Xây dựng hệ thống phát hiện mã độc trong thiết bị định tuyến dựa trên mô phỏng

Ngô Quốc Dũng, Lê Hải Việt, Trần Hoàng Anh, Lê Văn Hoàng, Nguyễn Việt Anh

***Tóm tắt*— Song hành cùng cuộc cách mạng công nghiệp lần thứ 4 là sự phát triển mạng lưới kết nối các thiết bị IoT. Để đảm bảo sự thông suốt trong toàn bộ quá trình trao đổi thông tin giữa các thiết bị IoT, thì thiết bị định tuyến đóng vai trò then chốt. Do đó, thiết bị định tuyến đã và đang trở thành mục tiêu tấn công phổ biến của tin tặc. Điều này dẫn tới nguy cơ không chỉ mất an toàn thông tin của thiết bị IoT nói riêng mà còn là nguy cơ gây mất an toàn, an ninh mạng nói chung. Trong bài báo này, nhóm tác giả đề xuất một phương pháp mô phỏng đầy đủ nhằm thu thập dữ liệu hoạt động của phần sụn (firmware) để phát hiện mã độc trong các thiết bị định tuyến.**

***Abstract*— Nowadays, along with the Fourth Industrial Revolution is the rapid development of IoT devices networks. The router, which is used as a core to ensure the stability of the entire interacting process between IoT devices, has became a potential target for hackers. This fact leads to the risk of not only IoT devices' security in particular but also the cyber security in general. In this paper, the authors propose a complete emulation method for collecting firmware's operation data to detect malware in routers.**

***Từ khóa*— Phân tích động; Mã độc; IoT; Thiết bị định tuyến; Mô phỏng.**

***Keywords*— Dynamic analysis; Malware; IoT; Router; Simulation.**

I. GIỚI THIỆU

Sự phát triển nhanh chóng về số lượng của các thiết bị IoT mang tới khả năng kết nối, trao đổi thông tin của mọi thiết bị với nhau thông qua mạng Internet. Trong đó có thiết bị được sử dụng phổ biến để kết nối nhiều thiết bị IoT với nhau là thiết bị định tuyến (Router). Tuy nhiên, vấn đề bảo mật cho thiết bị định tuyến còn chưa được quan tâm đúng mức.

Trong nghiên cứu của mình, Andrei Costin và cộng sự [1] đã phát hiện trong 32.256 firmware của các thiết bị mạng được phân tích, có hơn 38 loại lỗ hổng mới chưa được phát hiện trước đó. Thông qua các lỗ hổng này, tin tặc có thể tấn công làm chủ thiết bị, từ đó làm cơ sở để tấn công vào mạng mà các thiết bị này kết nối đến. Johannes Ullrich đã công bố nghiên cứu về mã độc Bashlite, một trong những mã độc lây nhiễm phổ biến trên các thiết bị IoT để xây dựng mạng botnet, nhằm thực hiện tấn công DDoS. Với hơn 1 triệu thiết bị IoT bị nhiễm độc, Bashlite có thể phát động cuộc tấn công DDoS lên tới 400 Gbps thông qua các kỹ thuật đơn giản như UDP hay TCP flood. Bashlite được coi là tiền thân của Mirai - loại mã độc ảnh hưởng tới nhiều loại thiết bị IoT. Mạng lưới botnet của Mirai được sử dụng trong cuộc tấn công DDoS đạt tới kỷ lục là 1,1 Tbps với 148.000 thiết bị IoT. Mục tiêu lây nhiễm chủ yếu của Mirai là các IP của camera, DVR và thiết bị định tuyến sử dụng trong gia đình. Để đối phó với các nguy cơ này, các nhà nghiên cứu về mã độc đã và đang phát triển các phương pháp và kỹ thuật phát hiện mới. Về cơ bản, các nghiên cứu này có thể chia làm hai nhóm chính là phân tích tĩnh và phân tích động.

Phân tích tĩnh là phương pháp phân tích, kiểm tra các phần mềm, mã độc trực tiếp trên mã nguồn, mã nhị phân tường minh trong các tập tin mà không cần thực thi chúng. Các nghiên cứu sử dụng phương pháp này trên các thiết bị IoT có thể kể đến như Angr [2]. Phân tích tĩnh cho phép chi tiết hóa toàn bộ luồng điều khiển (Control-Flow Graph) và luồng dữ liệu (Data-Flow Graph) cho từng tập tin hệ thống trong firmware. Từ đó, phát hiện mã độc bằng kỹ thuật phân tích đặc trưng như: mã trung gian (bytecode), header, system-calls API hay Printable-Strings-Information (PSI) [3]. Phương pháp phân tích tĩnh cho phép phân tích chi tiết các tập tin và đưa ra cái nhìn tổng quát về tất cả các khả năng kích hoạt của mã độc [4].

Tuy nhiên, phương pháp phân tích tĩnh khó áp dụng đối với các loại mã độc sử dụng các kỹ thuật gây rối phức tạp (obfuscations) như sắp xếp lại câu lệnh, chèn mã lệnh vô nghĩa [5]. Một hạn chế của phương pháp này là công nghệ dịch ngược các bản mã nhị phân thành bản mã bậc cao còn nhiều hạn chế [4] làm cho việc phân tích mất đi tính chính xác.

Bài báo được nhận ngày 25/05/2017. Bài báo được gửi cho phản biện thứ nhất vào ngày 20/06/2017 và nhận được ý kiến đồng ý đăng của phản biện thứ nhất vào ngày 27/07/2017. Bài báo được gửi phản biện thứ hai vào ngày 01/07/2017 nhận được ý kiến đồng ý đăng của phản biện thứ hai vào ngày 31/07/2017.

Do đó, theo Andreas Moser [5], phương pháp phân tích tĩnh nên được sử dụng như một phần bổ sung cho phân tích động.

Phân tích động là phương pháp giám sát, thu thập và phân tích các hành vi của hệ thống để từ đó phát hiện mã độc [6]. Kỹ thuật này dựa trên nguyên lý sử dụng tập luật bình thường để duy trì và xem xét một chương trình có cố ý vi phạm những tập luật được định trước hay không. Một số nghiên cứu phân tích động phát hiện mã độc trên thiết bị IoT có thể kể đến như Avatar [7], phân tích lỗ hổng bảo mật trên thiết bị định tuyến – Firmadyne [8]. Yêu cầu quan trọng nhất đối với phân tích động cho các thiết bị IoT là xây dựng một môi trường mô phỏng đầy đủ các chức năng cần có của thiết bị, có khả năng giám sát các hành vi của firmware khi thực thi và tránh lây nhiễm mã độc sang môi trường thực tế.

Để giải quyết yêu cầu trên, Jonas Zaddach và cộng sự giới thiệu về Avatar [7], cho phép mô phỏng hoạt động của CPU và tái sử dụng toàn bộ phần cứng của thiết bị định tuyến phục vụ mục đích mô phỏng trên. Tuy nhiên, hạn chế của Avatar là khả năng hoạt động thời gian thực, vì việc xử lý và phân tích thông tin giữa môi trường mô phỏng Qemu và thiết bị thật thông qua kênh UART, Jtag là rất chậm. Do đó, việc sử dụng công cụ Avatar phát hiện mã độc theo thời gian thực trên các thiết bị IoT là bất khả thi.

Mặt khác, Daming Chen và cộng sự đã trình bày về Firmadyne trong nghiên cứu của mình. Đây là hệ thống phân tích động với mục tiêu cụ thể là thiết bị định tuyến trong hạ tầng mạng [8]. Tuy nhiên, Firmadyne chỉ cho phép mô phỏng phần giao diện web quản trị của các thiết bị định tuyến với đầu vào là Firmware của chúng. Điều này phục vụ mục tiêu là quét lỗ hổng bảo mật của các thiết bị định tuyến bằng cách sử dụng các công cụ như Metaspoit và Nessus, chứ không cho phép phát hiện mã độc.

