Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.32, NO.1, Feb. 2022 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2022.32.1.67

심볼 테이블을 이용한 펌웨어 리눅스 커널 버전 정적 식별 기법*

김 광 준,[1†] 최 여 정,[1] 김 윤 정,[1] 이 만 희[2‡]
[1,2]한남대학교 (대학원생, 교수)

Static Identification of Firmware Linux Kernel Version by using Symbol Table*

Kwang-jun Kim,[1†] Yeo-jeong Choi,[1] Yun-jeong Kim,[1] Man-hee Lee[2‡]
[1,2]Hannam University (Graduate student, Professor)

요 약

장비 도입 시 해당 장비에 설치된 커널의 정확한 버전을 식별하는 것은 매우 중요하다. 특정 커널 버전에 취약점이 발견된 경우 이에 대해 조치 여부를 판단하거나, 특정 커널 버전의 제외 또는 포함 등에 대한 도입 요구 조건이 있는 경우 이를 판단하는데 사용될 수 있기 때문이다. 하지만 많은 시스템 및 네트워크 장비 제조업체들은 공식적으로 배포되고 있는 리눅스 기저 커널을 그대로 사용하지 않고, 장비에 최적화된 펌웨어를 제작하기 위해 커널을 수정하여 사용하므로 리눅스 커널 버전을 판단하기 어려운 상황이 발생한다. 또한, 커널의 패치가 공개될 경우 제조사는 수정한 커널에 패치 내용을 반영하므로, 이런 과정이 지속될 경우 커스터마이징된 커널은 리눅스 기저 커널과 매우 다른 형상이 된다. 따라서, 특정 파일 존재 여부 등의 단순한 방법으로는 리눅스 커널을 정확히 식별하기 어렵다. 새로운 리눅스 커널 버전이 공개될 때는 새로운 함수가 포함되기도 하고 기존 함수가 삭제되기도 한다. 본 논문에서는 심볼 테이블에 저장된 함수명을 이용하여 펌웨어 커널 버전의 정적 식별 방안을 제안하고 실험을 통해 그 실효성을 증명하였다. 100개의 리눅스 펌웨어를 대상으로 한 실험에서 99%의 정확도로 리눅스 커널 버전을 식별할 수 있었다. 본 연구를 통해 펌웨어 이용 환경의 보안성 향상에 기여할 것으로 기대한다.

ABSTRACT

When acquiring a product having an OS, it is very important to identify the exact kernel version of the OS. This is because the product's administrator needs to keep checking whether a new vulnerability is found in the kernel version. Also, if there is an acquisition requirement for exclusion or inclusion of a specific kernel version, the kernel identification becomes critical to the acquisition decision. In the case of the Linux kernel used in various equipment, sometimes it becomes difficult to pinpoint the device's exact version. The reason is that many manufacturers often modify the kernel to produce their own firmware optimized for their device. Furthermore, if a kernel patch is applied to the modified kernel, it will be very different from its base kernel. Therefore, it is hard to identify the Linux kernel accurately by simple methods

Received(11. 29. 2021), Modified(01. 25. 2022), Accepted(01. 25. 2022)
* 본 논문은 2021년도 한국정보보호학회 충청지부 학술대회에 발표한 우수논문을 개선 및 확장한 것임
* 이 논문은 ETRI부설연구소의 위탁연구과제[2021-088]로 수행한 연구결과입니다.(This work is the result of commissioned research project supported by the affiliated institute of ETRI[2021-088].)
* 이 논문의 내용을 발표하는 때에는 ETRI부설연구소에서 수행한 위탁결과임을 밝혀야 합니다.(When giving a presentation on this work, the presenter has to clarify that it is the result of the research commissioned by the affiliated institute of ETRI)
* 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(2021R1A4A200181).
† 주저자, kimkwangjun.kr@gmail.com
‡ 교신저자, manheelee@hnu.kr(Corresponding author)

such as a specific file existence test. In this paper, we propose a static method to classify a specific kernel version by analyzing function names stored in the symbol table. In an experiment with 100 Linux devices, we correctly identified the Linux kernel version with 99% accuracy.

Keywords: Firmware Kernel Version Identification, Static Analysis, Symbol Table, System map, Kallsyms

I. 서 론

IoT 산업이 발전함에 따라 임베디드 시스템에 사용되는 펌웨어의 수도 폭발적으로 증가하고 있다. 펌웨어란 하드웨어 장비의 효율적인 구동 및 제어를 위해 사용되는 소프트웨어를 의미한다. 이러한 펌웨어는 가정용 IoT 및 스마트 기기뿐만 아니라 네트워크 장비에 이르기까지 다양한 분야에서 사용되고 있다.

