A System Emulation for Malware Detection in Routers

Tran Nghi Phu, Ngo Quoc Dung, Le Van Hoang, Nguyen Dai Tho, Nguyen Ngoc Binh

Abstract: Nowadays, there are many discussions on the fourth industrial revolution with a combination of real physical and virtual systems (Cyber Physical Systems), Internet of Things (IoT) and Internet of Services (IoS). Along with this revolution is the rapid development of malicious code on IoT devices, leading to not only the risk of personal privacy information leaking but also the risk of network security in general. In this paper, we propose C500-toolkit, a novel tool for malware detection in Commercial-off-the-shelf routers, based on dynamic analysis approach. The main contribution of C500-toolkit is to provide an environment for fully emulating router firmware image including both operating system and web-interface. To show the advantage of C500-toolkit, experiments of this tool with embedded malwares Linux/TheMoon and Linux/Mirai are presented.

Keywords: IoT security, Router security, firmware, C500-toolkit.

I. INTRODUCTION

The term Internet of Things (IoT) has become popular in recent years and it is the main core of the forth industrial revolution. Kevin Ashton defines “Internet of Things” as an integration of sensors and embedded controllers inside devices that are connected through the Internet, wire or wireless [1]. In this trend, object and human are given an identification and able to exchange related data, information each other without the intervention of human or computer. IoT devices such as smartTVs, router, ipCamera, lamps, microwave oven and so on are becoming more and more intelligent. These smart things can communicate with each other to exchange data to perform their intended functionality. According to Cisco’s report [2], 50 billions IoT devices will connect to Internet by 2020 around the world opening significant opportunities for a large number of new applications that promise to improve the quality of our lives. Because smart things can receive privacy-sensitive information from their sensors such as user’s location, user’s activities, with whom this user is talking, or carry out a safety critical function such as actuators that lock the front door, errors in the firmware of devices, whether present result in an accidental mistake or purposeful malice, can have serious and varying implications in both the digital and physical world [3]. Therefore, in parallel with the development of IoT technology, there are many security problems such as disruption to operation, information leaking or, in some scenarios, even loss of lives [4] when some smart thing can become a spying device to collect information and interact with us anytime. Recent recorded attacks show that these scenarios become more and more critic. In September 2016, an IoT botnet built from the Linux/Mirai malware was responsible for a 600 Gbps DDoS attack, perhaps the largest botnet on record, targeting Brian Krebs’s security blog [5]. Another case that could be mentioned is Weeping Angel [6], one program used Samsung smart-televisions as secret listening devices. According to the Wikileaks news, even when it appears to be turned off, the television “operates as a bug, recording conversations in the room and sending them over the Internet to a covert C.I.A. server”. There exist a lot of vulnerabilities that attackers can use for getting privileges of IoT devices and OWASP has identified top ten issues [7] in which insecure firmware, insecure web interface and insufficient authentication are mentioned. These issues have been attracting much attention from researchers who are dealing with IoT network devices. Inside IoT network, network devices such as router, switch, IoT gateway play a important role in transmitting data between devices. To secure these IoT devices, especially routers, works presented in [8, 9, 10, 11, 12, 13] allow identifying vulnerabilities/malware in commercial off-the-shelf (COST) network devices. There are two main classes in firmware analysis: static and dynamic approach. Static approach, consists of analyzing and evaluating without executing them, uses techniques such as Data Flow Graph (DFG), Control Flow Graph (CFG), Symbolic Execution (SE) to analyze every single file found in firmware and identify malware characteristics [13] such as Printable-Strings-Information (PSI), bytecode, headers, system-calls, API etc. For example, Drew et al. [9] perform static analysis on embedded systems with FIE based on symbolic execution engine to detect bugs in firmware for the popular micro-controllers MSP430 family. This method is precise for a small range of open source firmware programs of 16-bits RISC processors, but it can not be applied to large-scale analysis for IoT devices in general. To deal with this problem of scalability, Shoshitaishvili et al. [3] presented Firmalice, Costin et al. [14] have built an automated framework for analysing router’s firmware images. Firmalice is a static analysis framework built on top of a symbolic execution engine to identify authentication bypass flaw that attackers could determine the required inputs to execute privileged operations. This framework could analyze firmware on the binary level, in a scalable manner and with no requirement to instrument code on the original device. While Costin’s framework scales to thousands of firmware images but it is

Revised Manuscript Received on September 03, 2019
* Correspondence Author
Tran Nghi Phu*, VNU University of Engineering and TechnologyInstitute of Informatics, Hanoi, Vietnam.

