EXAMENSARBETE INOM TEKNIK,
GRUNDNIVÅ, 15 HP
*STOCKHOLM, SVERIGE 2020*

# Towards Extending Probabilistic Attack Graphs with Forensic Evidence

An investigation of property list files in macOS

**OLLE HOVMARK**

**EMMA SCHÜLDT**

**KTH**
**SKOLAN FÖR ELEKTROTEKNIK OCH DATAVETENSKAP**

# Towards Extending Probabilistic Attack Graphs with Forensic Evidence

An investigation of property list files in macOS

Olle Hovmark
Emma Schüldt

Degree Project in Computer Science, DD142X
Date: June 2020
Supervisors: Robert Lagerström & Mathias Ekstedt
Examiner: Pawel Herman
KTH Royal Institute of Technology
School of Electrical Engineering and Computer Science

# Abstract

Cyber-attacks against all types of systems is a growing problem in society. Since the Mac operating systems are becoming more common, so are the attacks against them. Probabilistic attack graphs are a way to model cyber-attacks. The *Meta Attack Language* is a language that can be used to create domain-specific languages that in turn can be used to model an attack on the specific domain with a probabilistic attack graph. This report investigates how the Meta Attack Language can be extended so that it could be used for creating attack graphs with forensic evidence, by focusing on attacks on Mac operating systems that has left evidence in the form of property list files. The MITRE ATT&CK matrix is a knowledge base with information about cyber-attacks. A study of the matrix was made to examine what evidence has been found from attacks on a Mac operating system and also to motivate why this report focuses on evidence in the form of property list files. A study on grey literature was then made to investigate different types of attacks that has left evidence in the form of property list files. The studies showed that there are a multitude of evidence that could be left from an attack on a Mac operating system and that most evidence in the form of property list files was used by the adversary as persistence mechanisms. They also showed that the property list files often were placed at root level in the file system. The studies also showed that the adversary often tried to hide the files by giving them names that are common in a Mac operating system. After the studies were conducted a list of requirements for extending the Meta Attack Language was created. This list was based on the results from the studies and included requirements that says there must be a way of expressing the name and location of the files, detection evasion methods, connections between different types of evidence or between evidence and attack steps, and more.

## Sammanfattning

Cyberattacker mot alla typer av system är ett växande problem i samhället. Eftersom Mac-operativsystemen blir allt vanligare, blir attackerna mot dem också vanligare. Probabilistiska attackgrafer är ett sätt att modellera och visualisera cyberattacker. *Meta Attack Language* är ett språk som kan användas för att skapa domänspecifika språk som i sin tur kan användas för att modellera en cyberattack på den specifika domänen med en probabilistisk attackgraf. Denna rapport undersöker hur Meta Attack Language kan utvidgas så att det kan användas för att skapa attackgrafer som innehåller digitala forensiska bevis, genom att undersöka attacker mot Mac-operativsystem där bevis i form av property-list-filer har lämnats. MITRE ATT&CK-matrisen är en kunskapsbas med information om cyberattacker. En studie av denna matris gjordes för att ta reda på vilka olika typer av bevis som har hittats efter attacker på ett Mac-operativsystem samt för att motivera varför denna rapport fokuserar på bevis i form av property-list-filer. En studie av grå litteratur gjordes sedan för att undersöka olika typer av attacker som har lämnat bevis i form av property-list-filer. Studierna visade att det finns en mängd bevis som kan lämnas från ett angrepp på ett Mac-operativsystem och att de flesta bevis i form av property-list-filer användes av attackeraren för att göra attacken tålig mot sådant som omstart av systemet. De visade också att property-list-filerna ofta placerades i rotkatalogen i filsystemet. Studierna visade också att motståndaren ofta försökte dölja filerna genom att ge dem namn som vanligtvis används på ett Mac-operativsystem. Efter studierna genomförts skapades en lista med krav som måste uppfyllas av en utvidgning av Meta Attack. Denna lista baserades på resultaten från studierna och inkluderade krav som säger att det till exempel måste finnas sätt att uttrycka namnet och platsen för en fil, metoder som angriparen använder för att undvika upptäckt, samband mellan olika typer av bevis och samband mellan bevis och attacksteg.

# Contents

# 1 Introduction

In recent years cybercrimes have become more common as well as more sophisticated. Many enterprises and organisations are putting in a lot of resources to protect themselves and their valuable assets from attacks. To be able to do this, the organisations need to understand their system and its weak points. One way to get an overview of the vulnerabilities in a system is to create *attack graphs*. They show which attack steps an adversary could possibly take in a system to reach potential targets. They can be used to analyse security risks and assess the resilience of a system (Johnson, Vernotte, Gorton, Ekstedt & Lagerström, 2017). However, attack graphs also have other potential usages.

Examples of attack steps could be connecting to a malicious server or executing a malicious file. In each attack step, the adversary might leave traces on the system. The traces can be in the form of new or modified files, newly installed programs, and more (Reddy, 2019). Since the adversary leave different traces in different steps of an attack, traces found on a system could potentially be used to find out which step the attacker is currently on. That knowledge could then help in analysing what steps the adversaries might take next and decide which precautions to take, and further damage can thus hopefully be prevented. By connecting evidence to an attack graph, the attack graph can therefore gain a new purpose in incident response.

To do this, collection of data about attacks and potential traces are necessary. A help in this could be MITRE ATT&CK, which is "a globally-accessible knowledge base of adversary tactics and techniques based on real-world observations" (The MITRE Corporation, 2020). MITRE have presented their database as a matrix, where each cell is a different attack step/technique. For each cell, they also present some ways the technique can be detected. Several platforms are represented in the matrix, including operating systems such as Windows, Linux and macOS. Today there are countless threats against all types of operating systems, but according to the "2020 State of Malware Report" made by Malwarebytes in 2019 and 2020, macOS is now the system with the most threats. The report states that "this is likely because, with increasing market share in 2019, Macs became more attractive targets to cybercriminals" (Malwarebytes, 2020, p. 24).

According to the MITRE ATT&CK matrix, modification and addition of so-called property list (plist) files are involved in a number of cyber-attack techniques on macOS. Plist files are serialised objects, that are used to store a user's preferences, information about application and system configurations.

The Meta Attack Language (MAL) is a meta language, that can be used to design languages which in turn can be used to design models of attacks, specific for a domain. A domain can for example be *embedded systems* or *Ubuntu installations* (Johnson, Lagerström, Ekstedt, 2018). Today, MAL doesn't enable a way to include evidence in the graph. What modifications and extensions would be necessary to make this possible? How could for example evidence in the form of plist files be represented?

## 1.1 Problem statement

When a system is under attack there might be traces left behind by the adversary. Finding these traces and understanding what they mean could help in stopping the attack. The purpose of this report is to provide some insight on how forensic evidence in the form of property list files can be added to a probabilistic attack graph. This leads to the main research question:

1. *What requirements need to be fulfilled in an extended formalism of the Meta Attack Language, to enable creation of probabilistic attack graphs which include forensic evidence in the form of property list files?*

To answer this question this report will investigate and list which traces common cyber-attacks leave on a macOS system and then examine the traces that come in the form of plist files. Thus, this report also aims to answer the following supporting research questions:

2. What kinds of traces does common cyber-attacks leave on a Mac operating system according to the MITRE ATT&CK matrix?

3. What kinds of cyber-attacks leave traces in the form of added or modified plist files and what could these modifications include?
4. What could be other indications that the modification of the plist file was due to some malicious activity?
5. What are the essential steps that the adversary had to take to be able to make these modifications?
6. What subsequent steps could an adversary take to continue the attack and is it possible for an adversary to delete their traces in later stages of the attack?

## 1.1 Delimitations

This report focuses on attacks on Mac operating systems where plist files were used. Thus, this report gives a glimpse of how evidence could be used in an attack model, but if it should be possible to include other types of evidence there may be other problems that must be considered. The aim of this study is not to investigate all types of attacks and all types of evidence, but only to find requirements based on a few cases. The aim is also not to actually extend the Meta Attack Language, but to present some requirements that must be fulfilled when doing so. This study also mainly focuses on making graphs that could be used for incident response, and not for forensic analysis. Thus, reflections and requirements that handles specific problems with forensic analysis are purposely left out.

## 1.2 Vocabulary

| | |
|---|---|
| Persistence | "techniques that adversaries use to keep access to systems across restarts, changed credentials, and other interruptions that could cut off their access." (from the MITRE ATT&CK matrix, Persistence). |
| Backdoor | A method of bypassing normal authentication and getting remote access to a computer system (Maleh, Ezzati, & Belaissaoui, 2018). |
| Malware | A software with malicious intent. |

| | |
|---|---|
| Adware | A software that shows advertisement on the computer screen, often inside web browsers. |
| Spyware | A software that collects information about the user of a computer system and sends it to a third party. |
| C2-server | Command and Control Server, which is a server from which an adversary communicates with and controls a victim's computer system (Command and Control, the MITRE ATT&CK matrix). |
| Reconnaissance | Exploring, collecting information about a system. |

# 2 Background

*In the following section the reader is introduced to some of the concepts which are used in the rest of the report. The section is divided into subsections, first describing digital forensics, anti-forensics, incident response and attacks graphs more generally, and then describing the Meta ATT&CK language's formalism in more detail, with an example related to the subjects of this report. After that, the MITRE ATT&CK matrix is described, followed by some information about the Mac operating system. Then property list files are described and explained. Lastly, some related work is presented.*

## 2.1 Digital forensics

When digital devices are becoming more important for organizations and individuals, crimes involving these devices are also becoming more common. Crimes committed by using digital devices and/or targeting digital devices is often referred to as cybercrimes. As with all kinds of crimes a crucial part in the process to find and convict the person or organization who has committed the crime is the collection of physical evidence. When it comes to cybercrime this evidence is mostly the digital devices that was targeted or used by the attacker as well as the digital information inside these digital devices. Digital forensics is the practice of gathering and analysing such evidence (Reddy, 2019).

Just as in real life crime, it is also crucial to keep track of the chain of custody when handling evidence in cybercrime investigations. The chain of custody involves documenting the whole life cycle of a piece of evidence. It is highly important to keep track of and declare where and when the evidence was found, who owns the evidence, which individuals who participated in the forensic investigation, when they handled the evidence and how they handled it. If some of this information is lost the evidence might not be admissible in court (Reddy, 2019).

## 2.2 Incident Response

Van Wyk and Forno (2001) compares incident response operations to firefighting. They write in the first chapter of their book “Incident response” that “both disciplines strive to prevent, detect, contain, extinguish, and investigate bad things that may or do happen”. They describe incident response as a system for handling incidents where something has managed to get around the security controls and an entity’s information is at risk, for example if a virus is spreading in the network of an enterprise. When describing incident response further they divide it into eight parts. Firstly, an incident response program must have an understanding of the security mechanisms that exists on the system. Secondly, it must include planning ahead – before an actual incident occurs. Thirdly, there has to be a way of detecting when an incident occurs. Fourthly, the incident must be analysed when it occurs. Fifthly, the incident response program must try to contain the damage so that it does not spread further. The last three parts suggests what should be done after the incident has been handled. The authors write that an incident response operation should be performed in a way that facilitates a possible forensics investigation. They also write that the damaged systems need to be restored and that the incident should be reviewed. Then if possible, measures should be made to prevent it from happening again.

## 2.3 Anti-forensics

Since it is well known that hacking leave traces, many hackers and malware use different techniques to hide their trail. This is called anti-forensics. There are many common anti-forensic methods that can make digital forensics much more difficult.

One common method is to delete or overwrite incriminating or other sensitive data. Deleting data from a hard drive so that it cannot be recovered can be difficult and therefore it is common to overwrite it several times instead. This makes it more difficult to recover the data. Another way of eliminating the data is to physically harm the hard drive, so that its content no longer can be read or understood (Gabriška D. & Ölvecký M., 2018).

Another common method is to alter as many files as possible to throw the forensics investigators off so that they won't find which altered files actually was used in the attack (Zdzichowski P. et al. 2015).

Another method is to hide data. Data that an attacker does not want to be found can be hidden from investigators with several different methods. It can for example be encrypted or obfuscated so that the adversary can still read the information, but an investigator cannot, at least not easily (Zdzichowski P. et al. 2015).

## 2.4 Attack graphs

There are several ways to make a model of an attack; one is to create an attack graph. Attack graphs are used to show in a clear way which paths a possible adversary could take in a system. Attack graphs can be modelled in many different ways. One example is attack graphs modelled with the *pwnPr3d* approach. (Johnson, P. et al. 2017). Such graphs are directed and starts in the entry point(s) of the adversary. The graph then shows possible attack steps an attacker could take in the system. The nodes represent the attack steps, directed edges represent the path the attacker could take, and edge weight functions define "the probability distribution over time that an attacker will successfully attempt an attack step" (Johnson, P. et al. 2017, pp. 6). Every node, except the entry point nodes, has parent nodes. Either a subset of or all of parent nodes need to be exploited before the child node can be exploited (Johnson, P. et al. 2017).

*foreseeti* is a company that helps other companies manage threat modelling and attack simulations on systems. Their product securiCAD (Ekstedt et. al, 2020) can be used to make models and simulations of current or planned IT environments. This can provide valuable information about the general resilience of the system, can help identify potential risks and the critical paths to sensitive assets (Ekstedt et al., 2015). On their website they have a couple of example models, one of them is a model of a "Ukraine Attack Path" (foreseeti, n.d.). Figure 1 shows a scaled down version of the model, containing only the most essential steps of the attack.

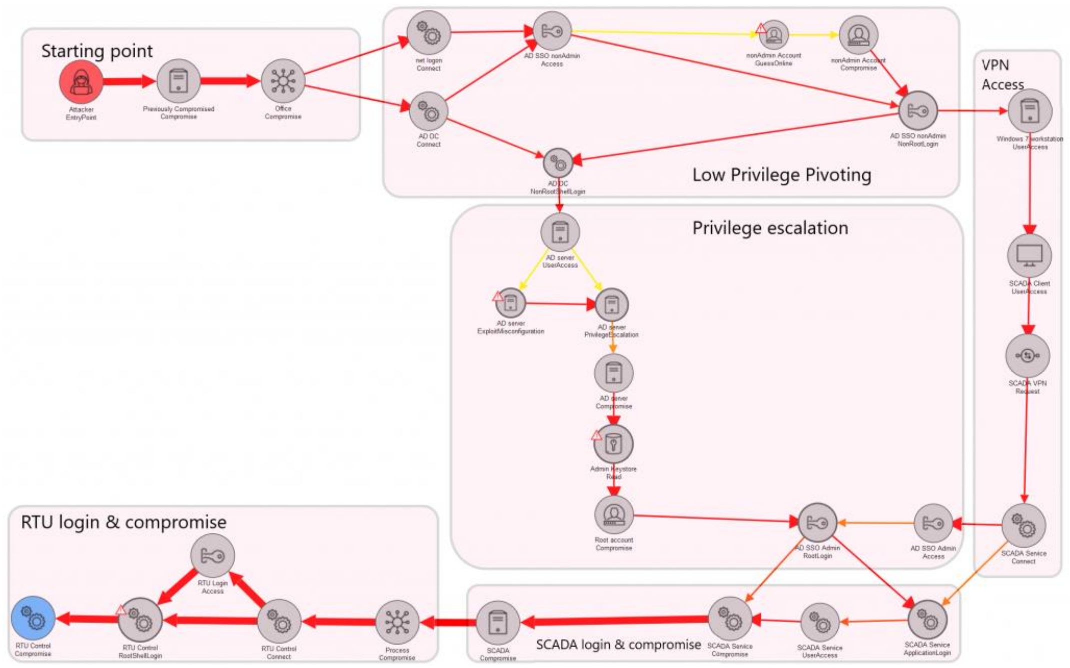


*Figure 1: An example of an attack graph (foreseeti, n.d.)*

Attack graphs are as of today mostly used either proactively, e.g. to analyse the system and discover vulnerabilities, or in retrospect; analysing an attack which has already happened, but they could also in incident response.

## 2.5 MAL

The Meta Attack Language (MAL) is a meta language which can be used to design attack languages for specific domains (Johnson, Lagerström, Ekstedt, 2018). Domains can both be more general such as *embedded systems* or more specialized such as *macOs installations*. A lot of systems within the same domain often have similar properties and have similar needs when building an attack graph. For example, all macOS units have the same file system, APFS (Reddy, 2019) and a lot of them are installed on laptops. Therefore, they may share attack steps that are focused on laptops. The Meta Attack Language can be automatically converted to programming languages such as Java and can be used to efficiently compute very large attack graphs.

The formalism will be introduced through part of the simplified attack example below:

*The adversary has been able to trick a user into downloading and executing a malicious script. The next objective of the adversary is to save the user's credentials. They attempt to do this by creating a pop-up window, which prompts the user to supply their credentials. The probability that the user falls for the trick depends on if the user has gone through a computer security course or not. If the user supplies their credentials, the adversary gains access to the user's account, which is therefore compromised.*

This example can be described through MAL's formalism. The user's credentials can be described as an *object*, which could be called for example *userCredentials*. All objects are divided between different classes (Johnson, Lagerström, Ekstedt, 2018). The *userCredentials* could belong to a class called *Credentials*. Other objects of that class could for example be *adminCredentials* or other user's credentials.

To each class there is also a set of attack steps (Johnson, Lagerström, Ekstedt, 2018). The attack step where the adversary prompts for the user's credentials could be described as *inputPrompt*. This attack step could also be described as belonging to the class Credentials. The attack step can be denoted by its class followed by the name of the attack step, i.e. *Credentials.inputPrompt*. When this step is completed, the next is that the adversary gains access to the user's credentials. This can be denoted as *Credentials.compromise*. The connection between these two attack steps can be expressed through an *edge*. The edges are directed from one attack step to another. It can be denoted by *e* = (*Credentials.inputPrompt, Credentials.compromise)*. However, there may be more edges leading to *Credentials.compromise,* i.e., other ways for the adversary to obtain the credentials of the user. For example by using a keylogger, which tracks what the user types on the keyboard. All the attack steps with edges directed to an attack step are called the *parent attack steps of the attack step*. Depending on which *type* the attack step has, either *all* or only one of the parent attack steps needs to have been performed. If it's the first one, the attack step has the type *AND*, if it's the second one, it has the type *OR* (Johnson, Lagerström, Ekstedt, 2018). The example above has the type *OR*, since both parents are independent of each other.

Each attack step is assigned a probability distribution, which specifies the time it would take for an attacker to perform it. This can be referred to as *local time to compromise*.

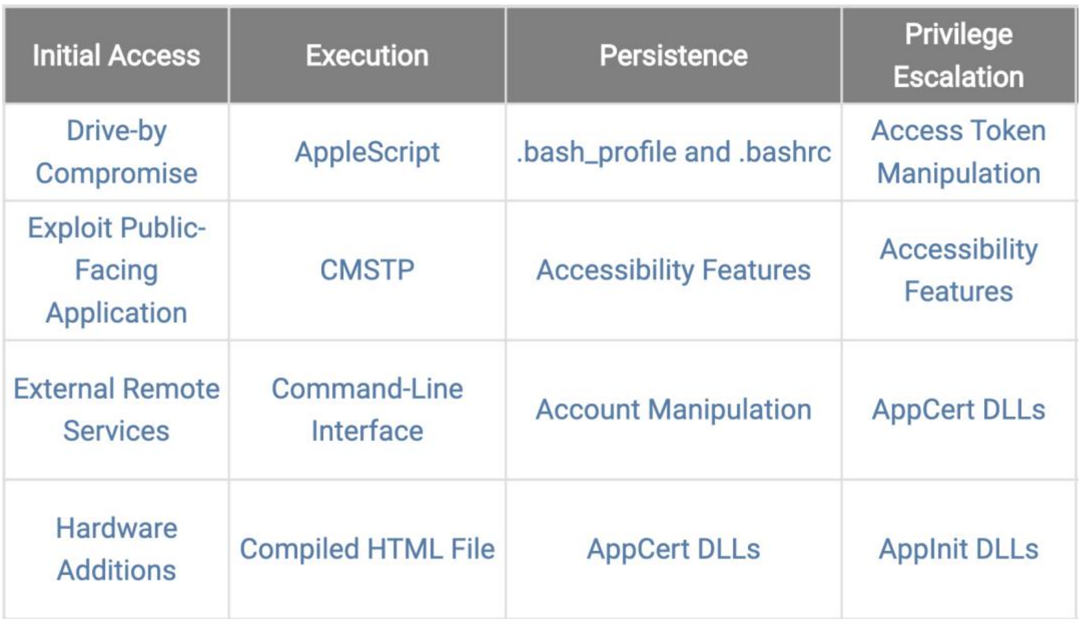
Classes can also have *defences*, which can either be activated or not activated. If they are, their state is defined as *TRUE*, otherwise *FALSE*. A defence can be the parent of an attack step (Johnson, Lagerström, Ekstedt, 2018). In the example, a defence is that the user has undergone a computer security course, which may prevent them from being tricked by the adversary into supplying their credentials. The defence can be denoted as *User.securityAware*, which means it belongs to the *User* class. If the defence is activated, it's child attack step/steps may be unable to perform. It may also affect the time to compromise distributions.

