Mac-A-Mal: macOS malware analysis framework resistant to anti evasion techniques

Duy-Phuc Pham¹ · Duc-Ly Vu² · Fabio Massacci²

Received: 4 August 2018 / Accepted: 10 June 2019 / Published online: 20 June 2019
© The Author(s) 2019

Abstract

With macOS increasing popularity, the number, and variety of macOS malware are rising as well. Yet, very few tools exist for dynamic analysis of macOS malware. In this paper, we propose a macOS malware analysis framework called Mac-A-Mal. We develop a kernel extension to monitor malware behavior and mitigate several anti-evasion techniques used in the wild. Our framework exploits the macOS features of XPC service invocation that typically escape traditional mechanisms for detection of children processes. Performance benchmarks show that our system is comparable with professional tools and able to withstand VM detection. By using Mac-A-Mal, we discovered 71 unknown adware samples (8 of them using valid distribution certificates), 2 keyloggers, and 1 previously unseen trojan involved in the APT32 OceanLotus.

Keywords Malware analysis · Static analysis · Dynamic analysis · Malware detection · MacOS · APT malware

1 Introduction

Contrary to popular belief, the Mac ecosystem is not unaffected by malware. In 2014, the first known ransomware appeared, and other ransomware has been discovered as Software-as-a-Service (SaSS), where malware is available as requests. Mac devices saw more malware attacks in 2015 than the past five years combined, according to a cybersecurity report from the Bit9 and Carbon Black Threat Research team.¹ In 2016, Mac malware grew 744% with around 460,000 instances detected, says McAfee report [1] and increases 270% between 2016 and 2017 (Table 1).²

There exist tools which support malware analysis of Windows, Linux or Android applications, while, investigation of macOS malware and development of tools supporting monitoring their behavior is still limited in functionalities or anti-analysis resistance, or both. For example, the open source Mac-sandbox [2] is vulnerable to anti-analysis techniques such as Dylib name verification. Cuckoo sandbox [3] does not support anti-analysis mitigation and human interaction under the macOS environment. The closed source FireEye monitor³ use a kernel extension which is resistant to anti-analysis techniques, but requires human intervention. VirusTotal Box of Apples sandbox⁴ executes malware to show screenshots of what an analyst would see, also reports network traffic and file operations but the underlying technology itself is enclosed.

Our goal is to design and implement a malware analysis framework, which can automatically capture malware behavior in an adversary environment, called Mac-A-Mal. It consists of two main modules implemented at user-space and kernel-space. At the kernel level, we implement system call hooking and process tracing techniques to capture system calls and their arguments. During sample execution, the analysis engines are customized to prevent analysis traces left

¹ <http://www.businessinsider.com/there-are-more-mac-malware-attacks-in-2015-than-last-five-years-combined-2015-11>.

² <https://blog.malwarebytes.com/101/2018/03/the-state-of-mac-malware/>.

✉ Duy-Phuc Pham
duy-phuc.pham@irisa.fr

Duc-Ly Vu
ducly.vu@unitn.it

Fabio Massacci
fabio.massacci@unitn.it
<http://disi.unitn.it/~massacci/>

¹ TAMIS, IRISA/INRIA, Rennes, France

² Department of Information Engineering and Computer Science, University of Trento, Trento, Italy

³ https://www.fireeye.com/blog/threat-research/2017/03/introducing_monitor.html.

⁴ <http://blog.virustotal.com/2018/03/additional-crispiness-on-macos-box-of.html>.

on the system while maximizing malware behavior exposure using memory patching and virtual machine hardening techniques. At the user level, we make specific handlers to deal with various file types.

Using Mac-A-Mal, we discovered 71 unknown Adware under 8 legitimate certificates, 2 keyloggers, and 1 trojan involved in the APT32 OceanLotus, which are unknown to many Anti-virus vendors for a long time. Only Ikarus and ESET-NOD32 products can recognize two of these unknown samples at the time of finding. Further, we observed that many of them are aimed with sophisticated anti-analysis techniques like one avoiding the bash execution to stay under the radar. After a few days since our analysis reports have published on social media, many anti-virus vendors had updated their engines to be able to detect these unknown samples (Table 2).

