Exorcist: Automated Differential Analysis to Detect Compromises in Closed-Source Software Supply Chains

Frederick Barr-Smith
University of Oxford
Oxford, United Kingdom

Tim Blazytko
Emproof B.V.
Eindhoven, Netherlands

Richard Baker
University of Oxford
Oxford, United Kingdom

Ivan Martinovic
University of Oxford
Oxford, United Kingdom

ABSTRACT

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.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**; **Intrusion detection systems**.

KEYWORDS

Supply Chain Security; Differential Analysis; Binary Analysis; Malware; Code Signing; Obfuscation; Advanced Persistent Threat

ACM Reference Format:
Frederick Barr-Smith, Tim Blazytko, Richard Baker, and Ivan Martinovic. 2022. Exorcist: Automated Differential Analysis to Detect Compromises in Closed-Source Software Supply Chains. In *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED '22), November 11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3560835.3564550

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SCORED '22, November 2022, Los Angeles, California USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9885-5/22/11…$15.00
https://doi.org/10.1145/3560835.3564550

1 INTRODUCTION

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, [26] 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 [24].

Advanced Persistent Threat (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 networks in which it is deployed.

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 [28]. 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 [25, 27]. The trojanised binaries use evasive mechanisms such as 'Living-Off-The-Land' (LotL) techniques and are highly sophisticated operations [6, 31, 32].

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, enumerated in Table 1 and Table 2. `Exorcist` then determines whether this new binary is malicious through weighting and aggregating these heuristics.

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`

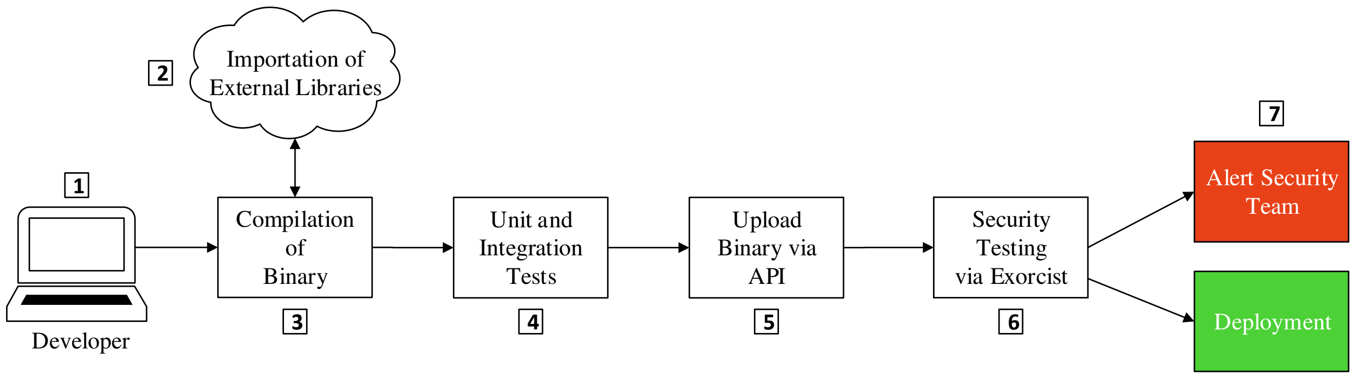
1
Developer
2
Importation of External Libraries
3
Compilation of Binary
4
Unit and Integration Tests
5
Upload Binary via API
6
Security Testing via Exorcist
7
Alert Security Team
Deployment

Figure 1: Positioning of `Exorcist`'s security testing within a typical continuous integration and deployment pipeline (stage 6).

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`.

From developing and testing the detection capabilities of our differential analysis approach implemented in `Exorcist`, the resultant key contributions of our work are:

- 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.
- We implement several detection heuristics that operate via differential analysis of two binaries. We evidence advantages of this approach over 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.

2 CHALLENGES

In this section, we explain the key challenges in systematic detection of closed-source supply chain attacks via differential analysis.

