OULUN AMMATTIKORKEAKOULU

Danila Timoshchenko

MALWARE ANALYSIS FOR ARM-BASED UNIX-LIKE SYSTEMS

MALWARE ANALYSIS FOR ARM-BASED UNIX-LIKE SYSTEMS

Danila Timoshchenko
Bachelor's Thesis
Spring 2024
Software Development Engineering
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Software Development Engineering

Author: Danila Timoshchenko
Title of the thesis: Malware Analysis for Arm-Based Unix-Like Systems
Thesis examiner(s): Teemu Korpela
Term and year of thesis completion: Spring 2024 Pages: 28

The goal of this thesis was to assess the current state of malware, developed and recompiled for ARM-based Linux and MacOS systems.

In the thesis, the threat of malware for newer ARM-based systems is reviewed for both Linux and MacOS. For analysis purposes, Ghidra was set-up using virtualised instance of MacOS as a secure environment using UTM.

The common measure of protection MacOS and Linux systems were mentioned, the threat of repurposed malicious code originally written for x86 architecture as well as Rosetta 2 being able to run x86 architecture malware.

To research the malware, the source code of Mirai botnet was reviewed as an example for Linux systems and their vulnerability to such attacks. For MacOS, the sample of GoSearch22 malware was analysed using Ghidra. With GoSearch22 analysis, the attention was drawn towards its anti-debugging behaviour as it utilised many techniques currently used to avoid research.

The thesis is finalised with the need for understanding low level programming, and obfuscation techniques of malicious binaries for proper malware analysis. Taking the nature of the analysed malware, some protection vectors of currently used systems are mentioned as well.

Keywords: Apple Silicon, Malware Analysis, Linux, Ghidra, Anti Debugging Logic.

CONTENTS

1 INTRODUCTION

Devices based on ARM architecture SoC are becoming more widespread on the market. With the introduction of Google's ChromeOS, Apple Silicon devices, and other ARM-based laptops and desktop computers, such devices are now directly competing with x86-64 based computers, with a positive trend of occupying more market share with each year.(1)

With this continuously expanding market-share of ARM-based computers, the first native malware sample for Apple silicon chip has been reported.(2) This included the concern of some of the antivirus engines having architectural specific signatures for known malware, making it harder to analyse ARM binaries. Many reports state that the most concerning threat for MacOS are potentially unwanted programs and adware, with malware accounting for only 1.5 percent of total detections in 2020.(3)

Unlike MacOS, Linux is heavily utilized as an operating system for servers and supercomputers, powering only 2 percent of desktops, making Linux systems a target for other types of malware compared to Apple's system.(4) Trojans and web shells pose a greater threat to Linux systems than potentially unwanted programmes and adware.(5), (6) This correlates with the claim that only 15 percent of Amazon AWS's cloud servers were ARM-based, and while ARM-based servers are growing in numbers, it is still difficult to compete with well-established x86 solutions, which could be the reason of slower distribution of Mirai family malware across ARM-based devices shown in some reports (7).

2 MALWARE PROTECTION MEASURES

2.1 Linux Protection Measures

Linux distributions come with package managers, which allow their users to install most programs via terminal, without the need of searching for the necessary software online. Most software distributed from such repositories is open source and any changes in the source code are reviewable by anyone. Such approach creates a community of trusted developers and proactive users, who take part in bug reporting the issues and contributing the development of the software for Linux systems.

Considering Linux being a relatively unpopular desktop solution, Canonical team claims there is no virus by definition in almost any known and updated Unix-like operating system (8). It is achieved by not running programs as a root in Linux and having less malicious actors actively developing malware for desktop Linux. However, Linux systems are not invincible to malware, and there are still ways of getting infected, rootkits and backdoors pose a great threat as well, being able to covertly infect the system and inspect files inside the system, leaking any sensitive data stored as plaintext.

Being a system with a wide choice of distributions with different software installed, there are no anti-virus software prebuilt in the system. However, there are ways of protecting the system such as installing various rootkit searching software, examples being *chkrootkit* and the *rootkit hunter project*. For other types of malware, *ClamAV* is recommended as an open-source anti-virus engine for Linux (9), (10), (12).

Some Linux distributions also provide extra security features, *Qubes OS* is a Linux distribution, which offers an isolation-based type of managing the system, where each application can be assigned to isolated virtual machine, that has no information accessible outside its allowed scope, including disposable virtual environments, capable of erasing any data left by the action of the program or the user, without any change to the system itself.

2.2 MacOS Protection Measures

Like Linux and Windows systems, MacOS allows its users to download and install applications in several ways:

- Downloading applications from Apple App Store.
- Downloading images of applications from 3rd party websites (13).
- Using package managers such as *Homebrew* or *Nix* (14), (15).

Such approach is different from iOS and iPad OS systems, which do not allow any applications installed outside of App Store (16). However, to comply with EU's Digital Markets Act, Apple might need to allow side-loading applications on their devices in near future (17), (18).

To ensure the safety of the system from malware, Apple uses 3 layers of defence, which consist of preventing the launch of the system with vetted App Store, blocking known malware from executing with Gatekeeper and Notarisation, and remediating the malware that has been executed with XProtect.(19)

Gatekeeper is responsible for quarantine of malicious software if such software is attempted to be executed. Its task is to verify if the software is notarized by Apple, otherwise, the application is prevented from executing and is assigned a quarantine extended attribute (20).

Notarisation is a process of submitting an application written for MacOS to review by Apple. This reduces the user warnings about launching the application and ensures the developer's compliance with Apple's policies and guidelines for distributing the software outside of App Store (21). Despite this, applications with no notarization can still be launched on MacOS devices after displaying the warning to the user.
XProtect acts proactively, scanning MacOS files every time the computer is idle or when an app has been changed or the signatures are updated (22). It acts as a rudimentary antivirus system that checks files for known malware based on signatures and other heuristics.

Additionally, there are several 3rd party solutions for active anti-malware software such as Avast Antivirus, Norton Antivirus or Kaspersky Antivirus for Mac (23), (24), (25).

3 UNIVERSAL AND ARM SPECIFIC MALWARE SAMPLES

3.1 Linux Samples

Most malware samples detected on Linux come in a form of binaries(5), to inspect if a binary files can be launched on linux for ARM, the 'file' command can be used in order to inspect the binary. This also works for libraries and any other file in the system, including images and text files.

```
~ file /bin/ls
/bin/ls: ELF 64-bit LSB pie executable, ARM aarch64, version 1 (SYSV), dynamically
linked, interpreter /lib/ld-linux-aarch64.so.1
```

Most of the binaries on Linux are ELF format binaries, they come in forms of x86, ARM and universal binaries. Using an ARM Linux distribution, it is possible to natively compile an aarch64 binary.

```
~ gcc -o hello_world_aarch64 hello_world.c
~ file hello_world_aarch64
hello_world_aarch64: ELF 64-bit LSB pie executable, ARM aarch64, version 1 (SYSV), dy-
namically linked, interpreter /lib/ld-linux-aarch64.so.1
```

Tools such as FatELF might allow creation of universal binaries for multiple architectures (26). However, such approach would require prior patching of the system where the binary is to be executed. The lack of patching of the target system requires the malware to be compiled and distributed with understanding of the final target, which may shield ARM based desktop Linux systems from already existing malware, compiled for x86.