In Sect. 2, we provide some background on macOS security. Then we review related literature on macOS malware analysis and detection in Sect. 3. In Sect. 4 we describe details of our approach. The anti-evasion techniques analysis and mitigation are presented in Sect. 5. Comparison of Mac-A-Mal and other dynamic analysis tools are conducted in Sect. 6. In Sect. 7, we show case studies of analyzing malware family based on our Mac-A-Mal. In Sect. 8 we present a summary of our approach and highlight future directions in Sect. 9.

2 Background on OSX kernel and application security

The basic principles of OSX Security follow the Common Data Security architecture, which consists of three layers: firmware, kernel-space, and user-space security [4]. Each layer contains specific features to secure data. In this paper, we only discuss the security properties of kernel-space and user-space layers. Some relevant firmware attacks are: Mac EFI rootkits [5] with boot device replacement techniques or ThunderStrike [6] permanently modifies firmware modifications. For other security aspects of components in these aforementioned layers, we refer [7].

Mac kernel is built from BSD and Mach. There are two main features in *Kernel-space security*: Portable Operating System Interface (POSIX) and Apple security features. The BSD itself defines access policies to system artifacts (e.g., files, computing resources) based on user and group IDs at various levels. Virtual memory pages are encrypted to prevent memory allocation exploitation attacks. These attacks are widely used by malware to force the processor to execute arbitrary code from another process' memory area. Address Space Layout Randomization (ASLR) is adopted to prevent exploitation by randomly offsetting location of modules and specific in-memory structures.

System Integrity Protection (SIP) is a built-in feature introduced in OS X El Capitan to protect system files and directories against the modifications caused by non-“entitlement” processes. It also keeps a process from code injection and runtime attachment attacks. A runtime debugging (e.g., DTrace) cannot interact with a protected process because SIP denies any attempt to load the unsigned kernel extensions (KEXT). Using KEXT as a device driver is a common technique of rootkit.

Sandboxing techniques restrict untrusted applications under the controlled environment such as files they can access or the ability to connect the internet. The built-in Mac Sandbox system consists of user-space library functions to initialize and configure environments for different processes, a Mach server which is responsible for handling logs from the kernel, a kernel extension that enforces security policies, and a kernel extension which provides regular expression matching for policy enforcement [8].

macOS uses its built-in XPCProxy tool to spawn new services. The child processes of Apple applications are often spawned via XPC services, which do not inherit their parent's environment.⁵ Security monitoring tools should keep track of “XPC children.” `/usr/libexec/xpcproxy` is a “trampoline” that's used by *launchd* to launch XPC services using `posix_spawn()` system call. It uses `xpc_pipe_routine()` to communicate back to *launchd* to get the information needed to spawn the XPC service – notably the path to the XPC service executable and the values that should be set in its environment.

The *Code signing* mechanism uses a digital signature used to verify the software's origin and integrity in case binaries were subverted. Built-in Anti-malware *XProtect* scan suspicious files by matching their signatures with malware definitions in the `Xprotect.plist` file. *Gatekeeper* prevents applications that are downloaded and installed from malicious websites. It suffers from the same restriction of XProtect [9]. In particular, Malware that signed with a valid Apple certificate can be able to bypass Gatekeeper. Any modification that invalidates the signature will cause the binary crash. However, malware authors can unsign an application, infect it with malicious code, and then execute it without crashing [8]. In contrast, all kernel extensions are required to be signed to prevent malicious code in kernel space. Moreover, the verification step is performed in user mode, and therefore malware author can turn on user-mode checking (on their signature) and load their kernel extensions.

⁵ For example, a user opens a new tab in Safari, rather than forking another process, it calls a service and *launchd* will spawn the new tab process. So if we use traditional hooking methods, it may lose that “child” behavior.

Table 1 Supported files of macOS analysis sandboxes

Sandbox Technologies	Supported packages
<i>Cuckoo Sandbox</i>	Mac application, Bash, Mach-O, Zip.
<i>Joe Sandbox</i>	Mac application, DMG, Mach-O, XAR
<i>Mac-sandbox</i>	Mac application, Bash, Mach-O, Zip, HTML, PDF, RTF, JAVA JAR, doc.
<i>VirusTotal Box of Apples Sandbox</i>	Mac application, Mach-O, PKG package.
<i>Mac-A-Mal Sandbox</i>	Mac application, DMG image, PKG package, Bash, Mach-O, Zip, Python, Perl, Office documents, URL, HTML, PDF, Java applications, RTF, XAR, generic files.