Published By: Blue Eyes Intelligence Engineering & Sciences Publication

DOI: 10.35940/ijitee.J9909.0981119
very generic and produces a large number of false positives. The main drawback of these methods is it cannot be used for malware detection purpose and firmwares containing complex obfuscation code. Therefore, using only the static analysis approach could not be sufficient to identify malware and should be combined with dynamic analysis [4].

Dynamic approach consists of monitoring the whole device during its run-time to detect abnormal behaviors. The principle of this method is to define a set of rules that are considered normal behavior to determine whether an observed device signals intentionally violates these predefined rules. To perform this approach, the most important parts were to build a virtual emulator for running firmware. In such environment, malware can only affect the virtual PC and not the physical one. In general, researchers used QEMU [15], a very popular open-source system emulator, to deal with emulator challenges. Indeed, QEMU supports many types of architecture processor such as Intel ATOM, X86, ARM, MIPS, PowerPC that are widely used in IoT devices.

In this perspective, Jonas et al. have built the Avatar framework to analyze embedded devices by orchestrating the execution of a QEMU based emulator and hardware of that embedded device [10]. To analyze a firmware, Avatar connects to the target device through the communication channel or the debugging link such as Jtag and Uart to the analysis model, in which QEMU was used for emulating its CPU. Then, every CPU operations will be observed and analyzed to detect abnormal behaviors. The offloading execution of firmware to actual hardware reduces the gap between the simulation and the reality. This point gives Avatar the possibility to get real CPU operation signals during its run-time that a native QEMU based emulator cannot do. However, a lot of low-cost embedded devices do not have standard debugging or communication channels as required. Besides, the data transmission between the actual device and the analysis module through Jtag or Uart is slow. Therefore, using Avatar for real-time malware detection on IoT devices is impossible.

Sandboxes are used as a core of IoT malware analysis and detection frameworks such as [16, 17] aiming at extracting behaviors of a targeted file during its run-time. However, these frameworks exist flaws that limit the malware detection capability. IoTPoT [17] has built a sandbox to capture and analyze Telnet behaviors of IoT malware that are used for DDoS attacks. This approach could be useful for detecting network abnormal behaviors, but can’t detect malware that behave mostly inside the operating system of the device such as Linux/TheMoon [18]. Rare [19] focused on how to activate malware on Router by discovering static and dynamic information to build a suitable environment for malware and Chang et al. [16] proposed IoT sandbox which can support 9 kinds of CPU architectures including ARM (EL, HF), MIPS, MIPS, POWERPC, SPARC, x86, x64 and sh4. But they only used OpenWrt to build emulated router, didn’t emulate NVRam and didn’t mention to detect malware on router.

Current sandboxes have been mostly built on basic environments including common Linux-based operating system, additional tools without the IoT devices specificity. This point has a strong impact to capture behaviors of a target executable file and the whole detection process afterwards. But with the firmware, it is built and packaged for specialized devices, with specialized functions in many different environments that don’t have much in common. Therefore, many programs that cannot be executed on a standard environment, we also can’t install additional required environments but need to create an environment like real firmware, which means that the sandbox environment needs to be very diverse and like the firmware Real equipment. That means, with IoT sandbox we not only create an environment but have to create a lot of IoT sandbox environment, each sandbox has a close environment with real devices. The installation of these environments is not feasible due to the complexity, and they themselves are packaged in unpublished firmware. To our knowledge, there is not any sandbox currently that has the ability to build an environment based on the actual device firmware, able to simulate the device’s NVRam and gather enough syscall to serve the malware detection.

Another well-known method to deal with vulnerabilities detection purpose in large-scale is Firmadyne [12] proposed by Chen et al.. Firmadyne aims at emulating router firmware’s web-interface using QEMU. Firmadyne is able to auto-configure a suitable emulation environment for a wide range of routers that allows to performing dynamic analysis of 23,035 firmware images gathered from 42 device vendors. This method does not rely on physical hardware to perform the analysis like Avatar but Firmadyne emulate perfectly the firmware non-volatile memory to execute the firmware web-interface. Once the router web-interface is emulated, the popular scanning framework Metasploit is used for exploring vulnerabilities and its corresponding exploits. Firmadyne is the best tool available currently that can emulate the actual device firmware with simulating NVRam. However, Firmadyne analyzes only the web interfaces by scanning from outside and ignore behaviors of the firmware operating system. Hence, these methods can not collect abnormal behaviors to detect malware and shows in the analysis of Linux/TheMoon and Linux/Mirai malware experiments.