Being used for various IoT applications, Linux for ARM shares the same threat both on desktop and IoT devices, therefore, most known malware for such devices can be labeled as Linux malware in general, sharing similar file structure and CPU architecture (27). Additionally, the need of targeting specific architecture and recompiling the code in order to target ARM devices makes open-source malware dangerous, as multiple malicious actors can target different types of devices, modifying the source code to suit their needs.

The example of such malware is Mirai family botnet, of which source code was publicly available. Mirai originally targeted IoT devices, infected devices then scanned the network for vulnerable targets and, using a list of most common root password combinations, acquired super-user priv-

ileges, spreading the botnet further. Being compiled for ARM devices, refactored version of Mirai might target ARM based desktop devices and gain control over the device and leak confidential data stored as plaintext. Such malware may infect desktop machines which have improper root usernames and passwords set up as well as poorly managed firewall rules for the system.

Additionally, as modern malware become more and more polymorphous, anti-virus software may struggle with performing static analysis and signature detection. Online anti-virus engines, both standalone or built into file exchange websites, have limitations for file sizes, the file limit can easily be reached with filling the end of the file with junk data (28). This would allow malware to avoid being sent for scanning via the anti-virus API and would further obfuscate the malicious software.

3.2 MacOS Samples

Unlike standard ELF binaries, Mach-O binaries can be compiled for both x86 and ARM-based MacOS systems, developers can compile their programs for both architectures, which allows malware to target all modern MacOS systems. Thus, malware developed for x86 does not need to be targeted to only one of the architectures as it can be compiled as a universal binary and be executed on either architecture (29).

One of the first malware samples found as a universal binary being compiled for Apple Silicon ARM system is GoSearch22 (SHA256 value: b94e5666d0afc1fa49923c7a7-faaa664f51f0581ec0192a08218d68fb079f3cf). Searching the malware databases and reports for malware, that could potentially run on newer Apple Silicon System on Chip lead to the sample. The filtering process included ‘arm’, ‘64’, ‘mach-o’ and ‘multi-arch’ tags. Using ‘file’ command again, GoSearch22 binary can be inspected to reassure it is compiled for ARM.

```
% shasum -a 256 GoSearch22
b94e5666d0afc1fa49923c7a7faaa664f51f0581ec0192a08218d68fb079f3cf  GoSearch22
% file GoSearch22
GoSearch22: Mach-O universal binary with 2 architectures: [x86_64:Mach-O 64-bit execut-
able x86_64] [arm64:Mach-O 64-bit executable arm64]
GoSearch22 (for architecture x86_64):Mach-O 64-bit executable x86_64
GoSearch22 (for architecture arm64):  Mach-O 64-bit executable arm64
```

At the moment, more binaries compiled for Apple Silicon can be found, searching the malware databases, including universal binaries for both x86 and Apple ARM architectures.

With introduction of Macintosh computers based on Apple Silicon arm chips, Apple provides its consumers with Rosetta 2 emulation tool, which allows to execute x86 programs using ARM-based devices (30). Rosetta 2 easies the transition from x86 to Apple Silicon ARM, but it can also be dangerous as it can aid in executing malware, developed for Intel based Macintosh computers.

Recent studies show that it is indeed possible to execute malware compiled for x86 on Apple Silicon devices with the help of Rosetta 2, more so, the malware written for x86 and running through Rosetta does not have to be signed in order to be executed. This may potentially create a greater threat to ARM macintosh devices as the absence of need of signing the malware removes one of the protection layers, reminding the user of potentially malicious capabilities of the program to be executed (31), (32).

4 ARM SPECIFIC MALWARE ANALYSIS TOOLS AND APPROACHES

4.1 Linux Malware Analysis

With the fast-growing market of IoT devices, Mirai has become a popular tool for malicious actors. However, due to its open code and structure, Mirai was easily available for evaluation and analysis.

Looking at Mirai botnet's structure we can see some of the key traits of this botnet on figure 1.