Table 2 Comparative analysis of competing solutions

Features	Cuckoo	Mac-sandbox	Fireeye Monitor	Mac-A-Mal	Virustotal Box of Apples	Joe sandbox
Automation	✓	✓		✓	✓	✓
Static analysis		✓		✓	✓	
Anti-debug detection & mitigation				✓		
Anti-analysis detection				✓		
Human interaction simulation		✓		✓	✓	✓
Process tree	✓	✓		✓	✓	✓
System calls monitor	✓	✓		✓		
XPCProxy process monitor			✓	✓	✓	
Dropped files capture	✓	✓		✓	✓	
Network graphical statistic	✓	✓		✓	✓	✓
Detection rule adoption	✓	✓		✓	✓	
Support any types of samples			✓	✓		
Modular framework	✓	✓		✓		

Mac-A-Mal shows many supported features that others are not able to perform. All systems do File operations monitor, Network monitor, DNS requests and replies, Dynamic library loads capture, Behavior filtering

Table 3 Survey of binary analysis frameworks for macOS executable

Framework	Hopper	IDA	Radare2	MachO View	lldb	Otool
Architecture	Intel, ARM, PPC	Intel, ARM, PPC	Intel, ARM	Intel	Intel	Intel
License	Comm.	Comm.	OS	OS	FW	FW
Dynamic analysis	Yes	Yes	Yes	No	Yes	No
Decompiler	Yes	Yes	No	No	No	No

Otool is equivalent to objdump and lldb is alternative of GDB on Linux. The spectrum of tools is available under commercial licenses (Comm), Open Source (OS), and Freeware (FW)

Table 4 Survey of sandboxes

Sandbox	Anubis	CWSandbox	Cuckoo	Virustotal Box of Apples	Joe sandbox
Architecture	Intel, ARM	Intel, ARM	Intel, ARM, PPC, SPARC, MIPS	Intel	Intel, ARM
Format	ELF, PE	ELF, PE, MachO, Dex, etc.	ELF, PE, MachO, Dex, etc.	MachO, Dex, PE	ELF, PE, MachO, Dex, etc.
License	Comm.	Comm.	OS	Comm.	Comm.

The spectrum of tools is available under commercial licenses (Comm) or Open Source (OS)

Table 5 Mac-A-Mal system calls hook

Files	<code>open, open_nocancel, access, access_extended, mkdir, mkdir_extended, chflags, utimes, utime, pathconf, truncate, rename, readlink, copyfile, unlink, chown, lchown, unlinkat, renameat, renameatx_np, linkat, faccessat, fchmodat, getxattr, fsetxattr, fgetxattr, fchownat, chdir, chroot, guarded_open_np, read, write, close</code>
Processes	<code>posix_spawn, execve, __mac_execve, fork, vfork</code>
Network	<code>socket, sendmsg, sendmsg_x, sendmsg_nocancel, recvmsg_x, recvmsg, recvmsg_nocancel, send, sendto, sendto_nocancel, recvfrom_nocancel, recvfrom, getsockopt, setsockopt, connectx, connect, connect_nocancel, getpeername, getsockname, bind, listen, accept</code>
Anti-analysis	<code>csrctl, ptrace, __sysctl, quotactl</code>

3 Related work

Static analysis tools demonstrated in Table 3 are common commercial and opensource disassemblers used by analysts to extract static information of malware such as file header, code disassembly. The commercial *IDA Pro*, *Hopper* and *Radare2* support more architectures than the others. They also provide *bindings* for scripting languages such as Python. Static features can be feed to machine learning algorithms to automatically classify malware samples in [4,10]. They extracted binary header, load commands, and segments features from the dataset of 2300 benign and 760 malicious samples and achieved 96.62% accuracy.