Classes can have *links* to other classes. For example, if there's an *Accounts* class, which consists of different user's accounts, there could be a *link* from the *Credentials* class to the *Accounts* class, which may have the label *authentication*, since the credentials are used to authenticate the account. There is also a link the other way, from *Accounts* to *Credentials*. These links are collected into associations. An association includes a link in both directions (Johnson, Lagerström, Ekstedt, 2018).

A mandatory class is the *Attacker* class, which consists of only one attack step: the initial step of the attack (Johnson, Lagerström, Ekstedt, 2018).

## 2.6 MITRE ATT&CK matrix

MITRE ATT&CK is "a globally-accessible knowledge base of adversary tactics and techniques based on real-world observations" (The MITRE Corporation, 2020). The information about the tactics and techniques is represented in a matrix where the columns represent different attack steps. An attack technique is placed in a column if it is known to have been used in the attack step which that column represents. The same attack technique can exist in multiple columns. For example, *Accessibility Features* exists in both the *Persistence* and the *Privilege Escalation* column, as can be seen in Figure 2. Every attack technique has their own page where more information about the technique can be found. There is information about how the technique works, examples of malware and hacker groups that have been known to use the technique and suggestions on how to detect if a system has been exposed to the technique. The part with suggestions on detection often includes information about traces that could be left behind by an adversary using the technique.



| Initial Access | Execution | Persistence | Privilege Escalation |
|---|---|---|---|
| Drive-by Compromise | AppleScript | .bash_profile and .bashrc | Access Token Manipulation |
| Exploit Public-Facing Application | CMSTP | Accessibility Features | Accessibility Features |
| External Remote Services | Command-Line Interface | Account Manipulation | AppCert DLLs |
| Hardware Additions | Compiled HTML File | AppCert DLLs | AppInit DLLs |

*Figure 2: A subset of the MITRE ATT&CK matrix (screenshot from the MITRE webpage)*

## 2.7 macOS, OS X & Mac OS X

macOS is the current name of the operating system of Apple computers. The Mac operating system was previously called OS X (from version 10.7 to 10.11.6), and before that Mac OS X (up to version 10.6.8) (Apple, 2020). The biggest difference except the name between macOS and previous versions, is the file system.

**APFS**
In the new operating system macOS, the old file system HFS+(Hierarchical File System plus) was replaced by the Apple File System (APFS) (Reddy, 2019). Some new things the APFS provides is improved file system fundamentals and new features, such as support for sparse files, cloning of files and directories and also supports full disk encryption. APFS is optimized for SSD's. According to Apple's documentation it's possible, without any code-changes, for high-level APIs such as FileManager to take advantage of the new behaviour (Apple, n.d.c).

**Privileges**
There are different levels of privilege of a user on the file system. There are basic user and admin users. The files belonging to the currently logged in user can only be modified by that user, if the user hasn't specified otherwise. Admin users can elevate themselves to root, which lets them perform actions that a normal user isn't allowed to do. This can be tasks such as manipulating file permissions and ownership, creating, reading, updating or deleting system and user files as well as changing system settings. If a process launches another process, the new process will have the same privileges as the process that launched it. This means that if there are a security leak in a program, and it's executed with root privileges, an adversary may be able to obtain these privileges as well (Apple, 2016b).

There is also an additional level of protection, called the *System Integrity Protection* (Apple, n.d.b). The purpose of this level is to prevent malicious actors to modify protected files and folders on Mac units. The following parts of the system are protected by System Integrity Protection: /System, /usr, /bin, /sbin, /var and apps that are pre-installed with macOS. Only processes signed by Apple and have special entitlements to write to system files are allowed to modify these parts. This could for example be Apple software updates and Apple installers (Apple, n.d.b).

## 2.8 Property list files – plist-files

Property lists are representations of object graphs that can be stored in the file system and reassembled later, i.e. serialized objects. They enable applications to store small amounts of data compactly and lightweight. Both the Apple-native APIs for its desktop operating system macOS, Cocoa Foundation (Wikipedia, 2020) and Core Foundation (Apple, n.d.c), have APIs related to property list serialization and deserialization (Apple, 2018).

Only certain types of data can be stored in a property list, the basic types being strings, numbers (integer and float), dates, binary data and Boolean values. There are also collection types, dictionaries and arrays, that can contain one or multiple data types, including dictionaries and arrays. Dictionaries are used to store key-value pairs.

Property lists can be formatted in both XML- and binary-format. The XML is more readable and can be edited manually if needed, but the binary format is much more compact, and is recommended by Apple in most scenarios (Apple, 2018). Tools such as *plutil* can be used to convert between the two types. There are also several applications and command-line tools that can be used to read, create and modify plist-files. One example of an application is the developer toolkit *XCode* (Marczak, Neagle, 2010).
All property list files have the filename extension plist, hence they are often called *plist files*.

**Usage of property list files**
As mentioned plist-files can be used by applications to store small amounts of data. One type of data that is often stored by apps are user preferences (Marczak and Neagle, 2010), and are often stored in plist-files. Another use for plist-files are *LaunchAgents* and *LaunchDaemons*, which are covered in the section about

*Standard* ASEPs (Auto-Start Extensibility Points).

**Naming of property list files**
The naming of the files is done according to something called the reverse domain name notation naming (reverse-DNS). Examples of this are that the registered domain name apple.com, with the specific program name "dock", becomes "com.apple.dock"(.plist). This is however not enforced, only a convention, so in reality there may be many plist-files that do not follow this (Marczak, Neagle, 2010).

**Formatting of property list files**
Figure 3 shows an example of how a common plist-file can be formatted, in this case the file *com.microsoft.update.agent.plist,* located in the "/Library/LaunchAgents" directory. The contents were accessed from an individual's Mac with the OS version 10.15.1 in 2020, using the command: *plutil -convert XML1 <file-name>*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
     <key>Disabled</key>
     <false/>
     <key>Label</key>
     <string>com.microsoft.update.agent</string>
     <key>MachServices</key>
     <dict>
          <key>com.microsoft.update.agent</key>
          <true/>
     </dict>
     <key>ProgramArguments</key>
     <array>
          <string>/Library/Application
Support/Microsoft/MAU2.0/Microsoft
AutoUpdate.app/Contents/MacOS/Microsoft         Update
Assistant.app/Contents/MacOS/Microsoft          Update
Assistant</string>
          <string>-checkForUpdates</string>
     </array>
     <key>RunAtLoad</key>
     <true/>
     <key>StartInterval</key>
     <integer>43200</integer>
</dict>
</plist>
```

*Figure 3: The contents of the file /Library/LaunchAgents/com.microsoft.update.agent.plist on an individual's Mac (2020).*

The file includes a header, which is very similar on all plist files, which plist version is used and a dictionary. The key-value pairs in the dictionary are what describes the plist file's function.

The purpose of the file is to check for updates for Microsoft products, through the application specified in the first string of the *ProgramArguments* field. It does this with a certain interval, in this case every 43200th second, which is specified in the *StartInterval* field.

The two different options for the header are shown in Figure 4.

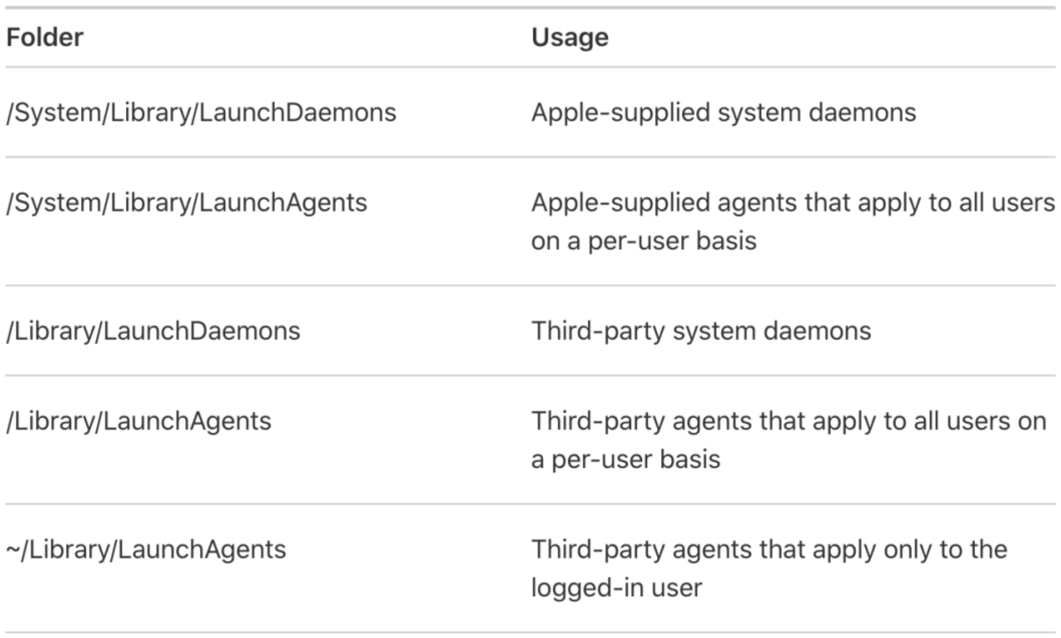
```
<!DOCTYPE plist PUBLIC    "-//Apple//DTD    PLIST
1.0//EN"  "http://www.apple.com/DTDs/PropertyList-
1.0.dtd">