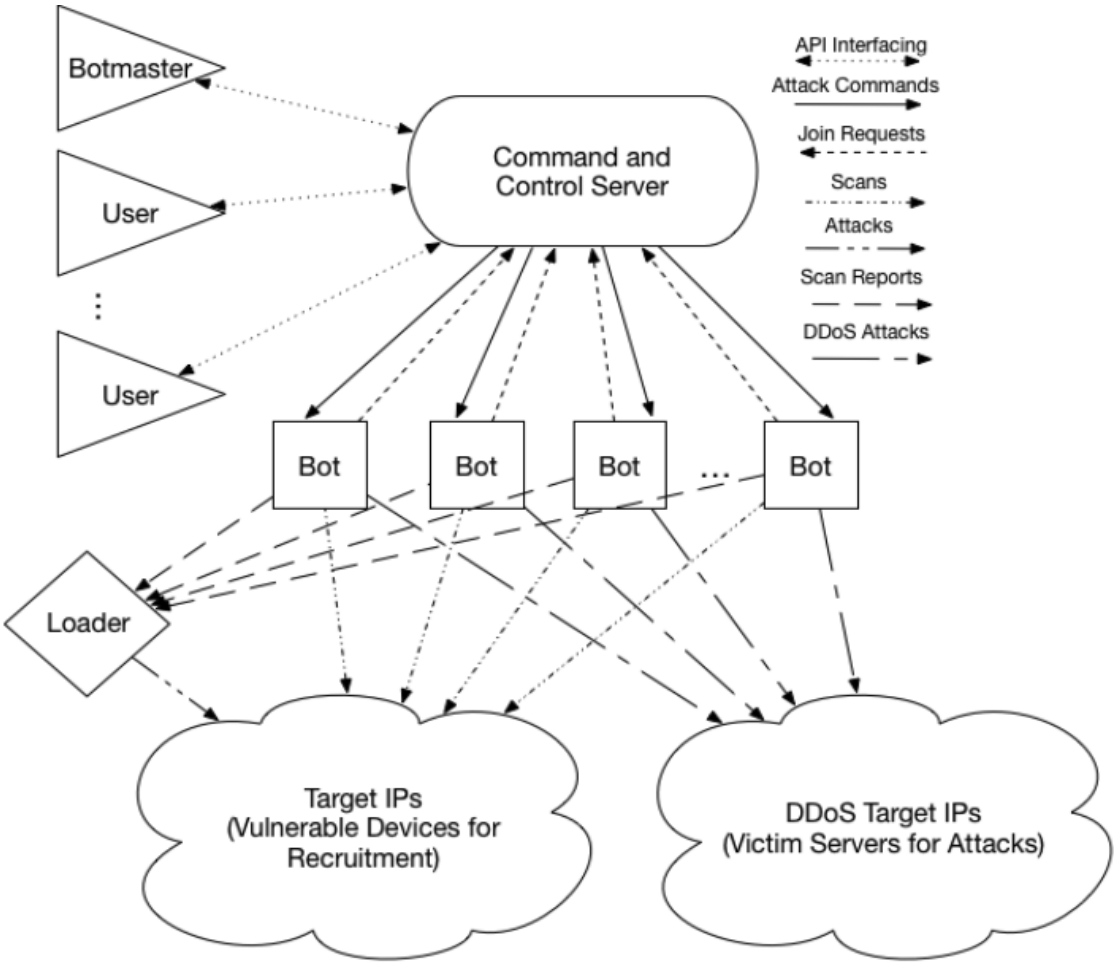

FIGURE 1. Mirai's botnet structure. Credit - Meghan Carole Riegel(37)

Instead of a more resilient structure of Peer-to-Peer bot network, Mirai uses Command and Control Server structure to command the network and issue orders to the botnet. Such structure is less persistent as if the C&C server is to be taken down by the law enforcement, the botnet will

end up being orphaned. To prevent the orphanage of the, the C&C server must be changed fairly often.

Infecting is performed with, firstly, scanning the IP addresses of possible IoT devices for open telnet ports and, secondly, attempts to gain root access on the device in order to install the malware. It utilizes hard-coded login-password pairs to attempt to gain root access, which can be found in mirai/bot/scanner.c file.

After establishing connection and gaining root access, the binary is downloaded via wget or tftp and installed. If neither wget or tftp are installed, the attacker loads the downloader binary with echo and installs the full binary after it. Once the binary is installed, the connection is closed and the newly recruited bot is starting to search for new hosts to attack and recruit new devices for the network (Figure 2).(37)

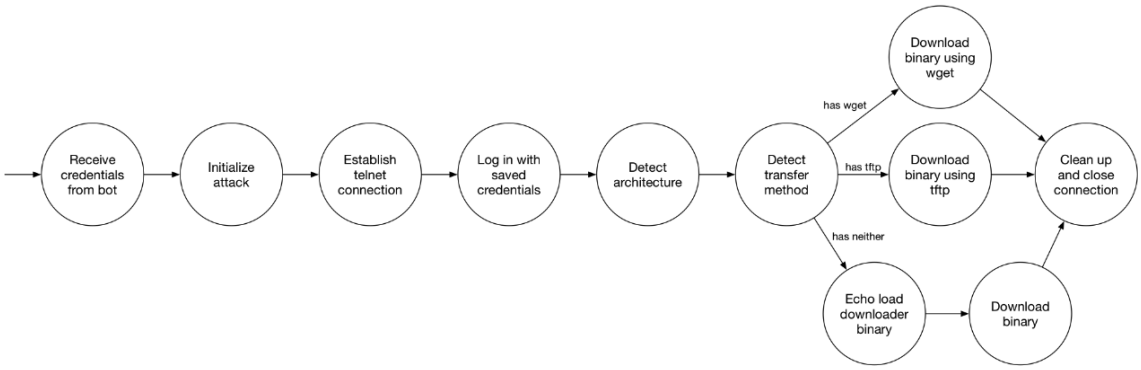

FIGURE 2. Sequence of attack of a bot in the network. Credit - Meghan Carole Riegel(37)

Mirai bots have been called 'territorial' for their never before seen behaviour of botnet killer process. It attempts to detect other malware processes on the infected device and kill them, trying to be the only malware running on the infected device. This behaviour can be seen, analysing the file mirai/bot/killer.c (38). (Figure 3)

```
511         m_qbot_report = table_retrieve_val(TABLE_MEM_QBOT, &m_qbot_len);
512         m_qbot_http = table_retrieve_val(TABLE_MEM_QBOT2, &m_qbot2_len);
513         m_qbot_dup = table_retrieve_val(TABLE_MEM_QBOT3, &m_qbot3_len);
514         m_upx_str = table_retrieve_val(TABLE_MEM_UPX, &m_upx_len);
515         m_zollard = table_retrieve_val(TABLE_MEM_ZOLLARD, &m_zollard_len);
516
517         while ((ret = read(fd, rdbuf, sizeof (rdbuf))) > 0)
518         {
519             if (mem_exists(rdbuf, ret, m_qbot_report, m_qbot_len) ||
520                 mem_exists(rdbuf, ret, m_qbot_http, m_qbot2_len) ||
521                 mem_exists(rdbuf, ret, m_qbot_dup, m_qbot3_len) ||
522                 mem_exists(rdbuf, ret, m_upx_str, m_upx_len) ||
523                 mem_exists(rdbuf, ret, m_zollard, m_zollard_len))
524             {
525                 found = TRUE;
526                 break;
527             }
528         }
```

FIGURE 3. mirai/bot/killer.c lines 511-528

After the competing malware processes are stopped, Mirai attempts to scan the network in order to gain access to new devices to infect.

Another trait of Mirai's scanning process is avoidance of some IP address ranges, for example, Mirai actively avoids the US Postal Service, as well as the US Department of Defence. Avoiding the unwanted attention towards the botnet may be the reason behind such behaviour, making the botnet less visible to the government law enforcements (figure 4) and potentially prolonging the window of opportunity for the botnet to act.

```
688         while (o1 == 127 ||                         // 127.0.0.0/8      - Loopback
689                (o1 == 0) ||                         // 0.0.0.0/8        - Invalid address space
690                (o1 == 3) ||                         // 3.0.0.0/8        - General Electric Company
691                (o1 == 15 || o1 == 16) ||            // 15.0.0.0/7       - Hewlett-Packard Company
692                (o1 == 56) ||                        // 56.0.0.0/8       - US Postal Service
693                (o1 == 10) ||                        // 10.0.0.0/8       - Internal network
694                (o1 == 192 && o2 == 168) ||          // 192.168.0.0/16   - Internal network
695                (o1 == 172 && o2 >= 16 && o2 < 32) ||// 172.16.0.0/14     - Internal network
696                (o1 == 100 && o2 >= 64 && o2 < 127) ||// 100.64.0.0/10    - IANA NAT reserved
697                (o1 == 169 && o2 > 254) ||           // 169.254.0.0/16   - IANA NAT reserved
698                (o1 == 198 && o2 >= 18 && o2 < 20) ||// 198.18.0.0/15    - IANA Special use
699                (o1 >= 224) ||                       // 224.*.*.*+       - Multicast
700                (o1 == 6 || o1 == 7 || o1 == 11 || o1 == 21 || o1 == 22 || o1 == 26 || o1 == 28 || o1 == 29 || o1 == 30 || o1 == 33 || o1 == 55 || o1 == 214 || o1 == 21
701         );
```

FIGURE 4. List of IP addresses Mirai actively avoids.