In some cases, Chen said that their NVRam emulation implementation didn’t work for all firmware images, and our test experiment shows that some of whom will run with NVRam information extracted from the real device, not emulated NVRam. In addition, COTS embedded devices, which directly extract from router devices, are usually controlled by vendors and firmware images could be different with those showcased on their websites.

In general, the considered methods have two drawbacks: Firstly, they focus on analyzing only standard Linux image or router firmware images that are downloaded from vendor websites and not dealing with those that are embedded the actual routers, so many firmwares still couldn’t operate. Secondly, the emulated firmware environment don’t enough a suitable environment for malware activate it’s behavior and don’t have tools to allow monitoring system behavior of malware. To overcome the shortcomings of previous works, a new tool is proposed in this paper to emulate actual router firmware images to perform the vulnerabilities and/or malware detection at the same time. This method is relied on a toolkit, named C500-Toolkit, with the capabilities of:

- The C500-toolkit is able to extract firmware image from actual router’s FLASH chip.
- The C500-toolkit is able to crawl firmware images from router vendor’s websites.
• The C500-toolkit automatically standardize router’s firmware image as required to fully emulate it including the operating system and the web interface.
• The C500-toolkit can insert monitoring tools such as Strace, TcpDump to collect data during the emulation of router.
• The C500-toolkit is able to perform dynamic analysis to detect malware and vulnerabilities in router’s firmware image.

II. ROUTER AND Firmware STRUCTURE

This section presents the router, the firmware image structure, and the challenges that we faced when building C500-toolkit.

The router is a network device in layer 3 of the Network Layer model OSI with main components presented in Fig. 1:

<table>
<thead>
<tr>
<th>Firmware</th>
<th>CPU</th>
<th>RAM</th>
<th>NVRAM</th>
<th>FLASH</th>
<th>ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig 1: Router structure

• CPU: is the heart of router that executes every instructions given by the operating system.
• ROM: is the Read-Only-Memory contains boot strap programming and basis software for starting up the router.
• RAM: is Random-Access-Memory and similar to the RAM used in traditional computer or PC, this volatile memory is lost all stored data when the router is shutdown or restart.
• FLASH: is a None-Volatile Memory where the firmware image is held, this component can be erasable/writable and keep data when the power is lost.
• NVRAM (None-Volatile RAM): has the same functionality as the FLASH but with less storage capacity. NVRAM contains configuration files of the device to ensure that when booting up, the default configurations of the router, including network and system configurations, will be automatically loaded correctly into the storing states. This component could be included in the FLASH.
• Interfaces: are physical hardware that ensure the transmission of packets come in and come out the router.

The composition of router is similar to PC except the architecture of CPU, about 86% of the routers use MIPS, ARM as in-house architecture because it was originally simpler and cheaper. Thanks to QEMU, a full-system emulation for any supported architecture including MIPS, ARM, X86, SPARC etc that allows creating an emulator for router. A router has the same components as a computer, except NVRam and the component is also not emulated in QEMU, it only updated by Firmadyne. It means origin QEMU couldn’t emulate router like PC. But Firmadyne only use emulated NVRam information, it can’t use information from actual NVRam of router devices.

The structure of firmware image may vary, depending on the function and the design of each manufacturer. Firmware image can be divided into the following types:

• Full-blown (full OS/kernel + boot-loader + libs + app): this is a complete firmware that contains a minimalist operating system. In this type, user-mode and kernel-mode are separated; applications run in user-mode allowing inserting custom tools for vulnerabilities and malware detection purpose.
• Integrated (apps + OS-as-a-lib): this is an incomplete firmware. Functions, operating system are built as a library and there is only kernel-mode in this type.
• Partial updates (apps or libs or resources or support): this type of firmware contains only files that used to update the firmware.

In this paper, firmware images of the first type (full-blown) are taken into account for the vulnerabilities and malware detection purpose. This choice is made because full-blown firmware images contain required components such as bootloader, linux kernel, file-system image, user-land binaries, web-interface and support files for the router emulation. These firmware images can be extracted from two sources, the first one is from vendor websites and the second one is to extract them from actual router devices. The first source is widely used by researchers such as Costin [14] or Chen [12]. The second source is rarely used in previous works even it has an important impact to determine if the embedded firmware image on actual routers contain malicious code or/and vulnerabilities.