Highly Evasive. Supply chain attacks are designed to be hard to detect. They are frequently conducted by APT groups [3], that Han et al. show using 'low-and-slow' techniques, wherein malware implants slowly and cautiously undertake their actions upon objectives [30]. APT attacks using evasive mechanisms such as LotL techniques have been investigated [31, 63], 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.

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.

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.

3 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.

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.

3.1 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 Portable Executable (PE) binaries [45]. This includes Dynamically Linked Libraries (DLLs) and Microsoft Installer (MSI) files.

3.2 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.

3.3 System Overview

To fulfill these requirements 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 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.

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.

4 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.

4.1 Automated Static Analysis

To determine the legitimacy of a given binary we deploy different static analysis methods, that we list in Table 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.

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

Table 1: Detection heuristics implemented for differential analysis of the static properties of successive binary versions.

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 [50].

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 [40]. 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 [13, 14]. 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 [54]. 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 [34], such as Stuxnet and Flame. 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 [61].

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 [41].

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 [1, 60]:

`Exorcist` calculates the Shannon entropy [55], 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 [39]. An entropy increase across an entire binary or a section between software versions may indicate compression or cryptographic operations. `Exorcist` adopts Mantovani et al. [41]'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 [4, 5, 16, 53]. 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 [10]. 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 [15, 67] which significantly increase code size, we compute average instructions per basic block. For control-flow obfuscations such as opaque predicates and range dividers [4], 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* [9] finds.

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.

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

Table 2: Detection heuristics implemented for differential analysis of dynamic behaviour of successive binary versions.

4.2 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 2.

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.

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 modification of the operating system 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.

4.3 Weighted Heuristic

Once the results of all our differential analyses are assembled, they are combined to produce a final determination of whether a binary

			Static					Dynamic				
	Campaign	Filetype	API Calls	Packing	Binary Signing	Obfuscation	Security (Dll) Characteristics	Registry Modification	Network Activity	Filesystem Modification	LotL	Score
	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
	Ewon 2	EXE	●	●	●	◐	●	◐	○	●	●	13.0
	Mbconnect 1	EXE	●	●	●	◐	●	●	○	●	●	13.5
	Mbconnect 2	EXE	●	●	○	◐	○	○	○	○	○	4.5
	Mesa Imaging	MSI	●	●	○	●	●	○	○	○	●	11.0
Solar	Sunburst	DLL	○	●	○	●	○	◐	○	◐	○	7.0
	Supernova	DLL	○	●	○	●	○	○	○	○	○	6.0

Table 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.

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:

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

Weights are then applied at a per-category level, as we list in Table 1 and 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 [35] 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 α. If $S \geq \alpha$ then the second iteration of the binary is considered malicious, otherwise it is deemed benign. The value of α 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 accomodate a small number of false positive indicators.

5 CASE STUDIES

To demonstrate the practical effectiveness of our approach, 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 3, with the exact versions and timelines in Appendix A.

5.1 SolarWinds

Multiple trojans were inserted in various SolarWinds products in 2020 in a closed-source supply chain attack [48].

Sunburst. Multiple threat intelligence reports recognise this campaign as evasive [58]. 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 5.

`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'.

5.2 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.

5.3 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 [38]. `Exorcist` detected this from the large amount of new registry operations added between versions, shown in Table 4. These were principally composed of registry read operations that identified the environment the binary was

	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 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.

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 in Rascagnere's analysis of the incident [52]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' [22] as anti-VM functionality.

5.4 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.