Ưu điểm nổi bật của phương pháp phân tích động là hiệu quả và độ chính xác, cho phép xác định nhanh chóng và tổng quát về mã độc được phân tích, thông qua các hành vi của chúng. So với phương pháp phân tích tĩnh trong việc dịch ngược, gỡ rối (deobfuscation) thì phương pháp động cho phép phân tích dễ dàng ngay cả với những mã độc có cấu trúc, mã nguồn phức tạp.

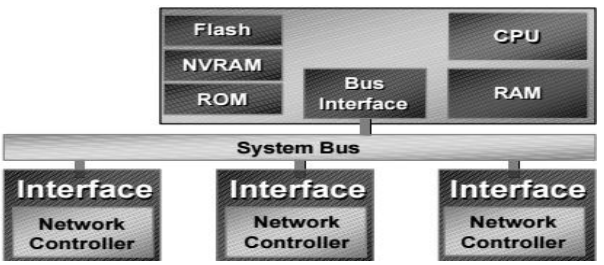
Tuy nhiên, phân tích động chỉ có thể giám sát đơn luồng thực thi. Điều này đã được T. Ronghua [4] chứng minh trong công bố của mình rằng: khi các điều kiện môi trường ảnh hưởng trực tiếp đến việc kích hoạt mã độc như time-bomb, bot,… thì phương pháp động không thể giám sát hết các hành vi tiềm tàng của mã độc. Việc giám sát được tất cả các khả năng thực thi của mã độc trong phân tích động đòi hỏi nhiều thời gian với dữ liệu ghi nhận là rất lớn.

Mặc dù có những hạn chế, nhưng phân tích động có ưu điểm nổi bật so với phân tích tĩnh ở khả năng áp dụng trên diện rộng và tránh được các kỹ thuật làm rối như đã nêu. Do đó, phương pháp đề xuất trong bài báo này dựa trên phương pháp phân tích động và bổ khuyết cho Firmadyne bằng cách xây dựng một môi trường mô phỏng đầy đủ, bao gồm cả phần giao diện web quản trị cho việc quét lỗ hổng và phần hoạt động của hệ điều hành thiết bị định tuyến cho việc phân tích mã độc.

Bài báo được trình bày theo bố cục sau: Sau Mục Giới thiệu, Mục II giới thiệu tổng quan về cấu trúc của các thiết bị định tuyến và firmware. Mục III trình bày quy trình xây dựng môi trường mô phỏng và phân tích. Kết quả thực nghiệm cụ thể sẽ được đưa ra trong Mục IV và cuối cùng là Mục Kết luận

II. TỔNG QUAN VỀ CẤU TRÚC CỦA CÁC THIẾT BỊ ĐỊNH TUYẾN VÀ FIRMWARE

A. Tổng quan cấu trúc thiết bị định tuyến



Hình 1. Cấu tạo của thiết bị định tuyến

Thiết bị định tuyến là thiết bị mạng 3 lớp của mô hình OSI (Open Systems Interconnection) với cấu tạo được thể hiện trong Hình 1, gồm các phần cụ thể như sau:

- *CPU:* điều khiển mọi hoạt động của bộ định tuyến trên cơ sở các hệ thống chương trình thực thi của hệ điều hành.
- *ROM:* chứa các chương trình tự động kiểm tra và có các thành phần cơ bản nhất sao cho bộ định tuyến có thể thực thi được một số hoạt động tối thiểu ngay cả khi không có hệ điều hành hay hệ điều hành bị hỏng.

- *RAM:* cấp phát vùng nhớ cho các quá trình như: lưu trữ các bảng định tuyến, các vùng đệm, tập tin cấu hình khi chạy, các thông số đảm bảo hoạt động của bộ định tuyến.
- *Flash:* là thiết bị nhớ có khả năng ghi và xóa, lưu dữ liệu khi mất nguồn. Thông thường, firmware của bộ định tuyến được lưu trữ ở đây. Tùy thuộc các thiết bị định tuyến khác nhau mà hệ điều hành sẽ được chạy trực tiếp từ Flash hay được tải lên RAM trước khi chạy. Tập tin cấu hình cũng có thể được lưu trữ trong Flash.
- *NVRAM (None-Volatile RAM):* có chức năng tương tự như *Flash* nhưng có khả năng lưu trữ ít hơn. *NVRAM* thường chứa tập tin cấu hình của thiết bị để đảm bảo khi khởi động, cấu hình mặc định của thiết bị định tuyến sẽ được tự động nạp về đúng trạng thái đã lưu giữ.

B. Tổng quan cấu trúc firmware

Cấu trúc của firmware rất đa dạng, phụ thuộc vào chức năng và thiết kế của từng nhà sản xuất. Các firmware được chia thành các kiểu như sau:

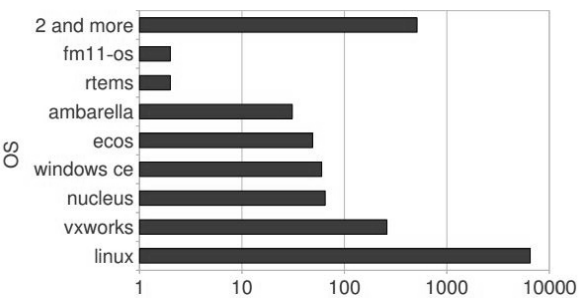
- Full-blown (full-OS/kernel + bootloader + libs + apps): đây thường là một Linux hoặc Windows firmware vì chúng chứa một hệ điều hành hoàn chỉnh đã được tối giản. Các ứng dụng có thể chạy trong chế độ người dùng, kernel modules, drivers.
- *Integrated (apps + OS-as-a-lib)*: đây là một bản firmware không đầy đủ, các chức năng và hệ điều hành được xây dựng như một thư viện chứ không có đầy đủ các thành phần cần thiết như trong bản Full-blown.
- *Partial updates* (các ứng dụng / các thư viện / các tài nguyên / các hỗ trợ): Loại firmware này chỉ chứa các tập tin dùng trong việc cập nhật cho bản firmware cần nâng cấp.

Bài báo này, tập trung phân tích loại Full-blown bởi vì khi chúng có chứa đầy đủ các tập tin hệ thống cần thiết cho việc phân tích động. Hệ điều hành hoàn chỉnh nhưng tối giản trong các firmware loại này thông thường chứa các tập tin hệ thống như sau:

- *Bootloader*: là một đoạn mã được thực thi trước khi hệ điều hành bắt đầu chạy và nó cho phép nhà sản xuất thiết bị quyết định những tính năng nào người sử dụng được phép dùng hoặc bị hạn chế.
- *Kernel*: được hiểu là hạt nhân - thành phần trung tâm của thiết bị định tuyến, có trách nhiệm giúp cho phần cứng và phần mềm của thiết bị có thể giao tiếp được với nhau. Kernel là thành phần quan trọng nhất trong thiết bị, nếu kernel bị hỏng thì thiết bị định tuyến sẽ dừng hoạt động. Kernel thường được lưu trữ trong bộ nhớ Flash của thiết bị.
- *File–system images*: chứa các tập tin hệ thống có chức năng tổ chức và kiểm soát quá trình hoạt động của thiết bị nhúng.
- *Web–server/web–interface*: đây chính là thành phần quan trọng giúp cho người dùng có thể tương tác được với thiết bị một cách dễ dàng. Tất cả các thông tin và cấu hình thiết bị phần lớn được thao tác thông qua Web–server và Web–interface.

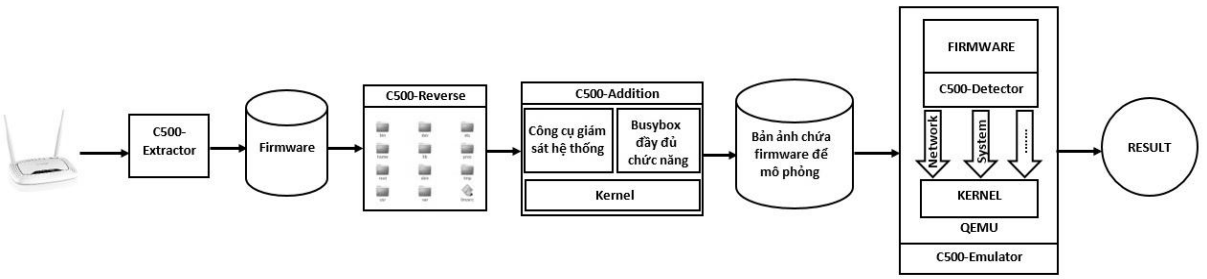
Khác với các thiết bị máy tính cá nhân khi bộ vi xử lý đa phần dùng nền tảng i386 và hệ điều hành Windows, 86% các thiết bị định tuyến dùng các bộ vi xử lý như MIPS và ARM với hệ điều hành Linux [11].

