졸업과제 중간보고서

웹 어셈블리의 Just-In-Time Compiled Code 보호 기술 개발

조 이름	인디고블랙파스타	
조원	학번	이름
	202155569	신채원(조장)
	201924486	배명진
	202155603	정윤서
분과명	하드웨어, 보안 (D)	
번호	52	
지도교수	권동현 교수님	

목차

1. 과제 개요 3

1-1. 과제 배경 3

1-2. 과제 목표 3

2. 요구조건 및 제약 사항 분석에 대한 수정사항 4

2-1. 요구조건 4

2-2. 제약사항 및 수정사항 4

3. 설계 상세화 및 변경내역 5

3-1. 설계 상세화 5

3-2. 변경 내역 7

4. 갱신된 과제 추진 계획 7

5. 구성원별 진척도 7

6. 보고 시점까지의 과제 수행 내용 및 중간 결과 8

7. 참고 문헌 9

1. 과제 개요

1-1. 과제 배경

V8 JIT-Compiler의 메모리 보호 정책

Google V8은 구글 브라우저 및 NodeJS에서 사용되는 JavaScript 및 WASM의 런타임 엔진이다. V8 엔진은 JavaScript와 WASM 코드를 빠르고 효율적으로 실행하기 위해 설계되었으며, Just-In-Time 컴파일 기술을 사용하여 뛰어난 성능을 제공한다.

V8 엔진의 JIT 컴파일러는 코드 실행 시 JavaScript 및 WASM 코드를 기계어로 변환한다. 이 첫 컴파일에서는 코드의 효율성보다는 컴파일 시간의 축소를 목표로 하여, 빠르게 컴파일이 가능하지만 코드에 대한 최적화는 거의 이루어지지 않는다. 이후 V8은 처음 변환된 기계어 프로그램을 수행하는 도중 해당 기계어에 대한 최적화를 실행하여, 빠른 컴파일 시간 및 효율적인 동작을 제공한다.[1]

이러한 최적화 동작을 위해 JIT 컴파일러는 전체 코드를 조각으로 나누어 각 조각별로 컴파일 및 코드 저장을 수행하며 코드 조각의 저장 위치 등은 점프테이블로 관리한다. 최적화 또한 코드 조각 별로 수행하고, 점프테이블의 정보를 변경하여 결과를 적용하기 때문에 수행 도중 작동 오류가 발생할 가능성이나 전체 기계 코드를 변경할 필요성이 없다.

V8의 JIT 컴파일러는 코드를 조각으로 나누어 각각 컴파일하는 특성으로 인해 독특한 메모리 보호 정책을 채택한다. V8에서는 Intel MPK[2]를 이용하여 메모리를 보호하는데, 기계 코드가 저장되는 코드스페이스에 protection key를 할당하고, 코드가 저장되는 동안에만 해당 protection key에 대해 쓰기 권한을 허용함으로써 코드가 실행되는 동안의 코드 변조를 막는다.

V8 JIT-Compiler 메모리 보호 정책의 문제점

그러나 V8의 JIT 컴파일러는 protection key를 하나만 사용하여 메모리 보호를 수행한다. 즉, 모든 코드스페이스를 같은 protection key로 보호하는 것이다. 이러한 동작의 허점은, 하나의 코드스페이스에 코드를 저장하기 위해 protection key의 권한을 변경하면 모든 코드스페이스가 쓰기 권한을 가진다는 데 있다. JIT 컴파일러는 동작 특성상 하나의 코드스페이스에 저장된 코드를 수행하는 도중에 다른 코드스페이스에 대한 컴파일 및 저장이 발생한다. 따라서 공격자의 공격 표면이 넓어지고, 보안성이 낮아진다.

1-2. 과제 목표

우리는 1-1에 명시된 V8의 보안적 문제점을 해결하기 위하여, 코드스페이스마다 서로 다른 protection key를 할당하여 메모리 보호를 수행하는 것을 과제 목표로 한다.

2. 요구조건 및 제약 사항 분석에 대한 수정사항

2-1. 요구조건

- 연속된 코드스페이스에 대해 서로 다른 protection key를 할당해야 한다.

- protection key의 할당이 예측 불가능하도록 한다.