제조업체는 장비를 보다 안정적이고 효율적으로 동작시키기 위해 리눅스 커널을 바탕으로 펌웨어를 제작한다. 커널은 하드웨어와 프로세스를 연결하는 핵심 인터페이스로 하드웨어의 자원을 관리하고 메모리를 제어하는 등 운영체제의 핵심 역할을 담당한다[1]. 즉, 커널은 하드웨어를 직접 제어할 수 있으므로 보안 위협으로 더욱 보호되어야 한다.

하지만 제조업체가 장비에 최적화된 펌웨어를 제작하는 과정에서 공식적으로 배포되고 있는 오픈소스 기반의 리눅스 커널을 그대로 사용하지 않고, 제조사의 장비에 맞게 자체적으로 커스터마이징하여 사용하는 사례가 다수 발견되고 있다[2][3]. 이 경우 기저 커널은 취약점을 개선한 보안패치가 지속적으로 진행되고 있으나, 커스터마이징한 커널은 제조사의 반영이 개입되어야 하므로 보안패치가 상대적으로 더디게 진행되고 있는 실정이다. 따라서 펌웨어의 운영체제와 커널 버전을 정확하게 식별할 수 있는 기술이 필요하다.

하지만 기존 커널 식별 도구는 동적으로 메모리의 시그니처를 이용하거나, 문자열 패턴 탐지를 바탕으로 버전을 식별한다[4][5][6][7][8]. 펌웨어는 특성상 동적 분석이 어려워 정적 분석을 수행해야 하나, 문자열 기반의 패턴 탐지는 커널 컴파일 시 손쉽게 버전 정보 변조가 가능하다는 한계가 존재한다. 따라서 단순한 문자열 기반의 패턴 탐지가 아닌, 커널의 바이너리에서 시그니처를 추출하여 정적으로 버전을 식별할 수 있는 연구개발이 필요하다.

본 논문은 다음과 같이 구성된다. 먼저 2장에서는 커널 식별에 활용할 심볼 테이블에 관한 개념적 지식 및 관련 연구를 기술하고, 3장에서는 심볼 테이블을 이용한 펌웨어 커널 식별 방안에 대해 소개한다. 그 후 실험을 통한 연구의 실효성을 입증한 후 4장에서 결론을 맺는다.

II. 관련 연구

2.1 Symbol Table

심볼 테이블이란 함수 및 변수의 이름과 타입을 주소와 바인딩하여 저장한 테이블을 의미한다(Fig.1.)[9]. 그중 커널의 모듈을 구현하는데 필요한 전역 함수 및 변수의 주소와 외부에서 참조할 수 있는 함수의 심볼을 저장한 테이블을 커널 심볼 테이블이라고 한다[10]. 커널 심볼 테이블은 커널의 모듈 및 함수 간의 효율적인 협업을 위해 활용된다.

2.2 System.map

system.map 파일은 리눅스 커널 컴파일 시 커널에 포함된 모듈의 심볼 정보를 기록한 심볼 테이블 파일이다[11]. 이 파일은 커널 부팅에는 영향을 주지 않고, 부팅 이후 커널을 디버깅하는 목적으로 사용된다.

커널 부팅 과정에서 커널 패닉(Kernel Panic) 등 커널과 관련된 문제가 발생했을 경우, 커널은 오류의 발생 위치를 16진수 주소로 반환한다. 이때, 커널 로그 메시지를 출력하는 klogd 데몬은 system.map 파일을 이용하여 오류의 위치를 16진수의 주소 대신 함수의 이름으로 반환하여 문제가 발생한 지점을 보다 편리하게 알 수 있도록 지원한다.

하지만 system.map 파일은 커널 컴파일 시 생성되는 심볼 테이블이므로, 주소에 해당하는 심볼이

```
root@yeojeong-VirtualBox:/proc# head kallsyms
0000000000000000 A fixed_percpu_data
0000000000000000 A __per_cpu_start
0000000000001000 A cpu_debug_store
0000000000002000 A irq_stack_backing_store
0000000000006000 A cpu_tss_rw
000000000000b000 A gdt_page
000000000000c000 A exception_stacks
0000000000010000 A entry_stack_storage
0000000000011000 A espfix_waddr
0000000000011008 A espfix_stack
```

Fig. 1. Symbol Table(kallsyms) Structure

실제로 존재하지 않을 수 있다. 따라서 커널 디버깅 시 참고 용도로만 사용한다.

2.3 Kallsyms

kallsyms 파일은 리눅스 커널이 부팅될 때 커널 메모리에 동적으로 접근하여 얻을 수 있는 커널 모듈의 공개 심볼 정보를 저장한 파일이다[12]. 이 파일은 system.map 파일과 달리 부팅이 정상적으로 이루어진 이후에 /proc 폴더 안에 생성된다.