5.5 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 services.

```
C:\Windows\System32\cmd.exe /c sc stop Talk2mVpnService
C:\Windows\System32\cmd.exe /c taskkill /F /T /IM
        Talk2MVpnService.exe
```

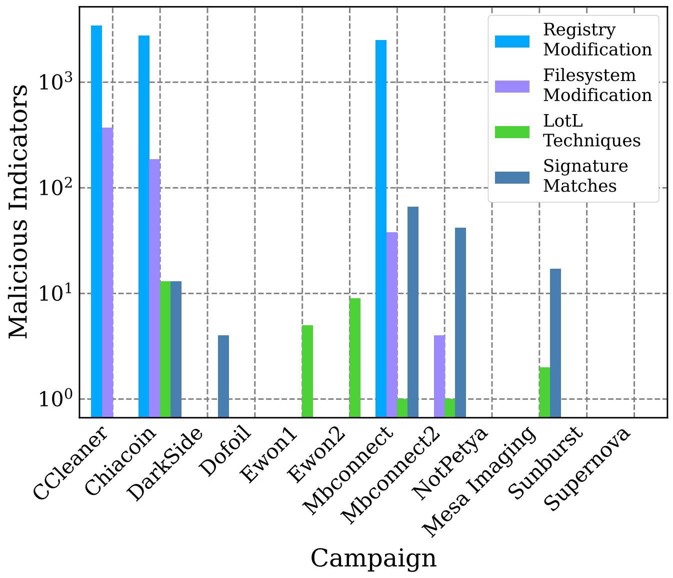

Figure 2: Comparative frequency of dynamic analysis indicators. The Y axis represents total number of maliciousness indicators in the trojanised binary and not the initial file.

`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 antivirus 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 2 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

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

addition of suspicious API calls. 'GetTickCount' detects if malware samples are executing in a virtualised environment [56]. 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'.

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:

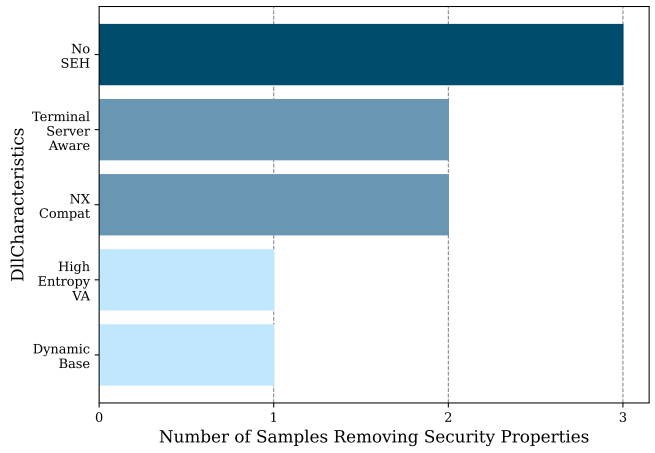

Figure 3: Frequency of dll characteristic alteration in samples between builds in APT malware dataset.

```
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
```

5.6 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.

5.7 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 4, the trojanised Chiacoin binary displayed different post-execution behaviour by reading 184 files and deleting 2.

6 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`.

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

Table 7: 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.

6.1 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 4, we tabulate the different indicators `Exorcist`'s heuristics detected.

We visualise the frequency of indicators of maliciousness within our APT dataset in Figure 2. 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.

6.2 Static Analysis Indicators

In Table 5, 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. 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 7.

Figure 3 shows the frequency of 'DLLCharacteristics' removed between build versions in our case studies. 'DLLCharacteristics' are a misnomer, as 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.

6.3 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 7.

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 2.

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[11].

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.

7 DISCUSSION

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.

7.1 Limitations

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.

Our scope is limited to binaries that have been trojanised during the supply chain, so we exclude notable real-world supply chain attacks such as Nobelium, Log4J and Hafnium [42, 51] attacks.

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 [2, 18, 19, 49, 64].

8 RELATED WORK

There has been recent research into open-source supply chain security [33, 46, 47, 59], 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.

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. [20, 46, 69], 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 [12, 65]. Detection and analysis of hardware and firmware backdoors are studied in the academic literature [17, 29, 57, 62, 66].

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 [34]. Our identification of malicious functionality via Windows API calls builds upon the ransomware detection developed by Kolodenker et al. [37].