After infecting enough devices, C&C Server can issue an attack for a botnet, the botnet readies the attack, creating multiple threads and if another attack is being performed, stops it, this can be inspected in mirai/bot/attack.c:

```
44  void attack_kill_all(void)
45  {
46      int i;
47
48  #ifdef DEBUG
49      printf("[attack] Killing all ongoing attacks\n");
50  #endif
51
52      for (i = 0; i < ATTACK_CONCURRENT_MAX; i++)
53      {
54          if (attack_ongoing[i] != 0)
55              kill(attack_ongoing[i], 9);
56          attack_ongoing[i] = 0;
57      }
58
59  #ifdef MIRAI_TELNET
60      scanner_init();
61  #endif
62  }
```

FIGURE 5. mirai/bot/attack.h lines 44-62 attack initialisation.

In mirai/bot/attack.h This can be seen in the code (Figure 5), responsible for initialising the attack and adding placing the attack methods into a dictionary for later accessing by the bot. It then receives a target for the DDoS attack from the C&C as well as the needed flags for the attack, which the bot can then query from the dictionary (Figure 6).

```
20  struct attack_target {
21      struct sockaddr_in sock_addr;
22      ipv4_t addr;
23      uint8_t netmask;
24  };
25
26  struct attack_option {
27      char *val;
28      uint8_t key;
29  };
30
31  typedef void (*ATTACK_FUNC) (uint8_t, struct attack_target *, uint8_t, struct attack_option *);
32  typedef uint8_t ATTACK_VECTOR;
33
34  #define ATK_VEC_UDP        0  /* Straight up UDP flood */
35  #define ATK_VEC_VSE        1  /* Valve Source Engine query flood */
36  #define ATK_VEC_DNS        2  /* DNS water torture */
37  #define ATK_VEC_SYN        3  /* SYN flood with options */
38  #define ATK_VEC_ACK        4  /* ACK flood */
39  #define ATK_VEC_STOMP      5  /* ACK flood to bypass mitigation devices */
40  #define ATK_VEC_GREIP      6  /* GRE IP flood */
41  #define ATK_VEC_GREETH     7  /* GRE Ethernet flood */
42  //#define ATK_VEC_PROXY      8  /* Proxy knockback connection */
43  #define ATK_VEC_UDP_PLAIN  9  /* Plain UDP flood optimized for speed */
44  #define ATK_VEC_HTTP       10 /* HTTP layer 7 flood */
45
46  #define ATK_OPT_PAYLOAD_SIZE    0   // What should the size of the packet data be?
47  #define ATK_OPT_PAYLOAD_RAND    1   // Should we randomize the packet data contents?
48  #define ATK_OPT_IP_TOS          2   // tos field in IP header
49  #define ATK_OPT_IP_IDENT        3   // ident field in IP header
50  #define ATK_OPT_IP_TTL          4   // ttl field in IP header
51  #define ATK_OPT_IP_DF           5   // Dont-Fragment bit set
52  #define ATK_OPT_SPORT           6   // Should we force a source port? (0 = random)
```

FIGURE 6. Defining attack types.

The attacks themselves are not unique, compared to other botnets, however, they may pose a great threat threat because of the numbers of the bots in the network, as well as the obfuscated direction of the attack, with the bots being spread across the globe.

The structure of the Command and Control Server consists of an SQL database with client list, which consists of bots in the network, the API for controlling the network and an interface for users to log-in.

Figures 7-8 shows the initialisation of the ClientList structure in mirai/cnc/clientList.go file as well as bot structure for the database in mirai/cnc/bot.go file.

```
type ClientList struct {
    uid         int
    count       int
    clients     map[int]*Bot
    addQueue    chan *Bot
    delQueue    chan *Bot
    atkQueue    chan *AttackSend
    totalCount  chan int
    cntView     chan int
    distViewReq chan int
    distViewRes chan map[string]int
    cntMutex    *sync.Mutex
}

func NewClientList() *ClientList {
    c := &ClientList{0, 0, make(map[int]*Bot), make(chan *Bot, 128), make(chan *Bot, 128), make(chan *AttackSend), make(chan int, 64), make(chan int), make(ch
    go c.worker()
    go c.fastCountWorker()
    return c
}
```

FIGURE 7. Initialisation of ClientLists in SQL database.

```
type Bot struct {
    uid     int
    conn    net.Conn
    version byte
    source  string
}
```

FIGURE 8. Initialisation of bots in SQL database.

The additional code can be found in the same mirai/cnc/ directory, Mirai provides a wide variety of settings regarding access to its C&C server, the owner of the server can set-up additional accounts, choose how much DDoS time the accounts are allowed to perform, the number of bots, dedicated to the user and the cooldown between the attacks. With this in mind, Mirai was designed as a part of the system, that could provide DDoS attacks as a service. This continues the trend of subleasing the botnets for hire, which lowers the barrier to entry for a DDoS attackers (39).

Mirai was one of the malware samples, which further threatens the ARM-based devices. The malware thrives upon the networks with poorly or not configured firewalls and exploits the standard login – password pairs. The protection measures, required to avoid becoming a target of such exploits are extremely basic, however, with the wide spread of IoT devices, such measures as randomly generated passwords and properly set-up firewalls may be reasonable to enforce upon the manufacturers of said devices, to strengthen the poorly protected market of IoT devices.

4.2 Preparing Ghidra for MacOS Malware Analysis

It is important to prepare the secure environment for working with malware. The most straight-forward approach would be setting up emulated or virtual machines with QEMU (33). For setting up

a virtual environment under MacOS, UTM will be used, it is based on QEMU and tailored more towards working with Apple Silicon based MacOS systems (34). UTM then needs to be installed via brew package manager or directly from the official website.