Do đó, yêu cầu đối với một môi trường mô phỏng đầy đủ cho các thiết bị định tuyến phải thoả mãn điều kiện là hỗ trợ đa nền tảng vi xử lý: MIPS, ARM, PowerPC, SPARC,… và đa nền tảng hệ điều hành, đặc biệt là Linux. Bên cạnh đó, một câu hỏi đặt ra là firmware cài đặt sẵn trong các thiết bị định tuyến lúc xuất xưởng và các bản firmware công bố trên mạng có khác nhau. Điều này quan trọng khi chúng ta chỉ có thể kiểm tra và phát hiện mã độc trên các bản firmware được công bố chứ không phải các bản cài đặt sẵn trên thiết bị. Vì vậy, việc bóc tách bản firmware trên thiết bị để phân tích và phát hiện mã độc là cần thiết.



Hình 2. Nền tảng hệ điều hành sử dụng phổ biến trong firmware trên các thiết bị định tuyến [1]

Để giải quyết những vấn đề trên, quy trình xây dựng môi trường phân tích mã độc trên các thiết bị định tuyến đề xuất được trình bày trong Mục III.



Hình 3. Tổng quan về bộ công cụ C500-Toolkit

III. QUY TRÌNH XÂY DỰNG MÔI TRƯỜNG MÔ PHỎNG VÀ PHÂN TÍCH

Quy trình xây dựng môi trường mô phỏng đầy đủ và phân tích được giới thiệu trong Hình 3 với 3 bước chính như sau:

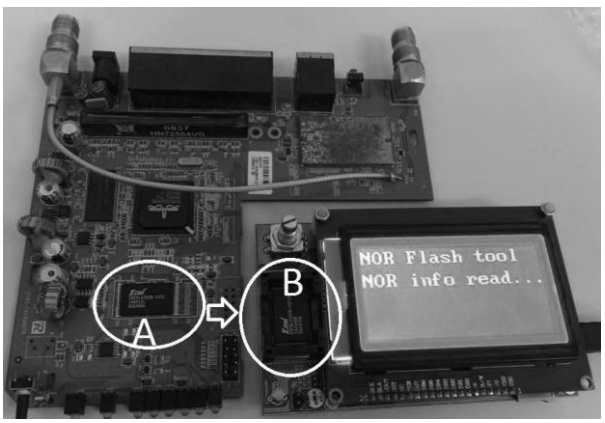
- Bước 1: Trích xuất firmware của thiết bị định tuyến với công cụ *C500-Extractor*.
- Bước 2: Tạo bản ảnh phù hợp cho môi trường mô phỏng đầy đủ hoạt động của thiết bị định tuyến với công cụ *C500-Standardization*.
- Bước 3: Phân tích các hành vi thu nhận được để phát hiện hành vi bất thường, từ đó đưa ra cảnh báo có mã độc hay không trong firmware của thiết bị định tuyến với công cụ *C500-Detector*.

A. Công cụ C500-Extractor

Việc trích xuất firmware được thực hiện trên các dòng thiết bị định tuyến SOHO (Small Office – Home Office) thông qua một số phương pháp như sử dụng chức năng backup của thiết bị, sử dụng cổng điều khiển nối tiếp (Serial) hay kênh kiểm tra lỗi (Debug Jtag). Tuy nhiên, trong một vài trường hợp, những phương pháp trích xuất này có thể không đạt được hiệu quả như mong muốn. Nguyên nhân là do nhà sản xuất đã khóa các cổng này trên bảng mạch và không cho phép người dùng can thiệp vào thiết bị của hãng. Trong trường hợp này, việc trích xuất trực tiếp dữ liệu firmware từ thiết bị là cần thiết, và là mục tiêu mà thiết bị trích xuất *C500-extractor* hướng tới.

C500-extractor được thiết kế để lấy dữ liệu từ chip Flash chứa firmware nằm trên bảng mạch của thiết bị định tuyến. Phiên bản mẫu được thiết kế và chế tạo có khả năng lấy dữ liệu từ các kiểu chip Flash trên các dòng thiết bị định tuyến phổ biến của TP-Link và Linksys, bao gồm: W25Q32FV, W25Q64FV, W25Q128FV (hãng Winbond); FM25Q64 (Fidelix); MX25L1606E (Macronix); EN29LV3 20B-70TC (Eon).

Thông qua tiến hành thực nghiệm trên những chip Flash này, hai chuẩn giao tiếp được sử dụng để đọc dữ liệu ra từ chip Flash là giao tiếp SPI (Serial Peripheral Interface) [9] và giao tiếp FSMC (Flexible Static Memory Controller) [10]. Do đó, mẫu thiết kế này đã được tích hợp hai cổng FSMC và cổng SPI. Trong đó, cổng FSMC được dùng để đọc chip Flash của hãng Eon và cổng SPI được sử dụng cho chip Flash của các hãng Winbond, Fidelix và Macronix. Vi điều khiển chính được sử dụng trên thiết bị trích xuất này là vi điều khiển dòng STM32F10. Vi điều khiển này có thể hỗ trợ giao tiếp với chip Flash thông qua cả SPI và FSMC.



Hình 4. Hoạt động của thiết bị C500-Extractor

Để tiến hành quá trình trích xuất, chip Flash cần được trích xuất sẽ được đặt trên khay đọc chip tương ứng trên thiết bị *C500-Extractor* như trên Hình 4. Sau đó, thiết bị trích xuất sẽ đọc ID code của chip Flash để xác định chip được đặt vào là loại chip nào và điều chỉnh các thông số phù hợp phục vụ cho quá trình đọc dữ liệu. Tất cả dữ liệu trên chip Flash được đọc ra và lưu vào một tệp có đuôi *.bin* và lưu trên thẻ nhớ nằm trong thiết bị trích xuất. Sau khi quá trình trích xuất kết thúc,

tệp nhị phân này sẽ được chuyển vào máy tính thông qua kết nối cáp micro USB. Dữ liệu trên tệp nhị phân này thường bao gồm các thông số của NVRAM, vmlinux, bootloader và rootfs. Sau khi các tệp nhị phân này được lấy ra, chúng sẽ được lưu trữ vào cơ sở dữ liệu phục vụ cho những quy trình tiếp theo.

B. Công cụ C500-Standardization

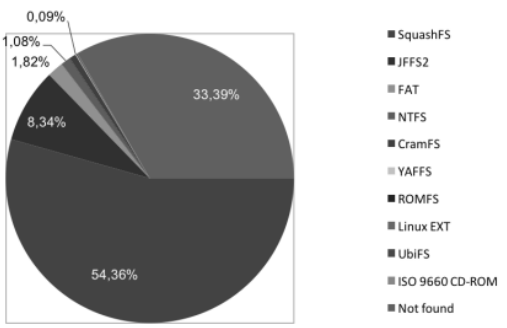
Sau khi đã trích xuất được firmware từ thiết bị định tuyến, việc chuẩn hóa firmware mà không làm thay đổi hoạt động của thiết bị định tuyến được hoàn thành thông qua hai bước sau:

- Bóc tách firmware và xác định kiến trúc phần cứng để bổ sung các công cụ thích hợp bằng công cụ *C500-Reverse*.
- Bổ sung thêm công cụ theo dõi lời gọi hệ thống (system-call), các hành vi mạng bằng công cụ *C500-Addition.*

C500-Reverse được xây dựng dựa trên mục tiêu *xác định kiến trúc phần cứng,* giải nén mã nguồn nhị phân của firmware thành các tập tin hệ thống tường minh và xác định phiên bản nhân Linux. Sau đó, quá trình bóc tách firmware từ tệp tin *.bin* được thực hiện để thu các tập tin hệ thống bởi công cụ *C500-Reverse*. Trên thực tế, đã có sẵ những công cụ cho việc bóc tách firmware, có thể kể đến như:

- *Firmware-mod-kit* (FMK) [11]: một tập các câu lệnh dưới dạng kịch bản (script) và công cụ tích hợp nhằm mục đích dịch ngược và đóng gói các firmware có nền tảng hệ điều hành là Linux. FMK chứa một tập các phiên bản khác nhau của bộ công cụ *unsquashfs*, sau khi dùng FMK để xác định điểm *offset* cũng như phiên bản nén của phần nhân chứa tập tin hệ thống, FMK sẽ thử lần lượt từng phiên bản khác nhau có trong bộ công cụ *unsquashfs* của mình để dịch ngược. Phương pháp này có một số điểm yếu, đáng chú ý nhất là việc dịch ngược bằng một bộ công cụ *unsquashfs* nhất định chưa chắc chắn là phiên bản chuẩn dùng khi biên dịch *squashfs*. Việc này dễ xảy ra nếu nhà sản xuất thực hiện việc thay đổi nội dung trong phần Header của tập tin.
- *Binwalk* [12]: công cụ dùng để phân tích, dịch ngược và giải nén dữ liệu chứa trong ảnh của các firmware. Binwalk được sử dụng phổ biến nhất hiện nay và dễ dàng tương thích với các công cụ phân tích mã độc khác vì được viết chủ yếu bằng ngôn ngữ Python. Bằng công cụ Binwalk, Costin và cộng sự [13] đã thử nghiệm phân tích tĩnh các bản firmware của thiết bị định tuyến. Theo kết quả thử nghiệm, nhóm tác giả đã chỉ ra rằng Binwalk không thể nhận dạng được một số firmware lưu dưới dạng tệp nhị phân, dẫn đến tỷ lệ dương tính giả khá cao bởi không thể tìm thấy dấu hiệu (signal) hoặc dấu hiệu đã bị thay đổi khi sử dụng đối sánh trong tập tin magic [8]. Đặc biệt, tỷ lệ này là cao đối với những dòng thiết bị của Cisco khi dữ liệu nén kiểu gzip. Nói cách khác, hạn chế của Binwalk không thể xác định được các loại firmware Integrated hay Partial updates.
- *Extracter* của Firmadyne: là công cụ được Daming D.Chen cùng đồng sự [8] sử dụng, trong đó đã làm mịn các chức năng có trong Binwalk. Tuy nhiên, khả năng dịch ngược của Firmadyne cũng còn hạn chế khi tỉ lệ thành công chỉ chiếm gần 33%.

Để cải thiện các tồn tại trên, môđun *C500-Reverse* đã đề xuất tại hội thảo SOIS 2016 [14] cho thấy có tỉ lệ bóc tách tốt hơn. Lý do của kết quả này là *C500-Reverse* có nhiều môđun để dịch ngược firmware theo các kiểu tập tin hệ thống khác nhau như *squashfs, cramfs, jefferson, yaffs,*… chiếm tới 97% trên tập thử nghiệm bao gồm 13275 firmware. Các kiểu tập tin hệ thống phổ biến nhất được trình bày tại Hình 5.



Hình 5. Các kiểu tập tin hệ thống phổ biến nhất [14]

Hình 6. Các tập tin hệ thống của firmware

Hơn nữa, *C500-Reverse* được xây dựng theo các môđun riêng cho phép việc cập nhật thường xuyên [14]. Các đặc trưng về phương pháp dịch ngược, sai số khi dịch ngược và kiểm chứng dịch ngược đã được trình bày cụ thể trong bài báo "Phát triển công cụ dịch ngược firmware trên thiết bị định tuyến" [14]. Một ví dụ về các tập tin hệ thống của firmware Netgear WNAP320 version 2.0.3 sau khi dịch ngược bằng *C500-Reverse* được trình bày trên Hình 6.

Bên cạnh những thư mục thường thấy trong hệ điều hành Linux như *bin, dev, etc*, những thư mục chứa giao diện web của thiết bị định tuyến được tìm thấy ở thư mục *www, cli* trong thư mục *home*. Ở giai đoạn này, việc xác định những thành phần còn thiếu của firmware để mô phỏng đầy đủ giao diện web và hệ điều hành là rất quan trọng. Để mô phỏng giao diện web, Firmadyne [8] đã trình bày chi tiết quá trình và cách sử dụng các công cụ quét để phát hiện các lỗ hổng. Dưới đây, chúng tôi giới thiệu cách để mô phỏng hệ điều hành để phát hiện mã độc dựa trên các lời gọi hệ thống (System-calls) và các hoạt động mạng.

Trên thực tế, để giảm kích thước firmware của thiết bị định tuyến xuống dưới 8 Mb, các nhà cung cấp đã tùy chỉnh firmware bằng cách loại bỏ các phần "không cần thiết". Vì vậy, Busybox được sử dụng rộng rãi trong các hệ thống nhúng vì nó tích hợp các phiên bản nhỏ của nhiều tiện ích phổ biến trên UNIX [15]. Tuy nhiên, để đảm bảo sự nhỏ gọn nên Busybox không được tích hợp các công cụ để theo dõi các lời gọi hệ thống và theo dõi các hoạt động mạng.

```
currently defined functions:
        [, [[, addgroup, adduser, adjtimex, ar, arp, arping, ash,
        awk, basename, brctl, bunzip2, bzcat, bzip2, cal, cat, catv,
        chat, chattr, chgrp, chmod, chown, chpasswd, chpst, chroot,
        chrt, chvt, cksum, clear, cmp, comm, cp, cpio, crond, crontab,
        cryptpw, cut, date, dc, dd, deallocvt, delgroup, deluser,
        depmod, df, dhcprelay, diff, dirname, dmesg, dnsd, dos2unix,
        dpkg, dpkg-deb, du, dumpkmap, dumpleases, echo, ed, egrep,
        eject, env, envdir, envuidgid, ether-wake, expand, expr,
        fakeidentd, false, fbset, fbsplash, fdflush, fdformat, fdisk,
        fetchmail, fgrep, find, fold, free, freeramdisk, fsck, fsck.minix,
        ftpget, ftpput, fuser, getopt, getty, grep, gunzip, gzip,
        halt, hdparm, head, hexdump, hostid, hostname, httpd, hwclock,
        id, ifconfig, ifdown, ifenslave, ifup, inetd, init, inotifyd,
        insmod, install, ip, ipaddr, ipcalc, ipcrm, ipcs, iplink,
        iproute, iprule, iptunnel, kbd_mode, kill, killall, killall5,
        klogd, last, length, less, linux32, linux64, linuxrc, ln,
        loadfont, loadkmap, logger, login, logname, logread, losetup,
        lpd, lpq, lpr, ls, lsattr, lsmod, lzmacat, makedevs, man,
        md5sum, mdev, mesg, microcom, mkdir, mkfifo, mkfs.minix,
        mknod, mkswap, mktemp, modprobe, more, mount, mountpoint,
        mt, mv, nameif, nc, netstat, nice, nmeter, nohup, nslookup,
        od, openvt, passwd, patch, pgrep, pidof, ping, ping6, pipe_progress,
        pivot_root, pkill, poweroff, printenv, printf, ps, pscan,
        pwd, raidautorun, rdate, readahead, readlink, readprofile,
        realpath, reboot, renice, reset, resize, rm, rmdir, rmmod,
        route, rpm, rpm2cpio, rtcwake, run-parts, runlevel, runsv,
        runsvdir, rx, script, sed, sendmail, seq, setarch, setconsole,
        setkeycodes, setlogcons, setsid, setuidgid, sh, sha1sum,
        slattach, sleep, softlimit, sort, split, start-stop-daemon,
        stat, strings, stty, su, sulogin, sum, sv, svlogd, swapoff,
        swapon, switch_root, sync, sysctl, syslogd, tac, tail, tar,
        taskset, tcpsvd, tee, telnet, telnetd, test, tftp, tftpd,
        time, top, touch, tr, traceroute, true, tty, ttysize, udhcpc,
        udhcpd, udpsvd, umount, uname, uncompress, unexpand, uniq,
        unix2dos, unlzma, unzip, uptime, usleep, uudecode, uuencode,
        vconfig, vi, vlock, watch, watchdog, wc, wget, which, who,
        whoami, xargs, yes, zcat, zcip
```

Hình 7. Busybox với đầy đủ chức năng

Theo kết quả nghiên cứu của Landley [15] tùy biến với các phiên bản Busybox khác nhau, chúng tôi có thể tùy chỉnh để có thêm nhiều chức năng hơn mà không làm thay đổi hay ảnh hưởng đến hoạt động của thiết bị định tuyến. Đối với mỗi firmware, một phiên bản Busybox thích hợp với đầy đủ chức năng hơn được thay thế cho Busybox vốn có của firmware tìm thấy trong thư mục *bin*. Một Busybox đầy đủ các công cụ được thể hiện ở trong Hình 7.