Dynamic analysis Existing tools shown in Table 2 proposed to monitor dynamic artifacts in the system such as processes in user-space and kernel-space. *FireEye monitor* collects information about processes, file and network activities, TTY events from the kernel-space. It, however, lacks any support of analysis automation for large-scale analysis. *DTrace* logs program execution trace (e.g., system calls, CPU performance) using *probes* callbacks from the user-space. Lindorfer [11] has used the tool for their analysis of honeypot captured samples. Moreover, it has three limitations: (i) requiring root privilege to run samples, (ii) inability to monitor official Apple binaries without placing them in a temporary folder and (iii) fail to follow a new process or fork triggered by `launchd`. The limitation (i) has been addressed by the opensource Cuckoo sandbox [3] via two running modes: executing DTrace as root permission or dropping privileges to the user with `sudo` command and alter the local `sudoers` file to reflect the change. This solution, however, appears to be trivial to detect a process executing under root privilege, analysts, therefore, must issue an additional `sudo` to drop the privilege. As a result, the analysis trace might unwillingly include benign `sudo` activities which confuse analysts. Only filtering them out would lead to significant information missing. Our observation in Table 6 shows that Mac-A-Mal found 31.03% samples of the dataset exhibits super-user permission (`sudo`) requests. *Mac-Sandbox* is an

extension of Cuckoo sandbox, which uses Dylib hijacking [12] to suspend process and hook system calls of particular interest. Compared to Cuckoo, it extended the static analysis and human interaction simulation. It, however, seems to leave analysis traces on the system caused by its suspending and resuming operations, and thereby highly be vulnerable to anti-analysis techniques. Moreover, relying on the parent-child process tracing mechanism at user-space, it likely neglects malicious processes spawned by system services.

Memory analysis using Volatility plugin [13] involves inspection of relevant macOS subsystems and volatile artifacts lie in the memory. J. Rentzsch [14] uses function overriding and code injection to execute the hooking code. They patch the target function to jump to a dynamically allocated executable code area (Table 4).

4 General system architecture

Figure 1 illustrates our Mac-A-Mal system design. The analysis machine submits samples to the monitor analysis machine. To prevent analysis information leak and infect to the analyst's host machine, we turn off data sharing features on the guest machine. To run applications from the Internet, we disable Gatekeeper feature and remove built-in XProtect malware detection as well as disable default Apple protection mechanisms to capture malware behavior explicitly. The Analyzer component of the analysis machine is responsible for distributing samples to one or more monitor machines and display analysis results to users. The monitor machine executes the examples and collect system call, process, file operation logs, and send them to the analysis machine. After the execution ends, the monitor machine is restored to a clean state by using snapshot mechanisms.

Using default Mac handler `-usr/bin/open` and *XPCProxy*, so we can open and trace up to at least 15 file types as shown in Table 1. It supports more file types than other sandboxes by executing samples via default `open` handler mechanism,

Fig. 1 Mac-A-Mal framework design

which is similar to how users open a file by double click. The challenge is that `open` spawns the new process which goes through the `mac_syscall` interface that prevents any tracing attempts regard to sandbox policy. Apparently, by tracing in kernel space, we can execute any samples in which DTrace and MacSandbox failed to. It means any types of files can be opened with the capability of process forks tracing (e.g., Office documents) without using other file handlers. We also classify file types for different ways of handling, for example, archives such as Apple disk image, we firstly mount via `hdiutil`, and then `mac-A-Mal` executes all applications inside the attached volume. For cross-platform malware variants using scripting application such as Java, Perl, Python, we use their own interpreters to execute (e.g. CrossRAT malware).

Then *System call logging* module will capture system calls using hooks at the kernel-space. By doing so, we monitor raw malware behaviors. Our module finds system call location in the `sysent` table by three steps: first we un-slide KASLR address to obtain kernel base image address, then walk through the kernel base image to locate kernel's load command segment, and finally perform system table lookups. Once the location of the `sysent` table is known, we replace its entries with our defined callback functions. As a result, when a malware invokes a system call, it will query the address of system call in the BSD kernel syscall table which matches our callback function address. The callback function then performs execution logging, malware evasion detection, and mitigation, and decide proper post-processing for the calls (forward or drop). Table 5 shows a list of system calls that are hooked by callbacks by categories, which can be easily extended by writing custom callback functions. With SIP enabled, the compiled KEXTs of our module must be signed with a developer ID for signing KEXTs certificate.

