Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.33, NO.2, Apr. 2023 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2023.33.2.337

웹어셈블리를 활용한 커널 모듈 보안성 강화*

임 하 정,[1†] 이 호 준[2‡]
[1,2]성균관대학교 (대학원생, 교수)

Enhancing Kernel Module Security Using WebAssembly*

Hajeong Lim,[1†] Hojoon Lee[2‡]
[1,2]Sungkyunkwan University (Graduate student, Professor)

요 약

리눅스를 비롯한 현대 OS들은 모놀리식 커널디자인을 채택하여 높은 확장성을 보여주지만, 모든 메모리 공간을 공유하기 때문에 취약한 보안을 가지고 있었다. 본 연구는 웹어셈블리를 활용하여 커널 내부에서 격리된 커널 모듈을 제시한다. 웹어셈블리는 메모리 안전성을 보장하면서도 저수준 명령어 집합을 정의하여 높은 성능을 보여주는 가상머신을 제공한다. 본 논문에서는 웹어셈블리 실행환경을 커널 내부에 구현하여 개발자가 커널 모듈의 동작을 제어할 수 있도록 허용하고 더 높은 보안성을 달성하였다.

ABSTRACT

Modern OSs, including Linux, show high scalability by adopting a monolithic kernel design, but have weak security because they share all memory space. This study presents a kernel module that are isolated inside the kernel using WebAssembly. WebAssembly provides a high-performance virtual machine by defining a low-level instruction set while guaranteeing memory safety. In this paper, the WebAssembly execution environment is implemented inside the kernel, allowing developers to control the operation of kernel modules and achieving higher security.

Keywords: WebAssembly, Kernel Module, Sandbox

I. 서 론

현대 OS의 커널은 디바이스를 지원하고 새로운 기능을 추가하기 위해 선택적으로 로드할 수 있는 모듈 프레임워크를 제공한다. 이러한 커널 모듈은 커널의 보안성을 희생하여 확장성을 달성하며, 하나의 모듈내의 버그는 다른 모듈들을 포함한 커널 전체의 보안을 위협한다.

현대 커널 드라이버 취약점의 근본적인 원인은 커널 모듈 코드의 유지 관리의 어려움에 있다. 많은 상용 디바이스 드라이버 개발자들이 자사의 하드웨어 드라이버를 개별적으로 개발하고 있다. 다양한 하드웨어를 지원하는 것은 수많은 커널 분산된 모듈 코드 작성을 요구하며, 이의 안전성은 오픈소스 커뮤니티에 의존하고 있다. 이러한 이유로 커널 모듈의 버그를 탐지하기 위한 다양한 방법들이 제시되고 있다. 정적 분석 방법[1, 2, 3]들이 연구되었으며, 커널 모듈에 대한 퍼징 기술[4, 5, 6]들이 제안되었다. 이는 많은 연구들이 커널 모듈의 권한을 최소화하고, 그 안전성 검증 방법론들을 지속적으로 고민해왔다는 것을 보여준다.

본 연구는 웹어셈블리로 격리된 커널 모듈을 제시

Received(02. 23. 2023), Modified(03. 13. 2023), Accepted(03. 17. 2023)
* 이 연구는 2023년도 정부의 재원으로 한국연구재단(교육부) 기초연구사업 (NRF-2022R1C1C1010494), 정보통신기획평가원(융합보안핵심인재양성 (No. 2019-0-01343), 국제공동연구사업 (No. 2020-0-00666))의 지원을 받아 수행된 연구임.
† 주저자, doolim98@gmail.com
‡ 교신저자, hjlee228@gmail.com(Corresponding author)

한다. 웹어셈블리는 아키텍처에 독립적인 격리 된 실행환경을 제시하는 표준이다. 웹어셈블리 표준 및 툴체인은 잘 관리되고 있으며 많은 개발자가 적극적으로 참여하고 있다. 또한 서버리스 컴퓨팅[7] 및 엣지 컴퓨팅[8], 스마트 계약[9] 및 베어메탈 장치용 액세스 제어 프레임워크 [10]에서의 채택이 증가하고 있다. 하드웨어 보안기술을 통한 가속[12 ,13]을 지원하며, 이는 개발자 커뮤니티가 활성화됨에 따라 웹어셈블리는 보안성은 지속해 발전하고 있음을 보여준다[11]. 또한 웹어셈블리의 표준 및 도구 체인에 대한 향후 업데이트는 구현에 쉽게 반영할 수 있다.