<!DOCTYPE plist PUBLIC   "-//Apple   Computer//DTD
PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-
1.0.dtd">
```

*Figure 4: The two different options for the header in plist files.*

**Standard ASEPs (Auto-Start Extensibility Points)**

An ASEP (Auto-Start Extensibility Point) is on macOS a file that can lead to automatic execution of a process. They are mostly stored in plist-files, specifically *LaunchAgents* and *LaunchDaemons* (Shakacon LLC, 2018). LaunchAgents and LaunchDaemons are daemons, i.e. background processes. The difference between them is that LaunchDaemons refers to system-level daemons, that can run on behalf of the root or any user specified, while LaunchAgents are run on the behalf of currently logged in user (Apple, 2016a). They are stored in the locations specified in Figure 5.



| Folder | Usage |
|---|---|
| /System/Library/LaunchDaemons | Apple-supplied system daemons |
| /System/Library/LaunchAgents | Apple-supplied agents that apply to all users on a per-user basis |
| /Library/LaunchDaemons | Third-party system daemons |
| /Library/LaunchAgents | Third-party agents that apply to all users on a per-user basis |
| ~/Library/LaunchAgents | Third-party agents that apply only to the logged-in user |

*Figure 5: Describing the locations for LaunchDaemons and LaunchAgents and their usages (Apple, n.d.d).*

They are launched using *launchd*. launchd is a service that is used by macOS to manage daemons and agents (Apple, n.d.d). It runs at boot to complete system initialization. To do this, it first goes through all the directories where LaunchDaemons are stored (see Figure 5). Then it follows the instructions set up by those, by for example registering sockets and file descriptors as well launching daemons that should always run. launchd also runs when a user logs in, and performs the same procedure, but on a user level (Apple, 2016a).

In Table 1 some keys commonly used in LaunchAgent and LaunchDaemon dictionaries are described.

*Table 1: Some commonly used keys in LaunchAgent and LaunchDaemon dictionaries (Apple, 2016a).*

| **Key** | **Description** |
|---|---|
| Label | A string that uniquely identifies the daemon to *launchd* (required) |
| ProgramArguments | The arguments used to launch the daemon (required to have either this or the Program key). |
| KeepAlive | Specifies if the daemon should be launched only on demand or if it always should be running. |
| Program | Specifies which file the daemon should execute. If missing, launchd pretends it has the same value as the first argument in ProgramArguments. |
| RunAtLoad | Specifies if the job should be launched once the daemon is loaded or not. |

launchd can also be launched manually by using the launchctl command. This command can be used to load or unload launchd daemons and agents at other times than at boot (support apple).

Jaron Bradley (Shakacon LLC, 2018) extends the information in Figure 5 a bit further in his talk. LaunchDaemons and LaunchAgents occur on three different levels of privilege. The most protected locations are on the *System Integrity Protection level*. These are the "/System/Library/LaunchDaemons" directory respectively the "/System/Library/LaunchAgents" directory. Only Apple should be able to write to these locations, even a root user cannot modify or add files in these locations. This is also confirmed by Apple (n.d.b). This only applies if System Integrity Protection is enabled, which is the default.

The directories /Library/LaunchDaemons and /Library/LaunchAgents are on the root-user-level. According to Bradely (Shakacon LLC, 2018), this is probably the most commonly used location for macOS malware. The last level is the User Level, which is located in the user's home folder, at "~/Library/LaunchAgents". On this level there are only LaunchAgents, not LaunchDaemons, since all daemons on this level only can run on behalf of the currently logged in user. Only ordinary user permissions are necessary to modify files in this directory.

Bradley (Shakacon LLC, 2018) also mentions schedulers such as *cron*, which is mostly used for old Linux malware and *periodic*, which he hasn't seen somebody use yet. Both of these can be used to schedule periodic jobs at fixed times.

Login items is somewhat deprecated but is something an application can use to persist on a macOS system. Login items are stored in a plist file called *com.apple.loginitems.plist* which is located in the user's Library/Preferences directory. These login items can also be viewed, added and removed in the GUI (Wardle, 2014).

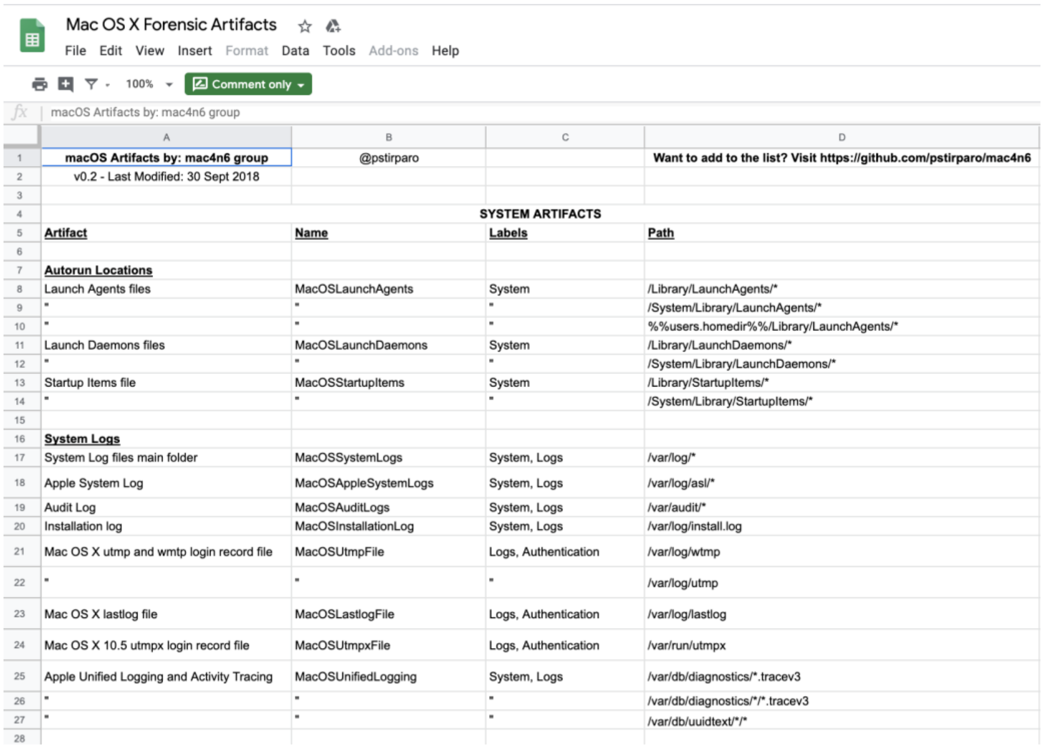
Startup items is just as login items a somewhat deprecated method for applications to persist on an macOS system. Startup items allow a script or command to be executed when the system starts. To create a startup item two files has to be added in a sub-directory to either the /System/Library/ StartupItems directory or the /Library/StartupItems directory. The two files should be one script file containing the script that should be executed on startup and one plist file called *StartupParameters.plist* containing a 'Provides' key with the name of the script file (Wardle, 2014).

## 2.9 Related work

*There has been some previous research of how to combine evidence and attack graphs, as well as what kind of artefacts that are common on Mac systems. In this section, two of these sources are presented.*

In their report "Mapping Evidence Graphs to Attack Graphs" (2012), Changewi Lui, Duminda Wijesekera and Anoop Singhal presents an algorithm that maps so called *evidence graphs* to formal attack graphs. These evidence graphs model intrusion evidence, and the dependencies between them. The authors write that “an evidence graph correlates attack evidence by using network configuration, time stamps and expert systems with fuzzy rules to reconstruct potential attack scenarios from large amount of noisy data” (pp. 121). In the end of the report the authors conclude that this mapping between the two types of graphs makes it possible to update attack graphs based on evidence and also that it can show if an adversary has used anti-forensics methods. They also state that this would help forensic investigators in their work.

Sarah Edwards and the *mac4n6* group have created a Google spreadsheet of macOS artefacts. The goal is to create one place for all artefacts and information about them in one place. Figure 6 displays a screenshot of a small part of the spreadsheet (Mac4n6 Group, 2018). Multiple plist files are mentioned, including LaunchAgents and LaunchDaemons. Other examples of plist files that are mentioned are System Preferences, Keyboard layout and application preferences.


Mac OS X Forensic Artifacts
File Edit View Insert Format Data Tools Add-ons Help
100%
Comment only
macOS Artifacts by: mac4n6 group

| | A | B | C | D |
|---|---|---|---|---|
| 1 | **macOS Artifacts by: mac4n6 group** | @pstirparo | | **Want to add to the list? Visit https://github.com/pstirparo/mac4n6** |
| 2 | v0.2 - Last Modified: 30 Sept 2018 | | | |
| 3 | | | | |
| 4 | | | **SYSTEM ARTIFACTS** | |
| 5 | **Artifact** | **Name** | **Labels** | **Path** |
| 6 | | | | |
| 7 | **Autorun Locations** | | | |
| 8 | Launch Agents files | MacOSLaunchAgents | System | /Library/LaunchAgents/* |
| 9 | " | " | " | /System/Library/LaunchAgents/* |
| 10 | " | " | " | %%users.homedir%%/Library/LaunchAgents/* |
| 11 | Launch Daemons files | MacOSLaunchDaemons | System | /Library/LaunchDaemons/* |
| 12 | " | " | " | /System/Library/LaunchDaemons/* |
| 13 | Startup Items file | MacOSStartupItems | System | /Library/StartupItems/* |
| 14 | " | " | " | /System/Library/StartupItems/* |
| 15 | | | | |
| 16 | **System Logs** | | | |
| 17 | System Log files main folder | MacOSSystemLogs | System, Logs | /var/log/* |
| 18 | Apple System Log | MacOSAppleSystemLogs | System, Logs | /var/log/asl/* |
| 19 | Audit Log | MacOSAuditLogs | System, Logs | /var/audit/* |
| 20 | Installation log | MacOSInstallationLog | System, Logs | /var/log/install.log |
| 21 | Mac OS X utmp and wmtp login record file | MacOSUtmpFile | Logs, Authentication | /var/log/wtmp |
| 22 | " | " | " | /var/log/utmp |
| 23 | Mac OS X lastlog file | MacOSLastlogFile | Logs, Authentication | /var/log/lastlog |
| 24 | Mac OS X 10.5 utmpx login record file | MacOSUtmpxFile | Logs, Authentication | /var/run/utmpx |
| 25 | Apple Unified Logging and Activity Tracing | MacOSUnifiedLogging | System, Logs | /var/db/diagnostics/*.tracev3 |
| 26 | " | " | " | /var/db/diagnostics/*/*.tracev3 |
| 27 | " | " | " | /var/db/uuidtext/*/* |
| 28 | | | | |

*Figure 6: A screenshot from the Mac4n6’s groups artefact spreadsheet.*

# 3 Method

*This section describes the method used for the studies in this report. The section is divided into subsections, where the first one describes the method used to answer the second research question: "What kinds of traces does common cyber-attacks leave on a macOS system according to the MITRE ATT&CK matrix?" and give a motivation to why plist files is a suitable evidence to focus on. The second subsection describes the method used for the main study, meant to answer research question 3, 4, 5 and 6 as well as to provide support for answering the main research question. The third and last subsection shortly describes the method used to analyse the results from previous studies in an attempt to answer the main research question: "What requirements need to be fulfilled in an extended formalism of the Meta Attack Language, to enable creation of probabilistic attack graphs which include forensic evidence in the form of property list files?"*

## 3.1 A compilation of the "Detection" sections in the MITRE ATT&CK matrix

To answer the second research question and to validate that plist files are interesting as evidence from cyber-attacks a study of the MITRE ATT&CK matrix was made. The MITRE ATT&CK matrix contains a great amount of information about different attack steps and more importantly, how to detect such attack steps. Therefore, the MITRE ATT&CK matrix seemed like a good tool that could be used to answer this question.

The ATT&CK Navigator tool, which is a tool for filtering information in the matrix, was used to show the attack techniques that according to MITRE have been used on MacOS. All "prepare" techniques were also filtered out, since the study focused on the actual attack. The sections called "Detection" from all the remaining techniques in the matrix was then studied and interpreted to find what kinds of traces the attack steps could leave behind. The found traces were then divided into categories based on the type of trace. The categories were chosen based on the groups of similar traces that were found during the study.

## 3.2 A grey literature review on the use of plist files in cyber attacks

To provide the basis for answering the main research question and answer the research questions 3, 4, 5 and 6, a literature review was made, with a focus on grey literature. Grey literature is according to the *Luxembourg definition*: “That which is produced on all levels of government, academics, business and industry in print and electronic formats, but which is not controlled by commercial publishers"(Schöpfel & Farace, 2010). Grey literature can be documents such as blog posts, government reports and clinical trial results, hence it has not gone through the traditional commercial or academic publishing channels. The motivation for focusing on grey literature was that incident response and digital forensics is mainly a practical topic that is handled by enterprises and organizations in real scenarios. Mac Forensics and incident response was also a relatively new field, and the research within the area had been limited.

There were, however, several more casual reports and blog posts about the subject. Since this kind of source material may not be peer-reviewed, source selection and criticism were crucial. The study mainly focused on incident reports and descriptions of the course of action that has been used in known attacks.

The literature was collected by searching Google, since the kind of Grey literature this report focus on normally is not collected in any large databases. A common web search engine with a broad range therefore seemed to be the best choice.

The browser used when searching was Google Chrome. To increase the generality of the search, some measures were taken in all searches. Firstly, to use incognito mode while browsing, to prevent the browser from storing browsing data. Secondly, to not use private or personalized data when querying, by turning of that feature in the search settings. Lastly, to not have the “safe search” filters activated.

Incident response is as stated earlier a common term for the job of trying to stop an ongoing attack, which includes collecting evidence. Intrusion analysis is sometimes mentioned as a crucial part of incident response and forensic analysis and has to do with detecting and analysing signs of an intrusion in the network (Fichera & Bolt, 2013). Since plist files only exist on Apple’s operating systems there was no need to include macOS or OS X among the search terms. Therefore, the chosen search terms were:

- 1: “plist” “intrusion” analysis
- 2: “incident response” “plist”

The sources had to fulfil the following criteria to be selected for the study:

- It was written by a known author (who preferably is known to be skilled within the area) or company/organisation who has business within the field.
- It contained information relevant to the study, such as real or possibly real cases.
- It dealt with some kind of attack and some kind of evidence on macOS.
- It was written in Swedish or English.
- It included enough information to add something to the study.
- It was from a safe URL
- It was not a book or study handling macOS forensics in general.

If a selected source contained a link to another source with further information about the described subject, that link was followed, and that new source was then also used in the study.

To understand more easily which information was interesting for the study, some questions were formulated. Then information was collected from each source, with the goal to answer the following questions:

- What is the purpose of the described attack?
- Questions regarding the plist file(s):
    - What is the specific purpose of the used plist file(s)?
    - What was the plist file(s) called?
    - Where was the plist file located?
    - Was the plist file(s) added by the adversary or did the adversary modify a plist file already existing on the system?
    - What was the content of the plist file(s)?
    - How long does the adversary need the plist file(s)?
    - Can the plist file(s) be removed by the adversary later?
- What other evidence was there of the attack?
- What steps did the adversary take and in what order?

Then, a short description, based on the answers to the above-mentioned questions, was made for each case. A graph, representing the timeline of each attack and the found evidence from each step, was also produced to get a clearer overview of the attacks. Details such as the locations and contents of the plist-file(s) was compiled into tables.

## 3.3 Extending MAL

After all data was collected, compiled and analysed the results were used to form a list of requirements. This was designed to answer the main research question: *What requirements need to be fulfilled in an extended formalism of the Meta Attack Language, to enable creation of probabilistic attack graphs which include forensic evidence in the form of property list files?* All such evidence found in the grey literature study should be possible to include in the models. Therefore, the list was based on the found evidence in the form of plist files and their relations to attack steps and other evidence.

# 4 Results

*This section presents and explains the results from the studies. In the first subsection the result from the study of the MITRE ATT&CK matrix, which mainly focused on the second research question: "What kinds of traces does common cyber-attacks leave on a macOS system according to the MITRE ATT&CK matrix?", is presented and explained. In the second subsection the result from the grey literature study is presented and explained. In this part the result is divided into cases which are based on different attacks or malware that was described in the sources found in the study. In the third and last subsection a list of demands for extending the Meta Attack Language is presented. This list was made based on the results in the first two subsections in an attempt to answer the main research question:*
*What requirements need to be fulfilled in an extended formalism of the Meta Attack Language, to enable creation of probabilistic attack graphs which include forensic evidence in the form of property list files?*

## 4.1 Evidence that indicates a malicious attack on a Mac operating system

The filtration in the MITRE ATT&CK Navigator resulted in 159 different techniques. Analysing the traces described in the detection section in all of these techniques resulted in 18 different categories of traces that could be found in a macOS system during or after an attack. These categories were:

1. Unusual network traffic or network connections
2. Changes to network settings
3. Unusual authentication attempts
4. Log files showing suspicious behaviour
5. Unusual API calls
6. Unusual command-line arguments or system calls
7. Unrecognized files or tools

8. Modified files and folders
9. File-deletion
10. Suspicious driver or certificate installation
11. Suspiciously accessed files
12. Read/write attempts to sensitive locations
13. Abnormal processes or executions
14. Strange input prompts
15. Unknown scripts
16. Suspicious computer resource behaviour
17. Malicious files or links in emails
18. Suspicious domains

The full list of the traces that were found mentioned in the MITRE ATT&CK matrix can be seen in Appendix. All traces were put in one of the categories even if there were more than one category that suited the trace. Some of the mentioned traces were not specifically described, such as "Command-line arguments for actions that could be done before or after code injection has occurred" and some traces included other traces that also were mentioned; for example, number 59: "Suspicious files written to disk" includes number 77: "New plist file on disk". All the traces can somehow be detected, but with varying difficulty. Some require special tools and some require extensive monitoring and logging. Some, such as "Unusual access by abnormal users or at abnormal times" also require knowledge of how the particular system is usually used. In some cases, the trace is a method used by the adversary, such as "Port scanning activity", but since there are tools that can notice it, it can be seen as a trace. The traces can be seen as indications of malicious activity, but to be certain it may be necessary to connect the traces to other suspicious activity or to compare it to usual behaviour on the system.

Since this study mainly focuses on plist files it is interesting to look at the traces that is in the form of plist files or connected to plist files. There were ten such traces mentioned, but a few of them are similar, so they could possibly be seen as the same trace. The traces were:

1. New plist file on disk
2. Plist files with the apple.awt.UIElement tag
3. Modified plist files. Especially by a user.
4. Unknown process execution resulting from login actions