We further trace processes to give insights into their characteristics, the interaction between them and exclude

Fig. 2 Process argument logging

irrelevant processes. We capture system calls, their arguments, and program path by making a tunnel to transfer data between user and kernel side as depicted in Fig. 2. In particular, the challenge is to transfer arguments which are typical arrays.⁶ Mac-A-Mal hooks system calls at kernel space and therefore can trace process spawns by `open`, `launchd`, and `xpcproxy` processes accurately, while DTrace, Cuckoo sandbox, and Macsandbox cannot trace. Information about process creation is collected by filtering MAC framework Kernel Authorization (KAuth)'s scope `KAUTH_FILEOP_EXEC` for process notifications. In particular, KAuth calls a defined callback function which logs a Vnode reference of the running program. We collect its full path, process ID (PID), and parent ID (PPID) in each process. The processes are visualized the tree structure that displays child and parent relationships. By Analyzing process tree, we can be able to identify suspicious processes such as orphaned processes.

Mac-A-Mal captures and automatically dissects arguments of shipped XPC-Proxy component and `posix_spawn()` function described in Fig. 3. To capture parent-child relationship among processes (see 2 on XPCProxy) The framework can keep track of the information of XPC children by using `XPCproxy`. In particular, `xpc_pipe_routine()` communicate back to `launchd` to get the information of the spawn XPC service such as the path to the service executable and environmental values. Then `XPCproxy` calls `posix_spawn()` function with the `POSIX_SPAWN_SETEXEC` flag to execute the XPC service. The general logging scheme optimizes the logger via internal sockets which is thread-safety and synchronous. Also, during malware execution, the extension does not leave any explicit traces on the system in comparison to previous approaches ([15,16]).

Automation The communication protocol between the analysis and monitoring machine is designed to be reliable

⁶ For example a program can spawns: `execve('/bin/ls', ['-lua', '/etc/passwd'], [])` to list `/etc/passwd` file and its attributes. After hooking from kernel space, we need to copy its values out to userspace's logger. It is trivial to log the first argument, which is a string, but second and third arguments which are arrays are not. However the address value of these arrays in the kernel space is collected, and each argument value has the different 8 bytes of address (for 64bit kernel). So we used an enumeration of each 8 bytes address to copy out the value of the array.

Fig. 3 Mac-A-Mal XPCProxy tracing

and straightforward. In particular, Mac-A-Mal sends data to user-space via a socket, which is thread-safe, synchronous, and fast. A thread is created to handle messages over the socket and wait for system call events. Once a system call is issued, the socket waked and started sending data to the daemon. The daemon then receives data and selects the next action for the process, such as doing a start or stop. Following steps describe the workflow of Mac-A-Mal sandbox: (i) Restore snapshots, (ii) Download sample(s), (iii) Static analysis, (iv) Sample execution, (v) Suspension, (vi) Produce reports.

Malware persistence mechanisms analysis and detection Malware employ persistence mechanisms to be hidden in the system for a long time. An identification of persistence indicators can be useful to fingerprint malware if it is unique enough. Simple malware can modify Unix startup files, install malware as a `launchd` daemon [17]. Advanced malware can leverage cron jobs, dynamic libraries hijacking, MachO binary infection, and application-specific installation (e.g., Safari, Google Chrome, iTunes, etc.) [9].

Human interaction simulation: The framework constantly captures changes in the analysis screen. The Quartz graphics layer library is used to detect interaction windows and react keyboard and mouse events. First, screenshot images are analyzed to seek for login forms or confirmation buttons. Mostly, Mac applications ask for superuser credentials, user agreement, installation location, etc. Mac-A-Mal passes username, password, and mouse clicks using Quartz's `KeyboardEvents` and `MouseEvents`.

5 Robust anti-evasion techniques

Mitigating anti-debugging techniques Apple introduces two standard methods to detect and prevent applications from

being attached by debuggers. `sysctl()` retrieves information about a process and determine its debugging status. Furthermore, `PT_DENY_ATTACH` protects a process from debuggers by using non-standard `ptrace` requests. Any attempts to trace the process will receive the exit status or a segmentation violation. We place a monitor module in the kernel-space to capture and analyze both `sysctl()` and `ptrace` anti-debug calls and their arguments.