2-2. 제약사항 및 수정사항

키 리소스 제한

Intel MPK에서는 0~15의 16가지 protection key만을 지원한다. 따라서 생성되는 코드스페이스의 개수가 16개를 초과하여 protection key가 모두 소진되었을 경우에는 어쩔 수 없이 같은 protection key를 사용하는 코드스페이스가 발생한다. 이때 공격 표면을 최소화할 수 있는 protection key 할당 알고리즘이 필요하다.(변경 없음)

시스템에서 예약되어 사용되는 pkey

Intel MPK의 protection key 중 0번은 기본적으로 모든 메모리 접근에 대한 권한을 허용하는 용도로 사용된다.[3] 즉 0번 키는 특별히 보호되지 않은 메모리 접근을 가능하게 하는 역할을 한다. 0번 키는 미리 예약되어 있어, 사용자가 임의로 제어할 수 없다. 따라서 코드스페이스의 protection key를 할당할 때, 0번은 제외하고 할당해야 한다.

V8의 기존 자료구조 변경 불가능

V8의 핵심 파일을 수정할 경우, 빌드가 되지 않는 제약사항을 확인했다. 따라서 코드스페이스를 관리하는 기존의 자료구조에 protection key를 표시하는 멤버 변수 등을 추가하는 방식의 구현은 불가능하다. 이에 따라 추가적으로 코드스페이스 별 protection key를 저장하는 테이블을 작성하거나, 코드스페이스의 특정 정보를 바탕으로 protection key를 계산하는 알고리즘 등을 고안해야 한다.

3. 설계 상세화 및 변경내역

3-1. 설계 상세화

코드스페이스의 시작 주소 align

Intel MPK는 PTE에 명시된 protection key를 확인하고, 해당 key에 대한 PKRU 비트를 수정하는 방식으로 메모리의 권한을 관리한다. 따라서 권한 관리는 페이지 단위로 이루어지게 된다. 코드스페이스 별로 권한을 관리하기 위해서는 하나의 페이지에 대해 하나의 코드스페이스만 저장되어야 한다. Intel 아키텍처는 페이지 당 4kB의 메모리 공간을 할당하므로, 코드스페이스의 시작 주소를 4kB에 대해 정렬한다.

protection key 할당

a. protection key에 대한 설계 조건

a-1. 연속된 코드스페이스에 대해 서로 다른 protection key 할당

코드스페이스의 시작 주소가 페이지 하나에 매핑되는 메모리 크기인 4kB에 대해 정렬되어 있고, MPK의 메모리 접근 권한 관리는 페이지 단위로 이루어지므로, 하나의 코드스페이스에 대해 권한을 변경할 경우 해당 코드스페이스가 포함된 4kB 공간 전체의 권한이 변경된다. 인접한 코드스페이스의 protection key가 서로 같을 경우, 그 중 하나의 코드스페이스에 코드를 쓰더라도 8kB의 큰 공간이 쓰기 권한을 얻게 되어 결과적으로 공격 표면이 증가한다. 따라서 연속된 코드스페이스에 대해 서로 다른 protection key를 할당한다.

a-2. 예측 불가능한 protection key

공격자가 protection key를 예측할 수 있다면, 하나의 코드스페이스에 대해 컴파일이 일어나고 있는 도중 같은 protection key를 사용하는 다른 코드스페이스에 대해서도 접근 가능하게 된다. 따라서 protection key를 하나만 사용하는 기존의 V8과 같은 보안 취약점을 가진다. 이를 방지하기 위해, 공격자가 protection key를 예측하기 어려운 할당 알고리즘을 설계한다.

b. protection key 할당 알고리즘

코드스페이스에 대한 protection key를 할당하고, 이후 식별하기 위해서 코드스페이스의 시작 주소에 대해 protection key를 매핑하였다. 시작 주소와 protection key를 매핑하는 데에는 세 가지 방법이 있다.