Other work has similarly recognised the limitations of static analysis for detecting maliciousness in purposely obfuscated binaries [1] and detection of malware in general [44].

Binary Similarity. Binary analysis has been covered extensively in the security literature. Egele et al. dynamically executed binaries to determine their similarity [23]. BinDiff is used for binary similarity detection, with DeepBinDiff augmenting its capabilities using machine learning [21]. Ming et al. [43] 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 [7, 8, 36]. VirusTotal has since become the default analysis platform, with sufficient material for Zhu et al. to analyse 15 academic papers using the platform [68].

9 CONCLUSION

In this paper, 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 accuracy.

ACKNOWLEDGEMENTS

This research was funded in whole or in part by EPSRC. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission.

REFERENCES

[1] Hojjat Aghakhani, Fabio Gritti, Francesco Mecca, Martina Lindorfer, Stefano Ortolani, Davide Balzarotti, Giovanni Vigna, and Christopher Kruegel. 2020. When Malware is Packin' Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features. In *Network and Distributed System Security (NDSS) Symposium*. Network and Distributed System Security Symposium. https://doi.org/10.14722/ndss.2020.24310

[2] Jia-Ju Bai, Julia Lawall, Qiu-Liang Chen, and Shi-Min Hu. 2019. Effective Static Analysis of Concurrency Use-After-Free Bugs in Linux Device Drivers. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 255–268. https://www.usenix.org/conference/atc19/presentation/bai

[3] Eric Baize. 2012. Developing Secure Products in the Age of Advanced Persistent Threats. *IEEE Security Privacy* 10, 3 (2012), 88–92. https://doi.org/10.1109/MSP.2012.65

[4] Sebastian Banescu, Christian Collberg, Vijay Ganesh, Zack Newsham, and Alexander Pretschner. 2016. Code Obfuscation against Symbolic Execution Attacks. In *Annual Computer Security Applications Conference (ACSAC)*. Los Angeles, CA, USA.

[5] Sebastian Banescu and Alexander Pretschner. 2018. A Tutorial on Software Obfuscation. *Advances in Computers* 108 (2018), 283–353.

[6] F Barr-Smith, X Ugarte-Pedrero, M Graziano, R Spolaor, and I Martinovic. 2021. Survivalism: Systematic Analysis of Windows Malware Living-Off-the-Land. In *2021 2021 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 806–823. https://doi.org/10.1109/SP40001.2021.00047

[7] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. 2009. Scalable, behavior-based malware clustering. In *NDSS 2009, 16th Annual Network and Distributed System Security Symposium, February 8-11, 2009, San Diego, USA*, ISOC (Ed.). San Diego.

[8] Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. 2009. A View on Current Malware Behaviors. In *Proceedings of the 2nd USENIX Conference on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More* (Boston, MA) *(LEET'09)*. USENIX Association, USA.

[9] Tim Blazytko. 2021. Automated Detection of Control-flow Flattening. https://synthesis.to/2021/03/03/flattening_detection.html.

[10] Tim Blazytko. 2021. Automated Detection of Obfuscated Code. https://synthesis.to/2021/08/10/obfuscation_detection.htmll.

[11] Juriaan Bremer. 2022. ignore.c @ github.com. https://github.com/cuckoosandbox/monitor/blob/master/src/ignore.c

[12] Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf Philipp Weinmann, Eric Rescorla, and Hovav Shacham. 2016. A systematic analysis of the Juniper Dual EC incident. *Proceedings of the ACM Conference on Computer and Communications Security* 24-28-Octo (2016), 468–479. https://doi.org/10.1145/2976749.2978395

[13] Binlin Cheng, Jiang Ming, Jianmin Fu, Guojun Peng, Ting Chen, Xiaosong Zhang, and Jean-Yves Marion. 2018. 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* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, Toronto, Canada, 395–411. https://doi.org/10.1145/3243734.3243771