본 논문에서는 먼저 관련연구와 커널 모듈 격리의 필요성 그리고 웹어셈블리의 특성들을 서술한다. 이를 바탕으로 본 연구에서 웹어셈블리를 채택한 이유에 대한 타당성을 보여주며, 웹어셈블리의 특성을 고려한 시스템설계 및 구현을 제시한다. 또 실제로 커널 내에서 간단한 커널 모듈을 컴파일하고 웹어셈블리 런타임 내에서 실행한 실험결과를 보여준다. 마지막으로 웹어셈블리를 활용한 커널 모듈의 격리의 시사점, 한계 및 향후연구를 서술하며 마무리한다.

II. 배경 및 관련연구

전통적으로 커널에서 코드를 실행하는 것은 하나의 버그나 취약성이 전체 시스템을 손상시킬 수 있기 때문에 위험하다. 커널 모듈은 이러한 위협에 취약하며, 이를 해결하기 위해 격리된 실행환경을 제공하려는 여러 연구들과 기술들이 있다.

2.1 커널 내부의 격리된 실행환경의 필요성

eBPF(Extended Berkeley Packet Filter)는 개발자가 커널 자체를 수정하지 않고도 리눅스 커널 내에서 코드를 작성하고 실행할 수 있는 기술이다. eBPF는 원래 네트워크 패킷을 필터링하고 조작하기 위해 만들어졌지만 이후 시스템 추적, 보안 및 성능 분석을 비롯한 광범위한 사용 사례를 지원하도록 확장되었다[14]. 하지만 eBPF는 패킷 필터링 및 코드 삽입의 용도로 개발 되었으며, 커널모듈작성과 같은 범용적인 기능들을 제공하지 않는다. 예를 들어, eBPF에서 호출할 수 있는 커널 API는 매우 제한적이며, 따라서 개발자들은 기존의 코드를 재사용할 수 없다.

본 연구와 eBPF는 커널에서 코드를 실행하는 안전하고 유연한 방법을 제공한다는 점에서 그 동기를 일부분 공유한다. eBPF는 엄격한 안전 제약을 적용하는 가상 머신에서 코드가 실행되는 코드 실행을 위한 샌드박스 환경을 제공하여 이 문제를 해결한다. 따라서 eBPF의 존재는 커널 내부의 격리된 실행환경의 필요성을 잘 보여주며, 본 연구에서는 eBPF가 아닌 웹어셈블리를 활용한 더욱 유연하면서도 안전한 격리된 실행환경을 제안한다.

2.2 웹어셈블리

웹어셈블리는 웹 기반 어플리케이션 개발의 성능 가속화를 위하여 개발되었으며, 따라서 네이티브 수준의 성능을 웹 기반 어플리케이션에서도 실현할 수 있다. 또한 웹의 특성상 클라이언트의 아키텍처에 종속적이지 않으며, 유연한 인터페이스를 제공한다.

웹어셈블리 생태계는 빠른 속도로 성장하고 있으며, 여러 플랫폼으로의 포팅이 가능하다[15]. 이러한 장점을 활용하여 웹 뿐만 아니라 다양한 분야에서 활용하고자 하는 움직임이 있다. 시스템에서 사용할 수 있는 WASI(WebAssembly System Interface)가 개발되고 있으며, 도커에서도 기존의 리눅스 컨테이너 뿐만 아니라, 웹어셈블리 컨테이너를 제공하고 있다.

메모리 안전성: 웹 어셈블리는 기존 연구들을 바탕으로 여러 가지 방법으로 메모리 안전을 보장한다. 메모리 경계를 검사함으로써, 프로그램이 할당된 공간 외부의 메모리에 액세스하려고 할 때마다 예외가 발생한다. 이를 위해 웹어셈블리는 선형 메모리 모델을 사용하는데, 이는 모든 메모리가 연속적으로 할당된다는 것을 의미하며, 이는 경계 검사를 단순화하고 보다 메모리 접근 작업을 용이하게 한다.