5. Process execution resulting from modified plist files

6. Programs installed via launchctl as launch agents or launch daemons.
7. Unusual process execution from launchctl/launchd or cron tasks
8. Programs that are executed from /Library/StartupItems that don't match the whitelist
9. Suspicious changes in Contents/Library/LoginItems
10. Unknown changes in Apple menu -> System Preferences -> Users & Groups -> Login items and the corresponding file locations

As stated, these are just indications of malicious activity and does not always mean there is malicious activity on the system. For example, legitimate software also adds or modifies plist files, quite often, but according to the MITRE ATT&CK matrix it is of interest to keep track of plist files in the file system in case of some suspicious activity. That there are ten different traces described in the matrix related to plist files shows that plist files can be an interesting evidence of a malicious attack on a Mac operating system.

## 4.2 Cases from the grey literature study

*In the following subsection the result from the grey literature study is presented and explained. The result is divided into cases which are based on different attacks or malware that was described in the sources found in the study. First a graph describing the timeline of the attack through its attack steps, the evidence and the connections between those is presented. Then the timeline is described in more detail in text. Every description of the attacks is either paraphrasing or summarizing parts from the sources. The potential evidences in the form of plist files are presented in a table and explained in text. Then other kinds of possible evidence in the case is briefly described. The choice of these possible pieces of evidence is based on the studied sources and the findings in the study of the MITRE ATT&CK matrix. If the kind of trace is specifically mentioned in the list in Appendix, from the study of the MITRE ATT&CK matrix, there is also a reference to that specific trace.*

The grey literature study focused on attacks on macOS system where plist files were potential evidence. Some of the attacks targeted several accounts on a system. All of the attacks targeted at least one account that had admin privileges. Most of the plist files were in the form of LaunchDaemons and LaunchAgents and were mainly used for establishing persistence for the adversary. Many of the attacks involved establishing a backdoor and through that gain control of the machine.

13 cases were found using the search terms described below. Some of the cases were found in both searches. The cases were numbered arbitrarily by the numbers from 1 to 13. The search term: “intrusion” “plist” analysis was used to find the following cases: 1, 2, 3, 4, 5, 6, 11, 12 and 13. The search term: “incident response” “plist” was used to find cases: 5, 6, 7, 8, 9, and 10.

The graphs show the timeline of the attacks and the evidence connected to the different attack steps. The ovals represent attack steps and the hexagons represent evidence. An oval with a hexagon inside represents an attack step that could be seen as an evidence. A rectangle is either adversary’s entry point or their goal. The arrows with solid lines and black heads represent a step between two connected attack steps. The arrows with dashed lines and white heads represent a relationship between an attack step and a possible evidence. The arrows with dotted lines and white heads connect two nodes that are the same evidence, but at different times, for example if a piece of evidence is moved to a new location. Lastly, the arrows with solid lines and white heads represents a direct connection between two pieces of evidence.

**Case 1**
The first case was a backdoor for macOS created by a hacker group called Lazarus, according to the sources. The backdoor was described on the Objective-See blog by Patrick Wardle in 2019 and mentioned in an article by Dennis Fisher in 2019 on duo.com. In Figure 7, the timeline of an example of an attack utilizing the backdoor is described through a graph.

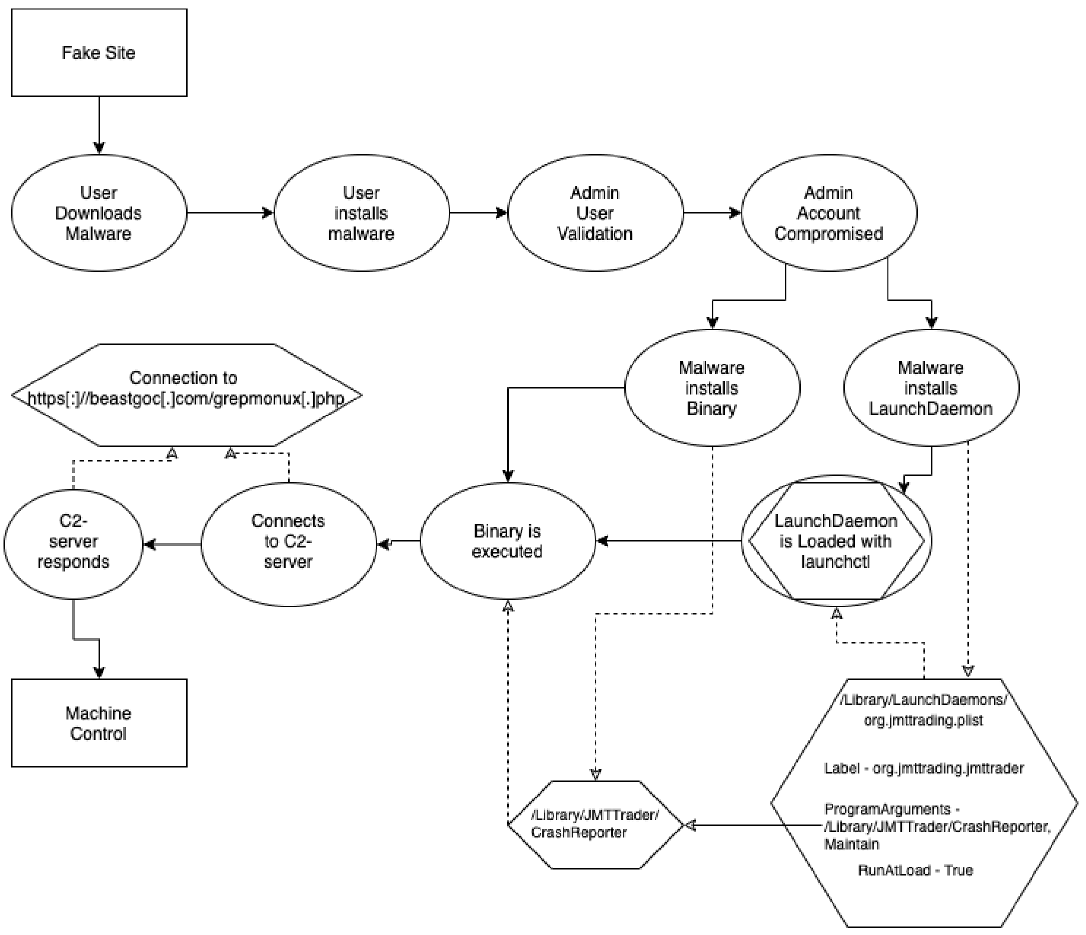

Fake Site
User Downloads Malware
User installs malware
Admin User Validation
Admin Account Compromised
Malware installs Binary
Malware installs LaunchDaemon
Connection to https[:]//beastgoc[.]com/grepmonux[.]php
C2-server responds
Connects to C2-server
Binary is executed
LaunchDaemon is Loaded with launchctl
Machine Control
/Library/LaunchDaemons/org.jmttrading.plist
Label - org.jmttrading.jmttrader
ProgramArguments - /Library/JMTTrader/CrashReporter, Maintain
RunAtLoad - True
/Library/JMTTrader/CrashReporter

*Figure 7: The timeline of an example of an attack utilizing the backdoor in case 1 described through its attack steps, evidence and how they are connected.*

The first known step of the adversary was to create a website for fake trading software. This was used to lure victims to install a backdoor on their computers. If the victim is tricked and downloads and installs the malware, then the adversary asks the user for admin privileges. If granted, the user's admin account is compromised. Then the adversary is free to carry on installing more malware now with root privileges. They install a binary and a LaunchDaemon. By loading the LaunchDaemon with the launchctl command, the binary is executed. The binary then tries to connect to a C2-server. If it's successful, and the C2-server responds, the machine can be commanded by the adversary.

The evidence that was left in the form of plist files was in this case a LaunchDaemon. Its properties can be observed in Table 2. It's placed in the root level Library, which means that it's a root level LaunchDaemon. The purpose of the plist file was to establish persistence for the backdoor, i.e. executing the file that connects to the C2-server.

*Table 2: Shows the properties of the plist file used in case 1.*

| **Location** | **Name** | **Key** | **Value** |
|---|---|---|---|
| /Library/LaunchDaemons/ | org.jmttrading.plist | Label | org.jmttrading.jmttrader |
| | | ProgramArguments | /Library/JMTTrader/CrashReporter Maintain |
| | | RunAtLoad | True |

Other possible evidence was:

- the *CrashReporter* binary, placed in /Library/JMTTrader.
- that the LaunchDaemon was loaded at another time than on startup, using launchctl. It also executed another process. This is mentioned as a possible trace in Appendix, number 127: "Unusual process execution from launchctl/launchd or cron tasks".
- an open connection to the for the user unknown server https[:]//beastgoc[.]com/grepmonux[.]php. This could fit in some mentioned traces in Appendix, for example number 4: "Processes that have not been seen before utilizing the network".

**Case 2**

The second case was a backdoor for MacOS, with the objective to collect as much data as possible according to the source. The timeline of the attack is shown in Figure 8. The attack was documented by Jaron Bradley and Karl Scheuerman at the Crowdstrike blog in 2018.

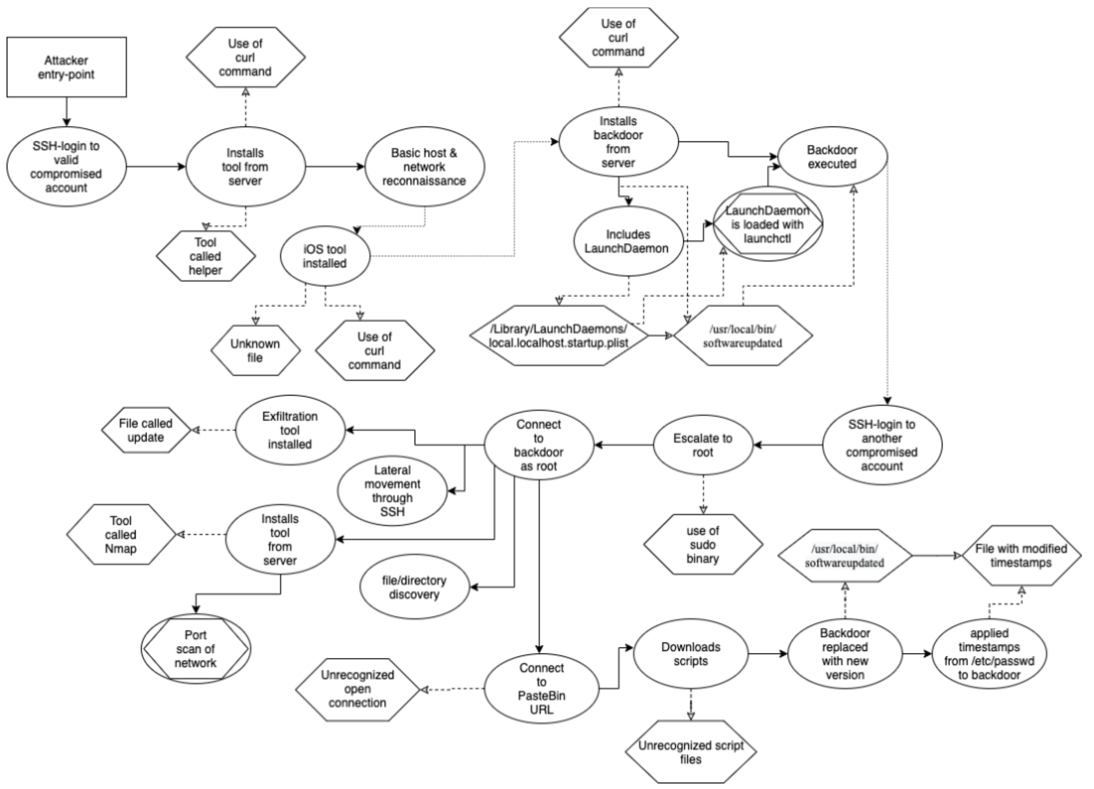

Attacker entry-point
SSH-login to valid compromised account
Installs tool from server
Use of curl command
Tool called helper
Basic host & network reconnaissance
iOS tool installed
Unknown file
Use of curl command
Installs backdoor from server
Use of curl command
Includes LaunchDaemon
LaunchDaemon is loaded with launchctl
Backdoor executed
/Library/LaunchDaemons/ local.localhost.startup.plist
/usr/local/bin/ softwareupdated
SSH-login to another compromised account
Escalate to root
use of sudo binary
Connect to backdoor as root
Exfiltration tool installed
File called update
Lateral movement through SSH
Installs tool from server
Tool called Nmap
Port scan of network
file/directory discovery
Connect to PasteBin URL
Unrecognized open connection
Downloads scripts
Unrecognized script files
Backdoor replaced with new version
/usr/local/bin/ softwareupdated
applied timestamps from /etc/passwd to backdoor
File with modified timestamps

*Figure 8: The timeline of the attack in case 2 described through its attack steps, evidence and how they are connected.*

The initial entry point of the attack is not known, but the subsequent steps indicate a pre-existing intrusion leading to compromised credentials.

The first known step of the adversary was to login to the target network via SSH, using the credentials of a valid account. After that, the curl command was used to install a tool from a server. Some basic host and network reconnaissance were executed, and then it seems like the adversary mistakenly installed a tool made for iOS and not macOS devices.

The next step for the adversary was to install another custom backdoor, which was masked by naming the backdoor after a standard Apple service. Included in the installation was a LaunchDaemon, which was loaded after the installations. This led to the execution of a new instance of the backdoor.

The adversary also accessed a second victim by using compromised credentials. The same backdoor was installed there. They escalated to root and connected to the backdoor running as root. They then continued to execute different commands.

The adversary performed considerable file and directory discovery. There didn't seem to be a specific target in the search, the goal seemed to be finding and gathering as much data as possible. The attacker also attempted to move laterally through SSH to other hosts.

The tool *Nmap* was downloaded from the previously accessed server, and through the use of that, extensive port scan of the network was executed. The file synchronisation tool rsync was employed to aid file transfers from the victim. A portable network penetration tool was retrieved from a new server, and then executed. The attacker was then able to tunnel through various hosts on the victim network and also back to their intended server.

The adversary later used their backdoor to return to the network, and then downloaded additional scripts from a PasteBin URL. The backdoor was replaced by a new version.

They tried to hide their traces by taking the timestamps from a legit file and applying them to their backdoor file.

The evidence that was left in the form of plist files was in this case a LaunchDaemon. The source didn't specify the dictionary fields, but its other properties can be observed in Table 3. The LaunchDaemon was placed in the root level Library, which mean it's a root level LaunchDaemon. The purpose of the plist file was to execute a new instance of the backdoor. This gives the backdoor persistence through reboots.

*Table 3: Shows the properties of the plist file used in case 2.*

| **Location** | **Name** | **Key** | **Value** |
|---|---|---|---|
| /Library/LaunchDaemons/ | local.localhost.startup.plist | - | - |

Other possible evidence was:

- a massive use of curl commands accessing and downloading programs from a server at the IP address 45.77.129[.]251.
- a tool called *helper*, which was executed after it was downloaded.
- a tool made for iOS devices, which was never used.

- the backdoor file called *softwareupdated*, which is named after a standard macOS service. This file was however placed in "/usr/local/bin" directory instead of the ordinary directory: "/System/Library/CoreServices/SoftwareUpdate.app/Contents/Resources/". This type of trace is also mentioned in Appendix, number 65: "Files with known names but in unusual locations".
- a tool called Nmap that was downloaded from the previously mentioned server.
- a connection to a server at the IP address: 43.245.48[.]189. From that server the file *update* was downloaded, which was an exfiltration tool named *earthworm* from the website *rootkiter*.
- The adversary also connected to a PasteBin URL and downloaded a new version of the *softwareupdated* file.
- Port scanning activity. This is mentioned as a trace in Appendix, number 5: "Port scanning activity".
- Unrecognized script files. This could be seen as trace number 70 in Appendix: "Script files with suspicious intent"
- that the LaunchDaemon was loaded at another time than on startup, using launchctl. It also executed another process. This is mentioned as a possible trace in Appendix, number 127: "Unusual process execution from launchctl/launchd or cron tasks".
- The file *softwareupdated* had its timestamps modified. This is mentioned as a trace in Appendix, number 88: "Files that have had their timestamps modified."

**Case 3**
The third case describes an adware/spyware combination created by a company called TargetingEdge. The timeline of the attack is shown in Figure 9. The attack was described by Amit Serper on the Cybereason blog in 2017.

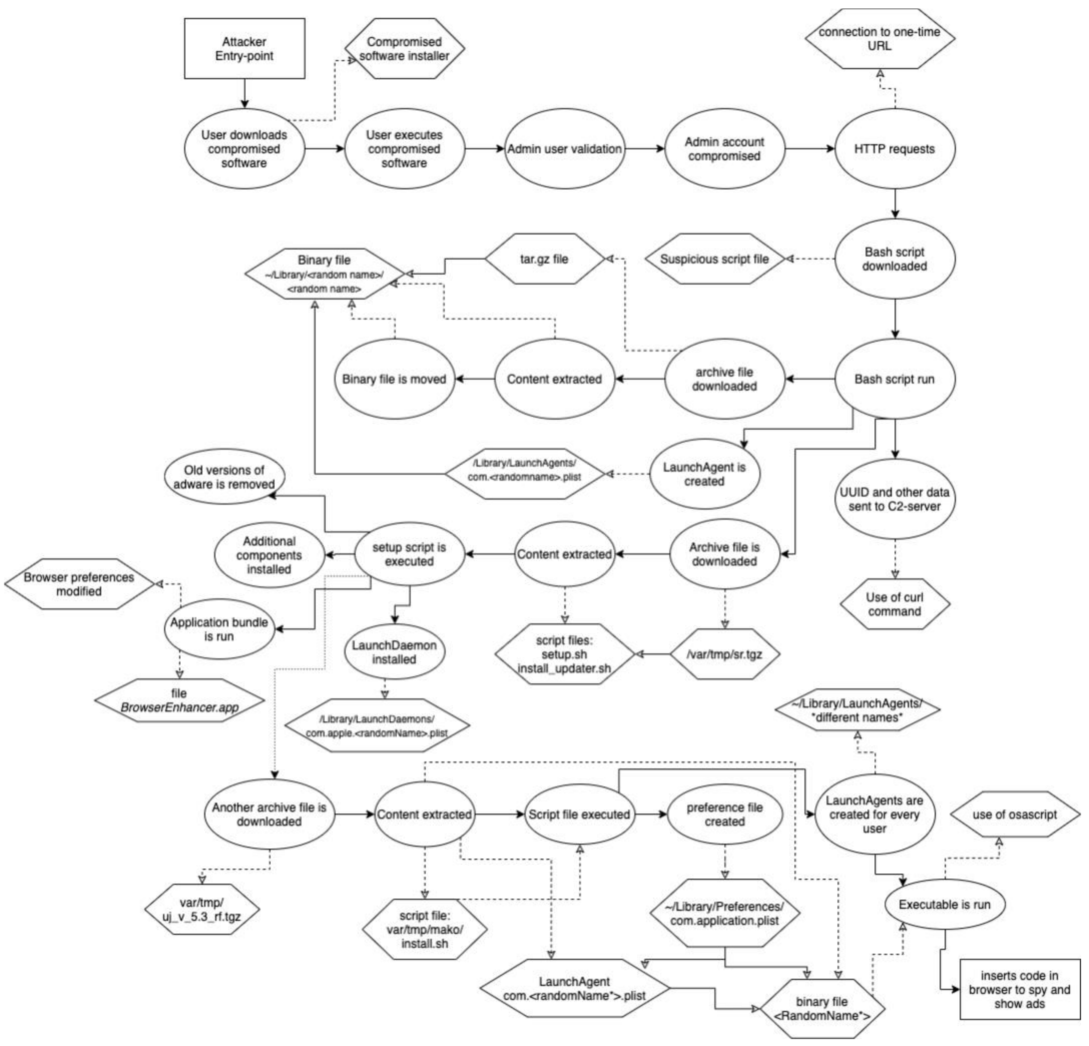

Attacker Entry-point
Compromised software installer
connection to one-time URL
User downloads compromised software
User executes compromised software
Admin user validation
Admin account compromised
HTTP requests
Binary file ~/Library/<random name>/ <random name>
tar.gz file
Suspicious script file
Bash script downloaded
Binary file is moved
Content extracted
archive file downloaded
Bash script run
Old versions of adware is removed
/Library/LaunchAgents/ com.<randomname>.plist
LaunchAgent is created
UUID and other data sent to C2-server
Additional components installed
setup script is executed
Content extracted
Archive file is downloaded
Browser preferences modified
Use of curl command
Application bundle is run
LaunchDaemon installed
script files: setup.sh install_updater.sh
/var/tmp/sr.tgz
file BrowserEnhancer.app
/Library/LaunchDaemons/ com.apple.<randomName>.plist
~/Library/LaunchAgents/ *different names*
Another archive file is downloaded
Content extracted
Script file executed
preference file created
LaunchAgents are created for every user
use of osascript
var/tmp/ uj_v_5.3_rf.tgz
script file: var/tmp/mako/ install.sh
~/Library/Preferences/ com.application.plist
Executable is run
LaunchAgent com.<randomName*>.plist
binary file <RandomName*>
inserts code in browser to spy and show ads

*Figure 9: The timeline of an example of an attack utilizing the malware in case 3 described through its attack steps, the evidence and how they are connected.*

This adware is sometimes hidden in an installation package of a legitimate software for Mac units.

*This is an example of how the adware/spyware could infect a victim's computer:*
When installing the software, the user was asked to enter their password, granting both the legitimate software and the adware admin privileges. This means that the process could then grant itself root privilege.

The installation script then sent HTTP requests to a one-time link URL. If these requests were successful, a bash script was downloaded and then run on the computer. The bash script created a new folder with a randomly generated name in the root Library folder. The bash script then extracted the UUID of the machine and sent it together with the adware's name and path to the adversary's C2 server, by using the curl command. An archive containing another executable file was downloaded from the server. The purpose of that program was to keep the adware/spyware updated. The executable file was then extracted from the archive and moved to the folder with the random name in the root Library folder. Next, a LaunchAgent was created, which when loaded would execute the executable.

An archive file which contained a browser hijacker as well as various installation and setup scripts was then downloaded. The content of the archive file was extracted, and a setup script was executed. This setup script installed several new components of the adware, including a LaunchDaemon. It also removed old versions of the adware and ran an application bundle that would change the properties of the browsers installed on the system.
Another archive file was then downloaded, and the contents extracted. These contents included an executable file, a setup script and a plist file with the preferences for the executable. A name was randomly generated for the executable and the plist file.

Next, the defaults command was used to create another plist file. This plist file included the random name generated for the executable and the name of the plist file that held the preferences for the executable.

LaunchAgents that run the executable when loaded were then created for all user accounts on the system. Then the LaunchAgent was loaded and the executable was run. The process used AppleScript to insert JavaScript code to the currently running browser. That code was used to extract information, track the user and to plant code in pages the victim was visiting.

The evidence that was left in the form of plist files in this case was several LaunchAgents, one LaunchDaemon, one with the preferences for the adware and one containing the names of adware as well as the name of preference file. The properties of these files can be observed in Table 4.

*Table 4: Shows the properties of the plist file used in case 3.*

| **Location** | **Name** | **Key** | **Value** |
|---|---|---|---|
| ~/Library/LaunchAgents/ | com.<randomName1>.plist | KeepAlive<br>RunAtLoad<br>Label<br><br>Program<br><br>UserName | True<br>True<br>~/Library/<randomName>/<RandomName><br>~/Library/<randomName>/<RandomName><br>root |
| /Library/LaunchDaemons/ | com.apple.<randomName2>.plist | - | - |
| ~/Library/Preferences/ | com.application.plist | Name<br><br>Pref<br><br>Service_pref | <randomName3><br><br>com.<randomName3>.plist<br><br>com.<randomName3>.plist |
| - | com.<randomName3>.plist | Domain<br><br><br>Machine_id<br><br>URL | http[:]//loadingpages[.]info/jo/is<br><br><Current machine's UUID><br><br>http://www.google.com |
| ~/Library/LaunchAgents/ | *Individual for every user* | - | - |

Other kinds of evidence were:

- Connecting to and downloading a script from an unknown server. This could be seen as trace number 70 in Appendix: "Script files with suspicious intent"
- Network traffic to another unknown server, i.e. the backdoor.
- Unknown files in the form of archive files (tar.gz-files) which contained bash scripts, executable files and an app bundle called *BrowserEnhancer.app*. Many of the files were named and placed in folders with names generated randomly.
- Modified search provider settings on the web browsers Firefox, Safari, Chrome and Internet Explorer. The browser's search provider then becomes either http://tika-search[.]com or http://delta-search[.]com, depending on browser settings or the user's geolocation.

- Use of osascript/AppleScript. This is mentioned as a possible trace in Appendix, number 157: “Execution of AppleScript through osascript”

**Case 4**
In this case, the goal was not specified in the source, but it seems like it was to take control of the system. The timeline of the attack is shown in Figure 10. It was documented on the AT&T Cybersecurity’s blog by Edurado De la Arada in 2013.

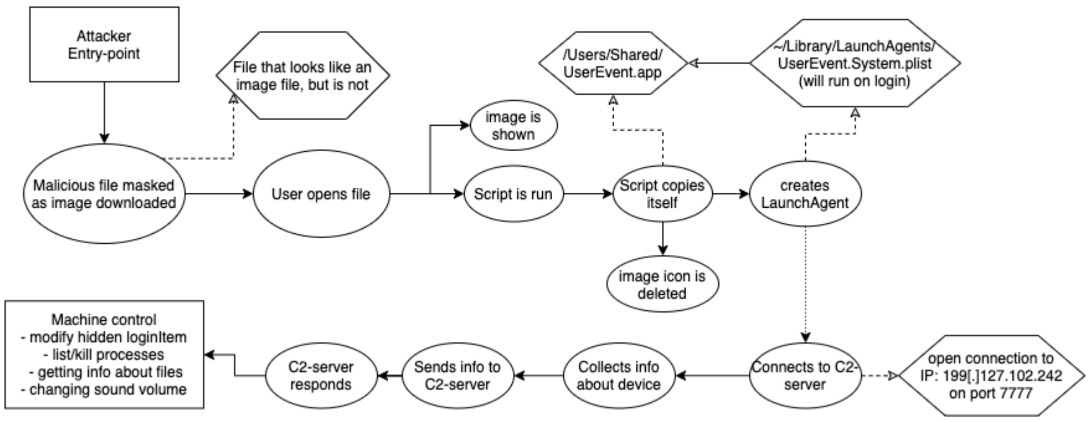

Attacker Entry-point
File that looks like an image file, but is not
/Users/Shared/ UserEvent.app
~/Library/LaunchAgents/ UserEvent.System.plist (will run on login)
image is shown
Malicious file masked as image downloaded
User opens file
Script is run
Script copies itself
creates LaunchAgent
image icon is deleted
Machine control
- modify hidden loginItem
- list/kill processes
- getting info about files
- changing sound volume
C2-server responds
Sends info to C2-server
Collects info about device
Connects to C2-server
open connection to IP: 199[.]127.102.242 on port 7777

*Figure 10: The timeline of the attack in case 4 described through its attack steps, evidence and how they are connected.*