To deal with the discussed challenges in the previous section, C500-Toolkit was built with the objectives of extracting, emulating and analyzing actual router firmware images, especially Linux-based firmwares. To showcase the capabilities of C500-toolkit, a real router (Tp-Link Wr842nd with the version of the FLASH chip is Winbond W25Q64FV) and malware (Linux/TheMoon and Linux/Mirai) were used for our experiments. To summarize, the contributions of this proposed toolkit are:

• The C500-toolkit has a module, named C500-Extractor, for extracting firmware image from an actual router device.
• The C500-toolkit has a module, named C500-Standardization, for completing the missing parts into a firmware image to get a pre-emulated image that can be performed by QEMU.
• The C500-toolkit has C500-Detector module, for detecting malware and vulnerabilities in a target firmware image.
III. C500-TOOLKIT

This section describes the C500-toolkit main components and their functions. The overview of this toolkit is shown in Fig. 2.

A. C500-Extractor

The extraction of firmware images on SOHO routers (Small Office - Home Office) can be performed through several methods such as using the device’s backup function, using the serial console port or the Jtag debug channel. However, in some cases these extraction methods may not be as effective as desired because the manufacturers remove these ports on the printed circuit board, therefore do not allow users to interfere with the device. In this case, extracting the firmware image directly from the FLASH chip is necessary, which is also the goal of the C500-Extractor.

The C500-Extractor is a hardware module and it is designed to read/write data from/to the FLASH chip containing the firmware image. The first prototype version was built to deal with popular FLASH chips used on Tp-Link and Linksys routers that are:

- The Winbond FLASH chips W25Q32FV, W25Q64FV, W25Q128FV
- The Fidelix FLASH chip FM25Q64
- The Macronix FLASH chip MX25L1606E
- The Eon FLASH chip EN29LV320B-70TC

To read/write data from these FLASH chips, two main interfaces were used are the SPI (Serial Peripheral Interface) [20] and the FSMC (Flexible Static Memory Controller) [21]. The FSMC port is employed for the Eon FLASH chip extraction, while the SPI port is applied to the Winbond, Fidelix, Macronix FLASH chips. This hardware module is controlled by a micro-controller STM32F10 family and support both communication channel SPI and FSMC. This module is shown in Fig. 3. To start the firmware image extraction process, the target FLASH chip is put into a corresponding compartment. The extractor would read the ID code of that FLASH chip to detect which one is put into, then it adjusts the appropriate configurations to extract data to get a binary file. The extracted binary file is then transferred to a computer through the micro USB connection. The extractor would read the ID code of that FLASH chip to detect which one is put into, then it adjusts the appropriate configurations to extract data to get a binary file. The extracted binary file is then transferred to a computer through the micro USB connection. This binary file usually contains NVRAM, Art, Kernel vmlinux, Bootloader, Rootfs in which three parts kernel vmlinux, Rootfs and NVRAM have an important role. The first part indicates the Linux kernel version used by the operating system that is found in the Rootfs. And the NVRAM indicates the required configurations for executing the router correctly, especially in user-mode. Once a firmware image is extracted, C500-toolkit standardizes this one by completing missing data, in using the extracted parts, to emulate it by QEMU.

B. C500-Standardization

The standardization of the firmware image is performed by two steps:

- Extraction and identification of hardware architecture to replace Busybox, named C500-Reverse.
- Addition of missing data including NVRAM, Linux kernel and Busybox, to get a pre-emulated image for QEMU.

First, C500-Reverse is called to identify the CPU architecture, its endianness, the kernel version and then, unzip the extracted firmware image to get file-system directories. There were several tools that are similar to C500-Reverse such as Firmware-mod-kit (FMK) [22], Binwalk [23] or the Extractor module of Firmadyne [12]. However, these modules have non-negligible drawbacks that were presented in [24]. As a result, C500-Reverse was built in using the most popular extraction tools for file-system formats such as Jefferson, Sasquatch, CRAMfs, Yaffs. These formats represent 97% of 13,275 tested firmware images that were crawled from router vendor websites.

The most popular file-system formats, based on...
13,275 tested firmware images, are shown in Fig. 4.