이 파일은 커널 컴파일 시, 리눅스 커널 바이너리인 vmlinux에서 심볼 정보를 추출하여 어셈블러 파일을 생성한 후 컴파일되어 원래의 바이너리와 링크된다. 이후 이 커널 심볼 테이블 정보는 부팅 시 /proc/kallsyms 파일을 생성하기 위해 사용되며, 커널의 주소를 무작위로 변환하는 KASLR(Kernel Address Space Layout Randomization)를 적용하여도 재배치된 주소를 반환하기 때문에 kallsyms에 저장된 주소는 실제 심볼이 저장된 메모리 주소와 일치한다.

리눅스 커널은 2.6.10 버전 이후부터 커널의 어셈블리어 소스에 kallsyms 심볼 정보를 포함하는 옵션이 기본적으로 활성화되어 있다. 따라서 해당 옵션이 활성화된 상태로 컴파일되었다면, 펌웨어에서 커널 바이너리를 추출한 후 kallsyms 심볼 정보를 추출할 수 있다.

2.4 커널 동적 식별 연구

Yufei Gu(2012)의 OS-SOMMELIER: Memory-Only Operating System Fingerprinting in the Cloud 연구는 커널 시그니처를 활용하여 VM(Virtual Machine)에서 구동 중인 운영체제 종류 및 버전을 식별하는 동적 분석 방법을 제안하였다[4]. 본 연구의 핵심은 다수의 PGD(Page Global Directory) Entry가 매핑된 PDE(Page Directory Entry)는 커널 메모리일 가능성이 높다는 특징으로, 커널 메모리를 덤프한 후 커널 페이지가 매핑된 PGD를 시그니처로 활용하여 커널을 식별한다. 따라서 구동 중인 VM의 메모리를 덤프 후 메모리에서 다수의 PGD Entry가 매핑된 PDE를 찾는다. 그 후 커널 코드가 참조하는 주소 및 호출 관계를 MD5로 해시화하여 커널의 시그니처를 생성한 뒤 이를 바탕으로 커널 종류 및 버전 매칭을 수행한다.

Vassil Roussev(2014)의 Image-based kernel fingerprinting 연구는 커널 버전 식별을 위한 핑거프린트 생성 도구인 Sdkernel을 제안하였다[5]. 해당 논문은 리눅스의 vmlinux 이미지를 이용하여 커널 버전별 고유 블록을 시그니처로 사용한다. 이때 시그니처를 위해 먼저 임시 파일 스토리지 tmpfs에 로드된 vmlinux 이미지를 일정 크기 블록으로 분할한다. 그 후 0으로 채워진 블록과 중복된 블록을 제거하여 걸러진 고유 블록에 대해 sdhash를 사용하여 해시값을 생성한다. 논문의 실험 결과 144개의 커널에서 112개 커널이 1,000개 이상의 고유 블록을 생성함에 따라 시그니처로의 활용 가능성을 충분히 보여주었다. 이렇듯 vmlinux 이미지를 통해 시그니처를 생성한 후 비교하고자 하는 대상 커널의 RAM 스냅샷을 똑같은 크기의 블록으로 잘라 sdhash의 해시값을 생성한다. 그 후 두 해시값의 sdhash 유사도를 비교하여 가장 유사한 커널 버전을 식별한다.

Manish Bhatt(2018)의 Leveraging relocations in ELF-binaries for Linux kernel version identification 연구는 리눅스 커널 버전을 식별하기 위해 커널 바이너리의 재배치 항목(Relocatable Entries)을 기반으로 시그니처를 생성하는 아이디어를 제안하였다[6]. 해당 논문은 readelf -r 명령줄 도구를 사용하여 리눅스 커널의 ELF 실행파일인 vmlinux의 재배치 항목 중 .rel.text 섹션에서 R_386_32 유형의 오프셋과 .text 섹션의 포인터 오프셋을 추출하였다. 그 후 추출한 오프셋을 바탕으로 시그니처를 생성하고, VM에서 구동 중인 리눅스에서 덤프한 메모리 스냅샷과 비교하여 커널 페이지 간의 균일성을 확인함으로써 커널 버전 및 커널이 매핑된 기본 주소(Base-address)를 식별한다. 분석 시간은 리눅스 커널 4점대를 기준으로 약 1000~1150초가 소요되었으며, 약 99%의 식별 정확도를 도출하였다.

Mihai Christodorescu(2009)의 Cloud Security Is Not (Just) Virtualization Security 연구는 루트킷 탐지를 위해 IDT(Interrupt Descriptor Table)를 기반으로 시그니처를 생성하는 아이디어를 제안하였다[7]. IDT는 핸들러 코드(인터럽트 및 예외 처리 코드)의 주소를 가리키는 테이블로 레지스터 IDTR(Interrupt Descriptor Table Register)에 저장된 IDT의

기본 주소를 바탕으로 IDT 주소를 확인할 수 있다. 이때 OS 유형, 버전 및 패치마다 핸들러 코드가 상이하다는 특징을 이용하여 동적으로 OS 및 커널 버전을 식별할 수 있는 부가적인 연구 성과를 소개하였다. 본 연구는 레지스터 IDTR에 저장된 IDT의 주소를 확인하여 핸들러 코드를 수집하고, 핸들러 코드를 기반으로 해시값을 생성한다. 그 후 생성한 해시값을 비교하여 OS 및 커널 버전을 식별한다.