Exactly how it was done is unknown, but somehow a malicious file masked as an image was downloaded to the target. Then the user opened the file, an image was shown, and malicious script was executed. The script was then moved to another directory. Next the script created a LaunchAgent which it placed with the user level LaunchAgents. The image icon was then deleted, to avoid that the user would open it up again. The adversary then tried to connect to a C2-server. They then proceeded to collect info about the device and sent this to the server.

Then the attacker was free to do a number of things: create and modify hidden loginItem to load software at start, get the current volume level of the target and change, list processes and their process id: s and kill them as well as getting file info and delete files.

The evidence in the form of plist files was in this case a user level LaunchAgent. Its purpose was to create persistence for malware. The properties in this file was not clearly stated, but the keys described in Table 5 was mentioned, however not their values.

*Table 5: Shows the properties of the plist file used in case 4.*

| **Location** | **Name** | **Key** | **Value** |
|---|---|---|---|
| ~/Library/LaunchAgents/ | UserEvent.System.plist | RunAtLoad<br>KeepAlive<br>Label<br>ProgramArguments | |

Other possible evidence was:

- A file that looked like an image file, but also executed other processes.
- The application named UserEvent.app which was located in the "/Users/Shared" directory.
- That the LaunchAgent executed new processes. This could be seen as trace number 124 in Appendix: "Unknown process execution resulting from login actions", since LaunchAgents on user level executes processes on login.
- Connections to servicesmsc[.]sytes[.]net, which used to resolve to the IP address 199[.]127.102.242, on port 7777, and send data to it.

**Case 5**

The fifth case seemed to be a version of *tinyshell*, which is linux backdoor. The exact purpose of the attack was unclear. It was described by Jaron Bradley in a presentation (Shakacon LLC, 2018). The timeline is described in Figure 11.

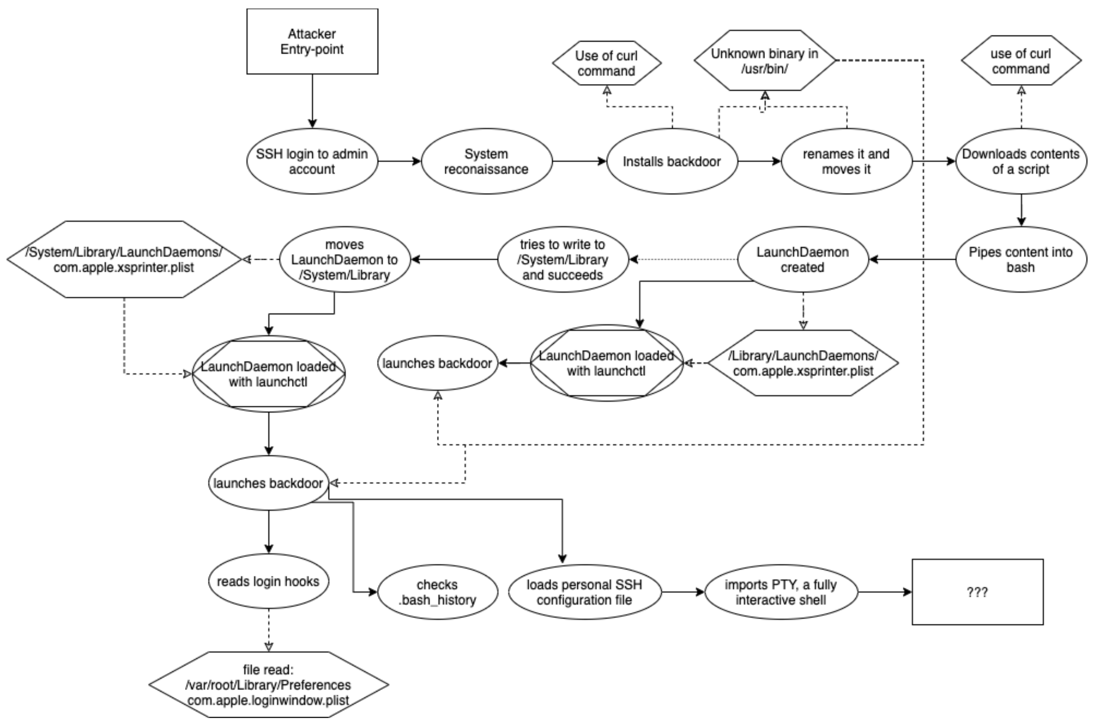

Attacker Entry-point
SSH login to admin account
System reconaissance
Installs backdoor
Use of curl command
Unknown binary in /usr/bin/
renames it and moves it
use of curl command
Downloads contents of a script
Pipes content into bash
LaunchDaemon created
tries to write to /System/Library and succeeds
moves LaunchDaemon to /System/Library
/System/Library/LaunchDaemons/ com.apple.xsprinter.plist
LaunchDaemon loaded with launchctl
launches backdoor
LaunchDaemon loaded with launchctl
/Library/LaunchDaemons/ com.apple.xsprinter.plist
launches backdoor
reads login hooks
checks .bash_history
loads personal SSH configuration file
imports PTY, a fully interactive shell
???
file read: /var/root/Library/Preferences com.apple.loginwindow.plist

*Figure 11: The timeline of the attack in case 5 described through its attack step and how they are connected.*

The initial access was through SSH, facilitated by pre-compromised credentials to an admin user account. Then the adversary did some basic reconnaissance of the system, i.e. checked the OS version, hardware details, listing of users as well as service users, and more. Next, a backdoor was installed, which was then renamed and moved to the "/usr/bin/" directory and given admin privileges.

Then the curl command was used multiple times. The adversary pulled down contents of a script and then piped it immediately into bash, instead of storing it on the file system. The backdoor talked to the server on the same IP that the tools were hosted on. Then a plist file was created for persistence. The plist file is linked to the backdoor, therefore when the plist file is loaded, the backdoor is launched.

The adversary was then able to write to "/System/Library", even though it should be protected through system integrity protection. The reason that the adversary was able to do that was because the machine's OS was outdated. The persistence plist was then placed in that folder instead and then loaded, so that the backdoor launched. They then read different login hooks and was probably letting persistence that already existed on the system kick of the backdoor instead.

Next, the adversary looked at what other systems the user had had access to through the .bash_history file. Then they loaded a personal configuration file which specified how they wanted SSH to work. After that PTY was imported, which is a fully interactive shell, used to communicate with executables without hanging. That was the last described step of the attack.

The evidence left in the form of plist-files was in this case a LaunchDaemon. No properties were specified, but its name and location are described in Table 6. The purpose of it was to enable persistence for the backdoor.

Also, the adversary read and may have edited the existing login hooks to enable persistence that way. The login hooks are also stored in a plist file called *com.apple.loginwindow.plist*.

*Table 6: Shows the properties of the plist file used in case 5.*

| **Location** | **Name** | **Key** | **Value** |
|---|---|---|---|
| /Library/LaunchDaemons/ then moved to /System/Library/LaunchDaemons/ | com.apple.xsprinter.plist | - | - |
| /var/root/Library/Preferences | com.apple.loginwindow.plist | - | - |

Other possible evidence was:

- that the adversary logged in remotely using SSH. This could possibly be seen as trace number 23 in Appendix: “User authentication, especially via remote terminal services like SSH, without new entries in that user's ~/.bash_history file”.
- the massive amounts of ping usage on a massive amount of systems on the network.
- a connection to a known bad IP and files that were downloaded from it.
- extensive use of the curl command.
- the attempt to use tools that does not exist on mac.

- that the LaunchDaemon executes other processes and was loaded at other times than boot. This is mentioned in Appendix, number 127: "Unusual process execution from launchctl/launchd or cron tasks".
- the backdoor file, which was moved to the "/usr/bin" directory.

**Case 6**
The sixth case describes a malware designed to take control of the victim's computer, probably with the purpose of stealing the victim's crypto currency wallets. Its timeline is described in Figure 12. The malware was described by Amit Malik at the Uptycs blog in 2019 and mentioned by Joshua Long in 2018.

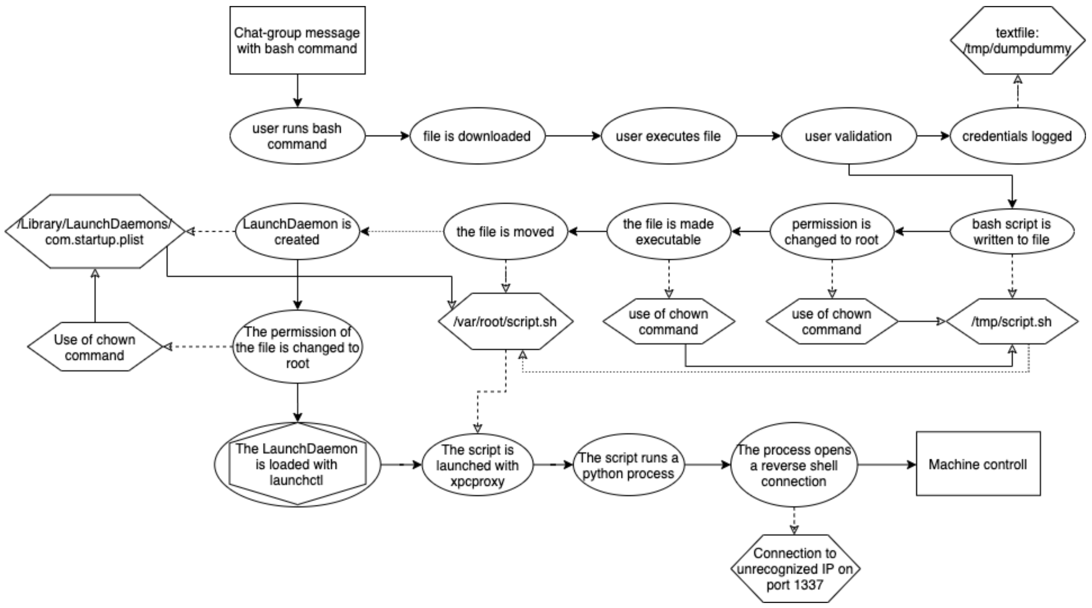

Chat-group message with bash command
textfile: /tmp/dumpdummy
user runs bash command
file is downloaded
user executes file
user validation
credentials logged
/Library/LaunchDaemons/ com.startup.plist
LaunchDaemon is created
the file is moved
the file is made executable
permission is changed to root
bash script is written to file
/var/root/script.sh
use of chown command
use of chown command
/tmp/script.sh
Use of chown command
The permission of the file is changed to root
The LaunchDaemon is loaded with launchctl
The script is launched with xpcproxy
The script runs a python process
The process opens a reverse shell connection
Machine controll
Connection to unrecognized IP on port 1337

*Figure 12: The timeline of an example of an attack utilizing the malware in case 6 described through its attack steps, evidence and how they are connected.*

*This is an example of how the malware could be used in an attack:*
The first step of the adversary was to write a message in a chat group with cryptocurrency users, i.e. doing a phishing attempt. The message encouraged the readers to copy a command and run it in their terminal. When a user did that, a malware was downloaded. The victim then had to execute the malware and also grant the malware privileges, for the account to be compromised. When the victim entered their password, it was logged and saved in a text file.

A bash script was written to a file and the permission of that file was changed to root. Then it was made executable and moved to the "/var/root" directory. A LaunchDaemon was created which purpose was to execute the script file. The permission of the LaunchDaemon was then changed to root. Next, the LaunchDaemon was loaded and therefore the script was executed. A system process called *Xpcproxy* then launched the script which then launches a python process. That process then attempted to open a reverse shell connection. If that was successful, the adversary could then control the machine, by for example sending shell commands.

The evidence that was left in the form of a plist file in this case was a LaunchDaemon. Its properties are described in Table 7. The value to the key Label only references the name of the plist file. The Program describes which program to start. The RunAtLoad field specify that the plist file should be started immediately upon load. The KeepAlive field specifies that the program should be restarted every time it ends or crashes.

In total, every time the LaunchDaemon is loaded, that is at boot or manually, the script specified is immediately executed. If it crashes or ends for some reason it will be relaunched, until the LaunchDaemon is stopped in some way.

*Table 7: Shows the properties of the plist file used in case 6.*

| **Location** | **Name** | **Key** | **Value** |
|---|---|---|---|
| /Library/LaunchDaemons/ | com.startup.plist | Label<br>Program<br>RunAtLoad<br>KeepAlive | com.startup<br>var/root/script.sh<br>1<br>1 |

Other possible evidence was:

- a text file called *dumpdummy* which stored the victim's credentials in the "/tmp/" directory.
- a script called *script.sh* also stored in "/tmp/", which was then moved to /var/root/. This could be seen as trace number 70 in Appendix: "Script files with suspicious intent"
- Use of the chown system call. This is mentioned as a trace in Appendix number 80: "Attempts to modify DACLs and file/directory ownership, such as use of chmod/chown"

- a connection to an IP address on port 1337.
- that A LaunchDaemon was loaded at another time than startup and executed another process. This is mentioned in Appendix, number 127: “Unusual process execution from launchctl/launchd or cron tasks”.

**Case 7**
The seventh case described a malware called *CresentCore* which come in several variations. Its timeline is described in Figure 13. The malware was described by Erika Noerenberg on VMware Carbon Black blog in 2019.

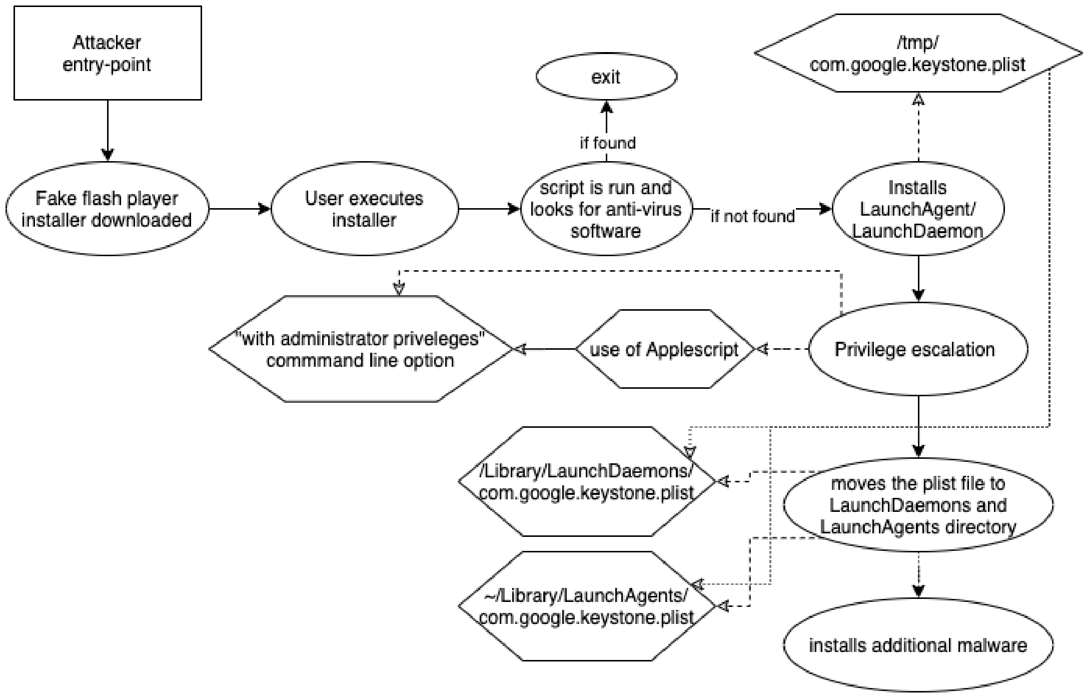

Attacker entry-point
Fake flash player installer downloaded
User executes installer
script is run and looks for anti-virus software
if found
exit
if not found
Installs LaunchAgent/ LaunchDaemon
/tmp/ com.google.keystone.plist
Privilege escalation
use of Applescript
"with administrator priveleges" commmand line option
moves the plist file to LaunchDaemons and LaunchAgents directory
/Library/LaunchDaemons/ com.google.keystone.plist
~/Library/LaunchAgents/ com.google.keystone.plist
installs additional malware

*Figure 13: The timeline of an attack utilizing the malware in case 7 described through its attack steps the evidence and how they are connected.*

The malware is often delivered in the form of a fake Adobe Flash Player installer.

*This is an example of how the malware could infect a victim’s computer:*
When a victim downloaded and executed this installer, the malware first checked for certain types of antivirus software on the system. If it found any of those, it would quit execution. If none were found the malware installed a LaunchAgent and a LaunchDaemon to the “/tmp/” folder.

Some of the variants then used AppleScript to pass the command line option "with administrator privileges". It then moved the LaunchAgent and the LaunchDaemon to the "/Library/LaunchDaemons" folder respectively the "~/Library/LaunchAgents". Some versions of the malware then continued to install other malicious software.

The evidence left as plist files in this case was a LaunchDaemon and a LaunchAgents. Their fields were not specified, but their locations are described in Table 8. The purpose of both files was described to be to launch the malicious application when loaded. The LaunchDaemon does this at boot, while the LaunchAgent does it when the user logs in.

*Table 8: Shows the properties of the plist file used in case 7.*

| **Location** | **Name** | **Key** | **Value** |
|---|---|---|---|
| /tmp/ then moved to /Library/LaunchDaemons | com.google.keystone.plist | - | - |
| /tmp/ then moved to ~/Library/LaunchAgents | com.google.keystone.plist | - | - |

Other possible evidence was:

- the use of AppleScript. This is mentioned in Appendix, number 157: "Execution of AppleScript through osascript".
- different files connected to additional malware.

## Case 8

The eighth case describes a made-up attack which uses a backdoor. The case was based on real scenarios. The timeline of the attack is described in Figure 14. The attack was described by Jason Bradley on the CrowdStrike blog in 2014.

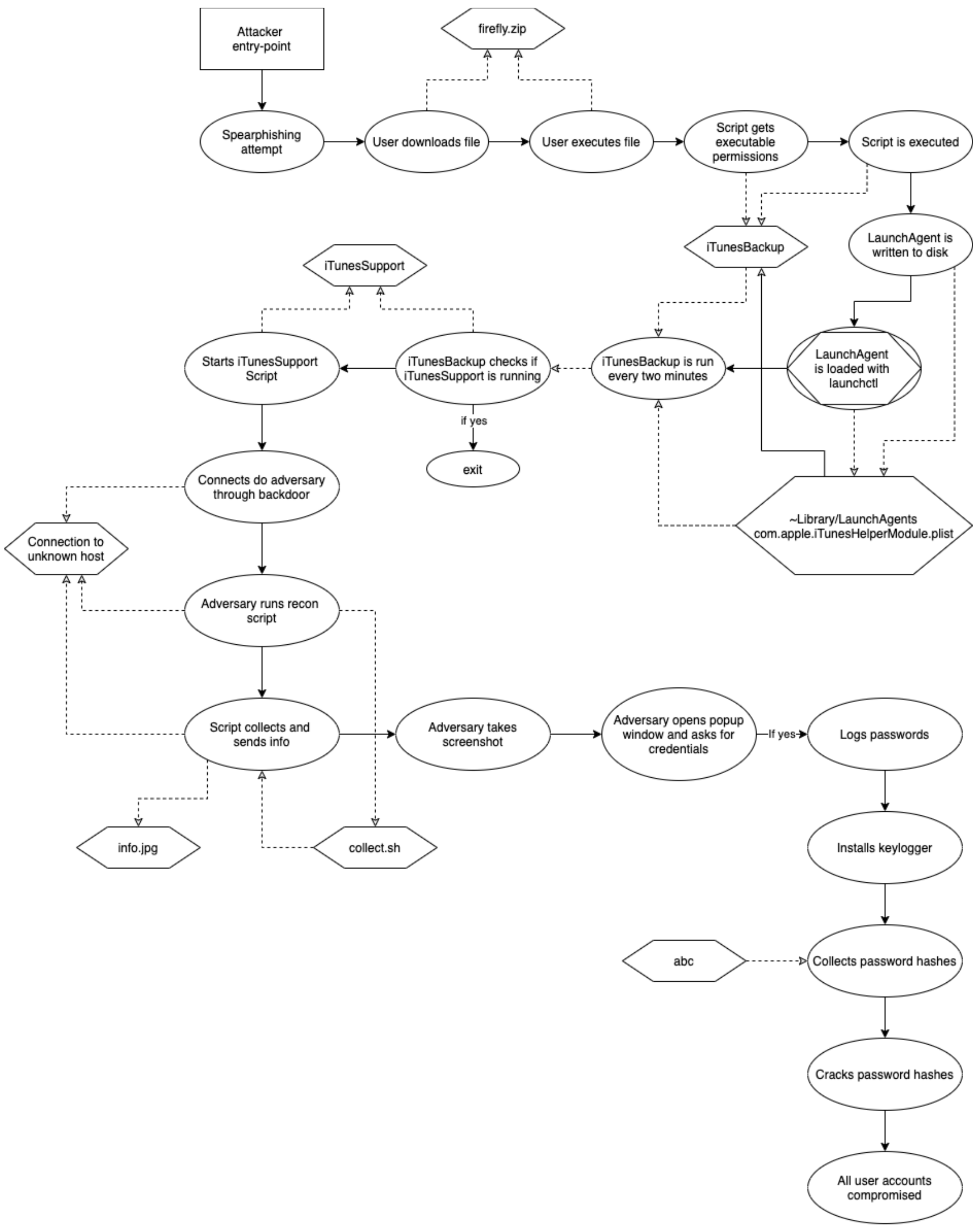

Attacker entry-point
firefly.zip
Spearphishing attempt
User downloads file
User executes file
Script gets executable permissions
Script is executed
iTunesBackup
LaunchAgent is written to disk
iTunesSupport
Starts iTunesSupport Script
iTunesBackup checks if iTunesSupport is running
iTunesBackup is run every two minutes
LaunchAgent is loaded with launchctl
if yes
exit
Connects do adversary through backdoor
~Library/LaunchAgents com.apple.iTunesHelperModule.plist
Connection to unknown host
Adversary runs recon script
Script collects and sends info
Adversary takes screenshot
Adversary opens popup window and asks for credentials
If yes
Logs passwords
info.jpg
collect.sh
Installs keylogger
abc
Collects password hashes
Cracks password hashes
All user accounts compromised

*Figure 14: The timeline of the attack in case 8 described through its attack steps, the evidence and how they are connected.*

The initial step of the adversary was to send a spear phishing email with a malicious file attached to the victim. The intention was to fool the victim into downloading the file and then executing the file. Then the script got executable permission, and was executed under another name, iTunesBackup, from a hidden directory.

A plist file was then written to disk and loaded with the launchctl command. The plist files function was to execute the iTunesBackup script every two minutes. The script checked if a python script called iTunesSupport was running. If not, it launched the python script, which then opened a connection to the adversary, i.e. the backdoor. The adversary then uploaded a short reconnaissance script to the victim's machine. The script collected information about the victim's machine and then uploaded that information to the adversary through the backdoor.

Then the adversary took a screenshot of the victim's screen. The adversary then asked for the victim's password through a popup-window. If the victim entered their password, it was logged and through that the adversary gained admin privileges, which means that they could elevate their privilege to root. Next, the adversary installed a keylogger on the victim's machine. Then, they used a password dumping tool to get the hashes for the passwords from all users on the system. Then they cracked all those hashes. When this was completed, they searched the system for data worth stealing. When found, they uploaded the documents as a tar file through the backdoor.

The evidence left in the form of a plist file in this case a LaunchAgent. Its properties are described in Table 9. The exact contents were not described, but the purpose of the file was to run the backdoor script every two minutes.

*Table 9: Shows the properties of the plist file used in case 8.*

| **Location** | **Name** | **Content** | |
|---|---|---|---|
| | | **Key** | **Value** |
| ~Library/LaunchAgents/ | com.apple.iTunesHelperModule.plist | - | - |

Other possible evidence was:

- that a LaunchAgent was loaded at another time than login. This is mentioned in Appendix, number 127: "Unusual process execution from launchctl/launchd or cron tasks".

- the (malicious) file, *firefly.zip*
- the open connection to a backdoor, i.e. an unknown ip-address.
- the reconnaissance script *collect.sh* which was downloaded through the backdoor, or the execution of the script. See Appendix, number 155: "Suspicious script execution and subsequent behaviour", or number 70: "Script files with suspicious intent."
- a file called *info.jpg*, which contained information collected by the script.
- the program "Dave Grohl" that was used to dump the passwords of the users from the "/Users" directory but was renamed to *abc*.
- that the adversary traversed the system to collect data and upload it through the backdoor.

**Case 9**
This case describes a malware called OS X Crisis. The purpose of malware was to monitor user activity by taking screenshots, copying messages and recording conversations on Firefox, Skype, Microsoft Messenger and Adium. A potential timeline is shown in Figure 15. The malware was described by Darina Stavniychuk on the MacPaw website in 2019.

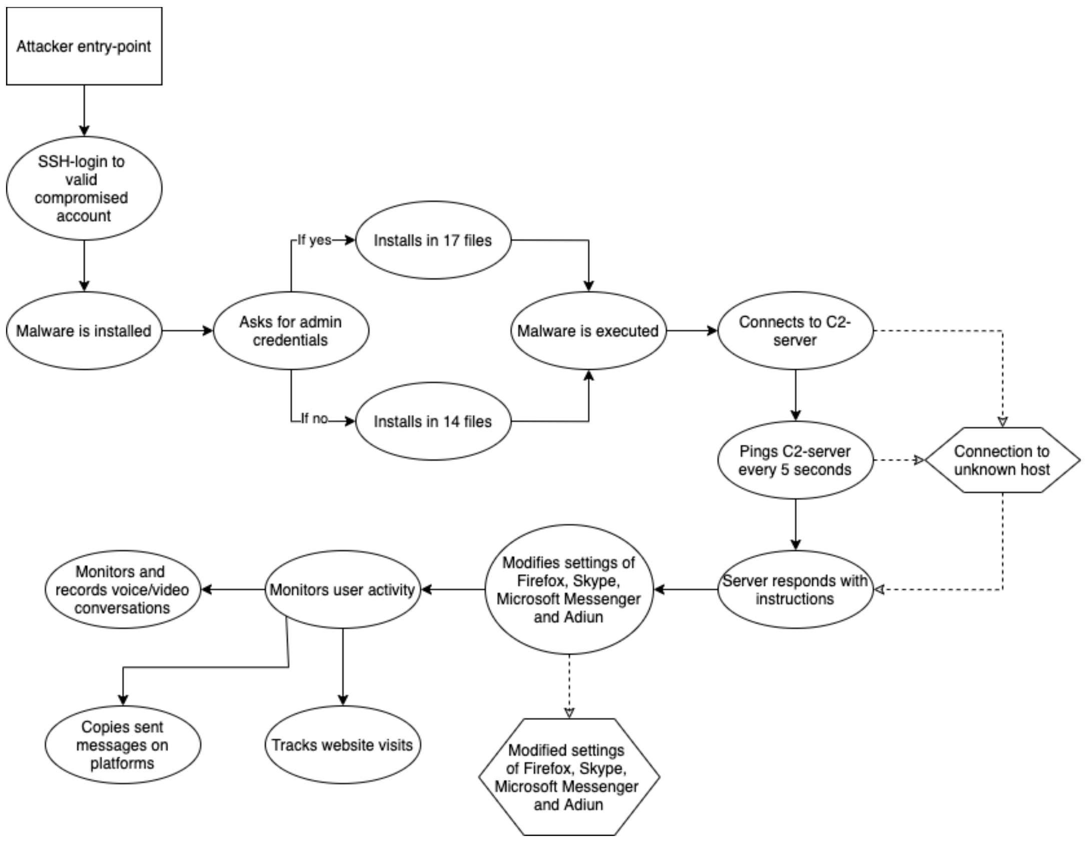

Attacker entry-point
SSH-login to valid compromised account
Malware is installed
Asks for admin credentials
If yes
Installs in 17 files
If no
Installs in 14 files
Malware is executed
Connects to C2-server
Pings C2-server every 5 seconds
Connection to unknown host
Server responds with instructions
Modifies settings of Firefox, Skype, Microsoft Messenger and Adiun
Modified settings of Firefox, Skype, Microsoft Messenger and Adiun
Monitors user activity
Monitors and records voice/video conversations
Copies sent messages on platforms
Tracks website visits

*Figure 15: The timeline of an example of an attack utilizing the malware in case 9 described through its attack steps, the evidence and how they are connected.*

*This is an example of how malware could infect a victim's computer:*
The initial step of the attack was not specified, there may therefore be several variants for initial access. The first known step was that the malware was installed. The malware then asked the user for admin credentials. If it was successful, the malware added seventeen files, if not, it added fourteen files. Then, the malware was executed. It connected to a C2-server, which was pinged every fifth second. If the server then responded with instructions, the malware could perform actions. They modified the settings of Firefox, Skype, Microsoft Messenger and Adiun, and then started to monitor the user's activity. This was done through monitoring and recording voice and video conversations, copying any messages sent on those platforms as well as tracking which websites were visited.

The evidence in the form of plist file was two with unspecified purposes. Their properties are described in Table 10. Their fields were not specified, but one of them was a LaunchAgent, and the other one was an info.plist file.

*Table 10: Shows the properties of the plist file used in case 9.*

| **Location** | **Name** | **Key** | **Value** |
|---|---|---|---|
| ~/Library/LaunchAgents | com.apple.mdworker.plist | - | - |
| ~/Library/ScriptingAdditions/appleHID/Contents/ | Info.plist | - | - |

Other possible evidence was:

- the file *com.apple.mdworker_server* in the "/System/Library/Frameworks/Foundation.framework/XPCServices/com.apple.mdworker_server.xpc/Contents/MacOS" directory.
- the directory: "/System/Library/Frameworks/Foundation.framework/XPCServices/com.apple.mdworker_server.xpc/Contents/Resources/".