[14] Binlin Cheng, Jiang Ming, Erika A Leal, Haotian Zhang, Jianming Fu, Guojun Peng, and Jean-Yves Marion. 2021. Obfuscation-Resilient Executable Payload Extraction From Packed Malware. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 3451–3468. https://www.usenix.org/conference/usenixsecurity21/presentation/cheng-binlin

[15] Christian Collberg, Sam Martin, Jonathan Myers, and Bill Zimmerman. 2014. The Tigress diversifying C virtualizer. http://tigress.cs.arizona.edu/

[16] Christian Collberg, Clark Thomborson, and Douglas Low. 1997. *A Taxonomy of Obfuscating Transformations.* Technical Report 148. Department of Computer Sciences, The University of Auckland. http://www.cs.auckland.ac.nz/$\sim$collberg/Research/Publications/CollbergThomborsonLow97a/index.html

[17] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. 2014. A Large-Scale Analysis of the Security of Embedded Firmwares. In *Proceedings of the 23rd USENIX Conference on Security Symposium* (San Diego, CA) *(SEC'14)*. USENIX Association, USA, 95–110.

[18] Johannes Dahse and Thorsten Holz. 2014. 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. https://www.ndss-symposium.org/ndss2014/simulation-built-php-features-precise-static-code-analysis

[19] Johannes Dahse and Thorsten Holz. 2014. Static detection of second-order vulnerabilities in web applications. *Proceedings of the 23rd USENIX Security Symposium* (2014), 989–1003. https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-dahse.pdf

[20] Ruian Duan, Omar Alrawi, Ranjita Pai Kasturi, Ryan Elder, Brendan Saltaformaggio, and Wenke Lee. 2021. 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. https://www.ndss-symposium.org/ndss-paper/towards-measuring-supply-chain-attacks-on-package-managers-for-interpreted-languages/

[21] Yue Duan, Xuezixiang Li, Jinghan Wang, and Heng Yin. 2020. 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. https://www.ndss-symposium.org/ndss-paper/deepbindiff-learning-program-wide-code-representations-for-binary-diffing/

[22] Edmund Brumaghin, Ross Gibb, Warren Mercer, Matthew Molyett, and Craig Williams. 2017. CCleanup: A Vast Number of Machines at Risk. *Cisco's Talos Intelligence Group Blog* (2017). http://blog.talosintelligence.com/2017/09/avast-distributes-malware.html

	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 15
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	OrionWeb v2020.2.15300.12766	OrionWeb v2019.4.5200.9083	2020-08-11	2020-03-24

Table 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 9: Versions and Release Dates of software in false positive dataset.

[23] Manuel Egele, Maverick Woo, Peter Chapman, and David Brumley. 2014. Blanket execution: Dynamic similarity testing for program binaries and components. *Proceedings of the 23rd USENIX Security Symposium*, 303–317. https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-egele.pdf

[24] W Enck and L Williams. 2022. Top Five Challenges in Software Supply Chain Security: Observations From 30 Industry and Government Organizations. *IEEE Security & Privacy* 20, 02 (mar 2022), 96–100. https://doi.org/10.1109/MSEC.2022.3142338

[25] FireEye. 2021. *M-Trends 2021 Report*. Technical Report. https://content.fireeye.com/m-trends/rpt-m-trends-2021

[26] Dan Geer, Bentz Tozer, and John Speed Meyers. 2020. For Good Measure: Counting Broken Links: A Quant's View of Software Supply Chain Security. *login Usenix Mag.* 45, 4 (2020). https://www.usenix.org/publications/login/winter2020/geer

[27] Mariano Graziano, Davide Canali, Leyla Bilge, Andrea Lanzi, and Davide Balzarotti. 2015. Needles in a haystack: Mining information from public dynamic analysis sandboxes for malware intelligence. *Proceedings of the 24th USENIX Security Symposium* (2015), 1057–1072. http://www.s3.eurecom.fr/docs/usenixsec15_graziano.pdf

[28] Andy Greenberg. 2018. The Untold Story of NotPetya, the Most Devastating Cyberattack in History.

