# Multi-Variant Execution Environment 연구 동향

조명현*, 장지원*, 남기빈*, 황동일*, 백윤흥*
*서울대학교 전기정보공학부, 반도체공동연구소
{mhcho, jwchang, kvnam, dihwang}@sor.snu.ac.kr, ypaek@snu.ac.kr

# A Study on Multi-Variant Execution Environment

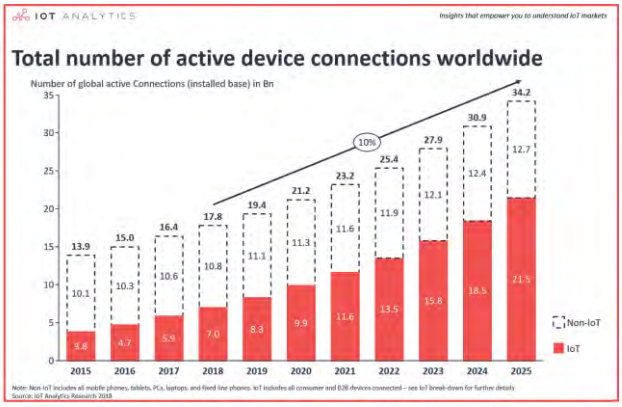
Myunghyun Cho*, Jiwon Chang *, Kevin Nam*, Dongil Hwang*, Yunheung Paek*
*Dept. of Electrical and Computer Engineering and Inter-University
Semiconductor Research Center (ISRC), Seoul National University.

**요 약**

C 와 C++은 비교적 자유로운 코딩 환경으로 많은 프로그래머들에게 사랑받는 프로그래밍 언어이다. 또한, 빠른 속도와 호환성 덕분에 현재 많은 IOT, 임베디드 시스템에 적용되고 있다. C 와 C++은 자유로운 환경을 가지고 있는 반면에 프로그래머의 부주의한 코딩 방식에 의해 여러 취약점을 발생시켜 공격 범위를 증가시킬 수 있다. 다음은 외부 침입자에게 공격에 필요한 좋은 소스를 제공할 수 있으므로 이러한 공격을 막기 위한 범용적인 기술이 필요하다. 본 연구에서는 다음 취약점에 대한 공격을 막을 수 있는 기술 중 하나인 Multi-Variant Execution Environment(MVEE) 기술을 소개하고 다음 기술의 핵심인 다양한 Variant 생성 방식과 기존 연구 분석을 통해 한계점을 고찰하고자 한다.

## 1. 서론

최근 IOT, 임베디드 기기의 증가에 따라 많은 사물들이 연결된 시대가 도래했다. 스마트 홈은 가스레인지, 냉장고, 컴퓨터, IP 캠 등과의 통신을 통해 사용자가 본인 휴대전화로 원격 제어가 가능하지만, 해킹에 대한 취약점을 갖고 있는 것이 화두가 되고 있다. 취약점을 제공하고 있는 원인 중 하나는 많은 임베디드 기기들이 C 와 C++을 기반으로 애플리케이션을 돌리고 있다는 점인데 다음 프로그래밍 언어의 특성으로 인해 공격 가능한 면적이 넓어질 수 있다.




(그림 1) IOT 트랜드.[1]

C 와 C++은 자유로운 코딩 환경과 빠른 실행환경으로 임베디드 기기뿐만 아니라 많은 애플리케이션 제작에 사용되고 있는 범용적인 프로그래밍 언어이다. 자유로운 코딩환경을 제공하는 특성을 가지지만 프로그래밍 개발자의 부주의로 인해 여러 가지 보안 문제를 가질 수 있다. 전 세계적으로 많이 사용하는 모놀리식 커널인 Linux, Windows, BSD 등은 프로그래머들이 코딩 방식에 많은 주의를 기울이고 있지만 여전히 안전하지 않은 프로그래밍 언어를 사용하므로 많은 버그가 존재한다. 예를 들어 보편적인 취약점에는 1) Uninitialized read, 2) Use-after-free, 3) Out-of-bounds 등이 있는데 다음을 이용해 공격자들은 민감한 커널 포인터 탈취, 암호화 키 탈취 등을 통해 Privilege escalation 과 같은 차후의 공격으로 연결할 수 있다. 만약 공격을 통해 공격자가 Root 권한을 탈취한다면 중요한 데이터에 대한 접근 권한을 가질 수 있고 IOT 기기들(IP 캠 등)을 통해 사용자의 개인정보가 유출될 수도 있다.