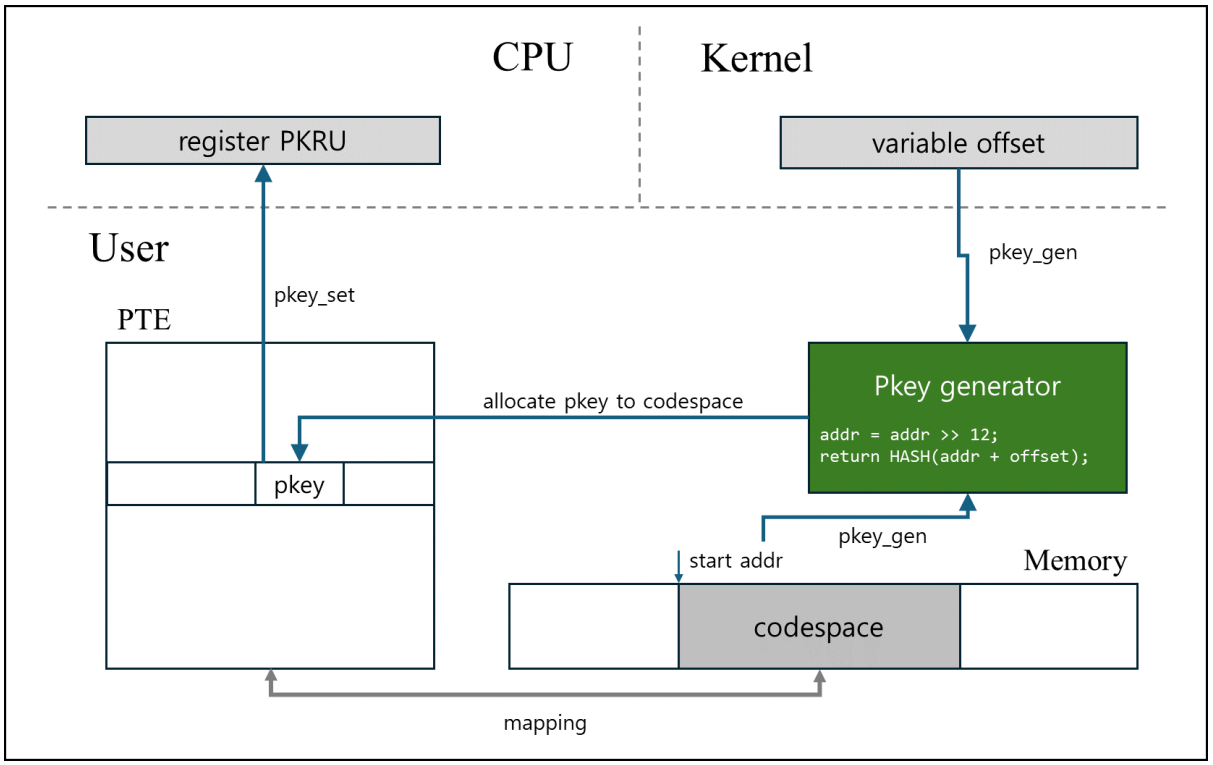
- 코드스페이스를 관리하는 자료구조에 protection key를 저장하는 멤버 변수 추가

- 시작 주소와 그에 따른 protection key를 저장한 테이블 구성
- 시작 주소에 대해 protection key를 계산하여 사용

이 중 1번은 2-2 제약사항 및 수정사항에 명시한 이유로 구현 불가능하다. 또한 2번과 3번을 비교하였을 때, 2번은 새로운 코드스페이스에 대하여 protection key를 할당하거나(연속된 코드스페이스에 대해 서로 다른 protection key를 할당해야 하기 때문에 이전에 할당된 코드스페이스에 대한 protection key 조회가 필요), 코드스페이스에 대한 권한을 변경할 때마다 테이블을 조회해야 하며, 테이블을 추가적으로 보호해야 하기 때문에 3번에 비해 비효율적일 것으로 예상된다. 따라서 3번 방법을 채택하여 사용하였다.

protection key 할당 알고리즘은 다음과 같다. 먼저 코드스페이스의 시작 주소에 대해 12만큼 right shift(>>) 연산을 한다. 이는 모든 코드스페이스가 4kB 정렬되어 있어 하위 12비트가 0으로 동일하기 때문이다. 그리고 연산 결과에 대해 해시 연산을 수행한 다음, 해시 연산의 결괏값을 15에 대해 나머지 연산하고 이에 1을 더해 protection key 값을 확정한다. 이렇게 만들어진 protection key 값은 1 이상 15 이하로 결정되며, 이는 Intel에서 지원하는 protection key 중 미리 예약된 0번을 제외한 나머지에 해당한다. 해시 함수는 입력 데이터에 아주 작은 변화가 발생하더라도 출력이 크게 변하도록 설계되어 있으므로 연속된 코드스페이스에 대해 다른 계산 결과가 도출될 가능성이 높다.

또한 protection key의 무작위성을 높이기 위해 해시 연산의 입력값에 일정한 오프셋 값을 더하여 해시 연산을 수행한다. 오프셋 값은 V8이 수행될 때마다 랜덤한 값으로 지정되며, 값의 유출을 막기 위해 커널 수준에서 저장되고 호출된다.