**Fig 4: C500-Extractor Component**

C500-Reverse module was built in modular manner allowing to be updated frequently. The details of this component is presented in [24] and the file-system directories of Tp-Link Wr842nd are shown in Fig. 5.

**Fig 5: Root file-system directories**

Along with the familiar directories found in linux root file system such as etc, bin, lib, we can find the router web-interface in www, cli directory. At this stage, the module C500-Addition starts to build an empty image format raw, then copy required data progressively into it to make a pre-emulated firmware. This process is similar to Firmadyne in order to emulate the web-interface [12] except two points:

- The initialization of the NVRAM
- The addition of tools for monitoring the network and the operating system behavior

For the first point, Chen et al. [12] have used a shared library, named libNvram.so to persist device-specific configuration parameters, for tested firmware images. Because of using a generic library for all of router firmware images, the firmware image emulation successful rate is quite low (4,992 firmware images out of 9,486). To overcome this shortcoming, C500-Addition module uses the extracted NVRAM of Tp-Link Wr842nd as shown in Fig. 6 in which we can retrieve information such as:

- The device Mac address is: 00:23:69:11:A6:D0
- The LAN interfaces are: vlan0, eth1, eth2, eth3
- The default Ip address is: 192.168.1.1
- The firmware version is: v4.21.5

The second point is not trivial to resolve. First, manufacturers always search to optimize the storage, therefore they have customized their firmware image by removing unnecessary packages. Hence, Busybox, a popular tool in embedded systems, combines tiny versions of many common UNIX utilities into a single small-size program [25], thought it is necessary to determine if the employed Busybox version has enough utilities for desired monitoring tools. In this paper, a customized Strace version and a customized Tcpdump are used for the vulnerabilities and/or malware detection purpose. Strace is a diagnostic, debugging and instructional user-space utility for a specific program in Linux, while Tcpdump is used for analyzing network traffic data. These tools are popularly used in malware analysis, based on abnormal behavior analysis. Thanks to Landley [25] with different pre-built Busybox versions, we can customize them to get more utilities. For each Linux kernel version, a suitable Busybox version is taken to replace the one found in bin directory, a full utilities system is then achieved as showing in Fig. 7. Second, about 93% of COTS router firmware images operate with Linux kernel versions 2.6.x. The difference between Linux kernel versions 2.6.x is not significant, therefore make a generic pre-built Linux kernel version 2.6.34 is the adopted solution for C500-Addition module.
A System Emulation for Malware Detection in Routers

5. We emulate these two couple images and two analysis are then performed:

- Using Metasploit to scan the emulated images to check if there are more vulnerabilities during Linux/TheMoon and Linux/Mirai execution.
- Using Strace and Tcpdump to scan two couple images and log every system-calls and network behaviors.

6. Based on obtained results from Strace and Tcpdump, C500-Detector module perform the analysis to warn abnormal behaviors.

These two malware samples are slightly different in terms of behavior. Linux/Mirai has to connect to a C&C server and infected firmware images are set into infinite loop waiting for commands from the C&C Server but it’s not necessary for Linux/TheMoon. Linux/TheMoon interacts a lot with operating system by creating and deleting files before executing shell commands on a vulnerable router that has Remote Management Access enabled, and downloads a copy of itself. Therefore, we present in details the results of Linux/TheMoon with Strace and Linux/Mirai with Strace and Tcpdump. First, we perform the vulnerabilities detection for raw images containing Linux/TheMoon and Linux/Mirai. This result is shown in Fig. 8. This means that Linux/TheMoon and Linux/Mirai did not create any vulnerability during its execution, therefore only Firmadyne is not enough to detect abnormal behaviors.

Fig 8: Obtained result with Metasploit

Results with Strace

Linux/TheMoon

Linux/TheMoon (EXr.pdf) was launched within the Linksys E2500 firmware image and some facts appeared as Fig. 9.
Fig 9: Captured results during Linux/TheMoon execution

At the first sight, we can conclude that this firmware image does not behave in normal way because there are deleting file behaviors. To be more precise, Strace is used for getting more analysis in detail on system-calls made by Linux/TheMoon. After launching Strace, C500-Detector detects deleting files with extensions .pdf, .png, .gif, .jpg, .tif etc. in /tmp directory as presented by Fig. 10. Actually, these extensions are used by Linux/TheMoon for its several variants, therefore this step tried to remove all traces it may have left in the system.