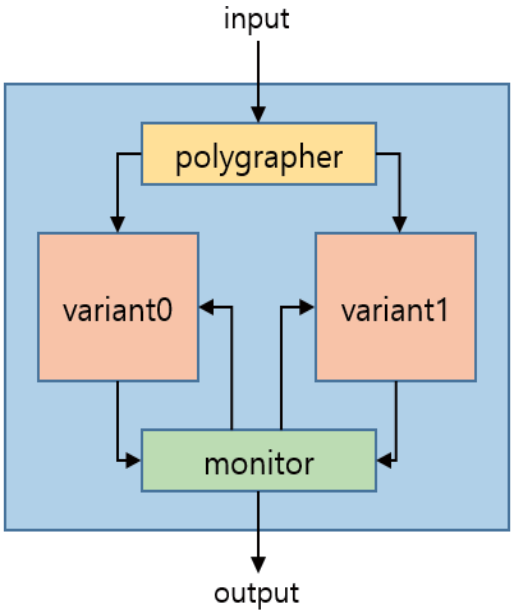
안전하지 않은 프로그래밍 언어를 사용할 때 개발자의 세밀한 코딩 방식으로는 한계가 있다. 그러므로 여러 가지 보안 기법들이 필요한데 다음 기법에는 정적 분석을 통한 변수 초기화, Bound check, 비정상적인 행동 모니터링 등과 같이 공격자에게 소스를 제공하는 것을 막는 기법들이 있고, ASLR(Address Space Layout Randomization), KASLR(Kernel Address Space Layout Randomization)와 같이 확률적인 방어 기법들이 존재한다. Fine-grained 기법에서는 여러 공격에 대한 높은 방어율을 보여주지만 높은 성능 저하를 발생시

키고 Coarse-grained 기법에서는 낮은 성능저하와 낮은 비용 등을 보여주지만 한정된 공격에 대한 방어만 가능하다.

많은 보안 기법들 중 본 논문에서는 MVEE(Multi-Variant Execution Environment)에 대해 2 장에서 설명하고 3 장에서는 여러 MVEE 기법에 대한 분석 마지막으로 4 장에는 결론과 MVEE 기법에 대해 고찰을 할 예정이다.

## 2. MVEE(Multi-Variant Execution Environment) 개념

NVP(N-Version Programming)는 1970 년 Chen and Avicienis[2]에 의해 고안되었다. NVP 의 아이디어는 여러 프로그래머로 이뤄진 팀마다 각각 독자적인 똑같은 프로그램을 만들어 병렬적으로 돌림으로써 버그를 찾아내는 방식에 의해 출발하였다. 하지만 업데이트가 필요할 때 모든 독자적인 프로그램들을 다시 업데이트해야 하므로 높은 유지비용을 발생시켰다.




(그림 2) N-variant System Framework.[3]

초기 NVP 의 높은 유지비용을 해결하기 위해 Variant[3]라는 개념을 도입하였다. Variant 란 본질적으로 동일한 행위를 하지만 방식을 Variant 마다 정해진 규칙에 따라 다각화하여 비정상적인 행동이나 버그가 생길 시 다른 행동을 하도록 만든 개체이다. MVEE 의 Variant 들은 동일한 소스 코드에서 자동적으로 생성되기 때문에 수동적으로 다시 업데이트할 필요가 없어 초기 NVP 의 단점인 높은 유지비용을 감소시켰다.