- the file *jl3V7we.app* in the "~/Library/Preferences" directory.
- the file *lUnsA3Ci.Bz7* in the "~/Library/ScriptingAdditions/appleHID/Contents/MacOS/lUnsA3Ci.Bz7" directory.
- the file *appleOsax.r* in the "~/Library/ScriptingAdditions/appleHID/Contents/Resources/" directory.
- the connection to and communication with the C2-server
- the alterations of program settings for Firefox, Skype, Microsoft Messenger and Adiun.

**Case 10**

This case describes a malware called CpuMeaner. The purpose of this malware was to use the victim's computer to mine cryptocurrency. The timeline of how the malware infects a victim's computer is shown in Figure 16. The malware was described by Arnaud Abbati and Phil Stokes on the SentinelOne blog in 2017 respectively 2018.

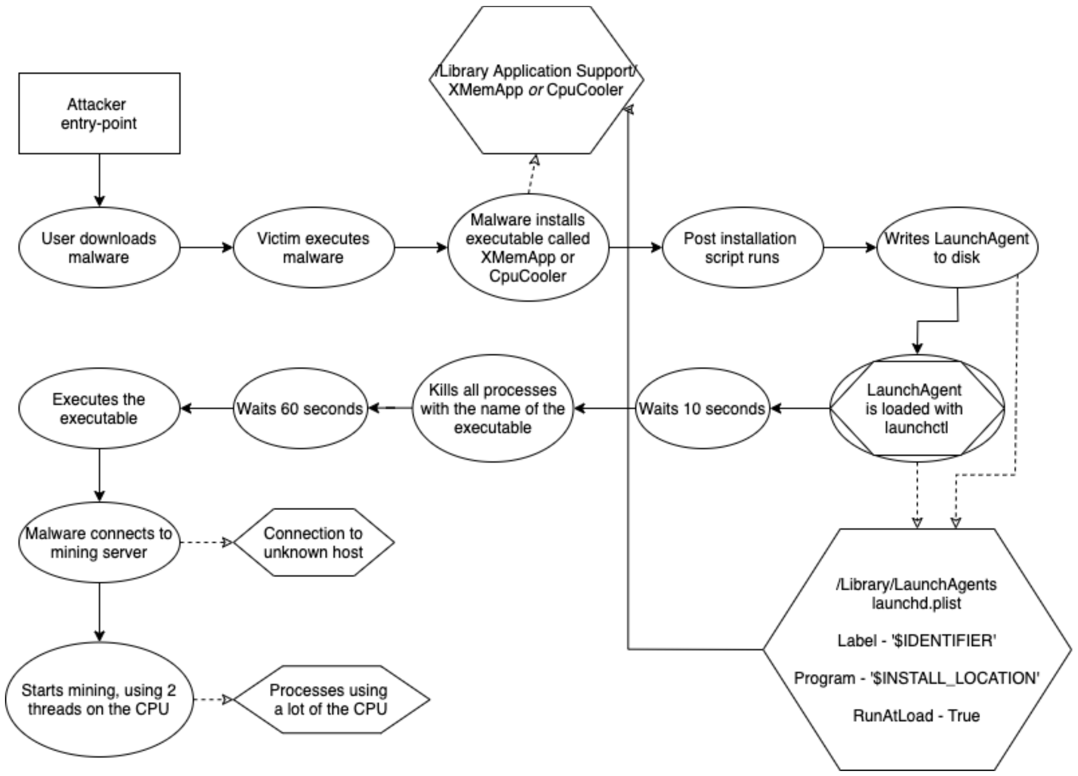

Attacker entry-point
User downloads malware
Victim executes malware
Malware installs executable called XMemApp or CpuCooler
/Library Application Support/ XMemApp *or* CpuCooler
Post installation script runs
Writes LaunchAgent to disk
LaunchAgent is loaded with launchctl
Waits 10 seconds
Kills all processes with the name of the executable
Waits 60 seconds
Executes the executable
Malware connects to mining server
Connection to unknown host
Starts mining, using 2 threads on the CPU
Processes using a lot of the CPU
/Library/LaunchAgents launchd.plist
Label - '$IDENTIFIER'
Program - '$INSTALL_LOCATION'
RunAtLoad - True

*Figure 16: The timeline of an attack utilizing the malware in case 10 described through its attack steps, the evidence and how they are connected.*

*This is an example of how the malware could be used in an attack against a victim's computer:*
The victim downloaded a package from a torrent which was disguised as pirated software. Then the victim started the installation of the software. The installation script installed an executable called XMemApp or CpuCooler. Then a post installation script was run, which wrote a LaunchAgent to "/Library/LaucnhAgents". The LaunchAgent was then loaded. Then, it waited ten seconds, then killed all processes with the name of the executable. Then it waited another sixty seconds and after that executes the executable. The malware then connected to a mining server, and started mining from the attacker's account, using two threads on the CPU.

*The evidence in the form of a plist-file was in this case a LaunchAgent. Its properties are described in*

Table 11. The exact purpose was not described.

*Table 11: Shows the properties of the plist file used in case 10.*

| Location | Name | Key | Value |
|---|---|---|---|
| /Library/LaunchAgents | launchd.plist | Label<br><br>Program<br><br>RunAtLoad | - '$IDENTIFIER' *<br><br>'$INSTALL_LOCATION' **<br><br>True |

*Comment 1: * (IDENTIFIER="com.osxext.cpucooler"), **(INSTALL_LOCATION="/Library/Application/Support/CpuCooler/cpucooler")*

Other possible evidence was:

- processes that were using a lot of the CPU. This is mentioned in Appendix, number 160: "Anomalous activity associated with malicious hijacking of computer resources such as CPU, memory, and graphics processing resources"
- the executable that started those processes, called either *XMemApp* or *CpuCooler*, placed in the "/Library/Application Support/" directory.
- network traffic to an unknown server.
- that the LaunchAgent was loaded at another time than on login and executed another process. This is mentioned in Appendix, number 127: "Unusual process execution from launchctl/launchd or cron tasks".

**Case 11**

This case described a backdoor used to install a cryptocurrency miner and steal credit card information. Its timeline is described in Figure 17. It was described by Thomas Reed on the Malwarebytes blog (2018a) and by Luis Magisa in 2019.

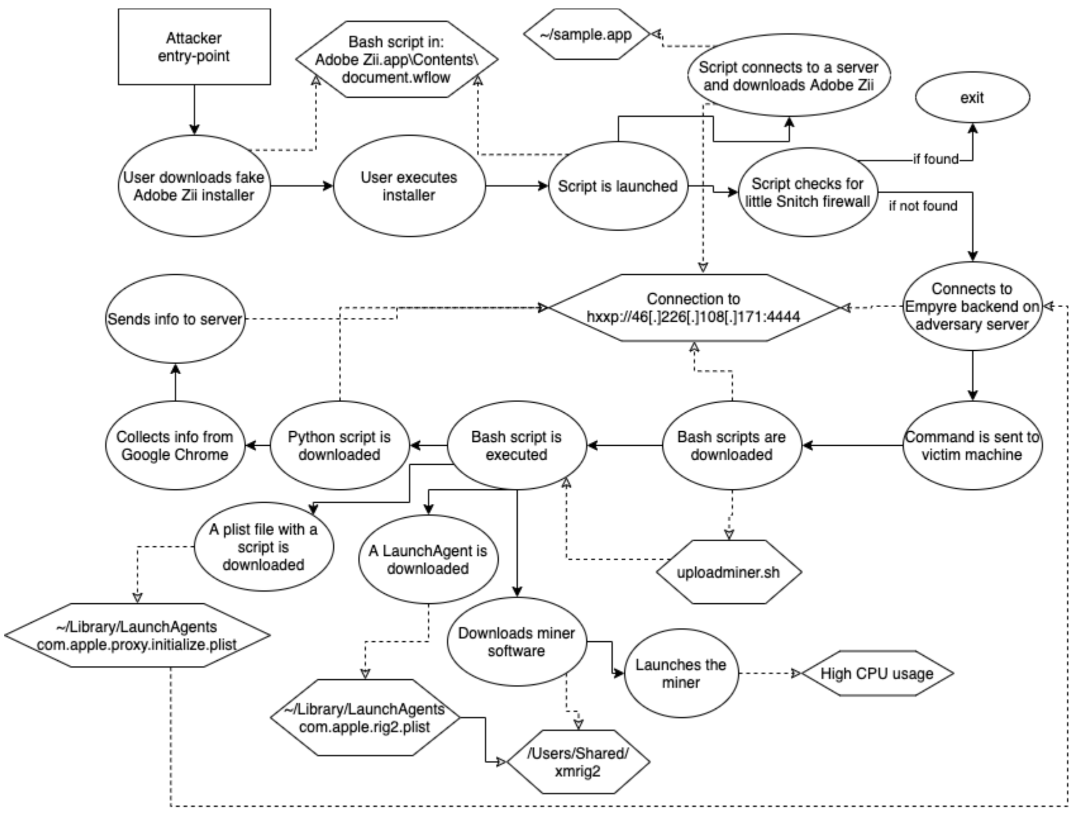

Attacker entry-point
Bash script in: Adobe Zii.app\Contents\ document.wflow
~/sample.app
Script connects to a server and downloads Adobe Zii
exit
User downloads fake Adobe Zii installer
User executes installer
Script is launched
Script checks for little Snitch firewall
if found
if not found
Sends info to server
Connection to hxxp://46[.]226[.]108[.]171:4444
Connects to Empyre backend on adversary server
Collects info from Google Chrome
Python script is downloaded
Bash script is executed
Bash scripts are downloaded
Command is sent to victim machine
A plist file with a script is downloaded
A LaunchAgent is downloaded
uploadminer.sh
~/Library/LaunchAgents com.apple.proxy.initialize.plist
Downloads miner software
Launches the miner
High CPU usage
~/Library/LaunchAgents com.apple.rig2.plist
/Users/Shared/ xmrig2

*Figure 17: The timeline of an example of an attack utilizing the backdoor in case 11 described through its attack steps, the evidence and how they are connected.*

*This is an example of how the malware could be used in an attack against a victim's computer:*
This malware posed as a legit *Adobe Zii* installer with the intention of getting the victim to download the malware and start its malicious installer. When a victim started the installer, a script that downloaded the real Adobe Zii software. This was a way for the malware to hide its existence and give the victim a false sense of security. The script also checked for a firewall software called *Little Snitch*, if found, the malware immediately exited. If not, it continued the attack by connecting to the adversary's server through an HTTP address. The adversary could then send commands to the machine through the server.

Next, a command that downloaded some bash scripts was run on the victim's system. One of these scripts was then executed. That script then downloaded a python script from a different address on the same server. The python script collected decrypted information as well as cookies from the Google Chrome browser, and then sends it back to the adversary's server.

The previously mentioned bash script then downloaded two plist files. The first one reconnected to the adversary's server when loaded, and the second one executed a cryptocurrency miner called *xmrig2* when loaded. The bash script then downloaded xmrig2 from the adversary's server and started to mine software using a command line argument.

The evidence left as plist files in this case were two LaunchAgents. Their specific fields were not described, but their other properties are described in Table 12. The purpose of the LaunchAgents was to open a backdoor respectively start a cryptocurrency miner when the user logs in.

*Table 12: Shows the properties of the plist file used in case 11.*

| **Location** | **Name** | **Key** | **Value** |
|---|---|---|---|
| ~/Library/LaunchAgents | com.apple.rig2.plist | - | - |
| ~/Library/LaunchAgents | com.apple.proxy.initialize.plist | - | - |

Other kinds of evidence were:

- Network traffic to a backdoor, i.e. an unknown server.
- Unknown files in the form of bash scripts, a python script and the cryptocurrency miner xmrig2.
- Processes that used a lot of the CPU. This is mentioned in Appendix, number 160: "Anomalous activity associated with malicious hijacking of computer resources such as CPU, memory, and graphics processing resources"

**Case 12**

This case describes a malware used to mine cryptocurrency on the victim's computer. Its timeline is described in Figure 18. The malware was described by Phil Stokes on the SentinelOne blog in 2018.

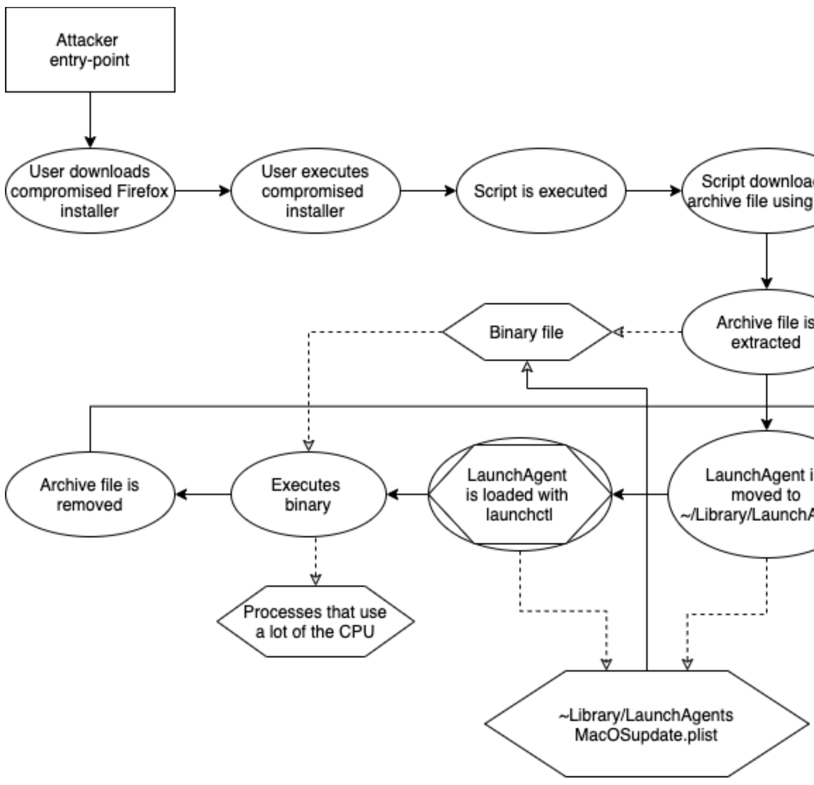

Attacker entry-point
User downloads compromised Firefox installer
User executes compromised installer
Script is executed
Script downloads archive file using curl
Archive file
Archive file is extracted
Binary file
LaunchAgent is moved to ~/Library/LaunchAgent
LaunchAgent is loaded with launchctl
Executes binary
Archive file is removed
Processes that use a lot of the CPU
~Library/LaunchAgents MacOSupdate.plist

*Figure 18: The timeline of an attack utilizing the malware in case 12 described through its attack steps, the evidence and how they are connected.*

*This is an example of how the malware could be used in an attack against a victim's computer:*
The malware posed as a real Mozilla Firefox version. When a victim was lured to download and install it, apart from installing the legit Firefox software, a malicious script was also downloaded and started. The script then downloaded an archive file using the *curl* command and extracted its contents. These included a binary file and a LaunchAgent. The latter was then moved to the "~/Library/LaunchAgents" directory. The LaunchAgent was then loaded using the command *launchctl* which led to the execution of the binary file, which was used to mine a cryptocurrency. The archive file was then removed from the system by the previously mentioned script.

The evidence left in the form of a plist file was in this case a LaunchAgent. Its fields were not described, but the rest of its properties are described in Table 13. The purpose of the plist file was to execute the miner binary.

*Table 13: Shows the properties of the plist file used in case 12.*

| Location | Name | Content | Value |
|---|---|---|---|
| ~/Library/LaunchAgents | MacOSupdate.plist | - | - |

Other kinds of evidence were:

- Unknown files in the form of an archive which was later deleted, as well as a script and a binary used for mining.
- Processes that used a lot of the CPU. This is mentioned in Appendix, number 160: "Anomalous activity associated with malicious hijacking of computer resources such as CPU, memory, and graphics processing resources"
- That the LaunchAgent was loaded at another time than on login and executed another process. This is mentioned in Appendix, number 127: "Unusual process execution
  from launchctl/launchd or cron tasks".

**Case 13**

This case described a malware which was used to mine a cryptocurrency on a victim's computer. Its timeline can be observed in Figure 19. The malware was described by Thomas Reed on the Malwarebytes blog (2018b) and mentioned by Phil Stokes on the SentinelOne blog in 2018.

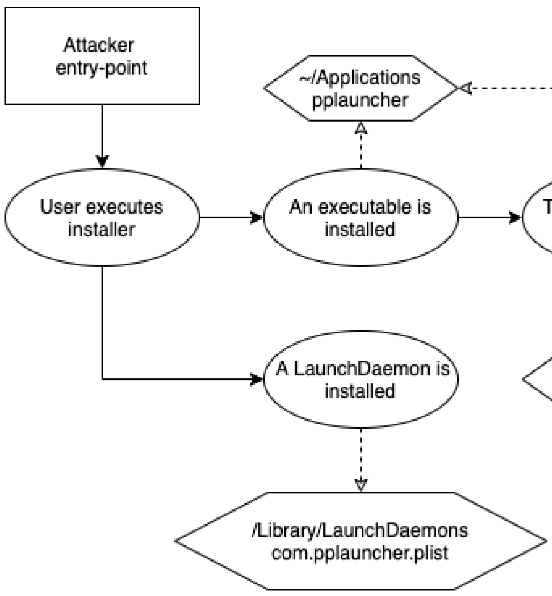
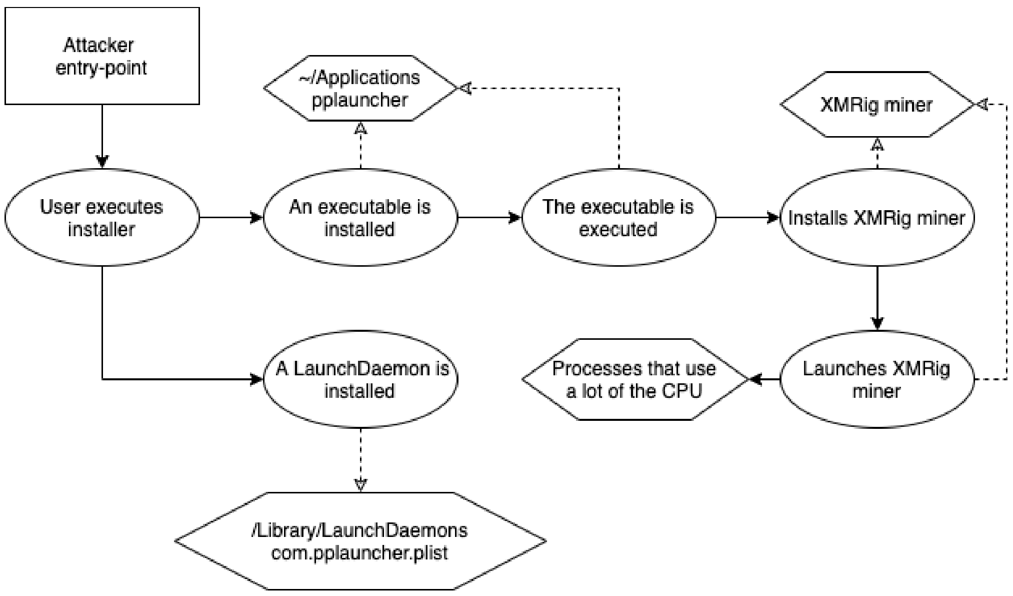

Attacker entry-point
~/Applications pplauncher
XMRig miner
User executes installer
An executable is installed
The executable is executed
Installs XMRig miner
A LaunchDaemon is installed
Processes that use a lot of the CPU
Launches XMRig miner
/Library/LaunchDaemons com.pplauncher.plist

*Figure 19: The timeline of an attack utilizing the malware in case 13 described through its attack steps, the evidence and how they are connected.*



*This is an example of how the malware could be used in an attack against a victim's computer:*
The attack started with tricking a victim into installing some software containing the malware. An executable called *pplauncher* was then installed into the "~/Applications" directory. A LaunchDaemon was also installed to make pplauncher keep on running. According to the article, this indicated that the dropper must have had root privileges. Then, the pplauncher file was launched. Its process installed and launched the cryptocurrency software *XMRig*.

The evidence left in the form of a plist file was in this case a LaunchDaemon. Its fields were not specified, but its other properties are described in Table 14. The purpose of the LaunchDaemon was to keep *pplauncher* running.

*Table 14: Shows the properties of the plist file used in case 13.*

| **Location** | **Name** | **Key** | **Value** |
|---|---|---|---|
| /Library/LaunchDaemons | com.pplauncher.plist | - | - |

Other kinds of evidence were:

- Unknown files in the form of the executable *pplauncher* and the cryptocurrency software *XMRig*.
- Processes that used a lot of the CPU. This is mentioned in Appendix, number 160: "Anomalous activity associated with malicious hijacking of computer resources such as CPU, memory, and graphics processing resources"
- That the LaunchAgent was loaded at another time than on login and executed another process. This is mentioned in Appendix, number 127: "Unusual process execution from launchctl/launchd or cron tasks".

## 4.3 Extending MAL's formalism

*This subsection presents an attempt to answer the main research question: What requirements need to be fulfilled in an extended formalism of the Meta Attack Language, to enable creation of probabilistic attack graphs which include forensic evidence in the form of property list files?*

*The analysis of the results presented in previous sections resulted in a list of requirements that need to be fulfilled if it should be possible to include such evidence as was found in this study in graphs created with the language. First the complete list of requirements is presented and then a short summary of what was looked at in the analysis is presented and all demands are explained.*

Below the list of requirements is presented. To be able to include these kinds of evidence with MAL there needs to be some way of expressing:

1. The plist file/evidence's location.
2. The plist file/evidence's name.
3. How the combination of #1 and #2 can be suspicious.
4. Which methods that can be used by adversaries to name files and folders to evade detection.
5. A relationship or connection between two pieces of evidence, which might be directed.
    a. Connecting properties of one piece of evidence with that of another piece of evidence.
6. The important contents of a piece of evidence.
7. That something is both an attack step and a piece of evidence.
8. If a program has persistence or not.
9. Events outside of the adversary's control.
10. How #7 affects the #8.
11. When and if evidence can be removed or restored to their original version.
12. A connection between an evidence and an attack step.

The majority of the plist files found in the study were either LaunchAgents or LaunchDaemons. The purpose of them was to give malicious programs such as backdoors persistence. They were placed on different levels of the system: the user level, the root level and the System Integrity Protection Level. Depending on the level, the adversary had to have different levels of privilege. Therefore, it's interesting to know the location of a file, since from that the adversary's privilege level can be deduced (#1).

Several of the plist file's name were intended to mimic legit apple services, such as "*com.apple.xsprinter*" from Case 5 and from Case 8, "*com.apple.iTunesHelperModule.plist*". Legit Apple service plist are however only placed in directories under the System Integrity Protection level, such as */System/Libraries/LaunchDaemons*. Therefore, the name and the location of a piece of evidence is interesting, as well as how these two are connected (#1, #2 and #3).

Some of the plist files and the folders they were put in were named using different kinds of random name generators. Others were named mimicking both legit Apple services, as mentioned, but also mimicking other known domains, such as Google. Therefore, it is interesting to have some way of explaining how the adversary names files and folders to evade detection (#4).

One of the most common purposes for plist files was to execute other files, such as *CpuCooler* in Case 10 and *CrashReporter* in Case 1. These other files can also be considered to be evidence. Therefore, it's interesting to be able to express some kind of relationship from one piece of evidence to another (#5). In some cases, the exact fields of the plist files were described. In the cases involving persistence, one of the fields referenced the file which was to be executed. It may be useful to be able to make some kinds of connections between specific properties of a piece of evidence to another piece of evidence (#5a).

As can be seen in Table 7 in Case 6, the properties of a plist file describes what it does. For example, the field *Program* specifies which program the plist file should execute. Another field, *RunAtLoad*, specifies that the plist file should be started immediately when it's loaded, which is not the case if this key (or *KeepAlive*) has been supplied. Therefore, there should be some way of describing the important contents of a plist file/evidence (#6).

Sometimes attack steps can also be evidence. One example is when the adversary (through a script or directly) uses the command *launchctl* to load a LaunchAgents and LaunchDaemons manually. This can be seen in a lot of the cases; one example is in Case 1 and another in
Case 13. It's an attack step since it's a step in the attack, but it is also an indication since they are usually loaded when a user login respectively when the system boots. Therefore, having some way of indicating that something is both an attack step and could be seen as evidence is necessary (#7).

Earlier examples of attack graphs often have a strictly chronological timeline, where the adversary drives the development. However, to include features such as persistence, which is important in almost all of the cases, it may be necessary to describe the timeline in another way. Events outside of the adversary's control can trigger attack steps. One example is in Case 10, where the user itself starts a downloaded malware. If the user logs out and persistence has not been established, the adversary will lose its grip of the system. To regain access the adversary is again dependent on that the user executes the malware. In other cases, depending of the way of gaining initial access, gaining back control could be practically impossible. Thus, some way of describing persistence would be helpful (#8), as well as some way of describing outside events (#9) and how these two are associated (#10).

To gain a complete picture of the evidence in the attack, it's interesting to see how long evidence in the form of for example plist files are needed, and if and when they can be removed. In for example Case 12, an archive file is removed, which shows that adversaries sometimes choose to delete potential evidence. As in Case 5, evidence could also be in the form of modifications of existing plist files. Therefore, there should be some way of describing when and if evidence can be removed alternatively restored to their original version (#11).

Evidence can be connected to either one or several attack steps, as can be seen in all of the cases. Therefore, there should be some way to connect evidence to attack steps (#12).

# 5 Discussion

*In this section the study and its results are discussed. The section is divided into subsections; first a section discussing the results, including some suggestions for how MAL could be extended based on the results, then a section discussing the methods used in this study, and last a section discussing the sources and their reliability.*

## 5.1 The result

The target of this report is to provide some insight on how forensic evidence in the form of property list files can be added to a probabilistic attack graph. As mentioned earlier the report focuses on making the graphs more useful for incident response; it does not focus on forensic analysis. This is an important distinction to make since when performing forensic analysis, the evidence should not be altered in any way, but this is not as important when it comes to incident response. Even though an incident response program should strive to facilitate a possible forensics investigation, the main focus is to stop the incident from causing further damage.