Fig 10: Deleting file behaviors

Then, Linux/TheMoon created an important amount of new and empty files, about 110 files, with two commands: touch and open behaviors as showning by Fig. 11.

Fig 11: Creating file behaviors

As a result, we can conclude that this firmware image has abnormal behaviors. As for the firmware image containing Linux/Mirai malware, Strace was launched for the first image and we can retrieve a lot of interesting information. One part of this result is shown in Fig. 12.

Linux/Mirai

As a result, we can conclude that this firmware image has abnormal behaviors. As for the firmware image containing Linux/Mirai malware, Strace was launched for the first image and we can retrieve a lot of interesting information. One part of this result is shown in Fig. 12.

First, to control watchdog timer in embedded device, Linux/Mirai opens the /etc/watchdog file in read-write state at the first line:

```bash
open("/dev/watchdog", O_RDWR) = 3
```

Then, PF_INET socket for TCP protocol is opened through a specific port (53) to self connect to Google DNS Server (8.8.8.8) to ensure that the Internet connection was established.

![Fig 13: Executed processes in both raw images](image_url)

![Fig 14: Executed processes in raw image without Linux/Mirai](image_url)

![Fig 15: Executed processes in raw image containing Linux/Mirai](image_url)

![Fig 16: Process 133 attached](image_url)

![Fig 17: Process 132 attached](image_url)

![Fig 18: Process 131 attached](image_url)

![Fig 19: Process 130 attached](image_url)

![Fig 20: Process 129 attached](image_url)

![Fig 21: Process 128 attached](image_url)

![Fig 22: Process 127 attached](image_url)

![Fig 23: Process 126 attached](image_url)

![Fig 24: Process 125 attached](image_url)

![Fig 25: Process 124 attached](image_url)

![Fig 26: Process 123 attached](image_url)

![Fig 27: Process 122 attached](image_url)

![Fig 28: Process 121 attached](image_url)

![Fig 29: Process 120 attached](image_url)

![Fig 30: Process 119 attached](image_url)

![Fig 31: Process 118 attached](image_url)

![Fig 32: Process 117 attached](image_url)

![Fig 33: Process 116 attached](image_url)

![Fig 34: Process 115 attached](image_url)

![Fig 35: Process 114 attached](image_url)

![Fig 36: Process 113 attached](image_url)

![Fig 37: Process 112 attached](image_url)

![Fig 38: Process 111 attached](image_url)

![Fig 39: Process 110 attached](image_url)

![Fig 40: Process 109 attached](image_url)

![Fig 41: Process 108 attached](image_url)

![Fig 42: Process 107 attached](image_url)

![Fig 43: Process 106 attached](image_url)

![Fig 44: Process 105 attached](image_url)

![Fig 45: Process 104 attached](image_url)

![Fig 46: Process 103 attached](image_url)

![Fig 47: Process 102 attached](image_url)

![Fig 48: Process 101 attached](image_url)

![Fig 49: Process 100 attached](image_url)

![Fig 50: Process 99 attached](image_url)

![Fig 51: Process 98 attached](image_url)

![Fig 52: Process 97 attached](image_url)

![Fig 53: Process 96 attached](image_url)

![Fig 54: Process 95 attached](image_url)

![Fig 55: Process 94 attached](image_url)

![Fig 56: Process 93 attached](image_url)

![Fig 57: Process 92 attached](image_url)

![Fig 58: Process 91 attached](image_url)

![Fig 59: Process 90 attached](image_url)

![Fig 60: Process 89 attached](image_url)

![Fig 61: Process 88 attached](image_url)

![Fig 62: Process 87 attached](image_url)

![Fig 63: Process 86 attached](image_url)

![Fig 64: Process 85 attached](image_url)

![Fig 65: Process 84 attached](image_url)

![Fig 66: Process 83 attached](image_url)

![Fig 67: Process 82 attached](image_url)

![Fig 68: Process 81 attached](image_url)

![Fig 69: Process 80 attached](image_url)

![Fig 70: Process 79 attached](image_url)

![Fig 71: Process 78 attached](image_url)

![Fig 72: Process 77 attached](image_url)

![Fig 73: Process 76 attached](image_url)

![Fig 74: Process 75 attached](image_url)

![Fig 75: Process 74 attached](image_url)

![Fig 76: Process 73 attached](image_url)

![Fig 77: Process 72 attached](image_url)

![Fig 78: Process 71 attached](image_url)

![Fig 79: Process 70 attached](image_url)