Tiếp theo, việc xác định các thư viện còn thiếu và tích hợp các công cụ giám sát vào firmware là cần thiết. Chúng tôi đã chọn *Strace* [16] để theo dõi các lời gọi hệ thống và *Tcpdump* [17] để theo dõi các hành vi mạng. *Strace* là công cụ cho phép người dùng giám sát các tương tác giữa các tiến trình và nhân Linux bao gồm các lời gọi hệ thống, sự phân phối tín hiệu và sự thay đổi trạng thái của tiến trình. *Tcpdump* là công cụ phân tích các gói dữ liệu mạng, cho phép chặn bắt và hiển thị các gói tin được truyền, nhận trên một mạng mà nó kết nối đến.

Công cụ *C500-Addition* xác định những thư viện còn thiếu để phục vụ cho việc cài đặt *Strace* và *Tcpdump*. Để dễ dàng hơn, chúng tôi sử dụng QEMU [18] để mô phỏng firmware trước tiên và sau đó sẽ bổ sung các thư viện còn thiếu. Ở giai đoạn này, QEMU đòi hỏi một tập các giá trị cấu hình mặc định của các thiết bị ngoại vi và lưu cấu hình này liên tục vào NVRAM. Tuy nhiên, những giá trị này đôi khi không có trong dữ liệu được trích xuất từ Flash, vì một số nhà cung cấp lưu trữ sẵn trong một thành phần khác của thiết bị. Để giải quyết vấn đề này, một giải pháp thay thế được đề xuất là sử dụng thư viện *libnvram.so* như Firmadyne đã làm. Thư viện này bao gồm các thiết lập cơ bản về cấu hình giao diện web như cài đặt mạng wifi, địa chỉ MAC và các xác thực truy cập cho giao diện web. Bằng cách sử dụng thư viện này, Firmadyne đã mô phỏng thành công 52,6% trong tổng số các firmware được bóc tách (4992 trong số 9486 firmware được kiểm tra) [8].

Sau đó, *C500-Addition* bắt đầu mô phỏng firmware và bổ sung tất cả những thư viện còn thiếu. Nhờ vào việc thay đổi Busybox với đầy đủ chức năng, chúng tôi có thể cài đặt được *Strace* và *Tcpdump* để thực hiện theo dõi hệ thống.

Công cụ *Strace* chạy ổn định trên phiên bản nhân Linux 3.2.x, trong khi trên phiên bản nhân Linux 2.6.x được tùy chỉnh bởi Firmadyne lại chưa hỗ trợ.

```
[*] Processing script.rc for ERB directives.
resource (script.rc)> setg RHOST 192.168.0.150
RHOST => 192.168.0.150
resource (script.rc)> setg RHOSTS 192.168.0.150
RHOSTS => 192.168.0.150
resource (script.rc)> spool exploits/exploit.0.log
[*] Spooling to file exploits/exploit.0.log...
resource (script.rc)> use exploits/linux/http/airties_login_cgi_bof
resource (script.rc)> exploit -z
[*] Started reverse TCP handler on 192.168.0.149:4444
[*] Accessing the vulnerable URL...
[-] Exploit aborted due to failure: unknown: 192.168.0.150:80 - Failed to access
 the vulnerable URL
[*] Exploit completed, but no session was created.
resource (script.rc)> spool off
[*] Spooling is now disabled
resource (script.rc)> sessions -K
[*] Killing all sessions...
resource (script.rc)> spool exploits/exploit.1.log
[*] Spooling to file exploits/exploit.1.log...
resource (script.rc)> use exploits/linux/http/belkin_login_bof
resource (script.rc)> exploit -z
[*] Started reverse TCP handler on 192.168.0.149:4444
[*] Accessing the vulnerable URL...
```

Hình 8. Kết quả quét với Metasploit

Với gần 86% thiết bị định tuyến sử dụng nhân Linux 2.6.x [1], nhóm nghiên cứu phải tùy chỉnh nhân Linux 2.6.x để các công cụ *Strace* và *Tcpdump* có thể chạy nhằm phục vụ cho việc thu thập thông tin phát hiện mã độc trong thiết bị định tuyến.

C. Công cụ C500-Detector

Công cụ này được xây dựng dựa trên kiểm tra các hành vi tương tác của mã độc với thiết bị theo thời gian thực. Các quy tắc phát hiện dựa trên tập các hành vi bất thường mà mã độc tương tác với các tập tin hệ thống, mạng và các tiến trình. Chúng tôi đã phân tích các thông tin được thu thập bởi công cụ *Strace* để xác định các hành vi bất thường về việc tạo, xóa các tập tin mà không có quyền của người sử dụng; công cụ *Tcpdump* giám sát các hành vi lắng nghe, mở và quét các cổng trái phép; kết nối đến IP trong danh sách nghi vấn. Kết quả thu được từ môđun *C500-Detector* được trình bày trong phần kết quả thực nghiệm.

IV. KẾT QUẢ THỰC NGHIỆM

Trong phần này, nhóm tác giả trình bày kết quả thực nghiệm mà bộ công cụ *C500-toolkit* giám sát mã độc Linux/Mirai thực thi trên thiết bị định tuyến. Mã độc Linux/Mirai được tải về từ mã nguồn công bố trên Internet [19]. Mã độc này lây nhiễm hàng loạt thiết bị IoT, biến chúng trở thành mạng botnet để thực hiện tấn công DDoS. Sự kiện này được đánh giá là cuộc tấn công từ chối dịch vụ lớn nhất từ trước đến nay. (MD5: 8e36a1fb6f6f718ec0b621a639437d8b) Kịch bản thử nghiệm như sau:

- Hai bản sao của firmware Netgear WNAP320 được sử dụng để thực hiện mô phỏng. Trong đó, một bản có lây nhiễm mã độc Linux/Mirai và bản còn lại thì không.
- Thực hiện các bước được trình bày tại Mục III để chuẩn hóa hai firmware.
- Tiến hành thực thi hai firmware này trên môi trường mô phỏng đầy đủ và thực hiện quá trình giám sát, phân tích: Sử dụng *Metasploit* quét cả hai firmware khi đang mô phỏng để kiểm tra liệu có thêm lỗ hổng nào trong suốt quá trình Linux/Mirai thực thi hay không; Sử dụng *Tcpdump* để giám sát hành vi mạng của cả hai bản firmware; Sử dụng *Strace* quét firmware bị nhiễm mã độc để ghi lại mọi hành vi thể hiện dưới dạng các cuộc gọi hệ thống.
- Dựa trên kết quả thu được từ *Metasploit, Tcpdump* và *Strace, C500-Detector* phát hiện các hành vi bất thường có thể có trong suốt quá trình mô phỏng firmware, từ đó quyết định xem có tồn tại mã độc trong firmware hay không.

Sau khi mô phỏng thành công hai firmware, kết quả thu được như sau:

Đối với công cụ *Metasploit*, kết quả thu được là một lỗ hổng giống nhau trên cả hai bản firmware, có mã là: CVE-2016-1555. Kết quả này đồng nghĩa với việc không xuất hiện thêm một lỗ hổng nào trong suốt quá trình Linux/Mirai thực thi. Kết quả quét với *Metasploit* được trình bày tại Hình 8. Đây cũng là đóng góp chính của môi trường mô phỏng đầy đủ đề xuất khi mà sử dụng Firmadyne không thể phát hiện ra mã độc trên các thiết bị định tuyến.