Arati Baliga(2008)의 Automatic Inference and Enforcement of Kernel Data Structure Invariants 연구는 루트킷을 탐지하기 위해 커널의 심볼 테이블 정보를 추출하여 시그니처로 활용하는 Gibraltar를 제안하였다[8]. 심볼 테이블 정보는 커널 컴파일 시 생성되는 system.map 파일에서 획득할 수 있으며 이 파일은 커널 심볼 및 유형 정의를 포함한다. 이때, 커널 데이터 구조를 나타내는 커널 심볼 및 유형 정의를 시그니처로 사용하여 커널을 식별할 수 있다. 해당 논문에서 제안한 Gibraltar은 먼저 분석 대상이 되는 커널을 가상환경에 실행한다. 그 후 Page Fetcher 단계에서 가상환경에 구동 중인 분석 대상의 물리 메모리 주소에 DMA(Direct Memory Access)한다. DMA를 통해 분석 대상의 Raw Memory Page를 DSE(Data Structure Extractor)에 전달한 후 메모리 페이지를 분석하여 분석 대상 커널의 데이터 구조를 추출한다. 추출한 데이터 구조와 커널의 system.map 파일의 데이터 유형을 바탕으로 데이터 구조를 식별하고 void 포인터 등 노이즈를 제거한 후 Kernel Snapshot에 저장한다. 그 후 Kernel Snapshot을 바탕으로 불변량(invariant) 데이터 구조를 추출하여 시그니처화 한 뒤 Monitor 과정에서 커널의 데이터 구조를 모니터링하며 앞서 추출한 시그니처와 비교하여 불변량이 다르거나 변경되었을 경우 에러 메시지를 출력하는 방식으로 루트킷을 탐지한다.

해당 연구는 루트킷 탐지가 주된 목적이었지만 그 과정에서 system.map 심볼을 이용한 커널 시그니처 생성 방안을 사용하였다. 따라서 본 논문에서의 커널 식별을 위한 시그니처 생성 접근 방식과 유사한 면이 있다.

앞서 소개한 논문들은 모두 동적 분석을 이용하여 커널을 식별하였다. 하지만 펌웨어는 하드웨어에 종속적인 특성상 동적 분석이 어려워 정적 분석을 수행해야 한다. 따라서 커널의 바이너리에서 시그니처를 추출하여 정적으로 버전을 식별할 수 있는 연구개발이 필요하다.

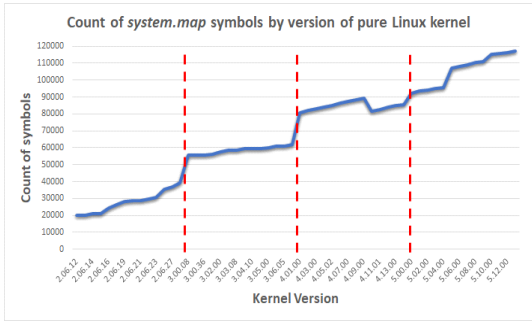

Fig. 2. Count of *system.map* symbols by version of pure Linux kernel

III. 커널의 심볼 시그니처 활용 방안

본 절에서는 커널의 심볼 정보를 시그니처로서의 활용 가능성을 확인하기 위해 순수 리눅스 커널을 버전별로 컴파일하여 system.map 심볼 정보를 확보한다. 그 후 커널 버전 별 심볼 개수 차이를 비교한다. 또한, 펌웨어의 kallsyms 포함 현황을 확인하기 위하여 제조사가 최근 1년 이내에 배포한 100종의 펌웨어를 수집 분석함으로써 심볼 시그니처 활용 방안의 가능성을 증명한다.

3.1 리눅스 커널 버전별 심볼 현황

리눅스 커널 컴파일 시 생성되는 system.map 심볼 정보를 이용한 시그니처 활용 가능성을 확인하기 위해 커널 버전별 system.map 심볼 현황을 정리하면 Fig.2.와 같다. 본 실험은 리눅스 커널 공식 분산 버전 관리 저장소에서 총 90개의 순수 커널 소스 코드와 49개의 리눅스 커널 소스 코드를 다운로드 및 컴파일한 후 system.map 파일을 획득하여 심볼 개수를 분석하였다.