샌드박스: 웹어셈블리는 샌드박스 환경 내에서 실행된다. 즉, 시스템 리소스에 대한 액세스가 제한되고 허용되지 않은 리소스에 접근하지 못하며, 모든 접근은 개발자에 의해 정의된 웹어셈블리 런타임에 의해 제어된다. 본 연구는 웹어셈블리의 성능과 보안성을 커널에 이식하여 더욱 안전한 커널 모듈 프레임워크를 제시한다.

2.3 관련연구

많은 연구들이 커널 모듈 격리체계를 제안하였으며[16-21], 구현방법에 따라 소프트웨어 혹은 하드

웨어 기반의 방법론을 제시한다. LXFI[10]는 커널의 API 무결성을 제시한다. LXFI는 SFI (Software Fault Isolation)을 기반으로 커널 모듈을 격리하며, 정적 분석을 바탕으로 API 무결성을 검증한다. 따라서 개발자는 각 API에 최소한의 권한만을 주어 의도된 사용을 규정할 수 있다.

LXDS[16]는 하이퍼바이저의 도움으로 커널 하위 시스템을 격리한다. LXDS는 각 커널 모듈의 정책을 설정하기 위해 자체 IDL(Interface Description Language)을 제공한다. 격리된 커널 도메인(커널 모듈)들은 커널 내부에서 실행되는 마이크로커널에 의해 관리되며, 여러 IPC(Inter-Process Communication)채널을 통해 다른 도메인과 통신한다. 도메인 간 통신을 위한 코드와 동기화 코드도 IDL을 기반으로 생성된다.

HAKC[17]는 커널 모듈을 서로 분리하는 최신 프레임워크이지만 이 또한 하드웨어 기능을 사용한다. Arm 최신 아키텍처에서만 사용이 가능하며, Arm 아키텍처의 PAC(Pointer Authentication Code)[22] 및 MTE(Memory Tagging Extension)[23]을 요구한다. PAC은 ARMv8.3부터 새로 도입된 하드웨어 기능으로, 포인터가 역참조되기 전에 악의적으로 수정되었는지 여부를 확인한다. MTE는 ARMv8.5부터 사용할 수 있는 메모리 제어 확장 기능이다. MTE는 64비트 주소에서 영향을 주지 않는 상위 바이트를 활용하여, 포인터를 역참조할 때에 동일한 상위 바이트값을 가진 주소만 접근 하도록 제어한다. HAKC는 포인터 무결성을 위해 PAC을 사용하고 메모리 구역을 나누기 위해 MTE를 사용하며, 코드와 데이터를 그룹화하여 데이터 액세스 정책을 세분화된 방식으로 구성한다.

본 연구의 주요 차별점은 대부분의 기존연구들이 독점 하드웨어 기능에 의존하고 특정 아키텍처에 의존하는 반면에, 사용하는 메모리 영역을 분리하기 위해 웹어셈블리의 소프트웨어 기반 메모리 격리 기술을 채택했다는 것이다. 또한 웹어셈블리는 기존의 연구들에서 제시했던 소프트웨어 기반 메모리 격리 기술과 비교하여, 활발한 커뮤니티와 표준을 가지고 있으며, 명료하면서도 보안성이 높은 선형 메모리 모델을 채택했다는 점에서 강점이 있다.

III. 설계 및 구현

웹어셈블리를 샌드박스 커널 모듈로 구현하는 것은 몇 가지 디자인 및 작업을 요구한다. 웹어셈블리에는 바이트 코드 가상 머신을 실행 환경에 연결하는 런타임이 필요하다. 기존의 웹어셈블리 런타임들은 유저공간에서 동작하도록 설계되어 있다. 하지만 유저 프로그램과 커널의 빌드 시스템은 프로그램 특성상 큰 차이를 가지고 있다. 커널에서는 일반적으로 사용되는 *libc*라이브러리를 사용할 수 없고, 기본적으로 부동소수점 CPU연산을 지원하지 않는다. 따라서 웹어셈블리를 커널에서 지원하기 위해서는 *libc*종속성을 제거하고 메모리 할당 및 미지원 CPU연산