[29] Eric Gustafson, Marius Muench, Chad Spensky, Nilo Redini, Aravind Machiry, Yanick Fratantonio, Aurélien Francillon, Davide Balzarotti, Yung Ryn Choe, Christopher Kruegel, and Giovanni Vigna. 2019. Toward the analysis of embedded firmware through automated re-hosting. *RAID 2019 Proceedings - 22nd International Symposium on Research in Attacks, Intrusions and Defenses* (2019), 135–150. https://www.usenix.org/system/files/raid2019-gustafson.pdf

[30] Xueyuan Han, Thomas F. J.-M. Pasquier, Adam Bates 0001, James Mickens, and Margo Seltzer. 2020. Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020.* The Internet Society. https://www.ndss-symposium.org/ndss-paper/unicorn-runtime-provenance-based-detector-for-advanced-persistent-threats/

[31] Wajih Ul Hassan, Adam Bates, and Daniel Marino. 2020. Tactical Provenance Analysis for Endpoint Detection and Response Systems. In *2020 IEEE Symposium on Security and Privacy (SP).* 1172–1189. https://doi.org/10.1109/SP40000.2020.00096

[32] Danny Hendler, Shay Kels, and Amir Rubin. 2018. Detecting Malicious PowerShell Commands Using Deep Neural Networks. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security* (Incheon, Republic of Korea) *(ASIACCS '18)*. Association for Computing Machinery, New York, NY, USA, 187–197. https://doi.org/10.1145/3196494.3196511

[33] Benjamin Hof and Georg Carle. 2017. Software Distribution Transparency and Auditability. *CoRR* abs/1711.07278 (2017). arXiv:1711.07278 http://arxiv.org/abs/1711.07278

[34] Doowon Kim, Bum Jun Kwon, and Tudor Dumitras. 2017. Certified Malware: Measuring Breaches of Trust in the Windows Code-Signing PKI. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 1435–1448. https://doi.org/10.1145/3133956.3133958

[35] Doowon Kim, Bum Jun Kwon, Kristián Kozák, Christopher Gates, and Tudor Dumitraş. 2018. The broken Shield: Measuring revocation effectiveness in the windows code-signing PKI. *Proceedings of the 27th USENIX Security Symposium* (2018), 851–868. https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-kim.pdf

[36] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and Xiao Feng Wang. 2009. Effective and efficient malware detection at the end host. *Proceedings of the 18th USENIX Security Symposium* (2009), 351–366. https://static.usenix.org/event/sec09/tech/full_papers/sec09_malware.pdf

[37] Eugene Kolodenker, William Koch, Gianluca Stringhini, and Manuel Egele. 2017. PayBreak : Defense against cryptographic ransomware. *ASIA CCS 2017 - Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security* (2017), 599–611. https://doi.org/10.1145/3052973.3053035

[38] Ralph Langner and Bruce Schneier. 2013. To Kill a Centrifuge. *The Langner Group* November (2013), 1–37. www.langner.com

[39] Robert Lyda and James Hamrock. 2007. Using entropy analysis to find encrypted and packed malware. *IEEE Security & Privacy* 5, 2 (2007), 40–45.