분석 결과의 예시로 리눅스 커널 2.6.12 버전의 system.map 심볼 정보는 19,854개이며, 3.0.12 버전은 55,611개, 4.1.0 버전은 80,590개, 5.13.0 버전의 심볼의 개수는 116,955개다. 즉, 커널의 고유 심볼 개수가 충분히 차이 나므로 심볼 정보를 시그니처로 활용하면 커널 버전을 식별할 수 있음을 확인할 수 있다.

Table 1. Count of *kallsyms* included in firmware

Device Classification	Count including kallsyms	Count without kallsyms
SOHO Router	37	28
VPN Router	4	-
Switch	4	-
AP	3	-
Extender	4	2
Controller	1	-
NAS	9	1
IP Camera	-	2
NVR	3	1
Wireless Video Transmitter	1	-
Total	**66**	**34**

3.2 펌웨어 내 kallsyms 포함 현황

펌웨어에 사용된 리눅스 커널의 심볼 정보를 이용하기 위해선 먼저 kallsyms의 심볼 정보를 추출해야 한다. 리눅스 커널 2.6.10 버전 이후 기본적으로 커널 컴파일 시 kallsyms 심볼 정보를 포함하는 옵션은 활성화되어 있지만, 펌웨어의 용량 절약과 디버깅 방지 등의 이유로 제조사가 해당 기능을 비활성화하는 경우가 존재한다. 따라서 가정용 라우터, IP 카메라 등 시중에서 쉽게 구할 수 있는 펌웨어 100종을 수집한 후 kallsyms 심볼 포함 여부를 분석하였다.

분석 결과 펌웨어 100종 중 kallsyms 심볼을 포함한 펌웨어는 66개로 절반 이상의 펌웨어에서 kallsyms 심볼 정보를 추출할 수 있었다(Table 1.). 따라서 분석 대상의 펌웨어 파일에서 kallsyms 심볼이 포함되어 있을 경우 리눅스 커널 식별의 시그니처로 활용할 수 있음을 확인하였다.

IV. 펌웨어 커널 버전 식별

본 절에서는 수집한 순수 리눅스 커널의 버전별 system.map 심볼 데이터와 펌웨어에서 추출한 kallsyms 파일의 심볼 데이터를 비교하여 펌웨어에 사용된 리눅스 커널 버전을 식별할 수 있는 기법을 제안한다. 그 후 실험을 통해 제안한 기법의 실효성과 가능성을 증명한다.

4.1 식별 기법

심볼 테이블을 시그니처로 활용하여 커스터마이징된 리눅스의 기저 커널 버전을 식별하기 위해 system.map에서 추출한 심볼 데이터를 이용하여 데이터베이스를 구축한다. 본 연구에서는 커널 버전을 식별하기 위한 시그니처로 커널 버전 2의 심볼 이름 74,876개, 커널 버전 3의 심볼 이름 103,703개, 커널 버전 4의 심볼 이름 138,673개, 커널 버전 5의 심볼 이름 162,520개, 합계 225,062개의 데이터베이스를 구축하였다.

구축한 데이터베이스를 바탕으로 미지의 커널 버전을 식별하기 위해 함수명 비교 알고리즘을 제안한다. 이 알고리즘은 '모든 함수', '공통 함수', '유니크 함수', '코어 함수'의 함수명을 바탕으로 커널 버전을 식별한다. 그 결과 기존의 문자열 기반 커널 탐지 도구가 식별하지 못한 변조된 커널 버전을 식별할 수 있다.

'모든 함수' 알고리즘은 순수 리눅스 커널 90개를 포함하여 라즈베리파이 리눅스 커널 등 수집한 리눅스 커널 139개에 포함된 모든 함수를 의미한다. 분석 대상 커널에서 kallsyms 심볼 테이블의 함수명을 추출한 후 각 함수를 포함하는 커널 버전을 데이터베이스의 질의하여 개수를 카운팅한다.

'공통 함수' 알고리즘은 각 메인 버전의 모든 마이너 버전에서 공통으로 포함하고 있는 함수를 의미한다. 분석 대상에서 kallsyms 심볼 테이블의 함수명을 추출한 후 각 함수가 리눅스 커널 2~5 버전의 공통 함수인지 차례로 데이터베이스에 질의하여 개수를 카운팅한다.

'유니크 함수' 알고리즘은 각 메인 버전에 유일하게 존재하는 함수를 의미한다. 분석 대상 커널에서 kallsyms 심볼 테이블의 함수명을 추출한 후 각 함수가 리눅스 커널 2~5 버전의 유니크 함수인지 차례로 데이터베이스에 질의하여 개수를 카운팅한다.

'코어 함수'는 커널 소스코드 중 하드웨어에 종속되지 않으며, 동시에 운영체제의 핵심 기능을 수행하는 영역의 함수를 의미한다. 커널 업데이트 시, 코어 함수가 추가로 생성 및 삭제되기도 하므로 코어 함수는 커널 버전을 식별하는데 중요한 시그니처로 활용될 수 있다. 코어 함수 데이터베이스를 구축하기 위해 42개의 커널 소스코드를 분석하여 init, mm, kernel, ipc 영역에서 코어 함수명을 수집하였다.