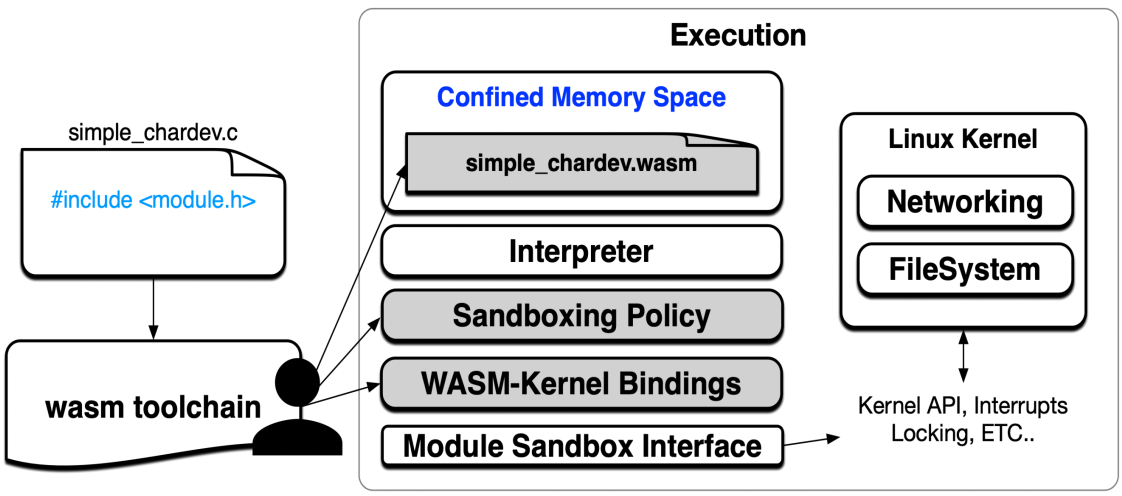

Fig. 1. WebAssembly kernel module design

지원기능을 추가하는 등 많은 엔지니어링 작업이 수반된다.

또한 모놀리식 커널과 커널 모듈 간의 권한과 인터페이스가 모호하게 정의되어 있다. 본 연구는 개발자가 이러한 인터페이스를 정의하고 엄격하게 제어할 수 있도록 유연성을 제공하여 효과적으로 커널 모듈을 격리할 수 있도록 한다. Fig .1.은 개발자가 작성한 모듈 및 정책의 컴파일과정과 실행 흐름을 보여준다.

3.1 웹어셈블리 바이너리 컴파일

웹어셈블리 아키텍처 대상으로 컴파일하는 동안 웹어셈블리가 x86을 지원해야 하기 때문에 커널 모듈용 웹어셈블리 런타임을 설계할 때 아키텍처의 불일치를 야기한다. 다행히도 여러 아키텍처를 지원해야하는 리눅스 커널의 매크로 및 함수들은 새로운 아키텍처를 포팅할 때 호환성을 제공하도록 잘 설계되어 있다.

호스트 API 접근: 외부 함수의 모든 호출은 웹어셈블리 런타임에 의해 통제된다. Fig. 2.에서 웹어셈블리 코드는 *import* 섹션에서 외부 함수 콜을 정의하며, 런타임은 이를 지원한다. 이 섹션은 컴파일 타임에 구성되며, 런타임에는 정의된 함수들이 인덱싱되고 호출 명령을 사용하여 웹어셈블리 인스턴스에서 호출할 수 있다. 내부 함수 호출과의 차이점은 제어 흐름을 런타임에 전달하여 요청을 처리한다는 것이다.

전역 변수 액세스: 기존의 커널 모듈은 링킹을 통해 전역 변수에 액세스한다. 기존의 커널 모듈 c 코드는 *extern* 키워드를 사용하여 커널 외부 전역 객체 변수를 지시한다. 따라서 커널은 커널 메모리에서 커널 모듈을 로드 할 때, 이러한 외부 심볼들을 재배치한다. 이는 사용자 공간에서의 링킹과 같은 역할을 한다.

```
(import "kernel" "spin_lock_irq"
   (func $spin_lock_irq (type 0)))
(import "kernel" "spin_unlock_irq"
   (func $spin_unlock_irq (type 0)))
(import "kernel" "radix_tree_lookup"
   (func $radix_tree_lookup (type 1)))
...
call $spin_lock_irq
```

Fig. 2. Example of Host Kernel API in WebAssembly

그러나 현재 웹어셈블리 C/C++ 컴파일러는 호스트의 전역변수 사용을 지원하지 않는다. 이는 모든 전역 변수는 웹어셈블리 모듈의 선형 메모리에 할당되며, 선형 메모리 외부로의 액세스를 허용하지 않기 때문이다. 하지만 호스트에서 직접 모듈의 선형 메모리에 액세스하는 것은 여전히 가능하다.