Mitigating Anti-Virtual machine techniques Analysts usually disable SIP to enable kernel debugging, load unsigned kernel extension, or remove DTrace restrictions. While these deactivation techniques allow bypassing constraints by Apple, their footprints can be leveraged by malware to detect the analysis machine. Malware can exploit the difference in cycles time between real and virtual processor to identify a virtual environment (Locky and KeRanger ransomware). We detect anti-virtual machine techniques by hooking and replacing `csrctl()` function call which is used to verify SIP. We catch attempts to gather environment artifacts such as configuration files, processor specifications, running processes and randomly change machine settings (e.g., MAC address, BIOS ID) to make malware harder to detect. Figure 4 shows how our memory patching technique is implemented to adjust CPU and SIP configuration information for mitigating anti-analysis malware⁷

6 Performance evaluation

We first analyze the performance of Mac-A-Mal as it needs to be close to bare performance as much as possible on the other hand. For the comparison, we select the opensource BYTE's Unix Benchmark Suite due to its controllability, repeatability, and durability. To make the suite compatible with macOS platform, we rebuilt it with some modifications. *Read Timestamp Counter (RDTSC) instruction* is used to determine how quickly the processor executes program's instructions. It returns the number of ticks counted since the last reboot. Slowness in the processor's execution might indicate the presence of malware analysis tools, such as a debugger. Hence, the faster an analysis tool runs, the less chance it has of getting detected by malware. We run Mac-A-Mal on the benchmark of 1000 system calls for 100 times and calculate the overall execution time. Figure 6 demonstrates that Mac-A-Mal performs faster than other competitors. In particular, Mac-A-Mal completes the benchmark with only 9.934 and 20,363 number of cycles on `stdout` writing and `localhost` connection operations, respectively.

⁷ If a malware uses `sysctl()` to retrieve information about the number of CPU on the machine, we will intercept its request and return a reasonable value. In practice, a value of 4 or 8 commonly used in a real computer.

Fig. 4 Mac-A-Mal anti-environment evasion module using hook and memory patching. (1) A system call is issued in the user space, (2) Hook the system call, (3) Return the control to the original system call, (4) Returns value to the user space and manipulate system call arguments if anti-evasion attempts are detection, (5) The return value is logged for further analysis using Logger component, (6) returns values from the kernel space is also to the analyzer in user-space, (7) the `exit()` system call, the is not hooked and forward back to the user-space

Fig. 5 We benchmark every system tracing time against a Macbook Pro running samples naturally without extra hooking/monitoring techniques

Fig. 6 We benchmark every system against a MacBook Pro running samples naturally without any extra hooking/monitoring techniques. Hence that is the best performance that one can achieve. Their baseline took 19.013 and 5.686 number of cycles to write to stdout and connect to the localhost, respectively. Note that anti-debugging is not required for Mac-A-Mal and FireEye Monitor

Figure 5 depicts that Mac-A-Mal's *tracing time* on the same benchmark, our tool takes only 1.1 seconds to complete the benchmark which is approximately 16 times faster than Macsandbox and 15 times than Cuckoo sandbox (Fig. 6).

Table 6 Analysis of malware behavior and techniques used by examples in the dataset

Features	Found (%)	Not found (%)
Persistence	60.92%	39.08%
Anti-analysis	13.79%	86.21%
Network	72.41%	27.59%
Tor	5.75%	94.25%
Open port	10.34%	89.66%
File write	82.76%	17.24%
Browser change	29.89%	70.11%
Invoke spawn	75.86%	24.14%
Superuser request	31.03%	68.97%

The percentage of samples exhibiting classified activities (FP = 0%). Detected by AVs with 97.70% accuracy and 2.30% false positive rate

7 Malware family analysis

Our *Dataset* consists of 2,000 unique macOS malware examples collected from VirusTotal Malware Intelligence Services (VTMIS),⁸ Macmalware,⁹ and Objective-see¹⁰ between January 2017 and July 2017. macOS malware evolution is investigated using heat map from 2016 to 2017. Figure 7 shows that only 2 samples use anti-analysis techniques while 5 samples do in 2017. We observed that some samples are an improvement of existing malware appeared in 2016. In 2017, we have also observed that the increasing trend of using malware to affect victim internet browser, to redirect them

⁸ <https://www.virustotal.com/#/intelligence-overview>.

⁹ <http://macmalware.manwe.io/>.

¹⁰ <https://objective-see.com/>.