After installing UTM, it will open with the welcome screen. The system can be virtualised or emulated, shown in figure 9:

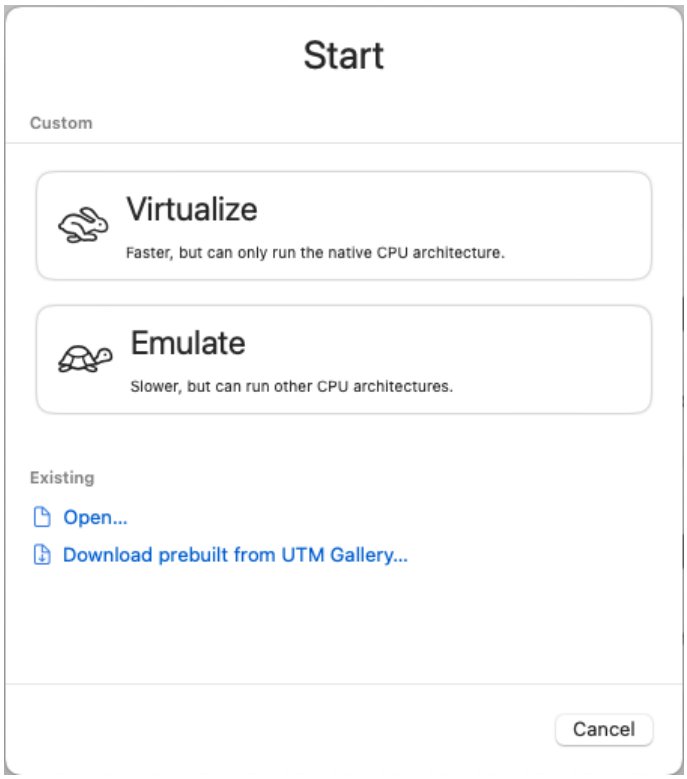

FIGURE 9. Creating UTM virtual environment.

Then it is possible to pick MacOS, it will be downloaded and installed as a virtual machine and appear in the most left column in UTM interface.

Such setup allows malware researchers to safely reproduce and observe malware's behaviour without causing any harm to the host system. It is possible to control what data is accessible by the virtual machine as well as restricting network access.

After initial setup of the system itself, brew package manager needs to be installed in the virtual environment as well (Figure 10):

```
% /bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

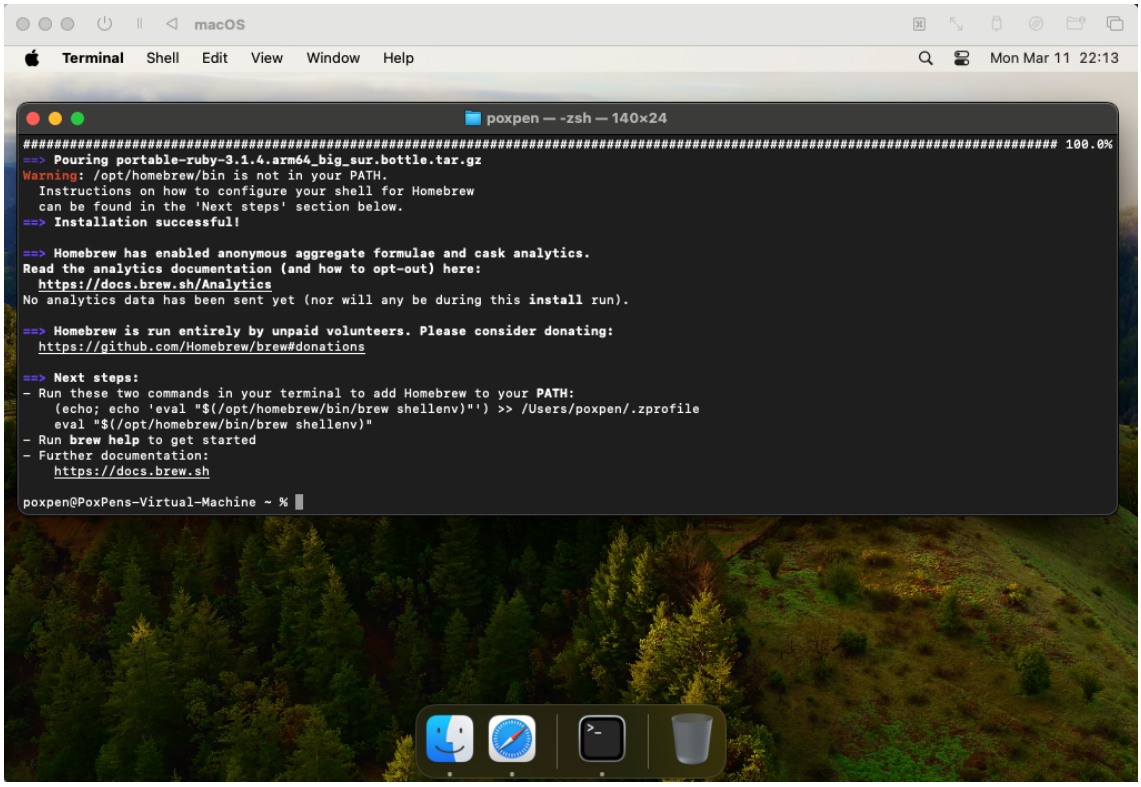

FIGURE 10. Installing brew package manager in UTM.

After it has been installed, it reminds to add Homebrew to system's PATH:

```
% (echo; echo 'eval "$(/opt/homebrew/bin/brew shellenv)"') >> /Users/User/.zprofile
% eval "$(/opt/homebrew/bin/brew shellenv)"
```

Brew can be then used to install other packages such as Ghidra and its dependencies.(35)

Ghidra is a powerful reverse-engineering tool, which allows dissecting the binary into assembly instructions. It is a crucial tool for the analysis as it allows breaking down the binary into readable instructions, which can be inspected then. Ghidra is an open-source tool, developed by NSA's Research Directorate. As an alternative, Hopper can be used as an alternative to Ghidra, however, it is a proprietary solution (41).

```
% brew install ghidra python3 gradle opendjk
```

Using brew, Ghidra, Python3, Gradle and OpenJDK respectively can be installed, which is seen in the command above. Then the variable can be exported, the versions of OpenJDK and Gradle

can be different depending on which version was installed, which can also be checked with HomeBrew.

```
export JAVA_HOME="/opt/homebrew/opt/openjdk"
export                GHIDRA_INSTALL_DIR="/opt/homebrew/Caskroom/ghidra/11.0.1-20240130/
ghidra_11.0.1_PUBLIC"
```

After exporting the variables, 'buildNatives' script needs to be launched in Ghidra, which will stop the gatekeeper from the denial of access to the decompiler.