Đối với công cụ *Tcpdump*, thông tin giám sát các hành vi mạng thu được là giống nhau. Kết quả này cho thấy, điểm hạn chế trong phương pháp mô phỏng đề xuất là chưa giám sát được hoàn toàn các hành vi mạng của mã độc. Kết quả giám sát được trình bày tại Hình 9.

Đối với công cụ *Strace*, thông tin thu được có nhiều điểm nghi vấn. Đối với bản firmware có chứa Linux/Mirai, 2 tiến trình mới được tạo và được thể hiện ở Hình 10 (hai tiến trình có PID là 537 và 538). Trước khi phân tích chi tiết hai tiến trình 537 và 538, phân tích sơ bộ ban đầu khi dùng *Strace* giám sát thông tin tương tác giữa Linux/Mirai và Kernel thì phát hiện hành vi mở tập tin */dev/watchdog* ở trạng thái cho phép đọc và ghi ở dòng đầu tiên:

```
open("/dev/watchdog", O_RDWR)
```

Sau đó, socket PF_INET cho giao thức TCP được mở thông qua một cổng đặc biệt (53) để kết nối với máy chủ DNS của Google (8.8.8.8):

```
socket(PF_INET, SOCK_DGRAM,
IPPROTO_IP)
connect(3, {sa_family=AF_INET,
sin_port=htons(53),
sin_addr=inet_addr("8.8.8.8")}, 16)
```

Ở giai đoạn này, các thông tin chưa có gì khác thường, tuy nhiên khi tiếp tục sẽ thấy Linux/Mirai mở một cổng TCP ngẫu nhiên (48178) dựa trên socket PF_INET trước đó tới một địa chỉ cụ thể là 192.168.0.150:

```
   getsockname(3, {sa_family=AF_INET,
sin_port=htons(43847),
  sin_addr=inet_addr("192.168.0.150")
}, [14])
```

Mặc dù những thông tin này chưa đầy đủ để xác định việc mở cổng hậu (backdoor) và sự thay đổi các tập tin hệ thống khi bị nhiễm mã độc nhưng nó thực sự thể hiện một số hành vi: sửa đổi tập tin watchdog, mở một cổng TCP đến một địa chỉ IP cụ thể, lắng nghe các kết nối từ bên ngoài. Để có thể khẳng định chính xác về các hành vi của mã độc này, kết quả thu được từ hai tiến trình 537 và 538 với *Strace* như sau:

- Với tiến trình 537: dễ dàng thấy được đây là một hành vi của backdoor khi kết nối với IP 65.222.202.53 thông qua HTTP ở cổng 80. Chi tiết được trình bày ở Hình 12.

```
20:02:58.494163 ARP, Request who-has 192.168.0.2 tell 192.168.0.150, length 28
20:02:58.494224 ARP, Request who-has 192.168.0.2 tell 192.168.0.150, length 28
20:02:59.493832 ARP, Request who-has 192.168.0.2 tell 192.168.0.150, length 28
20:02:59.493878 ARP, Request who-has 192.168.0.2 tell 192.168.0.150, length 28
20:03:00.493757 ARP, Request who-has 192.168.0.2 tell 192.168.0.150, length 28
20:03:58.598264 ARP, Request who-has 192.168.0.2 tell 192.168.0.150, length 28
20:03:58.598313 ARP, Request who-has 192.168.0.2 tell 192.168.0.150, length 28
20:05:01.682166 ARP, Request who-has 192.168.0.2 tell 192.168.0.150, length 28
20:05:01.682246 ARP, Request who-has 192.168.0.2 tell 192.168.0.150, length 28
20:05:02.681926 ARP, Request who-has 192.168.0.2 tell 192.168.0.150, length 28
20:05:02.681994 ARP, Request who-has 192.168.0.2 tell 192.168.0.150, length 28
20:05:03.681681 ARP, Request who-has 192.168.0.2 tell 192.168.0.150, length 28
20:05:03.681803 ARP, Request who-has 192.168.0.2 tell 192.168.0.150, length 28
```

Hình 9. Kết quả giám sát bằng công cụ Tcpdump

```
open("/dev/watchdog", O_RDWR)           = -1 ENOENT (No such file or directory)
chdir("/")                              = 0
socket(PF_INET, SOCK_DGRAM, IPPROTO_IP) = 3
connect(3, {sa_family=AF_INET, sin_port=htons(53), sin_addr=inet_addr("8.8.8.8")}, 16) = 0
getsockname(3, {sa_family=AF_INET, sin_port=htons(38723), sin_addr=inet_addr("192.168.0.150")}, [16]) = 0
close(3)                                = 0
socket(PF_INET, SOCK_STREAM, IPPROTO_IP) = 3
setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
fcntl(3, F_GETFL)                       = 0x2 (flags O_RDWR)
fcntl(3, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
bind(3, {sa_family=AF_INET, sin_port=htons(48101), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 EADDRINUSE (Address already in use)
connect(3, {sa_family=AF_INET, sin_port=htons(48101), sin_addr=inet_addr("0.0.0.0")}, 16) = -1 EINPROGRESS (Operation now in progress)
rt_sigprocmask(SIG_BLOCK, [CHLD], [INT], 16) = 0
rt_sigaction(SIGCHLD, NULL, {0x10000000, [], SA_NOCLDSTOP}, 16) = 0
nanosleep({5, 0}, 0x7fd6e560)           = 0
rt_sigprocmask(SIG_SETMASK, [INT], NULL, 16) = 0
close(3)                                = 0
socket(PF_INET, SOCK_STREAM, IPPROTO_IP) = 3
setsockopt(3, SOL_SOCKET, SO_REUSEADDR, [1], 4) = 0
fcntl(3, F_GETFL)                       = 0x2 (flags O_RDWR)
fcntl(3, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
bind(3, {sa_family=AF_INET, sin_port=htons(48101), sin_addr=inet_addr("127.0.0.1")}, 16) = 0
listen(3, 1)                            = 0
```

Hình 10: Kết quả phân tích sơ bộ với strace

Các tiến trình trước khi Mirai thực thi

```
PID USER      VSZ STAT COMMAND
  1 root     1412 S    init
  2 root        0 SW   [kthreadd]
  3 root        0 SW   [ksoftirqd/0]
  4 root        0 SW   [watchdog/0]
  5 root        0 SW   [events/0]
  6 root        0 SW   [khelper]
  7 root        0 SW   [async/mgr]
  8 root        0 SW   [sync_supers]
  9 root        0 SW   [bdi-default]
 10 root        0 SW   [kblockd/0]
 11 root        0 SW   [ata/0]
 12 root        0 SW   [ata_aux]
 13 root        0 SW   [khubd]
 14 root        0 SW   [kseriod]
 15 root        0 SW   [kmmcd]
 16 root        0 SW   [cfg80211]
 17 root        0 SW   [khungtaskd]
 18 root        0 SW   [kswapd0]
 19 root        0 SWN  [ksmd]
 20 root        0 SW   [aio/0]
 21 root        0 SW   [crypto/0]
 31 root        0 SW   [scsi_eh_0]
 32 root        0 SW   [scsi_eh_1]
 33 root        0 SW   [mtdblockd]
 42 root        0 SW   [kpsmoused]
 43 root        0 SW   [usbhid_resumer]
 88 root        0 SW   [flush-8:0]
 98 root     1404 S    /sbin/klogd -c 1
234 root     2224 S    /usr/sbin/configd
245 root     2628 S    /sbin/lighttpd -f /etc/lighttpd.conf
273 root     1360 S    /usr/sbin/dropbear
297 root     3444 S    /usr/local/sbin/snmpd -c /etc/snmpd.conf -Ln
375 root     3444 S    /usr/local/sbin/snmpd -c /etc/snmpd.conf -Ln
377 root     3444 S    /usr/local/sbin/snmpd -c /etc/snmpd.conf -Ln
378 root     3444 S    /usr/local/sbin/snmpd -c /etc/snmpd.conf -Ln
421 root     1320 S    /usr/sbin/nmbd -s /etc/smb.conf -D
484 root     1408 S    /bin/sh /usr/local/bin/ntpclient-wrapper time-b.netge
486 root     1404 S    /bin/sleep 30
500 root     1424 S    -sh
503 root     1416 S    sh
```