[40] Mandiant. 2014. Tracking Malware with Import Hashing. https://www.fireeye.com/blog/threat-research/2014/01/tracking-malware-import-hashing.html

[41] Alessandro Mantovani, Simone Aonzo, Xabier Ugarte-Pedrero, Alessio Merlo, and Davide Balzarotti. 2020. Prevalence and Impact of Low-Entropy Packing Schemes in the Malware Ecosystem. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020.* https://www.ndss-symposium.org/ndss-paper/prevalence-and-impact-of-low-entropy-packing-schemes-in-the-malware-ecosystem/

[42] Microsoft Threat Intelligence Center. 2021. HAFNIUM targeting Exchange Servers with 0-day exploits. https://www.microsoft.com/security/blog/2021/03/02/hafnium-targeting-exchange-servers/

[43] Jiang Ming, Dongpeng Xu, Yufei Jiang, and Dinghao Wu. 2017. BinSIM: Trace-based semantic binary diffing via system call sliced segment equivalence checking. *Proceedings of the 26th USENIX Security Symposium* (2017), 253–270. https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-ming.pdf

[44] A Moser, C Kruegel, and E Kirda. 2007. Limits of Static Analysis for Malware Detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. 421–430. https://doi.org/10.1109/ACSAC.2007.21

[45] Dario Nisi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. 2021. Lost in the Loader:The Many Faces of the Windows PE File Format. In *24th International Symposium on Research in Attacks, Intrusions and Defenses* (San Sebastian, Spain) *(RAID '21)*. Association for Computing Machinery, New York, NY, USA, 177–192. https://doi.org/10.1145/3471621.3471848

[46] Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. 2020. Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Clémentine Maurice, Leyla Bilge, Gianluca Stringhini, and Nuno Neves (Eds.). Springer International Publishing, Cham, 23–43.

[47] Marc Ohm, Arnold Sykosch, and Michael Meier. 2020. Towards Detection of Software Supply Chain Attacks by Forensic Artifacts. In *Proceedings of the 15th International Conference on Availability, Reliability and Security* (Virtual Event, Ireland) *(ARES '20)*. Association for Computing Machinery, New York, NY, USA, Article 65, 6 pages. https://doi.org/10.1145/3407023.3409183

[48] Sean Peisert, Bruce Schneier, Hamed Okhravi, Fabio Massacci, Terry Benzel, Carl Landwehr, Mohammad Mannan, Jelena Mirkovic, Atul Prakash, and James Bret Michael. 2021. Perspectives on the SolarWinds Incident. *IEEE Security & Privacy* 19, 2 (2021), 7–13. https://doi.org/10.1109/MSEC.2021.3051235

[49] Henning Perl, Sergej Dechand, Matthew Smith, Daniel Arp, Fabian Yamaguchi, Konrad Rieck, Sascha Fahl, and Yasemin Acar. 2015. 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)*. Association for Computing Machinery, New York, NY, USA, 426–437. https://doi.org/10.1145/2810103.2813604

[50] Daniel Plohmann. 2022. Top 100 Windows API functions / DLLs observed (413 families, .NET excluded). https://malpedia.caad.fkie.fraunhofer.de/stats/api_dll_frequencies

[51] Ramin Nafisi. 2021. FoggyWeb: Targeted NOBELIUM malware leads to persistent backdoor. https://www.microsoft.com/security/blog/2021/09/27/foggyweb-targeted-nobelium-malware-leads-to-persistent-backdoor/

[52] Paul Rascagneres. 2017. Disassembler and Runtime Analysis. https://blog.talosintelligence.com/2017/10/disassembler-and-runtime-analysis.html

[53] Rolf Rolles. 2009. Unpacking Virtualization Obfuscators. In *Proceedings of the 3rd USENIX Conference on Offensive Technologies* (Montreal, Canada) *(WOOT'09)*. USENIX Association, USA, 1.

[54] Moritz Schloegel, Tim Blazytko, Moritz Contag, Cornelius Aschermann, Julius Basler, Thorsten Holz, and Ali Abbasi. 2022. Loki: Hardening Code Obfuscation Against Automated Attacks. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 3055–3073. https://www.usenix.org/conference/usenixsecurity22/presentation/schloegel

[55] C. E. Shannon. 1948. A Mathematical Theory of Communication. *Bell System Technical Journal* 27, 3 (1948), 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

[56] Hao Shi, Jelena Mirkovic, and Abdulla Alwabel. 2017. Handling anti-virtual machine techniques in malicious software. *ACM Transactions on Privacy and Security (TOPS)* 21, 1 (2017), 1–31.

