and Government Organizations," *IEEE Security & Privacy*, vol. 20, pp. 96–100, mar 2022.

[208] A. Greenberg, "The Untold Story of NotPetya, the Most Devastating Cyberattack in History," 2018.

[209] FireEye, "M-Trends 2021 Report," tech. rep., 2021.

[210] N. Boucher and R. Anderson, "Trojan Source: Invisible Vulnerabilities," 2021.

[211] M. Ohm, A. Sykosch, and M. Meier, "Towards detection of software supply chain attacks by forensic artifacts," in *Proceedings of the 15th International*

*Conference on Availability, Reliability and Security*, ARES '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[212] B. Hof and G. Carle, "Software distribution transparency and auditability," *CoRR*, vol. abs/1711.07278, 2017.

[213] S. L. Thomas and A. Francillon, "Backdoors: Definition, Deniability and Detection," in *Research in Attacks, Intrusions, and Defenses* (M. Bailey, T. Holz, M. Stamatogiannakis, and S. Ioannidis, eds.), (Cham), pp. 92–113, Springer International Publishing, 2018.

[214] M. Zimmermann, C.-A. Staicu, C. Tenny, and M. Pradel, "Small world with high risks: A study of security threats in the npm ecosystem," in *28th USENIX Security Symposium (USENIX Security 19)*, (Santa Clara, CA), pp. 995–1010, USENIX Association, Aug. 2019.

[215] S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage, "When Private Keys Are Public: Results from the 2008 Debian OpenSSL Vulnerability," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, IMC '09, (New York, NY, USA), pp. 15–27, Association for Computing Machinery, 2009.

[216] S. Checkoway, J. Maskiewicz, C. Garman, J. Fried, S. Cohney, M. Green, N. Heninger, R. P. Weinmann, E. Rescorla, and H. Shacham, "A systematic analysis of the Juniper Dual EC incident," *Proceedings of the ACM Conference on Computer and Communications Security*, vol. 24-28-Octo, pp. 468–479, 2016.

[217] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel, and G. Vigna, "Firmalice - automatic detection of authentication bypass vulnerabilities in binary firmware," 2015.

[218] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A large-scale analysis of the security of embedded firmwares," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, (USA), p. 95–110, USENIX Association, 2014.

[219] J. Zaddach, A. Kurmus, D. Balzarotti, E.-O. Blass, A. Francillon, T. Goodspeed, M. Gupta, and I. Koltsidas, "Implementation and implications of a stealth hard-drive backdoor," in *Proceedings of the 29th Annual Computer Security Applications Conference*, ACSAC '13, (New York, NY, USA), p. 279–288, Association for Computing Machinery, 2013.

[220] A. Waksman and S. Sethumadhavan, "Silencing hardware backdoors," in *2011 IEEE Symposium on Security and Privacy*, pp. 49–63, 2011.

[221] E. Gustafson, M. Muench, C. Spensky, N. Redini, A. Machiry, Y. Fratantonio, A. Francillon, D. Balzarotti, Y. R. Choe, C. Kruegel, and G. Vigna, "Toward the analysis of embedded firmware through automated re-hosting," *RAID 2019 Proceedings - 22nd International Symposium on Research in Attacks, Intrusions and Defenses*, pp. 135–150, 2019.

[222] A. Moser, C. Kruegel, and E. Kirda, "Limits of Static Analysis for Malware Detection," in *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pp. 421–430, dec 2007.

[223] M. Egele, M. Woo, P. Chapman, and D. Brumley, "Blanket execution: Dynamic similarity testing for program binaries and components," pp. 303–317, 2014.

[224] Y. Duan, X. Li, J. Wang, and H. Yin, "Deepbindiff: Learning program-wide code representations for binary diffing," in *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*, The Internet Society, 2020.

[225] J. Ming, D. Xu, Y. Jiang, and D. Wu, "BinSIM: Trace-based semantic binary diffing via system call sliced segment equivalence checking," *Proceedings of the 26th USENIX Security Symposium*, pp. 253–270, 2017.

[226] E. Baize, "Developing Secure Products in the Age of Advanced Persistent Threats," *IEEE Security Privacy*, vol. 10, no. 3, pp. 88–92, 2012.

[227] D. Plohmann, "Top 100 Windows API functions / DLLs observed (413 families, .NET excluded)," 2022.

[228] B. Cheng, J. Ming, E. A. Leal, H. Zhang, J. Fu, G. Peng, and J. Y. Marion, "Obfuscation-resilient executable payload extraction from packed malware," *Proceedings of the 30th USENIX Security Symposium*, pp. 3451–3468, 2021.

[229] M. Schloegel, T. Blazytko, M. Contag, C. Aschermann, J. Basler, T. Holz, and A. Abbasi, "Loki: Hardening code obfuscation against automated attacks," in *31st USENIX Security Symposium (USENIX Security 22)*, (Boston, MA), pp. 3055–3073, USENIX Association, Aug. 2022.