The result in this report shows a multitude of different evidence and indications of attacks on a macOS system, but actually finding this evidence or noticing these indications may require analysing the system in a way that changes the evidence or makes the evidence less reliable. There may also be ways for the attacker to fabricate or delete evidence, which is important to be aware of to maintain the chain of custody. If these results should be applied to forensic analysis in the future, these areas have to be researched further.

However, the list of requirements can be used to form a suggestion on how to extend MAL so that evidence can be included, which could possibly help in incident response. The first conclusion is that there needs to be some way of representing evidence. Evidence has a lot of similarities with attack steps. For example, pieces of evidence should be able to be connected to each other (#5) as well as attack steps (#12), which is similar to how attack step are connected to each other. A suggestion would be to model evidence based on how attack steps are modelled. Evidence can be another type of node which also belongs to different classes. One example could be that the evidence *LaunchAgent* belongs to the class *launchd daemons/files or ASEP*. The evidence can then be denoted as *ASEP.LaunchAgent*. An example of an attack step belonging to the same class could be *ASEP.launchctlLoad*. In a scenario, the LaunchAgent is loaded using launchctl. The attack step and the evidence are therefore connected. Their relationship could be symbolised as another type of edge, for example denoted as *f = {ASEP.launchctlLoad, ASEP.LaunchAgent}*.

When the LaunchAgent is loaded, another process is started, which leaves evidence in the form of *executable.bashScript*. The connection between the two pieces of evidence could be described as another type of edge, denoted by *g = {ASEP.LaunchAgent, executable.bashScript}*.

The name (#1), location (#2) and the contents (#6) can be described as properties of the evidence. For example, the key *program* can be a property of the LaunchAgent mentioned earlier. It can be denoted as *ASEP.LaunchAgent.program*, and contains the value of that key. The *program* specified in this case could be the bashScript, which means that the property program has a connection to the evidence *bashScript* (#5b). Therefore, there should be an edge between that property and the evidence. This could be denoted as follows: *h = {ASEP.LaunchAgent.program, executable.bashScript}*. Now, the edge *g* is redundant, since in this case the exact property that is connected to the other piece of evidence is known. When choosing which edges to include in the specification, choose the one with the higher degree of detail.

The earlier example of an attack step, *ASEP.launchctlLoad,* could also be seen as a piece of evidence (#7). Therefore, there should be some way to define a node as both an attack step and as evidence. This is problematic since the formalism doesn't describe a lot of details about the attack step. Evidence and attack steps are also denoted in the same manner, so there needs to be some way to differentiate them. In MAL, attack steps can be of two different types *OR* and *AND*. One possible solution would be to define attack steps and evidence as more general *nodes*, and then let them have three different types: *Attack step* (*AS*)*, Evidence* (*E*) or both (*ASE*). Another solution could be to use different syntax depending on if the node represents an attack step, a piece of evidence or both. These different kinds of nodes would then have different properties.

The list of requirements also requested a way to express when and if a piece of evidence can be removed or restored to the original version (#11). The solution depends on what information is considered most important. Is it when the adversary has elevated to a certain privilege level on the system, which makes it possible to get rid of the evidence or is it when the evidence is not needed anymore, for example when persistence is no longer necessary? With some evidence, such as the usage of a command line argument, the first is clearly more interesting, but with other evidence the latter may be more important. For the second case, it may also be possible to delete the evidence before it is no longer needed, since the adversary can replace the functionality of the first with another. This makes it more difficult to define a specific time when the evidence can be deleted, so that is also something that has to be considered.

Requirements #10 says that requirement #8 and #9, i.e. some way to express if a program has persistence and some way to express outside events, are related to each other. Persistence could be acquired during certain attack steps. Since outside events are out of the adversary's control, it is in most cases difficult for the adversary to know if and when one might occur. One solution could be to define outside events as defences and then include a way of expressing when in the attack that defence could be made useless. In this case the way defences are expressed in the language may also have to be reconsidered. Another way would be to move the outside events out of the graph. They could for example be described as a list of objects, where each object describes what would happen in the attack if it occurred. Depending on when the event occurs and what the adversary has done to protect against the event. One example could be that the adversary has enabled persistence, so when the computer turns on again, the adversary still has control of the system. Another case would be that the adversary has not enabled persistence, and therefore loses control of the system.

Requirements #3, the connection between an evidence's properties, and #4, which methods can be used to name files and folders to evade detection, are more general information, and may be difficult to include in a graph. They may fit better in a description of the system which could be used as a complement to the graph. Since this information may be class specific, such as the tactic to mask malicious plist files by naming them “*com.apple.<name>.plist”,* it may be useful for each of the classes to have their own description.

This suggestion differs from other work that has provided suggestions on how evidence can be used in attack graphs. For example, there are many differences between this report and the report “Mapping Evidence Graphs to Attack Graphs” written by Liu, Singhal & Wijesekera in 2012 even though the long-term goal is similar, i.e. that attack graphs should be made more useful in forensic analysis or in this case in incident response, by adding evidence to the graphs. That report investigates the possibility to rethink the way attack graphs are created to make it possible to add evidence or indications of an attack, whereas Liu’s, Singhal’s & Wijesekera’s report studies the possibility to map evidence graphs to attack graphs in the shape they often have today. Liu’s, Singhal’s & Wijesekera’s approach does provide a solution that makes it easier to draw conclusions from an attack graph in a forensic investigation, but to make the graphs as useful as possible, also in incident response, rethinking the way they are created from the start may give a more universal solution. An indication of this from the results in this report is that some attacks could also be considered to be evidence, which is something that would have been missed in Liu, Singhal and Wijesekera’s method.

The resulting list of requirements were based only on a few specific cases but are probably applicable to more general cases as well. For example, the edges in evidence graphs (Liu, Singhal, Wijesekera, 2012) consist of dependencies between evidence which corresponds well with requirement #5 in the results, which states that there should be some way of connecting evidence to each other. Also, since MAL is a meta language which should be able to model many if not all attack cases, it’s important that all kinds of potentially useful evidence could be described through its formalism. Thus, there may be more interesting requirements for successfully extending MAL with forensic evidence, but the requirements specified in this report should also be met.

Since the cases analysed only covered one or a few endpoints, it can be assumed that simulating larger systems with this level of detail would lead to big and quite complicated graphs. The evidence part of the graph may take as much or even more space as the attack part. It could also potentially require a lot of resources to find the relevant evidence for a domain. However, since a lot of domains have similar potential evidence, it will not be necessary to start from scratch every time. Databases such as MITRE ATT&CK could be extended further with more descriptions of attack steps and their potential evidence.

Something unexpected about the results of the grey literature study was that the plist files involved in the cases were almost exclusively LaunchDaemons and LaunchAgents, while the mac4n6 database of macOS artefacts contains a lot of other types of artefacts in the form of plist files. The reason could be that since LaunchDaemons and LaunchAgents are well known tools for persistence, i.e. ASEP:s, and they are probably one of the most common artefacts for investigators to check. This may lead to that more attacks involving these artefacts are discovered compared to other not as well-known artefacts.

A lot of the cases in the study turned out to be malware, which also affected the results. Instead of a chronological timeline, the cases' timelines were often only an investigator's interpretation of the malware, not based on an actual real-life timeline. Often, instead of an active attacker, it was a pre-programmed script that executed the commands. In the future more types of attacks should be investigated, to gain more insights.

## 5.2 The method

The study of the MITRE ATT&CK matrix gave an overview of what kind of evidence that can be left from a cyber-attack on a MacOS system. However, it is clear that the main purpose of the MITRE ATT&CK matrix is not to present different types of evidence. The detection sections are often short and not very detailed. This sometimes made it difficult to understand what specific traces respective attack technique could induce.

The choice to use a common web search engine, Google, to find sources for the grey literature study resulted in a few issues. The most important one was that the collection of sources was time consuming since most hits from the searches were not useful for the study, but they still had to be examined to make sure nothing was missed. In the end only a few of the search hits were used in the study but they gave enough results to provide qualified answers to the research questions. Another way of finding sources, that may be less time consuming, could be to go through some of the blogs mentioned in this report and look at the different attacks and malware described there. The downside of that could be that selection gets more biased, since a lot of sources could potentially be missed.

By only investigating sources that specifically mentioned plist files, sources which did not but may still have provided interesting information related to plist files may have been missed out on. Potentially, the search could have included terms such as loginItems/startItems and preferences, which are terms often related to plist files. However, this could have made the results more biased, since it would have favoured the terms selected.

The decision to base the extension of MAL on the case study probably made the final list of requirements very specified towards these cases.

## 5.3 The sources

Since the result in this report is mostly based on grey literature it is important to discuss the reliability of this literature. All sources used in the grey literature study are written by named authors on blogs or company websites that focus on different types of cyber security. Many of the articles or blog posts describe a study of an attack or a malware conducted by the authors themselves. This suggests that the used sources are quite reliable. However, conclusions made by the authors in the posts are not always properly supported and in those cases we, the authors of this report, chose to trust the authors. A more experienced forensic analyst or incident responder may be able to draw different conclusions from the reported attacks. Some of the articles or blog posts were written in a way that made the timeline difficult to follow, which means the result presented in this report is based on our, the authors, understanding of those posts.

Something else that has to be considered is that several of the used sources are companies that sell products or services in the field of cyber security. This means that some of the information they provide may be slightly biased in a way that is supposed to make readers believe they need to buy the companies’ products or services. However, there are no suggestions that this has affected the result in this study. The presented scenarios are believable and the results from the study matches many of the descriptions of traces in the MITRE ATT&CK matrix, which indicates that the information from these grey literature sources are reliable.

# 6 Conclusion

The aim of this report has been to list common traces that cyber-attacks can leave on a macOS system and then investigating what requirements would be needed to extend the Meta Attack Language with forensic evidence, by focusing on evidence in the form of plist files. The questions this report has attempted to answer are:

1. *What requirements need to be fulfilled in an extended formalism of the Meta Attack Language, to enable creation of probabilistic attack graphs which include forensic evidence in the form of property list files?*
2. What kinds of traces does common cyber-attacks leave on a Mac operating system according to the MITRE ATT&CK matrix?
3. What kinds of cyber-attacks leave traces in the form of added or modified plist files and what could these modifications include?
4. What could be other indications that the modification of the plist file was due to some malicious activity?
5. What are the essential steps that the adversary had to take to be able to make these modifications?
6. What subsequent steps could an adversary take to continue the attack and is it possible for an adversary to delete their traces in later stages of the attack?

The results in this study has shown that there are a multitude of evidence that could be found of a cyber-attack on a MacOS system. It has also shown that extending the Meta Attack Language so that it is possible to include all these types of evidence is a difficult task and would require more research. This study has not formally presented a specific way to extend the MAL formalism or syntax but has provided some suggestions on how this could be done. To extend the language is left for future research. However, the results in this study give a good idea on what requirements the extended Meta Attack

Language must fulfil to make it possible to include evidence in the form of plist files and evidence directly connected to those plist files. The study has shown that plist files are left as traces from several types of cyber-attacks, mostly as a persistence mechanism for different types of software used by the adversary. It has also shown that these plist files can lead an investigator to other evidence if they are analysed. It has also shown that depending on the plist file's location and behaviour it can indicate what type of privilege an adversary has and therefore what subsequent steps they are able to take. All these answers have provided requirements that should be fulfilled if the Meta Attack Language should be extended to include forensic evidence.

# References

Apple, (n.d.a). About Apple File System. Retrieved February 26, 2020 from the Apple Developer Website: ldeveloper.apple.com/documentation/foundation/file_system/about_apple_file_system

Apple, (n.d.b). About System Integrity Protection on your Mac. Retrieved April 29, 2020 from the Apple Support Website: support.apple.com/en-us/HT204899

Apple, (n.d.c). Core Foundation. Retrieved March 2, 2020 from the Apple Developer Website: developer.apple.com/documentation/corefoundation

Apple, (n.d.d). Script Management with launchd in Terminal on Mac. Retrieved April 29, 2020 from the Apple Support Website: support.apple.com/en-gb/guide/terminal/apdc6c1077b-5d5d-4d35-9c19-60f2397b2369/mac

Apple, (n.d.e). UserDefaults. Retrieved March 2, 2020 from the Apple Developer Website: developer.apple.com/documentation/foundation/userdefaults

Apple, (2011). Syncing Preferences. Retrieved March 2, 2020 from the Apple Developer Website: developer.apple.com/library/archive/documentation/Cocoa/Conceptual/SyncServices/Articles/SyncingPreferences.html

Apple, (2016a). Creating Launch Daemons and Agents. Retrieved April 29, 2020 from the Apple Developer Website: developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/CreatingLaunchdJobs.html

Apple, (2016b). Elevating Privileges Safely. Retrieved May 4, 2020 from the Apple Developer Website: developer.apple.com/library/archive/documentation/Security/Conceptual/SecureCodingGuide/Articles/AccessControl.html

Apple, (2018). Property List. Retrieved March 2, 2020 from the Apple Developer Website: developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/PropertyList.html

Apple, (2020). Find out which macOS your Mac is using. Retrieved March 27, 2020 from the Apple Support Website: support.apple.com/en-us/HT201260

Bradley, J. (2016) *OS X Incident Response: Scripting and Analysis*. USA: Syngress. doi: 10.1016/C2015-0-00440-3

Cocoa (API), (n.d.). In *Wikipedia*. Retrieved March 3, 2020 from en.wikipedia.org/wiki/Cocoa_(API)

Ekstedt M., Gorton D., Johnson P., Lagerström R., Nydrén J. & Shahzad K. (2015), SecuriCAD by foreseeti: A CAD tool for enterprise cyber security management. *Proceedings of the 2015 Ieee 19th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations, Edowc 2015*, 152-155

Fichera J. & Bolt S. (2013). Network intrusion analysis methodologies, tools, and techniques for incident analysis and response. Amsterdam: Elsevier/Syngress.

foreseeti, (n.d.). Ukraine Attack Path. Retrieved March 2, 2020 from the securiCAD User Community Website: community.securicad.com/ukraine-attack-path

Gabriška D. & Ölvecký M. (2018). Wiping Techniques and Anti-Forensics Methods, *2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY)*, Subotica, pp. 000127-000132. doi: 10.1109/SISY.2018.8524756

Hurlow, M. (2011). Comparing the Mac OS X Property List to the Windows Registry. Retrieved March 31, 2020 from the Mac Forensics Lab Website: https://macforensicslab.com/2011/02/11/comparing-mac-os-x-property-list-windows-registry/

Jain, A. S., & Chhabra, G. (2014). Anti-forensics techniques: An analytical review. *2014 7th International Conference on Contemporary Computing, IC3 2014*, 412-418. doi: 10.1109/IC3.2014.6897209

Jain A. & Chhabra G. S. (2014). Anti-forensics techniques: An analytical review. *Seventh International Conference on Contemporary Computing (IC3)*, Noida, 2014, pp. 412-418.

Johnson, P., Vernotte, A., Gorton, D., Ekstedt, M. and Lagerström, R. (2017). Quantitative Information Security Risk Estimation Using Probabilistic Attack Graphs. Risk Assessment and Risk-Driven Quality Assurance,pp.37-52. doi: 10.1007/978-3-319-57858-3_4

Johnson, P., Lagerström, R. Ekstedt, M. (2018). A Meta Language for Threat Modeling and Attack Simulations. *Proceedings of the 13th International Conference on Availability, Reliability and Security, pp. 1-6.* doi: 10.1145/3230833.3232799