이를 해결하기 위해서 *setter/getter*를 구현한다. Fig. 3.은 *nr_cpu_ids* 전역 변수에 대한 접근 구현 예시이다. *setup_queues*함수는 CPU의 ID (*nr_cpu_ids*)에 접근한다. 이 변수는 커널 내에 저장되어 있는 전역변수로, 선형 메모리에 저장되어 있지 않다. 따라서 *__wasm_copy_nr_cpu_ids_from_host()*를 호출하여 이를 선형 메모리에 저장하고 웹어셈블리 모듈은 이를 사용한다. 반대로 웹어셈블리 내부에서 *nr_cpu_ids*를 변경 할 수도 있으므로, *__wasm_copy_nr_cpu_ids_to_host()*를 호출하여 커널로 값을 복사하여 동기화한다. 이는 웹어셈블리 런타임에 구현되어 있다.

```
// kernel's global variable
extern nr_cpu_ids;

// ORIGINAL CODE
// nr_cpu_ids access
static int setup_queues(struct nullb *nullb)
{
   nullb->queues = kcalloc(nr_cpu_ids, sizeof(struct
        nullb_queue),
            GFP_KERNEL);
   // ...
}

// WASM CODE
// Instrumented code
static int setup_queues(struct nullb *nullb)
{
   __wasm_copy_nr_cpu_ids_from_host(); // Getter
   nullb->queues = kcalloc(nr_cpu_ids, sizeof(struct
        nullb_queue),
            GFP_KERNEL);
   // ...
   __wasm_copy_nr_cpu_ids_to_host(); // Setter
}
```

Fig. 3. Example of Host Kernel API in WebAssembly

3.2 웹어셈블리 인터프리터

본 연구의 인터프리터는 WAMR(WebAssembly Micro Runtime)를 바탕으로 구현되었다. WAMR는 활발하게 개발되고 있는 웹어셈블리 런타임으로 현재 Linux, Linux-SGX 및 Windows와 같은

여러 플랫폼을 지원한다[15]. 하지만 리눅스 커널내부에서 동작하는 런타임을 제공하지 않으므로, 본 연구에서는 WAMR를 수정하여 커널 모듈로 컴파일하였다.

글로벌 런타임 구현: 웹어셈블리 바이너리는 리눅스 커널 모듈 로드 시스템과 유사하게 로드가 가능하다. 하지만 이를 위해 하나의 글로벌 런타임 커널 모듈을 필요로 한다. 글로벌 런타임 모듈은 기존 커널 모듈의 형태를 가지며 종속된 웹어셈블리 모듈에 런타임 API를 제공한다. 초기화를 할 때에 런타임 모듈은 WAMR API를 통해 웹어셈블리 실행 환경을 인스턴스화 한다.

3.3 커널 API 바인딩

매크로 기능 구현: 리눅스 커널 소스 코드에는 함수, 변수 및 컴파일러 특성을 캡슐화하는 많은 C 매크로가 있다. 예를 들어, Fig. 4.의 *cmpxchg*는 아토믹 연산(비교 및 교환)을 제공하는 매크로이며 인라인 어셈블리를 포함하는 아키텍처 특정 코드로 확장된다. 문제는 커널 코드 매크로에 아키텍처별(호스트 내장) 코드가 포함된 경우 웹어셈블리로 컴파일되지 않을 수 있다는 점이다. Fig. 4.의 *runtime_cmpxchg*는 *cmpxchg*의 런타임 구현을 보여준다. 본 연구는 x86 특정 인라인 어셈블리로 확장되는 매크로를 분석하고 소스 코드 수정 없이 웹어셈블리 런타임 API를 호출하도록 설계하였다.

```
// x86 architecture
cmpxchg(ptr,0,1);
// expanded code in x86
__typeof__(*(ptr)) __ret;
   __typeof__(*(ptr)) __old = (0);
   __typeof__(*(ptr)) __new = (1);
volatile u16 *__ptr = (volatile u16 *)(ptr);
asm volatile(lock "cmpxchgl %2,%1"
          : "=a" (__ret), "+m" (*__ptr)
          : "r" (__new), "0" (__old)
          : "memory");
//...
// expanded code in Web Assembly
__call_runtime_cmpxchg(ptr,0,1);

// Web Assembly Runtime Implementation
runtime_cmpxchg(wasm_ptr wasm_idx, int old, int new){
   void* host_ptr = wasm_linear_memory_base_addr +
        wasm_idx;
   cmpxchg(host_ptr,old,new);
}
```