![Fig 80: Process 69 attached](image_url)

![Fig 81: Process 68 attached](image_url)

![Fig 82: Process 67 attached](image_url)

![Fig 83: Process 66 attached](image_url)

![Fig 84: Process 65 attached](image_url)

![Fig 85: Process 64 attached](image_url)

![Fig 86: Process 63 attached](image_url)

![Fig 87: Process 62 attached](image_url)

![Fig 88: Process 61 attached](image_url)

![Fig 89: Process 60 attached](image_url)

![Fig 90: Process 59 attached](image_url)

![Fig 91: Process 58 attached](image_url)

![Fig 92: Process 57 attached](image_url)

![Fig 93: Process 56 attached](image_url)

![Fig 94: Process 55 attached](image_url)

![Fig 95: Process 54 attached](image_url)

![Fig 96: Process 53 attached](image_url)

![Fig 97: Process 52 attached](image_url)

![Fig 98: Process 51 attached](image_url)

![Fig 99: Process 50 attached](image_url)

![Fig 100: Process 49 attached](image_url)

![Fig 101: Process 48 attached](image_url)

![Fig 102: Process 47 attached](image_url)

![Fig 103: Process 46 attached](image_url)

![Fig 104: Process 45 attached](image_url)

![Fig 105: Process 44 attached](image_url)

![Fig 106: Process 43 attached](image_url)

![Fig 107: Process 42 attached](image_url)

![Fig 108: Process 41 attached](image_url)

![Fig 109: Process 40 attached](image_url)

![Fig 110: Process 39 attached](image_url)

![Fig 111: Process 38 attached](image_url)

![Fig 112: Process 37 attached](image_url)

![Fig 113: Process 36 attached](image_url)

![Fig 114: Process 35 attached](image_url)

![Fig 115: Process 34 attached](image_url)

![Fig 116: Process 33 attached](image_url)

![Fig 117: Process 32 attached](image_url)

![Fig 118: Process 31 attached](image_url)

![Fig 119: Process 30 attached](image_url)

![Fig 120: Process 29 attached](image_url)

![Fig 121: Process 28 attached](image_url)

![Fig 122: Process 27 attached](image_url)

![Fig 123: Process 26 attached](image_url)

![Fig 124: Process 25 attached](image_url)

![Fig 125: Process 24 attached](image_url)

![Fig 126: Process 23 attached](image_url)

![Fig 127: Process 22 attached](image_url)

![Fig 128: Process 21 attached](image_url)

![Fig 129: Process 20 attached](image_url)

![Fig 130: Process 19 attached](image_url)

![Fig 131: Process 18 attached](image_url)

![Fig 132: Process 17 attached](image_url)

![Fig 133: Process 16 attached](image_url)

![Fig 134: Process 15 attached](image_url)

![Fig 135: Process 14 attached](image_url)

![Fig 136: Process 13 attached](image_url)

![Fig 137: Process 12 attached](image_url)

![Fig 138: Process 11 attached](image_url)

![Fig 139: Process 10 attached](image_url)

![Fig 140: Process 9 attached](image_url)

![Fig 141: Process 8 attached](image_url)

![Fig 142: Process 7 attached](image_url)

![Fig 143: Process 6 attached](image_url)

![Fig 144: Process 5 attached](image_url)

![Fig 145: Process 4 attached](image_url)

![Fig 146: Process 3 attached](image_url)

![Fig 147: Process 2 attached](image_url)

![Fig 148: Process 1 attached](image_url)
connect(4, {sa_family=AF_INET, sin_port=hto ns(53), sin_addr=inet_addr("8.8.8.8")}, 16)

At this stage, everything seems fine but moving forward, we can see that there is an abnormal behavior consisting in opening a random TCP/port (48178), based on previous PF_INET socket, to the specific IP address (192.168.131.150) without user permission.

getsockname(4, {sa_family=AF_INET, sin_port=htons(56781), sin_addr=inet_addr("192.168.131.150")}, 16)

Then, Linux/Mirai opens another PF_INET socket at port 48101 from localhost IP address 127.0.0.1 and then starting to listen to the incoming connections and start its infinite loop.

bind(4, {sa_family=AF_INET, sin_port=htons(48101), sin_addr=inet_addr("127.0.0.1")}, 16) listen(4, 1)

Although this log is not exhaustive to identify backdoor opening and changes in infected system files but it collects significant behaviors: modifying watchdog file, opening a TCP/port to a specific IP, listening from outside to connect to previous port and continuously looping. Accordingly, we can conclude that these actions is abnormal behaviors. To be more precised, we analyzed in detail the logs captured by Strace for both of raw images. Concerning system behaviors, there are two supplement processes created, while Linux/Mirai was launched, are PID 747 and 749 as showing is Fig. 13.