'코어 함수' 데이터베이스 구축을 위해 분석한 코

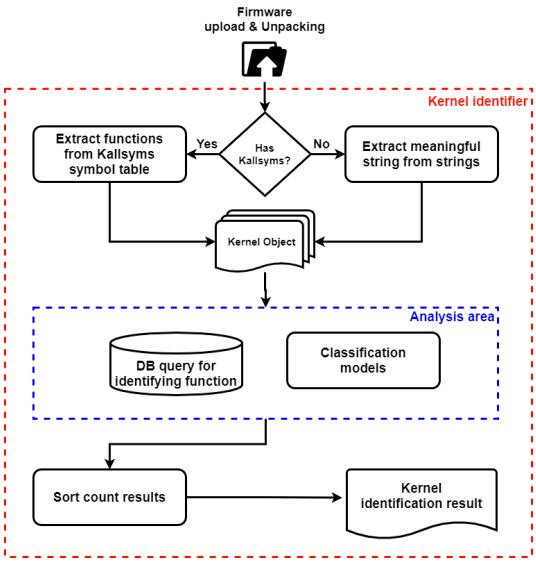

Fig. 3. Kernel Identifier Module Flowchart

어 영역 'init'는 하드웨어에 독립적인 커널 초기화 루틴의 소스 코드가 구현되어 있고, 'mm'은 하드웨어에 독립적인 메모리 관리(가상메모리, 페이징, 스와핑 등) 루틴이 구현되어 있다. 'kernel'는 하드웨어에 독립적인 커널 관리(스케쥴러, 시그널 관리 등) 루틴이 구현되어 있으며, 'ipc'는 프로세스간 통신(세마포어, 공유 메모리, 통신 프로토콜 등) 루틴이 구현되어 있다.

구축한 데이터베이스를 이용하여 미지의 커널 버전을 식별하는 알고리즘은 Fig.3.과 같은 흐름으로

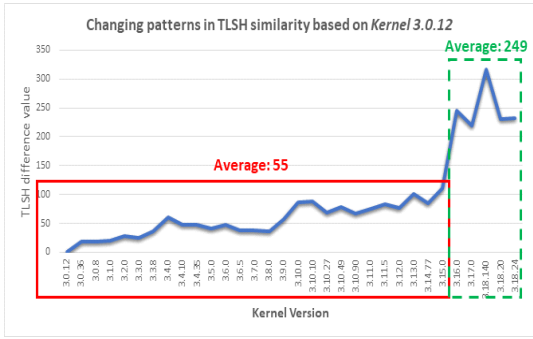

Fig. 4. Changing patterns in TLSH similarity based on Kernel Version 3.0.12

수행한다. 펌웨어 업로드 시 언패킹 및 커널 바이너리를 추출한 후 kallsyms가 포함되었을 경우, 심볼 테이블의 정보를 추출한다. 이때, kallsyms가 존재하지 않는다면 바이너리에 포함된 문자열 중 의미있는 문자열을 추출하여 심볼 테이블의 함수명과 동일하게 취급한다. 그 후 알고리즘별로 데이터베이스에 질의하여 커널에 포함된 각 함수의 버전을 카운팅한 뒤 가장 많이 식별된 커널 버전을 최종 버전으로 결정한다.

이때, 지속적인 버전 업데이트 수행 시, 커널의 후반대 마이너 버전은 다음 메이저 버전의 초기 버전과 유사하다는 특징이 있다. 따라서 비슷한 데이터일수록 해시값이 유사하게 나오는 해시 알고리즘인 TLSH(Trend Micro Locality Sensitive Hashing)를 이용하여 커널 바이너리의 유사도 분석을 수행하였다[13]. 그 결과 커널 버전 3과 5는

Table 2. Kernel identification result by algorithm

Index	Category	Model	Class	String	Identification Result	Unique Functions	Common Functions	All Functions	Core Functions
1	AP	W*	Kallsyms	3.3.8	3	3.A	3.A	3.A	3.B
2	AP	E*	Kallsyms	3.3.8	3	3.A	3.A	3.A	3.B
3	AP	E*	Kallsyms	2.6.36	2	2	2	2	2
4	NVR	H*	String	4.9.37	4	4	4	4	5.B
...									
96	Router	C*	String	3.3.8	3	3.A	3.A	3.A	3.B
97	Router	C*	Kallsyms	3.3.8	3	3.A	3.A	3.A	3.B
99	Switch	D*	Kallsyms	2.6.36	2	2	2	2	2
100	Switch	G*	Kallsyms	3.18.24	3	3.B	3.A	3.B	3.B
Count of Kallsyms		**66/100**	**Accuracy**		**99%**	**99%**	**86%**	**87%**	**78%**
Count of String		**34/100**	**Algorithm**		**Result**	**Unique**	**Common**	**All**	**Core**