Liu, C., Singhal, A., Wijesekera, D. (2012). Mapping evidence graphs to attack graphs. *2012 IEEE International Workshop Information Forensics and Security (WIFS),* Tenerife, 2012, pp. 121-126. doi: 10.1109/WIFS.2012.6412636

Mac4n6 Group, (2018). Mac OS X Forensic Artifacts. Retrieved May 4, 2020 from: docs.google.com/spreadsheets/d/1X2Hu0NE2ptdRj023OVWIGp5dqZOw-CfxHLOW_GNGpX8/edit#gid=1317205466

Maleh, Y., Ezzati, A., & Belaissaoui, M. (2018). Security and privacy in smart sensor networks. Hershey, PA: Information Science Reference, an imprint of IGI Global.

Malwarebytes, (2020). 2020 State of Malware Report. Retrieved February 17, 2020 from: resources.malwarebytes.com/files/2020/02/2020_State-of-Malware-Report.pdf

Marczak E., Neagle G. (2010). Property List Files. In: Andres C. et al. (eds) *Enterprise Mac Managed Preferences,* pp. 29-47. Apress. doi: 10.1007/978-1-4302-2938-4_4

Reddy, N. (2019). Practical cyber forensics: an incident-based approach to forensic investigations. Berkeley, CA: Apress. doi: 10.1007/978-1-4842-4460-9

Schembri, P. J. (2007). The Different Types of Scientific Literature. Retrieved April 27, 2020 from: um.edu.mt/__data/assets/file/0006/42981/The_different_types_of_scientific_literature.pdf

Schöpfel, J.; Farace, D.J. (2010). Grey Literature. In Bates, M.J.; Maack, M.N. (eds.). *Encyclopedia of Library and Information Sciences (3rd ed.)*. Boca Raton, Fla.: CRC Press. pp. 2029–2039.

Shakacon LLC. (2018). *Macdoored by Jaron Bradley*. Retrieved from: youtube.com/watch?v=h2dWoQobbRo

The MITRE Corporation, (n.d.). ATT&CK Matrix for Enterprise. Retrieved February 12, 2020 from: attack.mitre.org

Van Wyk, K., & Forno, R. (2001). Incident response (1st ed.). Sebastopol, CA: O'Reilly.

Wardle P., (2014). *Methods of Malware Persistence on Mac OS X*. Retrieved from: virusbulletin.com/conference/vb2014/abstracts/methods-malware-persistence-mac-os-x/ Retrieved May 4, 2019.

Zdzichowski P., Sadlon M., Väisänen T. U., Botas Munoz A. & Filipczak K. (2015). Anti-Forensic Study. CCDCOE. Retrieved March 2, 2020 from: ccdcoe.org

## Grey literature sources

Abbati, A. (2017). OSX.CpuMeaner: New Cryptocurrency Mining Trojan Targets macOS. Retrieved April 2, 2020 from the SentinelOne blog: sentinelone.com/blog/osx-cpumeaner-miner-trojan-software-pirates/

Bradley, J. (2014). Hunting Badness on OS X with CrowdStrike's Falcon Real-Time Forensic Capabilities. Retrieved April 15, 2020 from CrowdStrike's blog: crowdstrike.com/blog/hunting-badness-os-x-crowdstrikes-falcon-real-time-forensic-capabilities/

Bradley, J., Scheuerman, K. (2018). OverWatch Insights: Reviewing a New Intrusion Targeting Mac Systems. Retrieved April 1, 2020 from the Crowdstrike blog: crowdstrike.com/blog/overwatch-insights-reviewing-a-new-intrusion-targeting-mac-systems/

Fisher, D. (2019). Lazarus Group Deploying new macOS backdoor. Retrieved March 31, from the: duo.com/decipher/lazarus-group-deploying-new-macos-backdoor#d-nav-drawer

Long, J. (2018). New Mac malware targets cryptocoin 'dummies'. Retrieved on April 2, 2020 from the Intego website: intego.com/mac-security-blog/new-mac-malware-targets-cryptocoin-dummies/

Magisa, L. (2019). MacOS Malware Poses as Adobe Zii, Steals Credit Card Info and Mines Koto Cryptocurrency. Retrieved April 4, 2020 from: trendmicro.com/vinfo/fr/security/news/cybercrime-and-digital-threats/macos-malware-poses-as-adobe-zii-steals-credit-card-info-and-mines-monero-cryptocurrency

Malik, A. (2019). Mac Malware Analysis Using Osquery. Retrieved April 2, 2020 from the Uptycs Blog: uptycs.com/blog/malware-analysis-using-osquery

Noerenberg, E. (2019). Threat Analysis Unit (TAU) Threat Intelligence Notification: CrescentCore (macOS). Retrieved April 2, 2020 from the VMware Carbon Black blog: carbonblack.com/2019/12/19/threat-analysis-unit-tau-threat-intelligence-notification-crescentcore-macos/

Reed, T. (2018a). Mac malware combines EmPyre backdoor and XMRig miner. Retrieved April 4, 2020 from the Malwarebytes blog: blog.malwarebytes.com/threat-analysis/2018/12/mac-malware-combines-empyre-backdoor-and-xmrig-miner/

Reed, T. (2018b). New Mac cryptominer uses XMRig. Retrieved April 4, 2020 from the Malwarebytes blog: https://blog.malwarebytes.com/threat-analysis/mac-threat-analysis/2018/05/new-mac-cryptominer-uses-xmrig/

Serper, A. (2017). OSX.Pirrit Mac Adware Part III: The DaVinci Code. Retrieved April 1, 2020 from the Cybereason blog: cybereason.com/blog/targetingedge-mac-os-x-pirrit-malware-adware-still-active

Shakacon LLC. (2018). *Macdoored by Jaron Bradley*. Retrieved from: youtube.com/watch?v=h2dWoQ0bbRo

Stavniychuk, D. (2019). How to remove Crisis malware. Retrieved on April 2, 2020 from the MacPaw website: macpaw.com/how-to/remove-crisis-malware

Stokes, P. (2018). Crypto Mining Update: How macOS Malware is on the Rise. Retrieved April 3, 2020 from the SentinelOne blog: https://www.sentinelone.com/blog/macos-cryptomining-malware-rise/

Wardle, P. (2019). Pass the AppleJeus – a mac backdoor written by the infamous Lazarus apt group. Retrieved March 31, 2020 from the Objective-See blog: objective-see.com/blog/blog_0x49.html

# Appendix

*This Appendix shows a list of possible indications of a cyber-attack mentioned in the "Detection" sections in the MITRE ATT&CK matrix (The MITRE Corporation, 2020). This list was created in the study aimed to answer the second research question of this paper: "What kinds of traces does common cyber-attacks leave on a macOS system according to the MITRE ATT&CK matrix?".*

**Unusual network traffic or network connections**

1. Uncommon data flows in the network
2. Processes utilizing the network that do not normally have network communication
3. Network activities disassociated from user-driven actions from processes that normally require user direction
4. Processes that have not been seen before utilizing the network
5. Port scanning activity
6. Network traffic that contains abnormal data for the network, for example ICMP messages or other protocols
7. Traffic to known anonymity networks (such as Tor) or known adversary infrastructure that uses [Multi-hop Proxy]
8. Sudden increases in network or service utilization
9. Sudden surge in one type of protocol
10. Network connections to the same destination [regularly]
11. Use of utilities, such as FTP, that does not normally occur
12. ARP spoofing and gratuitous ARP broadcasts.
13. Packets that do not belong to established flows, sent to or from the system
14. Packet content that shows communication that do not follow the expected protocol behavior for the port that is being used.
15. Command and control traffic. (May be encrypted.)
16. Abnormal patterns of activity from a user to an external web service.

17. Unrecognized devices, services, or changes to the MBR.
18. Loss of availability for a service that may be targeted by an Endpoint DoS.
19. Computer systems or network devices that should not exist on a network.
20. Unplanned content changes in internal or external websites.
21. Improper inputs attempting exploitation in web application. Can be detected by firewall.

**Changes to network settings**

22. Changes to host adapter settings, such as addition and/or replication of communication interfaces.

**Unusual authentication attempts**

23. User authentication, especially via remote terminal services like SSH, without new entries in that user's ~/.bash_history file
24. Suspicious administrator logins or configuration changes.
25. Suspicious account login activity on the deployment system.
26. New unrecognized accounts or accounts assigned to admin roles.
27. A “user's actual ID and effective ID are different”. (Can be monitored and indicates use of sudo.)
28. Unusual access by abnormal users or at abnormal times.
29. Suspicious access patterns after a remote login.
30. Suspicious account behavior across systems that share accounts [...] for example one account logging in to multiple systems simultaneously.
31. User that does not show up on the login screen.
32. Several remote access tools trying to access the system. (May be difficult to detect.)
33. Access to and use of Legitimate Credentials to access remote systems within the network. (Can possibly be detected through security logs).
34. Authentication logs showing a high amount of login failures of a Valid Account or several different accounts.

### Log files showing suspicious behaviour

35. Software deployment logs from third-party software showing suspicious or unauthorized activity.
36. Missing security logs or event files.
37. Logs in the web server, application server or database server that indicates DoS-attack against a web application.

### Unusual API calls

38. Unusual "access to certain APIs and dashboards that may contain system information" in cloud-based systems
39. Unusual process access to "APIs associated with devices or software that interact with the microphone/video camera, recording devices, or recording software"
40. "API calls to functions associated with enabling and/or utilizing alternative communication channels"

### Unusual command-line arguments or system calls

41. Unusual process execution with command-line arguments.
42. "Abnormal commands shown in ~/.bash_profile and ~/.bashrc"
43. Command-line arguments for applications that may be used by an adversary to gain Initial Access
44. Command-line arguments for script execution and suspicious subsequent behavior.
45. Command shell execution of source and subsequent processes that are started as a result of being executed by a source command
46. AuthorizationExecuteWithPrivileges is being called. (Possibly shown in MacOS system logs)
47. "Linux specific calls such as the ptrace system call, the use of LD_PRELOAD environment variable, or dlfcn dynamic linking API calls"
48. "Command-line arguments for actions that could be done before or after code injection has occurred"
49. "Command-line arguments that stop security tools from running."

50. “Command-line deletions functions that can be correlated with binaries or other files that an adversary may drop and remove”
51. “Commands containing indicators of obfuscation and known suspicious syntax such as uninterpreted escape characters like "'^'" and """"."”
52. “Common file system commands and parameters on the command-line interface within batch files or scripts.”
53. “Command-line arguments for known compression utilities”
54. “Execution of kextload commands”
55. “Command-line arguments for actions indicative of hidden windows”
56. “Suspicious words or regular expressions that may indicate searching for a password in command line arguments.”
57. Command-line arguments for actions that could be taken to collect files from a system, network share or removable media
58. Suspicious system calls to the keychain

**Unrecognized files or tools**

59. “Suspicious files written to disk”
60. Unverified distributed binaries
61. Suspiciously large files.
62. Unrecognized files that are encrypted, hidden or uses steganography
63. “New files that could be executed with the source command”
64. “Files that have the setuid or setgid bits set”
65. “Files with known names but in unusual locations”
66. “Known software packers or artifacts of packing techniques.”
67. “Compressed or encrypted data in publicly writeable directories, central locations, and commonly used staging directories (recycle bin, temp folders, etc.)”
68. “Non-native binary formats and cross-platform compiler and execution frameworks [...] that does not have a legitimate purpose on the system.”
69. “Suspicious or overly broad trap commands” in code files.
70. Script files with suspicious intent.
71. “Non-standard extensions in file names”
72. “File names that do not match their expected hash”
73. Malicious signatures in downloads.

74. “Suspicious/unexpected values in application binary file hashes”
75. “New files being created with a leading "."”
76. Multiple dylib files with the same name
77. New plist file on disk
78. Plist files with the apple.awt.UIElement tag

## Modified files and folders

79. Changes in the /etc/sudoers file.
80. “Attempts to modify DACLs and file/directory ownership, such as use of chmod/chown”
81. “Files that are modified outside of an update or patch”
82. “modification of the HISTFILE and HISTFILESIZE environment variables or the removal/clearing of the ~/.bash_history file”
83. Unrecognized changes in the /Library/StartupItems folder
84. Unknown changes in Apple menu -> System Preferences -> Users & Groups -> Login items and the corresponding file locations.
85. Suspicious changes in Contents/Library/LoginItems.
86. “Changes to files in the Web directory of a Web server that do not match with updates to the Web server's content”
87. “removal of the com.apple.quarantine flag by a user”
88. Files that have had their timestamps modified.
89. “Large quantities of file modifications in user directories”
90. An added or modified LC_MAIN entry point of a binary file
91. “Files added or modified by unusual accounts outside of normal administration duties”
92. Unknown changes in the /etc/rc.common file
93. The moving, renaming, replacing, or modifying of dylibs.
94. “Files created or modified in /etc/emond.d/rules/ and /private/var/db/emondClients”
95. “Changes to application binaries and invalid checksums/signatures”
96. “Changes to binaries that do not line up with application updates or patches”
97. Modified plist files. Especially by a user.
98. Specific plist files associated with reopening applications can indicate when an application has registered itself to be reopened
99. Users checking or changing their HISTCONTROL environment variable

## File-deletion

100. “Known deletion and secure deletion tools” added to the system
101. “Improper deletion or modification of indicator files.”

## Suspicious driver or certificate installation

102. “Unusual kernel driver installation activity.”
103. “New suspicious certificates installed on a system”

## Suspiciously accessed files

104. “Unusual processes performing sequential file opens and potentially copy actions to another location on the file system for many files at once”
105. Unknown file access on removable media
106. “Strange access patterns to files and directories related to cryptographic keys and certificates.”
107. Attempts to access files and repositories on a local system that are used to store browser session cookies
108. “Access to information repositories performed by privileged users.”
109. “Users retrieving and viewing a large number of documents and pages from information repositories.”
110. “User SSH-agent socket files being used by different users.”
111. Browser files with credentials being read by anything but the browser.
112. Web servers accessing files that are not in the Web directory
113. Suspicious process opening the /proc//maps file.

## Read/write attempts to sensitive locations

114. Attempts to read/write to sensitive locations such as: the partition boot sector or BIOS parameter block/superblock or the master boot record and the disk partition table

## Abnormal processes or executions

115. "Abnormal behaviour of processes"
116. Abnormal behaviour of internet/public-facing applications.
117. Named pipe creation. (Could indicate infected process.)
118. "Processes using tools and command line arguments that may indicate they're being used for password policy discovery."
119. Unrecognized applications and processes related to remote admin tools
120. Process periodically writing files to disk that contain audio data/video or camera image data
121. "Suspicious process trees"
122. "Processes for actions that could be done before or after code injection"
123. "Processes that stop security tools from running."
124. "Unknown process execution resulting from login actions"
125. Processes for actions indicative of hidden windows
126. Suspicious processes executed through trap interrupts
127. Unusual process execution from launchctl/launchd or cron tasks
128. "Process activity that does not correlate to known good software"
129. "/usr/libexec/security_authtrampoline executions"
130. "Processes that may be used to modify binary headers"
131. "Process execution resulting from modified plist files"
132. Programs that are executed from /Library/StartupItems that don't match the whitelist
133. Unusual processes that are executed during the bootup process
134. Suspicious processes that gather a variety of system information
135. "Processes that execute when removable media is mounted"
136. "Attempts by programs to inject or dump browser process memory"
137. "Processes that appear to read files from disparate locations and write them to the same directory or file"
138. "Unrecognized processes or scripts that appear to be traversing file systems and sending network traffic"
139. "Execution of known compression utilities"
140. "Execution and command-line arguments of binaries involved in deleting accounts or changing passwords" or "data destruction activity" or "shutting down or rebooting systems"

141. "Changes in the set of dylibs that are loaded by a process that do not correlate with known software, patches, etc."
142. "Process execution from rc.common script"
143. Unstable or crashing processes. (May indicate exploitation.)
144. "Programs installed via launchctl as launch agents or launch daemons."
145. Unusual certificate characteristics and outliers on executed software.
146. Execution of applications that may be used by an adversary to gain Initial Access
147. "Unknown or unusual process launches outside of normal behavior on a particular system occurring through remote interactive sessions"
148. "Unusual processes with external network connections creating files on-system"
149. "Common cryptomining software process names and files on local systems that may indicate compromise and resource usage"
150. "Processes and command-line arguments for actions that could be taken to gather system and network information"
151. Processes for actions that could be taken to collect files from a system or its network share or some removable media.
152. A process that maintains a long connection during which it consistently sends fixed size data packets.
153. A process that opens connections and sends fixed sized data packets at regular intervals

**Input prompts**

154. Strange banners, text, timing, and/or sources in an input prompt.

**Unknown scripts**

155. Suspicious script execution and subsequent behaviour
156. Malicious instances of scripting that could be used to prompt users for credentials
157. Execution of AppleScript through osascript (if related to other suspicious behaviour)
158. Non-granted attempts to enable scripts running on a system

159. Scripts not commonly used on a system, but enabled, scripts running out of cycle from patching or other administrator functions

## Suspicious computer resource behaviour

160. Anomalous activity associated with malicious hijacking of computer resources such as CPU, memory, and graphics processing resources
161. Suspicious use of network resources associated with cryptocurrency mining software

## Malicious files or links in emails

162. Malicious files or links in internal emails
163. Malicious files or links in external emails

## Suspicious domains

164. A pseudo-randomly generated domain name.
165. Recently registered domain name or rarely visited domain.
166. HTTP header that doesn’t match HTTPS SNI and is not in a whitelist of domain names or that matches a blacklisted domain name

TRITA-EECS-EX-2020:357