```
% cd ${GHIDRA_INSTALL_DIR}/support
% ./buildNatives
```

Some extensions for Ghidra may require Python, for this it is essential to install Ghidraton extension(36). MacOS does not have Python 2 installed, according to Ghidraton documentation, it will switch to Python 3 interpreter when `ghidrathon_configure.py` is launched.

Ghidra now be used to create projects and start analysing the code of the malicious software via reverse-engineering.

4.3 MacOS Malware Analysis

GoSearch22 was one of the first discovered malware samples compiled natively for Apple Silicon, using Ghidra to create a project and start analysing its behaviour.

Distributed as a .app application file for MacOS, the victim had to grant permission to execute the program. The malware changes the search engine of the default web-browser of the victim, acting as an adware.

Infected devices would have their browser sessions compromised, having the search engine changed to GoSearch website. It could also leak sensitive data and could potentially push unwanted content and malicious code through the browser extension.

The focus of GoSearh22 analysis will be pointed towards its mechanisms of debugging avoidance. The sample actively tries to avoid being debugged and researched under controlled environment. The malware applies several logical checks and procedures in order to make sure if it is being debugged:

1. Invoking supervisor call (svc instruction)
2. Invoking sysctl API call
3. Check if the malware is running in a virtual environment
4. Check if SIP is disabled

Other malware samples have also implemented anti-debugging techniques, dealing with it is now a required practice for malware analysis(42). Instruction obfuscation techniques and adding junk instructions to the binary is also becoming a widely used practice to further harden the malware against dynamic analysis tools.

To analyse the logic of an svc instruction, a project in Ghidra needs to be created to open the malware sample inside the project. After uploading the file, it is possible to select the needed language and then check the file for any mistakes while uploading or any differences with SHA sums (Figure 11).

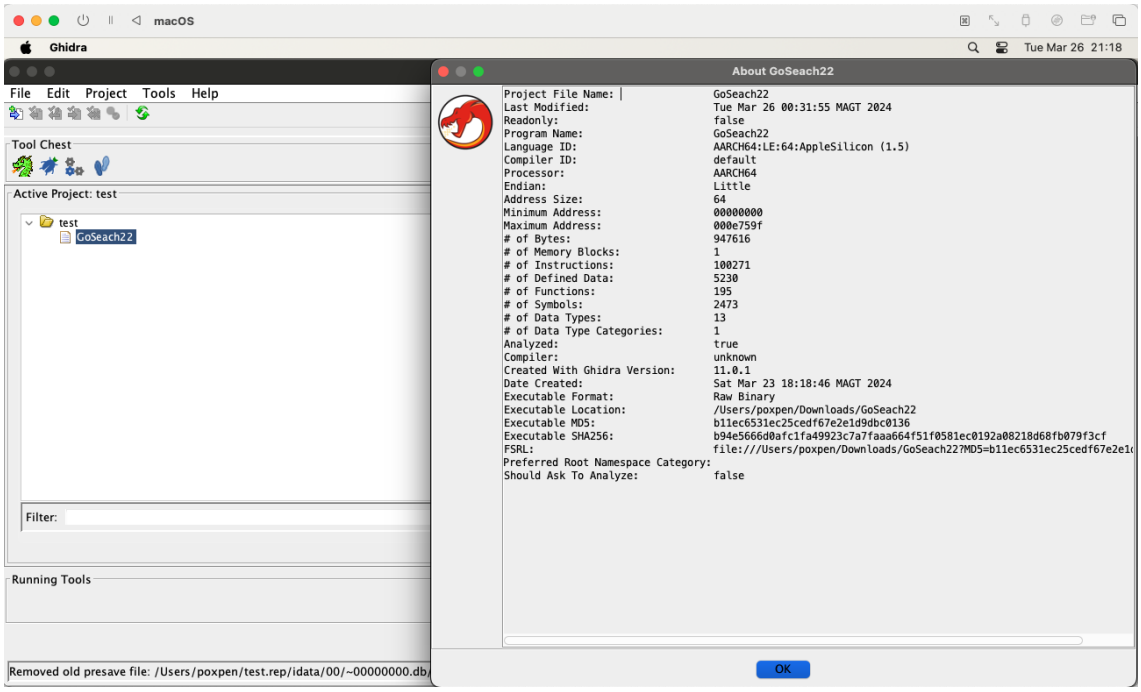

FIGURE 11. Creating and inspecting GoSearch22 project in Ghidra.

Using Ghidra's code browser, the malware can be analysed in search of the needed instruction. Searching for the instruction mnemonic, the first svc call at `00bc12c` can be found, shown on figure 12.

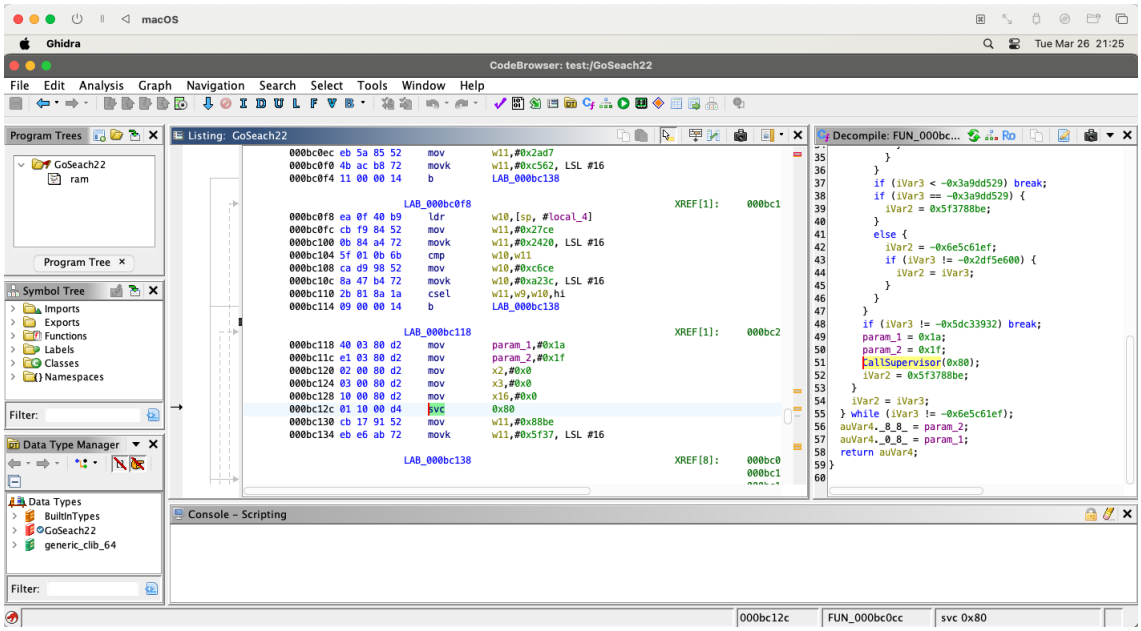

FIGURE 12. First svc interrupt call.

Another svc call can be found at address `000bc1fc`. (Figure 13)

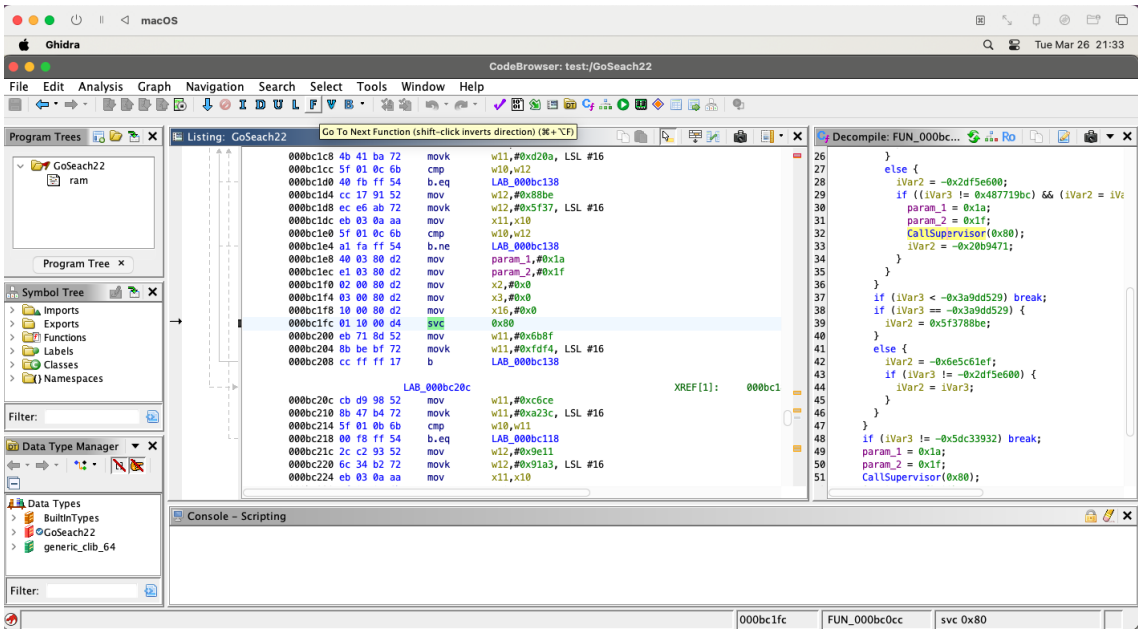

FIGURE 13. Second svc interrupt call.

To bypass such svc calls, it is possible to set a breakpoint at the instruction address in a debugging tool and then skip over the problematic call. The rest of the program will continue as if the supervisor call was never there in the first place. This concludes the anti-debugging logic of this malware sample.

However, GoSearch22 implements another tactic of making a call to sysctl API. The API can be used for various purposes, including the information about the state of the process. With this, the malware can be determined whether it is being debugged. Once again, it is invoked with the bl instruction. (Figure 14)

Figure 14. Sysctl API invoked with bl instruction.

This check can be avoided similarly to svc call, just skipping over the problematic instruction in the debugger. After locating the instruction, where the p_trace is attached, it is possible to skip over the instruction, leaving the register unchanged for later checks.

The malware sample also checks for various artifacts in the system, which can hint to the program that it is being run in an emulated environment. In the section below, the parsing process is displayed, which is used to determine whether the malware is being executed under the virtual environment.