MVEE(Multi-variant execution environment) 기술은 악의적인 공격 시 프로세스의 상태가 달라지는 특성을 모니터링하므로 프로그램의 버그 탐지뿐만 아니라 보안에도 적절하다. MVEE 의 동작은 (그림 2)와 같이 폴리그래퍼가 입력을 받으면 다음 입력을 복사하여 각각의 Variant 들에게 똑같이 전달한다. 여기서 Variant 들은 본질적으로 동일한 행위를 하지만 내부의 구조가 다르므로 내부 상태에 맞게 알맞은 입력을 줘야한다. 예를 들어, 각 입력에 해당하는 포인터의 위치는 다를 수 있지만, 본질적인 내용은 같아야 하고 버퍼를 사용할 시 위치가 다르더라도 버퍼 안의 내용은 같아야 한다. 다음과 같이 각 Variant 은 동일한 행위를 수행하고 결괏값을 모니터로 전달하는데 이 때 모니터는 각각의 모니터의 값을 비교하여 같은 값이 나왔는지 비교한다. 만약에 동일한 입력에 대해 다른 출력이 나왔다면 다음을 공격으로 간주하고 민감한 데이터를 0 으로 만들어 유출을 막거나 프로그램을 종료시키는 등 미리 정해진 규칙에 따라 동작하도록 만든다.

다음과 같이 MVEE 는 Multi-Variant 들을 동시에 실행시키므로 현대 멀티코어 시스템에서 매우 적합하다. 또한, 특정 환경에 맞게 출력 값만 볼 것인지 아니면 내부 진행상태를 정해진 범위내에서 볼 것인 것 정하여 granularity 를 조정할 수 있으므로 각각의 디바이스에 맞는 보안 수준을 제공할 수 있다.

## 3. MVEE(Multi-Variant Execution Environment) 연구 동향

MVEE 은 런타임 모니터링 기술로써 Lock-step 방식을 통해 동일한 입력을 동기화하여 각각의 Variant 에 넣는다. 보안성을 위해 각각의 Variant 들은 메모리 Isolate 기술을 사용하여 모니터와 물리적으로 분리되어 있다. 또한, 성능향상을 위해 Salamat B, Jackson T, Wagner G, et al[4]은 각각의 Variant 의 메모리 블록을 공유시키는 방식을 고안했는데 모니터가 Variant 들에게 데이터를 제공할 때는 공유된 메모리에 써서 Variant 들이 읽은 뒤 자신의 할당된 주소에 쓰도록 하였고, 동일하게 모니터가 각각의 데이터를 요청할 때는 자신의 주소에서 읽어 공유된 메모리에 쓰도록 하였다.

현재 연구된 MVEE 은 다양한 모니터의 구현과 Variant 방식을 가지고 있다. 대부분의 MVEE 는 User-space 에서 구현되는데 그 이유는 User-space 프로그램들은 엄격하게 정의된 Syscall 이라는 I/O 인터페이스를 가지고 있기 때문이다. 그러므로 각각의 Variant 들은 Syscall 을 통해 동일한 시간 값과 네트워크 트래픽을 가질 수 있다. 하지만 Kernel-space 에서는 그러한 인터페이스가 없으므로 추가적인 동기화 구조를 만들어야 한다. 최신 연구인 kMVX[5]는 Kernel-space 에 동기화 구조를 추가적으로 만들어 MVEE 를 구현할 수 있음을 보여줬다.