Fig. 7 Behavior heatmap of macOS malware in 2017. The behaviors can be grouped as following categories: (i) persistence, file write, browser modification (ii) root request, bash execution (iii) anti analysis (iv) network, Tor, open port (v) detection rate by AVs on the x-axis. ChromePatch and Mughthesecc are classified as Adware, Macransom is ransomware and the rest of variants is trojan/backdoor on the y-axis. We clustered operations as they are often needed one another. For example: persistence needs to write file locations, modify startup locations. Browser modification (like adware installing malicious plugins on the browser), also make file write operation

to fraud web traffic or unwanted advertising for profit.¹¹ In 2017, we have also observed that the trend of using malware to affect victim internet browser to redirect them to fraud web traffic or unwanted advertising for profit.

Table 6 shows suspicious behaviors captured by Mac-A-Mal from malware samples in the dataset. Many of them execute network operations, writing data to the victim machine, spawn processes or threads, and perform persistence techniques. Among anti-analysis techniques, we observed common system calls such as `ptrace()`, `sysctl()`, `sleep()`

We write analysis rules to identify suspicious characteristics or behaviors such as network activity, anti-debug techniques, persistence mechanisms, etc. We then apply the rules to analyze malware in the dataset. Figure 7 shows our analysis result for malware families. Family *OSX/Fake File Opener* appears to do something benignly with neither network activity nor persistence mechanisms. However, further analysis on the samples we found a large embedded Info.plist file which registers 232 file extensions to the operating system. Such behavior makes the malware look potentially like an unwanted application that masquerades

as a document handler. *macOS/Mughthesecc* tries to hijack victim's browsers, install the SafeFinder adware, and use query victim's Mac address to detect virtual machine, download additional malicious applications such as a scam utility Advanced Mac Cleaner. *OSX/Dok* employs anti-analysis technique, uses Tor to communicate with its command and control server or try to tamper victim network traffic, modify browser settings, spawn shell, and request superuser permission to disable system update as well as XProtect signature.

More importantly, Mac-A-Mal shows significant approaches in detecting and analyzing unknown malware using VTMISS. We provide an analysis of two undetected malware campaign. The first one is *OSX/Mughthesecc* adware campaign (founded in August 2017) where Mac-A-Mal discovered 71 signed archives, 10 Apple legitimate developer certificates (only 2 of them revoked later). Another one is the APT32-OceanLotus campaign where we catch the second generation of Mac APT32. More details our finding and malware analysis artifacts, refer to [18], [19].

8 Threats to validity

Although we implement our logging mechanism in the user-space, which might be detected by malware, its counterpart kernel module, however, is armed with advanced rootkit functionalities [20] such as KEXT, process and file hider can minimize the traces. Besides, our framework could also not capture all anti-analysis techniques by malware authors as they are becoming more and more complex. But with its

¹¹ For example XAgent is an upgraded version of the Komplex malware which participated in APT28 targeting individuals macOS computers in the aerospace industry in 2016 and Trojan Downloader.Agent.X.CXB spreads malware by executing malicious Visual Basic for Applications (VBA) code which contains payloads for both macOS and Windows. OceanLotus.B has been evolved from its variant in 2016 avoiding Bash command execution, which is more vulnerable to anti-analysis techniques.

nature open-source¹² enables analysts to add their own handlers for a specific technique.

9 Conclusion and future work

In this paper, we presented a hybrid malware analysis framework where static and dynamic analysis are combined to support the analysts. Its key feature is a kernel hooking module, which implements various analysis techniques including system calls and services invocations that are macOS specific. We provide many hooks for arguments monitoring, process tracing, anti-analysis detection, and mitigation. Mac-A-Mal performs best among other monitor tools on the same benchmark both regarding tracing process execution time and many CPU counts. With Mac-A-Mal, we discovered 71 unknown Mac malware variants currently using new evasion techniques. Our behavior classification rules allow us to filter out behaviors that are unrelated to malware. While 85% of the collected samples are adware, most of the classified variants (49%) belong to the trojan/backdoor family. We discovered a significant improvement of Mac evasion methods such as sleep probing, detect MAC address, etc.

An interesting future development is to use Mac-A-Mal combined with machine learning techniques to detect and classify malware automatically. To this extent, Mac-A-Mal provides a rich set of traces (e.g., system call, network activity, persistence mechanisms, anti-debug techniques) that machine learning tool can exploit.