[57] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. 2015. Firmalice - Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware. (2015). https://www.ndss-symposium.org/ndss2015/firmalice-automatic-detection-authentication-bypass-vulnerabilities-binary-firmware

[58] Microsoft 365 Defender Research Team. 2020. Analyzing Solorigate, the compromised DLL file that started a sophisticated cyberattack, and how Microsoft Defender helps protect customers. https://www.microsoft.com/security/blog/2020/12/18/analyzing-solorigate-the-compromised-dll-file-that-started-a-sophisticated-cyberattack-and-how-microsoft-defender-helps-protect/

[59] Sam L Thomas and Aurélien Francillon. 2018. Backdoors: Definition, Deniability and Detection. In *Research in Attacks, Intrusions, and Defenses*, Michael Bailey, Thorsten Holz, Manolis Stamatogiannakis, and Sotiris Ioannidis (Eds.). Springer International Publishing, Cham, 92–113.

[60] Xabier Ugarte-Pedrero, Davide Balzarotti, Igor Santos, and Pablo Bringas. 2015. SoK: Deep Packer Inspection: A Longitudinal Study of the Complexity of Run-Time Packers. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7163053

[61] Daniel Uroz and Ricardo J. Rodríguez. 2020. On Challenges in Verifying Trusted Executable Files in Memory Forensics. *Forensic Science International: Digital Investigation* 32 (2020), 300917. https://doi.org/10.1016/j.fsidi.2020.300917

[62] Adam Waksman and Simha Sethumadhavan. 2011. Silencing Hardware Backdoors. In *2011 IEEE Symposium on Security and Privacy*. 49–63. https://doi.org/10.1109/SP.2011.27

[63] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, Carl A. Gunter, and Haifeng Chen. 2020. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/you-are-what-you-do-hunting-stealthy-malware-via-data-provenance-analysis/

[64] Yang Xiao, Bihuan Chen, Chendong Yu, Zhengzi Xu, Zimu Yuan, Feng Li, Binghong Liu, Yang Liu, Wei Huo, Wei Zou, and Wenchang Shi. 2020. MVP: Detecting Vulnerabilities using Patch-Enhanced Vulnerability Signatures. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 1165–1182. https://www.usenix.org/conference/usenixsecurity20/presentation/xiao

[65] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. 2009. 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)*. Association for Computing Machinery, New York, NY, USA, 15–27. https://doi.org/10.1145/1644893.1644896

[66] Jonas Zaddach, Anil Kurmus, Davide Balzarotti, Erik-Oliver Blass, Aurélien Francillon, Travis Goodspeed, Moitrayee Gupta, and Ioannis Koltsidas. 2013. Implementation and Implications of a Stealth Hard-Drive Backdoor. In *Proceedings of the 29th Annual Computer Security Applications Conference* (New Orleans, Louisiana, USA) *(ACSAC '13)*. Association for Computing Machinery, New York, NY, USA, 279–288. https://doi.org/10.1145/2523649.2523661

[67] Yongxin Zhou, Alec Main, Yuan X. Gu, and Harold Johnson. 2007. Information Hiding in Software with Mixed Boolean-Arithmetic Transforms. In *Proceedings of the 8th International Conference on Information Security Applications* (Jeju Island, Korea) *(WISA'07)*. Springer-Verlag, Berlin, Heidelberg, 61–75.

[68] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. 2020. Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2361–2378. https://www.usenix.org/conference/usenixsecurity20/presentation/zhu

[69] Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, and Michael Pradel. 2019. Small World with High Risks: A Study of Security Threats in the npm Ecosystem. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 995–1010. https://www.usenix.org/conference/usenixsecurity19/presentation/zimmerman

A VERSIONS

In Table 8 we list the versions and release dates of the APT case studies we analyse. In Table 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.