MVEE 의 단점 중 하나는 다음 기술을 적용 시 큰

성능 저하를 발생시킨다는 점이다. 특히 I/O 에 관련된 Syscall 이 연속적으로 요청되었을 때 더 큰 성능저하를 보여주었다. 대부분 모니터의 구현은 Syscall 을 감지하는 ptrace 인터페이스에 의해 구현되는데 4byte 씩 읽는 특성 때문에 느리다는 단점이 있다. 그러므로 성능 저하를 줄이기 위해 여러 기법들의 구현이 필요하다. VARAN[6]같은 경우에는 Fast shared memory ring buffer 를 이용하여 ptrace 인터페이스의 오버헤드를 감소시켰고, ReMon[7]은 모니터로 Context switching 이 일어날 때 큰 성능 저하가 발생하는 것을 대상으로 하여 Syscall 중 민감한 Syscall 은 cross-process 모니터를 이용하여 외부에서 안전하게 처리하고 민감하지 않은 Syscall 은 in-process 모니터를 이용하여 빠르게 처리하여 성능 저하를 감소시켰다. 하지만 다음 노력에도 불구하고 몇 가지 벤치마크에서는 아직도 큰 성능 저하를 보여주었다.

MVEE 의 핵심적인 요소는 Variant 의 설정이다. 어떻게 Variant 를 정할지에 따라 보안 수준과 성능이 달라지므로 적절한 매커니즘을 고안해야 한다. Variant 를 통한 보안 수준 향상을 위해 주의할 점은 Variant 자체의 엔트로피보다는 공격에 대해 각각의 Variant 마다 서로 다른 행동을 보여줌으로써 공격에 대한 탐지를 가능하게 하는 것이다. 예를 들면 KASLR 같은 경우 커널 메모리의 레이아웃에 대한 복잡도가 증가하므로 자체 엔트로피는 크지만, 공격에 대한 Variant 마다의 상이함을 잡아낼 수 있는 능력이 높다고는 말할 수 없다.

<표 1> 취약점 및 대응하는 Variant 기법들

| 취약점 | Variant 기법 |
|---|---|
| 포인터 | 1) 주소공간 파티셔닝 |
| 초기화 안된 변수 | 2) 리버스 스택 |
| 초기화 안된 변수 | 3) 스택 패딩 |
| 초기화 안된 변수 | 4) 스택 구조 랜덤화 |
| 코드 삽입 | 5) 명령어 세트 랜덤화 |
| 포인터 | 6) 시스템 콜 번호 랜덤화 |
| 버퍼 오버 플로우 | 7) 데이터 공간 랜덤화 |
| Use-after-free | 8) Type-based SLAB allocator |