```
/bin/sh -c -c,
readonly VM_LIST="VirtualBox\|Oracle\|VMware\|Parallels\|qemu";is_hwmodel_vm() { ! sysctl -n hw.model|grep "Mac">/dev/null;};is_ram_vm(){(($(($(sysctl -n hw.memsize)/1073741824))<4));};is_ped_vm(){ local -r ped=$(ioreg -rd1 -c IOPlatformExpertDevice);echo "${ped}"|grep -e "board-id" -e "product-name" -e "model"|grep -qi "${VM_LIST}"||echo"${ped}"|grep "manufacturer"|grep -v "Apple">/dev/null;};is_vendor_name_vm(){ ioreg -l|grep -e "Manufacturer" -e "Vendor Name"|grep -qi "${VM_LIST}";};is_hw_data_vm(){ system_profiler SPHardwareDataType 2>&1 /dev/null|grep -e "Model Identifier"|grep -qi "${VM_LIST}";};is_vm() { is_hwmodel_vm||is_ram_vm||is_ped_vm||is_vendor_name_vm||is_hw_data_vm;};main(){ is_vm&&echo 1||echo 0;};main "${@}
```

This can be avoided the same way, skipping over the problematic instruction. Another solution could be setting up the virtual environment in a way, that mimics the set-up on hardware, repeating the same entries as the system running on bare metal.

In order to debug malware under MacOS, it might be necessary to disable System Integrity Protection(40). This leaves the system vulnerable to malicious code, however, it is not an issue under a virtual environment. As malware researchers may have the SIP disabled, GoSearch22 attempts to execute the following:

```
-c command -v csrutil > /dev/null && csrutil status | grep -v "enabled" > /dev/null &&
echo 1 || echo 0
```

If the SIP is disabled, the command will echo 1, telling the malware to stop from running further.

The object can be printed out to examine the value of the register using lldb:

```
(lldb) po $x0
<NSConcreteTask: 0x1058306c0>
(lldb) x/s $x1
0x1e9fd4fae: "launch"
```

The object can be seen as an instance of an NSConcreteTask. With x/s it is possible to see the second argument, which is a launch method, which will execute the task and interrupt the program.

Concluding the techniques of GoSearch22 evading the analysis, the malware sample utilises several techniques, which are essential to understand for any malware analyst, this also highlights the need of lower level programming skills and ARM instructions.

5 CONCLUSION

Comparing Linux and MacOS systems on ARM, the distinction between desktop and IoT solutions is clearly seen, the transition of desktop solutions from x86 to ARM has not yet finished, but the malicious code has already been found compiled for desktop systems running on ARM, the translation layers such as Rosetta may also pose a threat(32).

Being used for various IoT applications, ARM has already become one of the leading architectures for such devices. Unfortunately, analysing Mirai source code and its impact on the DDoS threats, The vulnerability of these devices can be easily seen if left unprotected. Default login-password pairs set-up by manufacturers have created a significant dent in the whole class of devices, which are always online and surround us everywhere. The process of randomising the login-password pairs during the initial set-up or during the production will eliminate such a straight-forward exploit in these machines.

Setting up firewalls, sysadmins can leverage the sources of incoming packets, which a device or a network can receive. Restricting the use of the device to the local network will also protect the IoT device entirely from all the malicious activity coming from outside the network.

For desktop systems, the ways of protecting the device comes down to administrating the software, allowed for executing. As mentioned before, the ways of protecting the desktop systems come in a way of vetting the applications. However, GoSearch22 and Rosetta 2 showed us an example of malicious software being temporarily signed or allowed to run without signing from Apple.

Securing the desktop systems in business environment once again brings up a need for IT security consulting, sysadmins setting up and managing the user accounts in the working environment, and most importantly, teaching the employees cybersecurity fundamentals, to know how to secure their system and not to fall victim of a malicious actor.

The slow transition towards ARM architecture for desktop systems now brings a need for understanding both x86 and ARM instructions on lower levels of software development. The techniques of obfuscating the binary code is now widely used for both architectures, as well as the anti-de-

bugging methods. Better knowledge of system APIs as well as studying ARM assembly is now essential not only for software development engineers working on embedded systems, but also for malware analysts.

REFERENCES

1. Counterpoint Research, ‘Arm-based PCs to Nearly Double Market Share by 2027’. Search date 1.2.2024 https://www.counterpointresearch.com/insights/arm-based-pcs-to-nearly-double-market-share-by-2027
2. Patrick Wardle, VB2021 Localhost Conference, ‘ARM’d & dangerous an introduction to analysing arm64 malware targeting macOS’ October 8, 2021. Search date 1.2.2024 https://vblocalhost.com/uploads/VB2021-Wardle.pdf
3. Malwarebytes.com, ‘2021 State of Malware Report’. Search date 1.2.2024 https://www.malwarebytes.com/wp-content/uploads/sites/2/2023/09/mwb_stateofmalwarereport2021.pdf
4. Truelist.co, ‘Linux Statistics – 2023’, January 9, 2023. Search date 1.2.2024 https://truelist.co/blog/linux-statistics/
5. AV Atlas, ‘Total amount of malware and PUA under Linux’. Search date 1.2.2024 https://portal.av-atlas.org/malware/statistics
6. Trend Micro, ‘The Linux Threat Landscape Report’. Search date 1.2.2024 https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/the-linux-threat-landscape-report
7. Crowdstrike.com, ‘Mirai Malware Variants for Linux Double Down on Stronger Chips in Q1 2022’. Search date 1.2.2024 https://www.crowdstrike.com/blog/linux-mirai-malware-double-on-stronger-chips/
8. Ubuntu Help Page, ‘So You Want to Know How to Use Anti-virus Software on Ubuntu?’. Search date 17.2.2024
9. Chkrootkit.org Main Page. Search date 17.2.2024 https://www.chkrootkit.org/
10. The Rootkit Hunter Project. Search date 17.2.2024 https://rkhunter.sourceforge.net/
11. ClamAV Main Page. Search date 17.2.2024 https://www.clamav.net/
12. Red Hat Blog, ‘3 antimalware solutions for Linux systems’. Search date 17.2.2024
13. Apple.com, ‘Install and uninstall apps from the internet or a disc on Mac’. Search date 13.2.2024 https://support.apple.com/guide/mac-help/install-and-uninstall-other-apps-mh35835/mac
14. Brew.sh, ‘Homebrew The Missing Package Manager for macOS (or Linux)’. Search date 13.2.2024 https://brew.sh/

15. Nixos.org, ‘Nix: the package manager’. Search date 13.2.2024 https://nixos.org/download.html#nix-install-macos
16. Apple Discussions, ‘Run DMG file in iPad Air 5’. Search date 13.2.2024 https://discussions.apple.com/thread/253921700?sortBy=best
17. European Commission, ‘About the Digital Markets Act’. Search date 13.2.2024 https://digital-markets-act.ec.europa.eu/about-dma_en
18. Bloomberg Blog, ‘Apple to Allow Outside App Stores in Overhaul Spurred by EU Laws’. Search date 13.2.2024 https://www.bloomberg.com/news/articles/2022-12-13/will-apple-allow-users-to-install-third-party-app-stores-sideload-in-europe
19. Apple Support, ‘Protecting against malware in macOS’. Search date 16.2.2024 https://support.apple.com/guide/security/protecting-against-malware-sec469d47bd8/web
20. Apple Support, ‘Gatekeeper and runtime protection in macOS’. Search date 16.2.2024 https://support.apple.com/guide/security/gatekeeper-and-runtime-protection-sec5599b66df/1/web/1
21. Apple Developer Documentation, ‘Notarizing macOS software before distribution’. Search date 16.2.2024 https://developer.apple.com/documentation/security/notarizing_macos_software_before_distribution
22. Appleinsider.com Blog, ‘Apple XProtect is now proactive with periodic malware scans’. Search date 16.2.2024 https://appleinsider.com/articles/22/08/31/apple-xprotect-is-now-proactive-with-periodic-malware-scans
23. Avast.com Download Page. Search date 17.2.2024 https://www.avast.com/index#mac
24. Norton.com Download Page. Search date 17.2.2024 https://us.norton.com/products/norton-360-antivirus-plus#
25. Kaspersky.com Download Page. Search date 17.2.2024 https://www.kaspersky.com/mac-antivirus
26. FatELF Web Page, ‘FatELF: Universal Binaries for Linux’. Search date 20.2.2024 https://icculus.org/fatelf/
27. ‘A Survey on Cross-Architectural IoT Malware Threat Hunting’. Search date 22.2.2024 https://arxiv.org/abs/2306.07989
28. Virus Total Documentation, ‘Upload a file’. Search date 22.2.2024 https://docs.virustotal.com/reference/files-scan

29. Apple Developer Documentation, ‘Building a Universal macOS Binary‘. Search date 23.2.2024 https://developer.apple.com/documentation/apple-silicon/building-a-universal-macos-binary
30. Apple Support Page, ‘If you need to install Rosetta on your Mac’. Search date 19.2.2024 https://support.apple.com/en-us/HT211861
31. Apple Support Page, ‘Rosetta 2 on a Mac with Apple silicon’. Search date 28.2.2024 https://support.apple.com/guide/security/rosetta-2-on-a-mac-with-apple-silicon-secebb113be1/web
32. Science Direct, ‘Assessing the threat of Rosetta 2 on Apple Silicon devices’. Search date 28.2.2024 https://www.sciencedirect.com/science/article/pii/S2666281723001300?ref=cra_js_challenge&fr=RR-1
33. QEMU website. Search date 11.3.2024 https://www.qemu.org/
34. UTM website, ‘Securely run operating systems on your Mac’. Search date 11.3.2024 https://mac.getutm.app/
35. Ghidra repository. Search date 18.3.2024 https://github.com/NationalSecurityAgency/ghidra
36. Ghidraton repository. Search date 18.3.2024 https://github.com/mandiant/Ghidrathon
37. Meghan Carole Riegel. ‘TRACKING MIRAI: AN IN-DEPTH ANALYSIS OF AN IOT BOT-NET’. Search date 01.04.2024 https://etda.libraries.psu.edu/files/final_submissions/15003
38. Mirai Source Code Repository. Search date 29.3.2024 https://github.com/jgamblin/Mirai-Source-Code
39. Imperva blog, ‘Booters, Stressers and DDoSers’. Search date 01.04.2024 https://www.imperva.com/learn/ddos/booters-stressers-ddosers
40. Apple Developer Documentation, ‘Disabling and Enabling System Integrity Protection’. Search date 02.04.2024 https://developer.apple.com/documentation/security/disabling_and_enabling_system_integrity_protection
41. Hopper Front Page. Search date 04.04.2024 https://www.hopperapp.com/
42. SentinelLabs Blog, ‘Defeating macOS Malware Anti-Analysis Tricks with Radare2’. Search Date 04.04.2024 https://www.sentinelone.com/labs/defeating-macos-malware-anti-analysis-tricks-with-radare2/