Table 3. Experimental Environment

CPU	Intel Xeon E5-2620 CPU @ 2.40GHz
Memory	Hynix DDR4-2133 32GB
OS	Ubuntu 18.04.1 LTS

전/후반부의 커널 유사도 양상이 뚜렷함을 확인하였다(Fig.4.). 따라서 커널 전/후반부 차이에 따라 커널 버전 3과 5는 2구간으로 나누어 분류함으로써 식별의 정확도를 향상했다. 이때, 커널 버전 3.0 ~ 3.15까지는 3.A, 3.16~3.19는 3.B로 분류하였으며, 커널 버전 5.0 ~ 5.9까지는 5.A, 5.10 ~ 5.15는 5.B로 분류하였다.

4.2 실험 결과

본 논문에서 제안한 심볼 테이블을 이용한 펌웨어 커널 정적 식별 알고리즘의 실효성을 검증하기 위하여 앞서 수집한 펌웨어 100종에 대하여 커널 버전 식별 실험을 수행하였다. 그 결과 '유니크 함수' 알고리즘 99%, '공통 함수' 알고리즘 86%, '모든 함수' 알고리즘 87%, '코어 함수' 알고리즘 78%의 정확도를 도출하였으며, 최대 99%의 정확도로 커널 버전을 식별할 수 있었다. 정확도를 측정한 본 실험은 문자열 기반의 버전 정보가 정확하다는 가정하에 수행하였다(Table 2.).

결과표의 카테고리는 기기의 종류를 의미하며 전체 종류는 Table 1.의 기기 분류와 같다. 모델은 펌웨어 사용 기종의 첫 글자만 표기하였다. 또한, 분석 대상인 kallsyms 심볼 정보의 포함 여부에 따라 클래스를 Kallsyms와 String으로 분류하였으며, 문자열 기반의 버전 정보를 String으로 표기하였다.

본 실험에 사용한 시스템은 Table 3.과 같고, 펌웨어에 포함된 커널 버전 식별을 위한 분석 소요시간은 Table 4.와 같다. 용량이 서로 다른 임의의 펌웨어 3개의 분석 소요시간은 평균 약 6 ~ 9초로 매우 빠르게 커널의 버전 식별이 가능하다. 최초 분석 시 커널에서 추출한 kallsyms 정보를 json 형태로 캐싱하기 때문에, Table 4.의 Reanalysis time과 같이 재분석의 소요시간을 파일 용량 15.3MB 기준 최대 33% 감소하였다. 또한, 다중 파일 분석의 성능 향상을 위해 모듈 실행 시 최초 1회 DB의 내용을 모두 메모리에 올림으로써 초기 구동 시간은 약 6초 정도 소요되지만, 그 이후론 매우 빠르게 분석을 수행할 수 있게 된다.

V. 결 론

본 논문은 심볼 테이블을 이용한 펌웨어 커널 정적 식별 기법을 제안하였다. 기존의 문자열 패턴 탐지를 이용한 커널 식별은 변조가 쉽다는 한계점이 존재하나, 심볼 테이블을 이용한 커널 식별은 고유한 심볼 정보를 이용하므로 매우 높은 정확도로 버전을 식별할 수 있다. 그를 위해 90개의 순수 리눅스 커널과 49개의 라즈베리파이용 리눅스 커널을 컴파일한 후 system.map 파일을 획득하여 심볼 데이터베이스를 구축했다. 또한, 수집한 펌웨어 100종과 제안한 알고리즘을 통해 99%의 정확도로 커널 버전을 식별할 수 있음을 증명하였다. 본 연구결과는 낮은 리눅스 커널 버전 사용에 따른 취약점 존재와 연계될 수 있는 문제이므로, 리눅스 커널을 기반으로 제작된 펌웨어를 도입했거나 도입 중인 조직에서 반드시 확인해야 할 사안으로써 그 중요성이 있다고 할 수 있다.

향후 연구 방향으로는 본 연구의 알고리즘으로 이용된 심볼 테이블의 함수명 외 다른 시그니처를 더욱 개발하여 커널 버전 식별의 정확도와 해상도를 높일 예정이다. 또한, 개발한 시그니처를 머신러닝과 연계

Table 4. Analysis Time for Kernel Identification

Manufacturer - Firmware Name	File Size (MB)	Source of Functions	Initial analysis time	Reanalysis time (Caching)
Ne* - M*	1.93	String	7s	6s
ip* - T*	4.88	Kallsyms	9s	7s
TP* - C*	15.30	Kallsyms	9s	6s
Kernel identify		**Total analysis time**	**DB Loading time**	**Kernel identify time**
Initial analysis time		**about 8.33s**	6.7s	2.5s
Reanalysis time		**about 6.33s**	6.0s	0.2s

할 경우 펌웨어 분류 및 커널 버전 식별 측면에서 의미 있는 결과를 도출할 수 있을 것으로 예상한다.