3-2. 변경 내역

- 착수보고서에서는 코드스페이스의 protection key를 관리하는 구조체를 찾고, protection key에 해당하는 변수를 수정하여 구현한다고 언급하였다. 그러나 3-1에 작성된 내용에 따라 구조체의 protection key 변수를 수정하는 것이 아닌, 새로이 protection key를 계산하는 방식으로 디자인을 변경했다.

- 프로젝트를 착수한 이후 각자 담당한 업무가 균형적이지 않음에 따라 구성원별 담당 업무 및 세부 목표를 새로 설정하였다.

4. 갱신된 과제 추진 계획

	8월 5주	9월 1주	9월 2주	9월 3주	9월 4주	9월 5주	이후
Pkey 관리 알고리즘 작성							
Offset 관리 코드 작성							
Pkey 관리 알고리즘 v8에 적용하기							
보안성 검증 방법 탐색							
실험							
마무리							

5. 구성원별 진척도

구성원	진척도	
신채원	목표	Protection key 할당 알고리즘을 작성하고, 실제 V8에 대해 적용함
	진행 상황	- V8의 코드스페이스 관리 정책 및 권한 관리에 대한 코드 분석 완료함 - protection key 할당 알고리즘 설계 및 작성 완료함 - V8에 대해 적용하는 작업 진행 중
배명진	목표	V8 엔진에서 관리하는 코드스페이스의 시작 주소를 확인하여, protection key 할당 알고리즘에서 연속된 코드스페이스를 탐색할 수 있도록 함
	진행 상황	- V8 엔진 메모리 구조 확인 - 'CodeDesc' 구조체를 디버깅하여, 개별 코드스페이스의 시작 주소 탐색

		- 현재, JIT Compile 외에, 처음으로 Compile된 코드의 저장 공간을 탐색 중
정윤서	목표	V8 엔진 내에서 protection key를 할당하여 메모리 공간들의 권한을 개별적으로 관리할 수 있도록 함
	진행 상황	- V8 엔진에서의 pkey 할당 매커니즘 분석 완료 - 메모리 권한 관리를 위한 알고리즘 삽입 지점 탐색 완료 - Intel에서 미리 예약해서 사용하는 pkey 값 확인 완료

6. 보고 시점까지의 과제 수행 내용 및 중간 결과

6-1. V8의 권한 관리 관련 코드 분석

기존 V8의 권한 관리에 관련한 코드를 분석했다.

V8의 코드스페이스 권한 관리는 주로 WritableJitAllocation 구조체에서 이루어진다. WritableJitAllocation 구조체의 생성자에서 RwxMemoryWriteScope의 optional 생성자를 호출하고, RwxMemoryWriteScope의 생성자에서 SetWritable 함수를 호출한다. SetWritable 함수는 protection key에 쓰기 권한을 설정한다. 이후, 코드 쓰기가 모두 끝나고 WritableJitAllocation 구조체가 소멸하면, RwxMemoryWriteScope 또한 자신의 scope가 종료됨에 따라 소멸하는데, 이때 RwxMemoryWriteScope의 소멸자에서 SetExecutable 함수를 호출해 protection key에 대한 쓰기 권한을 빼앗는다.

6-2. address에 대한 protection key 할당 알고리즘 작성

address를 입력으로 받았을 때 protection key를 계산하는 알고리즘을 작성했다. 단, offset은 커널 드라이버를 추가적으로 작성하여 이를 통해 관리해야 하기 때문에 현재는 포함하지 않았다.

```
1  pkey_gen (Address begin) {
2
3          unsigned char bytes[sizeof(Address) – 2];
4          begin = begin >> 12; //start address is 4kB aligned
5
6          while (len(begin))
7                  bytes[i] = (begin >> (i * 8)) & 0xFF;
8          //convert address to byte array
9
10         SHA256(bytes, sizeof(int), hash);
11         return hash[0] & 0xF + 1;
12 }
```

6-3. V8의 권한 관리 관련 코드에 protection key 할당 코드 추가

V8의 권한 관리 코드에 protection key를 할당하는 코드를 추가했다. 현재 기존 V8 동작과 충돌하는 코드에 대한 수정 작업 중에 있다.

7. 참고 문헌

[1] "V8 JavaScript engine". 2024년 08월 23일 접속. https://v8.dev/

[2] "Memory Protection". 2024년 08월 23일 접속. https://www.gnu.org/software/libc/manual/html_node/Memory-Protection.html

[3] "Pkeys(7) linux programmer's manual". 2024년 08월 23일 접속. http://man7.org/linux/man-pages/man7/ pkeys.7.html.