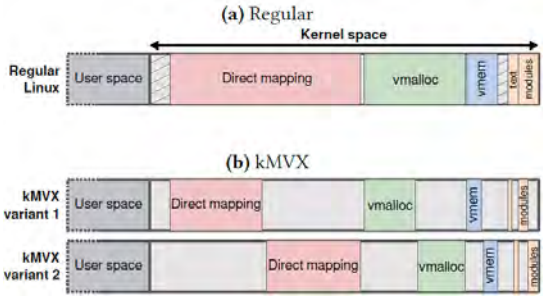
현재 연구된 Variant 방식은 1)주소 공간 파티셔닝, 2)리버스 스택, 3)스택 패딩, 4)스택 구조 랜덤화, 5)명령어 세트 랜덤화, 6)시스템 콜 번호 랜덤화, 7)데이터 공간 랜덤화, 8)Type-based SLAB allocator 등 여러 가지가 있다. 1)주소 공간 파티셔닝 기법은 가장 보편적으로 쓰이는 방식으로 Variant 마다 다른 절대 주소 공간을 할당하여 공격자가 민감한 포인터에 악의적인 목적으로 접근 시 알람을 울리게 하는 방법이다. 한 Variant 주소 공간에 적법한 민감한 포인터가 다른 Variant 주소 공간에서는 적법하지 않을 수 있는 것을 이용한다. 스택의 포맷을 바꾸는 방식은 초기화가 안 된 변수를 통한 데이터 유출을 막는 데 도움이 된다. 예를 들어 스택 프레임이 pop 되고 새로운 스택 프레임이 push 되었을 때 새로운 스택 프레임에 초기화가 안 된 변수가 있으면 이전 프레임의 값을 가지고 있을 수 있다. 2)리버스 스택은 스택이 자라는 방향을 다르게 하는 기법이고, 3)스택 패딩은 프레임마다 랜덤하게 공간을 만들어 타겟이 되는 데이터 값을 읽지 못하게 하는 방법이다. 또한, 4)스택 구조 랜덤화를 이용해 필드의 저장 순서를 랜덤화 하여 원하는 데이터의 위치를 알기 어렵게 만들 수 있다. 5)명령어 세트 랜덤화는 명령어에 각각의 Variant 에 맞는 랜덤한 값을 추가해 악의적인 목적의 명령어 실행 시 특정 Variant 에서는 가능하지만 다른 Variant 에서는 불가능하게 만들어 모니터가 탐지하게 만드는 기법이다. 6) 시스템 콜 번호 랜덤화는 실행마다 시스템 콜번호에 해당하는 서비스 루틴을 랜덤화 하여 공격자가 포인터 탈취를 통해 악의적인 서비스 루틴을 실행하지 못하도록 막는다. 7)데이터 공간 랜덤화[8]는 버퍼 오버플로우를 탐지하는 기법으로 각 Variant 의 버퍼와 민감한 데이터마다 다른 키를 할당하여 XOR 연산을 하게 만든다. 만약에 버퍼 오버 플로우가 발생한다면 민감한 데이터 앞의 버퍼의 키 값이 민감한 데이터와의 연산으로 올라와 다른 Variant 와 다른 결과를 야기할 것이다. 8)Type-based SLAB allocator 은 kmalloc 의 수정을 통해 SLAB allocator 을 타입 기반으로 바꿔 use-after-free 를 방지하는 기법이다. 다음과 같이 다양한 기법들이 존재하지만, 상황에 맞지 않은 적용들은 오히려 성능 저하를 발생시킬 수 있다.

<표 2> MVEE 연구 동향[9]

| Technique | Synchronization | Defense | Implementation | Privilege | Security Benchmark |
|---|---|---|---|---|---|
| **GHUMVEE** | program points | | ptrace | user space | |
| **DCL** | program points | control flow hijacking exploits | ptrace | user space | CVE-2013-2028<br>CVE-2010-4221<br>CVE-2012-4409<br>CVE-2014-0749 |
| **ReMon** | syscall | unknown attacks and low-level memory errors | ptrace | kernel space | logical argument |
| **MvArmor** | syscall | memory error exploits | ptrace | user space | CVE-2004-0488<br>CVE-2014-0160<br>CVE-2014-0195 |
| **kMVX** | I/O and syscall | kernel information leaks | | kernel space | CVE-2016-4569<br>CVE-2013-2237<br>CVE-2016-0728 |
| **Varan** | syscall | | binary rewriting | context | |

여러 모니터링 기법, Variant 기법, Memory-space 기법 등을 통해 MVEE 에 관한 연구들이 진행되어 왔다. Petr Hosek 은 Bionic C 라이브러리 기반의 자신만의 C library function 을 구현한 'VARAN[6]'을 발표해 성능

향상을 보여줬고, K. Koning 은 hardware-assisted 가상화 기반의 'MvArmor[10]'을 구현함으로써 성능 개선에 도움을 주었다. S.Volckaert 는 주소 파티셔닝 기법을 이용해 ROP (Return Oriented Programming) 공격을 막는 'DCL[11]'을 보여주었고 중요한 Syscall 만 cross-process 모니터를 통해 선별적으로 안전하게 검사하는 'ReMon[7]'을 고안하여 성능 향상을 보여주었다. 최근에는 S. Osterlund 가 초기화가 안 된 변수를 통한 민감한 정보의 유출을 타겟으로 Kernel-space 에서 두 커널을 Variant 로 설정하여 주소 파티셔닝, 스택과 힙 영역의 다각화 등을 통해 MVEE 시스템을 구현했다. 두 커널에 시간과 네트워킹을 동기화할 수 있는 I/O sync 와 Copy_to_user 와 같은 커널 정보가 유저에게 유출될 수 있는 상황을 탐지할 수 있는 Syscall sync 를 구현해 Kernel space 에서 공격을 탐지할 수 있는 'kMVX[5]'를 발표했다.