Acknowledgements The project leading to this paper has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 675320 (NeCS: European Network for Cyber Security). This work was also partially supported by Securify B.V.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Beek, C., Frosst, D., Greve, P., Gund, Y., Moreno, F., Peterson, E., Schmugar, C., Simon, R., Sommer, D., Sun, B., Tiwari, R., Weafer, V.: McAfee labs threats report. <https://goo.gl/bKfcW3> (2017). Accessed 19 July 2017
2. Walkup, E.: Mac-sandbox: an os x extension for the open-source cuckoo sandbox for malware analysis. <https://github.com/ewalkup/cuckoomac> (2014). Accessed 20 July 2017
3. Guarnieri, C., Tanasi, A., Bremer, J., Schloesser, M.: Automated malware analysis—cuckoo sandbox. <http://www.cuckoosandbox.org> (2012). Accessed 30 Aug 2017
4. Hsieh, S., Wu, P., Liu, H.: Automatic classifying of mac os x samples, VB2016. <https://www.virusbulletin.com/conference/vb2016/abstracts/automatic-classifying-mac-os-x-samples> (2016). Accessed 20 July 2017
5. Loukas, K.: Mac efi rootkits, BlackHat. http://ho.ax/De_Mysteriis_Dom_Jobsivs_Black_Hat_Paper.pdf (2012). Accessed 19 July 2017
6. Hudson, T., Kovah, X., Kallenberg, C.: Thunderstrike 2: Sith strike. Black Hat USA Briefings (2015)
7. Apple: Mac os x security configuration for mac os x version 10.6 snow leopard. [https://ssl.apple.com/support/security/guides/docs/\\$SnowLeopard_Security_Config_v10\\$6.pdf](https://ssl.apple.com/support/security/guides/docs/$SnowLeopard_Security_Config_v10$6.pdf) (2010). Accessed 19 July 2017
8. Blazakis, D.: The apple sandbox. BlackHat DC (2011)
9. Wardle, P.: Methods of malware persistence on mac os x. In: Proceedings of the Virus Bulletin Conference (2014)
10. Pajouh, H.H., Dehghantanha, A., Khayami, R., Choo, K.-K.R.: Intelligent OS X malware threat detection with code inspection. J. Comput. Virol. Hacking Tech. **14**(3), 213–223 (2018)
11. Lindorfer, M., Miller, B., Neugschwandtner, M., Platzer, C.: Take a bite-finding the worm in the apple. In: 2013 9th International Conference on Information, Communications and Signal Processing (ICICS), pp. 1–5. IEEE (2013)
12. Wardle, P.: Dylib hijacking on os x, CanSecWest 2015. <https://www.virusbulletin.com/virusbulletin/2015/03/dylib-hijacking-os-x> (2015). Accessed 20 July 2017
13. Case, A., Richard, G.G.: Advancing mac os x rootkit detection. Digit. Investig. **14**, 25–33 (2015)
14. Rentzsch, J.W.: Dynamically overriding mac os x. <http://rentzsch.com/papers/overridingMacOSX>; <https://papers.put.as/papers/macosx/2003/overridingMacOSX.pdf> (2003). Accessed 30 July 2017
15. Lindorfer, M., Kolbitsch, C., Milani Comparetti, P.: Detecting environment-sensitive malware. In: Recent Advances in Intrusion Detection, pp. 338–357. Springer, Berlin (2011)
16. Balzarotti, D., Cova, M., Karlberger, C., Kirda, E., Kruegel, C., Vigna, G.: Efficient detection of split personalities in malware. In: NDSS (2010)
17. Yonts, J.: Mac osx startup, Malicious Streams (2011)
18. Phuc, P.D.: What is safefinder/operator mac campaign?. <https://wp.me/p72HMD-b5> (2017). Accessed 30 Aug 2017
19. Phuc, P.D., Massacci, F.: Mac-a-mal: an automated framework for mac malware hunting. BlackHat Asia (2018)
20. Ming-chieh Pan, S.-t.T.: A mac os x rootkit uses the tricks you haven't known yet. BlackHat Asia (2014)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

¹² <https://github.com/phdphuc/mac-a-mal>.