References

[1] David A. Rusling, "The Linux Kernel," Version 0.8-3, http://pdinda.org/ics/doc/linux-kernel.pdf, Jan. 1999.

[2] Asif Shaik, "Google slams Samsung for making unnecessary changes to Linux kernel code," SamMobile, https://www.sammobile.com/news/google-slams-samsung-making-unnecessary-changes-linux-kernel-code, Feb. 2020.

[3] Jean-Luc Aufranc, "Allwinner News - Root Exploit in Linux and Fake Pine A64 Boards," CNX Software, https://www.cnx-software.com/2016/05/13/allwinner-news-root-exploit-in-linux-and-fake-pine-a64-boards, May. 2016.

[4] Yufei Gu, YangChun Fu, Aravind Prakash, Zhiqiang Lin and Heng Yin, "OS-SOMMELIER: Memory-Only Operating System Fingerprinting in the Cloud," Proceedings of the Third ACM Symposium on Cloud Computing, pp. 1-13, Oct. 2012.

[5] Vassil Roussev, Irfan Ahmed and Thomas Sires, "Image-based kernel fingerprinting," Digital Investigation, vol. 11, pp. 13-21, Aug. 2014.

[6] Bhatt, Manish, and Irfan Ahmed, "Leveraging relocations in ELF-binaries for Linux kernel version identification," Digital Investigation, vol. 26, pp. 12-20, Jul. 2018.

[7] Mihai Christodorescu, Reiner Sailer, Douglas Lee Schales, Daniele Sgandurra and Diego Zamboni, "Cloud security is not (just) virtualization security: a short paper," Proceedings of the 2009 ACM workshop on Cloud computing security, pp. 97-102, Nov. 2009.

[8] Arati Baliga, Vinod Ganapathy and Liviu Iftode, "Automatic Inference and Enforcement of Kernel Data Structure Invariants," 2008 Annual Computer Security Applications Conference (ACSAC) IEEE, pp. 77-86, Dec. 2008.

[9] Linuxbase, "Symbol Table," https://refspecs.linuxbase.org/elf/gabi4+/ch4.symtab.html, Jan. 2022.

[10] Alessandro Rubini and Jonathan Corbet, Linux Device Drivers, 2nd Ed., O'Reilly Media, Inc., pp. 27-29, Jun. 2001.

[11] Linux Kernel Newbies, "System.map," https://kernelnewbies.org/FAQ/System.map, Dec. 2017.

[12] Unix, "kallsyms - Extract all kernel symbols for debugging," https://www.unix.com/man-page/redhat/8/kallsyms, Apr. 2000.

[13] Oliver, Jonathan, Chun Cheng and Yanggui Chen, "TLSH—a locality sensitive hash," 2013 Fourth Cybercrime and Trustworthy Computing Workshop. IEEE, pp. 7-13, Nov. 2013.

〈저자소개〉

김 광 준 (Kwang-jun Kim) 학생회원
2017년 2월: 한남대학교 컴퓨터공학과 학사
2019년 2월: 한남대학교 컴퓨터공학과 석사
2019년 3월~현재: 한남대학교 컴퓨터공학과 박사과정
〈관심분야〉 정보보호, 침입 탐지, 네트워크/시스템/공급망 보안

최 여 정 (Yeo-jeong Choi) 학생회원
2021년 2월: 한남대학교 수학과, 컴퓨터통신무인기술학과 졸업
2020년 3월~현재: 한남대학교 컴퓨터공학과 학석사연계과정
〈관심분야〉 양자암호, 취약점 탐지

김 윤 정 (Yun-jeong Kim) 학생회원
2021년 2월: 한남대학교 수학과 및 컴퓨터통신무인기술학과 졸업
2020년 3월~현재: 한남대학교 컴퓨터공학과 학석사연계과정
〈관심분야〉 시스템 보안, 펌웨어 분석, 취약점 분석, 양자암호

이 만 희 (Man-hee Lee) 종신회원
1995년 2월: 경북대학교 컴퓨터공학과 공학사
1997년 2월: 경북대학교 공학석사
2008년 8월: Texas A&M 대학교 컴퓨터공학과 공학박사
1997년~2003년: 한국과학기술정보연구원 연구원
2008년~2009년: Cisco Systems, San Jose
2010년~2012년: 국가보안기술연구소 선임연구원
2012년~현재: 한남대학교 교수
〈관심분야〉 네트워크/시스템/스마트폰/공급망 보안, 고성능 시스템, 컴퓨터교육