(그림 3) kMVX 메모리 공간.[5]

## 4. 결론 및 고찰

본 논문에서는 C 와 C++의 취약점을 보완할 수 있는 MVEE 에 대한 개념과 핵심 기법인 Variant 를 소개했고, 과거 MVEE 기법부터 최신 MVEE 기법까지 분석을 했다. 모든 보안 기법에도 그렇듯 성능과 보안은 Trade-off 관계에 있다. 최신 연구인 kMVX[5]도 20~50%의 높은 성능 저하를 보여주는데 앞으로 다음 문제점을 개선하기 위해서는 공격이 가해질 때 포괄적으로 반대 행동을 하는 Variant 에 대한 연구, 병렬적인 연산을 해결할 수 있는 하드웨어 모니터링, 필요한 방어 기법들만 유동적으로 적용할 수 있는 모듈화에 관한 연구 등이 필요할 것이다.

## 5. ACKNOWLEDGEMENT

본 연구는 2020 년도 정부(과학기술정보통신부)의 재원으로 한국연구재단(NRF-2017R1A2A1A17069478), 2020 년도 두뇌한국 21 플러스사업, 2020 년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터(No.2017-0-00213, 능동적 사전보안을 위한 사이버 자가변이 기술 개발)의 지원을 받아 수행된 연구임.

## 참고문헌

[1] https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/

[2] Liming Chen., Algirdas V. Avizienis. "N-version programming: A fault-tolerance approach to reliability of software operation" in Annual International Conference on Fault-Tolerant Computing, Toulouse, 1978, pp.3-9.

[3] B. Cox et al. "N-variant systems: A secretless framework for security through diversity"in *USENIX Security*, Canada, 2006, pp. 105-120

[4] Salamat, B., Jackson, T., Wagner, G., Wimmer, C., Franz, M., "Runtime defense against code injection attacks using replicated execution" in Department of Computer Science, United States, 2011, pp. 588-601.

[5] S. Österlund., K. Koninh., P. Olivier., A.Barbalace., H.bos., C. Giuffrida. " kMVX: Detecting Kernel Information Leaks with Multi-variant Execution" in ASPLOS, United States, 2019, pp. 559.

[6] Hosek, P., & Cadar, C. "Varan the unbelievable: An efficient n-version execution framework." in International Conference on Architectural Support for Programming Languages and Operating Systems – ASPLOS, Istanbul, 2015, Vol. 50, No. 4, pp. 339-353

[7] Volckaert, S., Coppens, B., Voulimeneas, A., Homescu, A., Larsen, P., De Sutter, B., & Franz, M. "Secure and efficient application monitoring and replication" in USENIX Annual Technical, United States, 2016, pp. 167-179

[8] Hwang, Shin, et al. "Data Randomization for Multi-Variant Execution Environment.", in International SoC Design Conference (ISOCC), Jeju, 2019, pp. 291-292

[9] Zhenwu Liu, Zheng Zhang, Jiexin Zhang and Hao Liu. "Multi-Variant Execution Research of Software Diversity" in Journal of Physics: Conference Series, China, 2019, Volume 1325.

[10] Koning, K., Bos, H., & Giuffrida, C. "Secure and efficient multi-variant execution using hardware-assisted process virtualization." in 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), France ,2016, pp. 431-442.

[11] Volckaert, S., Coppens, B., & De Sutter, B. "Cloning your gadgets: Complete ROP attack immunity with multi-variant execution" In IEEE Transactions on Dependable and Secure Computing, 2015, 13(4): 437-450.