Fig. 4. WebAssmebly's x86 macro API example code

3.4 샌드박싱 정책 생성

커널 웹어셈블리 모듈은 미리 정의된 함수 및 인자만을 허용한다. 웹어셈블리의 특성 및 런타임 구현이 어떻게 샌드박싱을 수행하는지 보여준다.

호출 가능한 함수 제어: 웹어셈블리는 외부 함수를 호출 할 때에 미리 정의된 함수만 호출 할 수 있도록 디자인 되었다. 허용된 함수 정보들은 컴파일된 바이너리에 저장되어 있으며, Fig. 2. 에서 함수들의 정의를 볼 수 있다.

함수 호출 정책 생성: 웹어셈블리 런타임 모듈에서는 웹어셈블리 커널 모듈과 커널 간의 모든 상호작용을 중재할 수 있다. 개발자는 호스트 API접근을 할 때에 허용되는 인자들을 바탕으로 **새로운 정책**을 만들 수 있도록 하였다.

일반적으로 C 함수 호출은 여러 인수 전달로 구성되며 인수/반환 값은 웹어셈블리 모듈의 격리를 손상시킬 수 있다. 커널 API 자체는 구성 요소 격리를 염두에 두고 설계되지 않았으며 본질적으로 안전하지 않다. 따라서 커널 모듈은 API를 악용하여 웹어셈블리 커널 모듈의 샌드박싱을 약화시킬 수 있는데, 이는 부작용이 아닌, 리눅스 커널 디자인상의 근본적인 문제이다. 본 연구에서는 API호출을 컨트롤할 수 있는 수단을 제공하여 리눅스 커널 API의 문제점을 보완한다.

IV. 실 험

16GB RAM과 intel i7-9700 하드웨어, 리눅스 커널 5.15에서 웹어셈블리 런타임 커널 모듈을 실행한다. 런타임 커널 모듈 내에서 실행하는 커널 모듈은 간단한 디바이스 모듈인 character 디바이스를 구현하여 실행한다. 해당 디바이스 파일은 read/write 파일 입출력 기능을 제공하며, 간단한 선입선출 기능을 수행한다.

4.1 실험결과 및 분석

Table 1.은 같은 커널 모듈 드라이버를 웹어셈블리 런타임에서 실행시킬 때의 성능을 보여준다. 기존의 커널모듈과의 비교를 위해 같은 코드를 일반적인 커널 프레임워크를 활용해 빌드한 코드(native

Table 1. Simple Character Device IO

IO operation	IO(KB/ms)	switch cycle
wasm read	2314	1231cycle
wasm write	2311	1244cycle
native read	1823	430cycle
native write	1844	431cycle

read/write)가 포함되었다. 유저공간 프로그램에서 read/write 시스템콜을 호출하였을 때에 오버헤드는 switch cycle로 소요된 cycle수로 측정하였다. IO는 1ms에 입력 및 출력한 데이터의 킬로바이트 수를 나타낸다. 매 시스템콜마다 1,000byte의 데이터를 처리하였으며, 10,0000번의 실행결과의 평균이다.

웹어셈블리 모듈에서는 기존의 모듈보다 더 높은 오버헤드를 보여준다는 것을 알 수 있다. 하지만 데이터 입출력에서는 큰 차이를 보여주지 않는 다는 것을 볼 수 있으며, 커널 모듈의 리소스 소모가 크면 클수록 컨텍스트 스위치의 오버헤드의 영향은 적다는 것을 의미한다.