Các tiến trình sau khi Mirai thực thi

```
PID USER      VSZ STAT COMMAND
  1 root     1412 S    init
  2 root        0 SW   [kthreadd]
  3 root        0 SW   [ksoftirqd/0]
  4 root        0 SW   [watchdog/0]
  5 root        0 SW   [events/0]
  6 root        0 SW   [khelper]
  7 root        0 SW   [async/mgr]
  8 root        0 SW   [sync_supers]
  9 root        0 SW   [bdi-default]
 10 root        0 SW   [kblockd/0]
 11 root        0 SW   [ata/0]
 12 root        0 SW   [ata_aux]
 13 root        0 SW   [khubd]
 14 root        0 SW   [kseriod]
 15 root        0 SW   [kmmcd]
 16 root        0 SW   [cfg80211]
 17 root        0 SW   [khungtaskd]
 18 root        0 SW   [kswapd0]
 19 root        0 SWN  [ksmd]
 20 root        0 SW   [aio/0]
 21 root        0 SW   [crypto/0]
 31 root        0 SW   [scsi_eh_0]
 32 root        0 SW   [scsi_eh_1]
 33 root        0 SW   [mtdblockd]
 42 root        0 SW   [kpsmoused]
 43 root        0 SW   [usbhid_resumer]
 98 root     1404 S    /sbin/klogd -c 1
234 root     2224 S    /usr/sbin/configd
245 root     2628 S    /sbin/lighttpd -f /etc/lighttpd.conf
273 root     1360 S    /usr/sbin/dropbear
297 root     3444 S    /usr/local/sbin/snmpd -c /etc/snmpd.conf -Ln
375 root     3444 S    /usr/local/sbin/snmpd -c /etc/snmpd.conf -Ln
377 root     3444 S    /usr/local/sbin/snmpd -c /etc/snmpd.conf -Ln
378 root     3444 S    /usr/local/sbin/snmpd -c /etc/snmpd.conf -Ln
421 root     1320 S    /usr/sbin/nmbd -s /etc/smb.conf -D
484 root     1412 S    /bin/sh /usr/local/bin/ntpclient-wrapper time-b.netge
500 root     1424 S    -sh
503 root     1424 S    sh
533 root      548 S    /usr/local/bin/ntpclient -s -h time-b.netgear.com -p
537 root      208 S    f4uwcrca8fms0lme371e6ql1
538 root      212 S    f4uwcrca8fms0lme371e6ql1
541 root     1416 R    ps
```

Hình 11: Các tiến trình trước và sau khi Linux/Mirai thực thi

```
nanosleep({1, 0}, 0x7ffdbf80)           = 0
rt_sigprocmask(SIG_SETMASK, [INT], NULL, 16) = 0
socket(PF_INET, SOCK_STREAM, IPPROTO_IP) = 0
fcntl(0, F_GETFL)                       = 0x2 (flags O_RDWR)
fcntl(0, F_SETFL, O_RDWR|O_NONBLOCK)    = 0
connect(0, {sa_family=AF_INET, sin_port=htons(80), sin_addr=inet_addr("65.222.202.53")}, 16)
_newselect(4, [3], [0], NULL, {10, 0})  = 0 (Timeout)
close(0)                                = 0
rt_sigprocmask(SIG_BLOCK, [CHLD], [INT], 16) = 0
rt_sigaction(SIGCHLD, NULL, {0x10000000, [], SA_NOCLDSTOP}, 16) = 0
nanosleep({1, 0}, 0x7ffdbf80)           = 0
rt_sigprocmask(SIG_SETMASK, [INT], NULL, 16) = 0
socket(PF_INET, SOCK_STREAM, IPPROTO_IP) = 0
```

Hình 12. Linux/Mirai kết nối với IP 65.222.202.53 qua cổng 80

```
sendto(0, "E\0\0(\3718\0\0@\6\361\371\300\250\0\226\304\335\t\224\365\6\0\27\304\335\t\224\0\0\0\0"...,
40, MSG_NOSIGNAL, {sa_family=AF_INET, sin_port=htons(23), sin_addr=inet_addr("196.221.9.148")}, 16) = 40
sendto(0, "E\0\0(\3718\0\0@\6?\263\300\250\0\226\245v\333A\365\6\0\27\245v\333A\0\0\0\0"..., 40, MSG_NOS
IGNAL, {sa_family=AF_INET, sin_port=htons(23), sin_addr=inet_addr("165.118.219.65")}, 16) = 40
sendto(0, "E\0\0(\3718\0\0@\6\247K\300\250\0\226\16q\n\257\365\6\0\27\16q\n\257\0\0\0\0"..., 40, MSG_NOS
IGNAL, {sa_family=AF_INET, sin_port=htons(23), sin_addr=inet_addr("14.113.10.175")}, 16) = 40
sendto(0, "E\0\0(\3718\0\0@\6\225f\300\250\0\226\325\322U2\365\6\0\27\325\322U2\0\0\0\0"..., 40, MSG_NOS
IGNAL, {sa_family=AF_INET, sin_port=htons(23), sin_addr=inet_addr("213.210.85.50")}, 16) = 40
sendto(0, "E\0\0(\3718\0\0@\6\263\304\300\250\0\226\264\tX\235\365\6\0\27\264\tX\235\0\0\0\0"..., 40, MS
G_NOSIGNAL, {sa_family=AF_INET, sin_port=htons(23), sin_addr=inet_addr("180.9.88.157")}, 16) = 40
sendto(0, "E\0\0(\3718\0\0@\6a\236\300\250\0\226\232Y\304s\365\6\0\27\232Y\304s\0\0\0\0"..., 40, MSG_NOS
IGNAL, {sa_family=AF_INET, sin_port=htons(23), sin_addr=inet_addr("154.89.196.115")}, 16) = 40
sendto(0, "E\0\0(\3718\0\0@\6\233<\300\250\0\226\272Vj\330\365\6\0\27\272Vj\330\0\0\0\0"..., 40, MSG_NOS
IGNAL, {sa_family=AF_INET, sin_port=htons(23), sin_addr=inet_addr("186.86.106.216")}, 16) = 40
sendto(0, "E\0\0(\3718\0\0@\6!\233\300\250\0\226+\265s\33\365\6\0\27+\265s\33\0\0\0\0"..., 40, MSG_NOSIG
NAL, {sa_family=AF_INET, sin_port=htons(23), sin_addr=inet_addr("43.181.115.27")}, 16) = 40
sendto(0, "E\0\0(\3718\0\0@\6H\217\300\250\0\226k\36\f\276\365\6\0\27k\36\f\276\0\0\0\0"..., 40, MSG_NOS
IGNAL, {sa_family=AF_INET, sin_port=htons(23), sin_addr=inet_addr("107.30.12.190")}, 16) = 40
sendto(0, "E\0\0(\3718\0\0@\6\332Y\300\250\0\226Q\327\224:\365\6\0\27Q\327\224:\0\0\0\0"..., 40, MSG_NOS
IGNAL, {sa_family=AF_INET, sin_port=htons(23), sin_addr=inet_addr("81.215.148.58")}, 16) = 40
sendto(0, "E\0\0(\3718\0\0@\6\35\231\300\250\0\226\274X\346y\365\6\0\27\274X\346y\0\0\0\0"..., 40, MSG_N
OSIGNAL, {sa_family=AF_INET, sin_port=htons(23), sin_addr=inet_addr("188.88.230.121")}, 16) = 40
sendto(0, "E\0\0(\3718\0\0@\6\233\332\300\250\0\226E\344\336\254\365\6\0\27E\344\336\254\0\0\0\0"..., 40
, MSG_NOSIGNAL, {sa_family=AF_INET, sin_port=htons(23), sin_addr=inet_addr("69.228.222.172")}, 16) = 40
```

Hình 13. Linux/Mirai quét cổng telnet để tự động lây nhiễm

- Tiến trình 538 có chức năng quét các cổng telnet (cổng 23) từ các địa chỉ IP khác như 189.34.200.158 và 153.55.105.31,…. Chức năng này phục vụ cho mục đích tự động lây lan sang các thiết bị khác của mã độc này. Chi tiết được trình bày ở Hình 13.