The analysis of the process PID 747 with Strace shows that this process behaves as a Backdoor to connect via Http on IP 65.222.202.53 via the port 80. Details of this connection are shown in Fig. 14.

To capture the network behaviors of Linux/Mirai, Tcpdump was launched and the obtained result is shown in Fig. 16

**Results with Tcpdump**

Tcpdump shows that Linux/Mirai try to reach a wide range random IP by sending ICMP packets. Because of this firmware image was running within our emulator and there is no Internet connection, therefore we cannot observe further the network behaviors that Linux/Mirai perform. At this point, we can conclude that all Linux/Mirai behaviors could not be traced in details but two interesting comportments, without user permission, are captured and analyzed:

- The behavior of connection to an outside IP and starting to listen to the incoming connections.
- The behavior of scanning a wide range of random IP to check if these devices are alive.

Based on these two behaviors, we can conclude that the examined firmware image could be infected by malware.

**V. CONCLUSION**

In this paper, we presented the C500-toolkit to deal with the detection of malware and vulnerabilities in SOHO routers. Three main contributions of this toolkit respectively are: extracting firmware from an actual device FLASH chip; auto-completing missing parts to get a pre-emulated firmware image by QEMU; capturing and analyzing abnormal behaviors during its run-time.

By extracting and using NVRAM for configuring firmware images, the C500-toolkit could get a higher successful emulation rate. Our proposed tool could use Firmadyne as a complement to perform a full device scanning in order to identify vulnerabilities. Although there remain some limitations in malware detection rules or FLASH chip types that can be extracted, but it showed that we could detect malware by using C500-toolkit based on abnormal behaviors captured. Dynamic analysis could be interested in many aspects such as the obfuscation issue avoiding and the scalability, but it could not explore all malware trigger conditions (time, system events or network inputs) as static analysis approach. Therefore, our next challenge is to use static analysis as a pre-treatment step to automatically identify malware.
trigger conditions to use as inputs to our simulator.

REFERENCES

7. Internet of things top 10 project. URL www.owasp.org/
15. QEMU. URL http://www.qemu.org/.

AUTHORS PROFILE

Tran Nhi Phu is currently working as lecturer and network security specialist, People’s Security Academy, Hanoi, Vietnam. He has completed his Master degree in Software Engineering at VNU University of Engineering and Technology (2014). He has been doing research, application and teaching since then in the fields of network security, artificial intelligence, IoT and more recently in malware analysis. Currently, he is a PhD student at the University of Engineering and Technology – VNU, Hanoi, Vietnam. He has owned some applications in network security field at http://firmware.vn.

Le Van Hoang currently a graduate Department of Information Technology and Information Security in People's Security Academy, Hanoi, Vietnam. His research area includes malware analysis, artificial intelligence, natural language processing.

Dr. Ngi Quoc Dung is currently working as Lecturer in the Department of Information Technology, Posts and Telecommunications Institute of Technology, Hanoi, Vietnam. He has completed his doctoral degree in Informatics applied in automatic and manufacturing at the Grenoble Institute of Technology, Grenoble, France. He has been actively participated in all the research activities. He has many books and has more than 10 research papers to his credit. He also has guest edited several edited books. His research area includes network security, malware analysis, artificial intelligence, optimal energy.

Dr. Dai Tho Nguyen received his PhD from University of Technology of Compiegne, France. He is currently the Head of Laboratory of Information Security, Faculty of Information Technology, University of Engineering and Technology, Vietnam National University, Hanoi. He teaches various courses on network security, information security management, digital forensics, and ethical hacking. His research areas include information security, distributed computing and networking, with interests ranging from theoretical concerns to concrete use cases and scenarios.

Nguyen Ngoc Binh is currently working as Visiting Professor, Faculty of Computer and Information Sciences (CIS), Hosei University, Tokyo, Japan. He is PhD in Information and Computer Sciences from Osaka University, Japan and is Honorary Doctor from Toyohashi University of Technology, Japan. His profile is at http://het.vnu.edu.vn/~nbhinh/.