V. 결 론

웹어셈블리 생태계는 빠른 속도로 성장하고 있으며, 이러한 장점을 활용하여 웹 뿐만 아니라 다양한 분야에서 활용하고자 하는 움직임이 있다. 본 연구는 이러한 흐름에 따라 커널 내부에서 웹어셈블리를 활용하는 방안을 제시하였다. 여러 가지 분야에서 웹어셈블리를 활용하고 있지만, 아직 커널내부 구현체는 제공하지 않는다. 본 연구는 기존에 없었던 커널 웹어셈블리 런타임을 구현하여, 웹어셈블리 생태계에 기여했다는 점에서 의미하는 바가 크다.

5.1 시사점 및 한계

본 연구는 웹어셈블리의 메모리 모델은 높은 보안성을 보여주지만, 커널 모듈으로 이식할 때에 문제가 있을 수 있음을 보여주었다. 격리된 모듈 내에서 전역변수에 접근하지 못하며 이에 대한 전역변수 *getter/setter* 방법론을 제시하였다. 또한 기존의 리눅스 컴파일 툴체인을 이식하는 과정에서 아키텍처 종속적인 매크로 및 함수들이 단순히 이식되지 않으며, 런타임 내에 구현되도록 하였다. 따라서 웹어셈블리로 컴파일된 커널 모듈을 다루는 첫 논문 이라는 점에서 이러한 문제점들을 보여주고 해결방안을 제시했다는 점에서 가치가 있다.

향후연구: 현재는 간단한 커널 모듈만을 실험하였지만, 더욱 다양한 커널 API을 지원할 수 있도록 일반적인 방향을 제시하였으며, 더욱 복잡한 드라이버에서도 동작할 수 있도록 고려되었다. 따라서 실제로 사용되는 네트워크 드라이버(e1000/e1000e), 테스트용 드라이버(nullblk) 및 파일 시스템 모듈을 비롯한 실제로 사용되는 대상으로 실험을 진행 할 계획이다.

References

[1] J. Bai, J. Lawall, Q. Chen, and S. Hu, "Effective static analysis of concurrency use-after-free bugs in linux device drivers," in 2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019 (D. Malkhi and D. Tsafrir, eds.), pp. 255 - 268, USENIX Association, 2019.

[2] J. Bai, T. Li, K. Lu, and S. Hu, "Static detection of unsafe DMA accesses in device drivers," in 30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021 (M. Bailey and R. Greenstadt, eds.), pp. 1629 - 1645, USENIX Association, 2021.

[3] A. Machiry, C. Spensky, J. Corina, N. Stephens, C. Kruegel, and G. Vigna, "DR. CHECKER: A soundy analysis for linux kernel drivers," in 26th USENIX Security Symposium (USENIX Security 17), (Vancouver, BC), pp. 1007 - 1024, USENIX Association, Aug. 2017.

[4] J. Corina, A. Machiry, C. Salls, Y. Shoshitaishvili, S. Hao, C. Kruegel, and G. Vigna, "Difuze: Interface aware fuzzing for kernel drivers," CCS '17, (New York, NY, USA), Association for Computing Machinery, 2017.

[5] H. Peng and M. Payer, "USBFuzz: A framework for fuzzing USB drivers by device emulation," in 29th USENIX Security Symposium (USENIX Security 20), pp. 2559 - 2575, USENIX Association, Aug. 2020.

[6] D. Song, F. Hetzelt, J. Kim, B. B. Kang, J.-P. Seifert, and M. Franz, "Agamotto: Accelerating kernel driver fuzzing with lightweight virtual machine checkpoints," in 29th USENIX Security Symposium (USENIX Security 20), pp. 2541 - 2557, USENIX Association, Aug. 2020.

[7] S. Shillaker and P. R. Pietzuch, "Faasm: Lightweight isolation for efficient stateful serverless computing," in 2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020 (A. Gavrilovska and E. Zadok, eds.), pp. 419 - 433, USENIX Association, 2020.

[8] M. Nieke, L. Almstedt, and R. Kapitza, "Edgedancer: Secure mobile webassembly services on the edge," in EdgeSys@EuroSys 2021: 4th International Workshop on Edge Systems, Analytics and Networking, Online Event, United Kingdom, April 26, 2021 (A. Y. Ding and R. Mortier, eds.), pp. 13 - 18, ACM, 2021.