Đến đây, chúng tôi có thể khẳng định, mẫu mã độc này có những hành vi bất thường như mở cổng hậu kết nối với IP 65.222.202.53 và quét cổng telnet để phục vụ cho mục đích lây nhiễm của mã độc này.

VI. KẾT LUẬN

Bộ công cụ *C500-toolkit* thực hiện quy trình mô phỏng đầy đủ nhằm thu thập, phân tích và phát hiện mã độc trong các thiết bị định tuyến. Quy trình này cho phép chạy các firmware thiết bị định tuyến trong môi trường mô phỏng và giám sát các hành vi của firmware này. Từ những thông tin thu thập được, có thể xác định được các hành vi bất thường của mã độc lây nhiễm trong firmware của các thiết bị định tuyến một cách nhanh chóng mà không cần tới thiết bị thật. Với ưu thế sử dụng môi trường mô phỏng để chạy các firmware của nhiều dòng thiết bị định tuyến khác nhau mà không phụ thuộc vào phần cứng thật, bộ công cụ *C500-toolkit* có khả năng phân tích diện rộng với nhiều dòng thiết bị định tuyến khác nhau.

Do tập luật mà nhóm tác giả sử dụng để phát hiện mã độc trên thiết bị định tuyến còn ít, vì vậy chưa thể phát hiện được chính xác mã độc trong một số trường hợp. Trong tương lai gần, nhóm tác giả sẽ bổ sung tập các hành vi bất thường, kết hợp với kỹ thuật học máy áp dụng vào nghiên cứu các dữ liệu thu thập được từ bộ công cụ *C500-toolkit* để phát triển công cụ đầy đủ để tự động xác định mã độc xuất hiện trong firmware của thiết bị định tuyến. Bên cạnh đó, nhóm tác giả sẽ cải tiến *C500-toolkit* để nâng cao khả năng trích xuất, ảo hóa các thiết bị phần cứng nhằm giúp cho môi trường mô phỏng đầy đủ hơn, mô phỏng các firmware của nhiều dòng thiết bị định tuyến, áp dụng cho nhiều loại CPU hơn (ARM, PowerPC,…) nhằm phát hiện các loại mã độc mới xuất hiện trên thiết bị định tuyến trong tương lai gần.

TÀI LIỆU THAM KHẢO

[1] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti, "A Large-Scale Analysis of the Security of Embedded Firmwares", USENIX Security, pp. 95-110, 2014.

[2] C. Kruegel and Y. Shoshitaishvili, "Using Static Binary Analysis To Find Vulnerabilities And Backdoors In Firmware", Black Hat USA, 2015.

[3] D. Davidson, B. Moench, S. Jha, and T. Ristenpart, "FIE on Firmware, Finding vulnerabilities in embedded systems using symbolic execution", USENIX Security, 2013.

[4] T. Ronghua, "An Integrated Malware Detection and Classification System", No. Ph. D. Deakin University, 2011.

[5] A. Moser, C. Kruegel, and E. Kirda, "Limits of Static Analysis for Malware Detection", Computer security applications conference, pp. 421-430, 2007.

[6] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and Classification of Malware Behavior". Springer Berlin Heidelberg, pp. 108-125, 2008.

[7] J. Zaddach, L. Bruno, A. Francillon, and D. Balzarotti, Avatar: "A Framework to Support Dynamic Security Analysis of Embedded Systems Firmwares", NDSS. 2014.

[8] D. Chen, M. Egele, M. Woo, and D. Brumley, "Towards Automated Dynamic Analysis for Linux-based Embedded Firmware", ISOC Network and Distributed System Security Symposium (NDSS), 2016.

[9] L. Frédéric, "An introduction to I 2 C and SPI protocols", IEEE Instrum. Meas. Mag, vol. 12, pp. 8–13, 2009.

[10] E. Volpi, F. Sechi, and T. Cecchini, "System study for a head-up display based on a flexible sensor interface", Sensors and Microsystems, Springer Netherlands, pp. 413–417, 2010.

[11] Firmware mod kit [Online] https://code.google.com/archive/p/firmware-mod-kit/.

[12] Binwalk [Online] http://binwalk.org .

[13] J. Zaddach and A. Costin, "Embedded Devices Security and Firmware Reverse Engineering", Black-Hat USA, 2013.

[14] T. N. Phú, N. H. Trung, and N. Q. Dũng, "Phát triển công cụ dịch ngược firmware trên thiết bị định tuyến", SOIS, 2016.

[15] https://www.busybox.net.

[16] https://strace.io.

[17] F. Fuentes and C. Kar, "Ethereal vs. Tcpdump: a comparative study on packet sniffing tools for educational purpose", J. Comput. Sci. Coll. Consort. Comput. Sci. Coll. USA, Apr. 2005.

[18] F. Bellard, "QEMU, a fast and portable dynamic translator", USENIX Annual Technical Conference, FREENIX Track, pp. 41-46, 2005.

[19] Linux/Mirai [Online] http://www.malwaremustdie.org.

SƠ LƯỢC VỀ TÁC GIẢ

TS. Ngô Quốc Dũng

Đơn vị công tác: Học viện An ninh nhân dân, Bộ Công An.

Email : quocdung.ngo@gmail.com

Quá trình đào tạo: Nhận bằng Kỹ sư tại Đại học Bách Khoa Nantes; Nhận bằng Thạc sỹ tại Đại học Bách Khoa Nantes và Đại học Lyon 2; Bảo vệ Tiến sỹ tại Đại học Bách khoa Grenoble, Cộng Hòa Pháp.

Hướng nghiên cứu hiện nay: Đảm bảo an toàn, an ninh thông tin trên các thiết bị IoT.

ThS. Lê Hải Việt

Đơn vị công tác: Học viện An ninh nhân dân, Bộ Công An.

Email: vietlhsa@gmail.com

Quá trình đào tạo: Nhận bằng Kỹ sư và bằng Thạc sỹ tại Đại học tổng hợp kỹ thuật quốc gia Irkutsk, Liên bang Nga. Đang là nghiên cứu sinh tại Khoa CNTT – Học viện Khoa học và Công nghệ, Viện Hàn lâm khoa học Việt Nam.

Hướng nghiên cứu hiện nay: phân tích phát hiện mã độc trong các thiết bị IoT và ứng dụng.

CN. Trần Hoàng Anh

Đơn vị công tác: Học viện An ninh nhân dân, Bộ Công An.

Email : anhthk55@gmail.com

Quá trình đào tạo: Nhận bằng cử nhân ĐTVT tại Đại học Công nghệ - Đại học Quốc gia Hà Nội.

Hướng nghiên cứu hiện nay: xây dựng hệ thống trích xuất, phân tích dữ liệu từ thiết bị nhớ trong hệ thống nhúng.

Lê Văn Hoàng

Đơn vị công tác: Học viện An ninh nhân dân, Bộ Công An.

Email: levanhoang.psa@gmail.com

Quá trình đào tạo: Sinh viên Khoa Công nghệ và An toàn thông tin, Học viện An ninh nhân dân.

Hướng nghiên cứu hiện nay: phân tích phát hiện mã độc trong hệ điều hành Linux và ứng dụng cho thiết bị nhúng.

Nguyễn Việt Anh

Đơn vị công tác: Học viện An ninh nhân dân, Bộ Công An.

Email : vietanhnguyen142857@gmail.com

Quá trình đào tạo : Sinh viên Khoa Công nghệ và An toàn thông tin, Học viện An ninh nhân dân.

Hướng nghiên cứu hiện nay: phân tích phát hiện mã độc trong hệ điều hành Linux và ứng dụng cho thiết bị nhúng.