[230] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[231] S. Banescu and A. Pretschner, "A Tutorial on Software Obfuscation," *Advances in Computers*, vol. 108, pp. 283–353, 2018.

[232] S. Banescu, C. Collberg, V. Ganesh, Z. Newsham, and A. Pretschner, "Code Obfuscation against Symbolic Execution Attacks," (Los Angeles, CA, USA), 2016.

[233] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," tech. rep., Department of Computer Science, The University of Auckland, New Zealand, 1997.

[234] R. Rolles, "Unpacking virtualization obfuscators," in *Proceedings of the 3rd USENIX Conference on Offensive Technologies*, WOOT'09, (USA), p. 1, USENIX Association, 2009.

[235] T. Blazytko, "Automated detection of obfuscated code." `https://synthesis.to/2021/08/10/obfuscation_detection.htmll`, 2021.

[236] Y. Zhou, A. Main, Y. X. Gu, and H. Johnson, "Information hiding in software with mixed boolean-arithmetic transforms," in *Proceedings of the 8th International Conference on Information Security Applications*, WISA'07, (Berlin, Heidelberg), p. 61–75, Springer-Verlag, 2007.

[237] C. Collberg, S. Martin, J. Myers, and B. Zimmerman, "The Tigress diversifying C virtualizer," 2014.

[238] T. Blazytko, "Automated detection of control-flow flattening." `https://synthesis.to/2021/03/03/flattening_detection.html`, 2021.

[239] S. Peisert, B. Schneier, H. Okhravi, F. Massacci, T. Benzel, C. Landwehr, M. Mannan, J. Mirkovic, A. Prakash, and J. B. Michael, "Perspectives on the SolarWinds Incident," *IEEE Security & Privacy*, vol. 19, no. 2, pp. 7–13, 2021.

[240] M. . D. R. Team, "Analyzing Solorigate, the compromised DLL file that started a sophisticated cyberattack, and how Microsoft Defender helps protect customers," 2020.

[241] R. Langner and B. Schneier, "To Kill a Centrifuge," *The Langner Group*, no. November, pp. 1–37, 2013.

[242] P. Rascagneres., "Disassembler and Runtime Analysis," 2017.

[243] Edmund Brumaghin, Ross Gibb, Warren Mercer, Matthew Molyett, and Craig Williams, "CCleanup: A Vast Number of Machines at Risk," *Cisco's Talos Intelligence Group Blog*, 2017.

[244] J. Bremer, "ignore.c @ github.com," 2022.

[245] A. Kerckhoffs, "La cryptographie militaire," *Journal des Sciences Militaires*, pp. 161–191, 1883.

[246] D. Dolev and A. C. Yao, "On the Security of Public Key Protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[247] Ramin Nafisi, "FoggyWeb: Targeted NOBELIUM malware leads to persistent backdoor," 2021.

[248] Microsoft Threat Intelligence Center, "HAFNIUM targeting Exchange Servers with 0-day exploits," 2021.

[249] J. Dahse and T. Holz, "Simulation of Built-in PHP Features for Precise Static Code Analysis," in *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, The Internet Society, 2014.

[250] J.-J. Bai, J. Lawall, Q.-L. Chen, and S.-M. Hu, "Effective static analysis of concurrency Use-After-Free bugs in linux device drivers," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, (Renton, WA), pp. 255–268, USENIX Association, July 2019.

[251] Y. Xiao, B. Chen, C. Yu, Z. Xu, Z. Yuan, F. Li, B. Liu, Y. Liu, W. Huo, W. Zou, and W. Shi, "MVP: Detecting vulnerabilities using Patch-Enhanced vulnerability signatures," in *29th USENIX Security Symposium (USENIX Security 20)*, pp. 1165–1182, USENIX Association, Aug. 2020.

[252] H. Perl, S. Dechand, M. Smith, D. Arp, F. Yamaguchi, K. Rieck, S. Fahl, and Y. Acar, "VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, (New York, NY, USA), pp. 426–437, Association for Computing Machinery, 2015.

[253] J. Dahse and T. Holz, "Static detection of second-order vulnerabilities in web applications," *Proceedings of the 23rd USENIX Security Symposium*, pp. 989–1003, 2014.

[254] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, detection and analysis of malware for smart devices," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 2, pp. 961–987, 2014.

[255] U.S. Army Criminal Investigation Division, "Cid lookout: Unsolicited smartwatch via mail." Online, 2023.

[256] P. A. N. U. 42, "Sockdetour – a silent, fileless, socketless backdoor – targets u.s. defense contractors," 2022.

[257] D. Rabadi and S. G. Teo, "Advanced Windows Methods on Malware Detection and Classification," *ACM International Conference Proceeding Series*, pp. 54–68, 2020.