[9] N. He, R. Zhang, H. Wang, L. Wu, X. Luo, Y. Guo, T. Yu, and X. Jiang, "EOSAFE: security analysis of EOSIO smart contracts," in 30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021 (M. Bailey and R. Greenstadt, eds.), pp. 1271 - 1288, USENIX Association, 2021

[10] R. Liu, L. Garcia, and M. Srivastava, "Aerogel: Lightweight access control framework for webassembly-based bare-metal iot devices," in 2021 IEEE/ACM Symposium on Edge Computing (SEC), pp. 94 - 105, 2021.

[11] WebAssembly, 2021. Reference Types for WebAssembly. https://github.com/WebAssembly/reference-types/blob/master/proposals/reference-types/., 2022. 2.11

[12] C. Disselkoen, J. Renner, C. Watt, T. Garfinkel, A. Levy, and D. Stefan, "Position paper: Progressive memory safety for webassembly," in Proceedings of the 8th International Workshop on Hardware and Architectural Support for Security and Privacy, HASP '19, (New York, NY, USA), Association for Computing Machinery, 2019.

[13] dlehmann@google.com. "Write-protection of generated code with PKEYs/PKU." https://bugs.chromium.org/p/v8/issues/detail?id=11714#c26. , 2022.5.8.

[14] Ebpf.io, "What is eBPF? An Introduction and Deep Dive into the eBPF Technology.", https://ebpf.io/what-is-ebpf/, 2023.2.21

[15] WAMR, "WebAssembly Micro Runtime,", https://bytecodealliance.github.io/wamr.dev/, 2023.2.21.

[16] S. Boyd-Wickizer and N. Zeldovich, "Tolerating malicious device drivers in linux," in 2010 USENIX Annual Technical Conference, Boston, MA, USA, June 23-25, 2010 (P. Barham and T. Roscoe, eds.), USENIX Association, 2010.

[17] Y. Mao, H. Chen, D. Zhou, X. Wang, N. Zeldovich, and M. F. Kaashoek, "Software fault isolation with api integrity and multi-principal modules," in Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, SOSP '11, (New York, NY, USA), p. 115 - 128, Association for Computing Machinery, 2011.

〔18〕 V. Narayanan, A. Balasubramanian, C. Jacobsen, S. Spall, S. Bauer, M. Quigley, A. Hussain, A. Younis, J. Shen, M. Bhattacharyya, and A. Burtsev, "Lxds: Towards isolation of kernel subsystems," in 2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019 (D. Malkhi and D. Tsafrir, eds.), pp. 269 - 284, USENIX Association, 2019.

〔19〕 D. McKee, Y. Giannaris, C. O. Perez, H. Shrobe, M. Payer, H. Okhravi, , and N. Burow, "Preventing Kernel Hacks with HAKCs," in 29th Annual Network and Distributed System Security Symposium, NDSS 2022, April 24-28, 2022, The Internet Society, 2022.

〔20〕 Xiong, Xi, Donghai Tian, and Peng Liu. in Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011, The Internet Society, 2011.

〔21〕 Y. Zhou, X. Wang, Y. Chen, and Z. Wang, "Armlock: Hardware-based fault isolation for arm," in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, (New York, NY, USA), p. 558 - 569, Association for Computing Machinery, 2014.

〔22〕 ARM Ltd, "ARMv8.5-A Memory Tagging Extension." https://developer.arm.com/-/ media/ArmDeveloperCommunity/PDF/Arm_Memory_Tagging_Extension_ Whitepaper.pdf, 2019. Accessed Mar 8 , 2022.

〔23〕 ARM Ltd, "Pointer Authentication on ARMv8.3" https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/pointer-auth-v7.pdf, 2017. Accessed Mar 8 , 2022.

〈저자소개〉

임 하 정 (Hajeong Lim) 학생회원
2022년 2월: 성균관대학교 소프트웨어학과 졸업
2022년 3월~현재: 성균관대학교 소프트웨어학과 석사과정
〈관심분야〉 정보보호, 시스템보안

이 호 준 (Hojoon Lee) 정회원
2010년 12월: The University of Texas at Austin 졸업
2013년 8월: KAIST 석사
2018년 2월: KAIST 박사
2019년 11월~현재: 성균관대학교 소프트웨어대학 교수
〈관심분야〉 정보보호, 프로그램 분석, 소프트웨어보안, 시스템보안, TEE, 클라우드 AI보안