Multi-Variant Execution Environment의 레플리카 생성 기법에 대한 연구

황동일*, 신장섭*, 정선화*, 백세현*, 백윤흥*
*서울대학교 전기·정보공학부
e-mail:dihwang@sor.snu.ac.kr

A Study on Replica Generation Techniques for Multi-Variant Execution Environment

Dongil Hwang*, Jangseop Shin*, Seonhwa Jung*, Sehyun Baek*
and Yunheung Paek*
*Dept of Electrical Engineering and Computer Science,
Seoul National University

요 약

C와 C++는 컴파일된 결과물의 실행 속도가 타 프로그래밍 언어로 작성된 것보다 빠르고 기존의 시스템과 호환성이 좋다는 점 등의 장점을 갖고 있기 때문에 많은 프로그래머들로부터 여전히 사랑받는 프로그래밍 언어이다. 하지만 C와 C++로 생성된 프로그램은 버퍼 오버플로 취약점과 같은 다양한 메모리 커럽션 취약점을 갖고 있기 때문에 쉽게 외부의 공격자들에게 노출된다. 따라서 사용자가 이와 같은 언어로 생성된 프로그램을 안전하게 실행하기 위해서는 시스템에 신뢰할 수 있는 방어 기법이 미리 적용되어 있어야만 한다. 본 연구에서는 메모리 커럽션 취약점을 이용한 공격을 탐지하기 위한 기법 중 하나인 Multi-variant Execution Environment (MVEE)를 소개하고, MVEE의 핵심 요소 기술인 레플리카 생성 기법들에 대하여 설명하고 비교·분석하고자 한다.

1. 서론

프로그래밍 언어 관련 시장 조사 업체인 TIOBE에서 조사한 바에 따르면 2017년 9월에 사용된 프로그래밍 언어 중 C와 C++는 7.382%, 5.565%의 점유율을 기록하여 Java에 이어 각각 2위, 3위를 기록한 것으로 나타났다[1].

<표 1> 2017년 9월의 TIOBE Index[1]

Rank	Programming Launguage	Ratings
1	Java	12.687%
2	C	7.382%
3	C++	5.565%
4	C#	4.779%
5	Python	2.983%
6	PHP	2.210%
7	JavaScript	2.017%
8	Visual Basic .NET	1.982%
9	Perl	1.952%
10	Ruby	1.933%

보다 직관적으로 쉽고 편하게 사용할 수 있는 다양한 프로그래밍 언어가 개발된 오늘날에도 이처럼 C와 C++가 널리 사용되고 있는 것은 기존 시스템과의 호환성이 좋고 컴파일된 결과물의 실행 속도가 빠르다는 장점이 있기 때문이다. 하지만 C나 C++을 사용하여 프로그래밍 된 프로그램에는 악용될 여지가 있는 메모리 커럽션 취약점이 존재한다.

예를 들면 한 프로그램이 기존에 선언된 어레이에 접근하고자 할 때, 미리 할당된 메모리 범위 내의 인덱스에 올바르게 접근하는지를 매번 확인하는 Java와는 달리 C와 C++는 이러한 확인 과정을 거치지 않는다. 이로 인해 C나 C++로 구현된 프로그램은 공격자가 시스템의 메모리에 원하는 값을 쓰는 것을 가능토록 하는 버퍼 오버플로 취약점을 갖게 된다.

위와 같은 메모리 커럽션 취약점의 존재는 이를 이용한 공격 기법들의 등장으로 이어졌다. 예를 들면, 취약점을 사용하여 실행하고자 하는 악성 코드를 공격 대상이 될 시스템의 스택에 삽입한 뒤, 리턴 주소나 점프 주소 등을 조작하여 프로그램의 제어 흐름 (control flow)을 삽입한 악성 코드의 위치로 이동하여 이를 실행시키는 공격이 가능하다. 이와 같은 공격 방식을 “코드 삽입 공격”이라고 한다.

하지만 코드 삽입 공격은 W⊕X protection이라고 하는 간단한 방어 기술로 무력화 시킬 수 있다. 메모리 영역을 “쓰기 가능 (writable)” 혹은 “실행 가능 (executable)” 중 한 가지 권한만 가질 수 있도록 설정함으로써 스택과 같은 데이터 영역에 공격자가 새로 삽입한 악성 코드의 실행을 차단하는 것이다.

W⊕X protection과 같은 방어 기술을 우회하기 위해 새롭게 개발된 공격 방식이 코드 재사용 공격 (Code

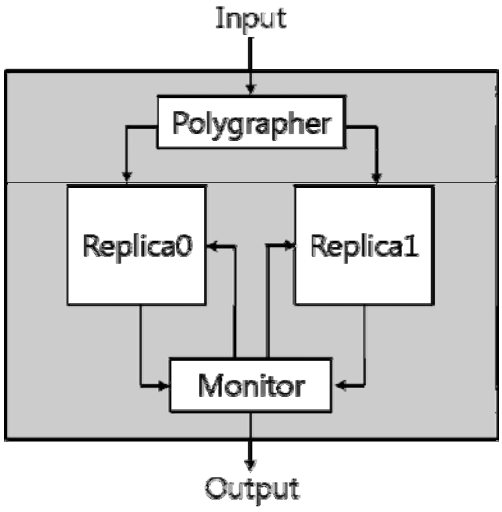
Reuse Attack, CRA)으로 오늘날에도 수많은 시스템에게 위협이 되고 있다. CRA는 공격자가 실행하고자 하는 악성 코드를 스택에 삽입하는 것이 아니라 실행 가능 영역의 코드의 조각 (gadget)들을 모아서 공격자가 원하는 행동을 하도록 만드는 공격 방식이다. CRA는 코드 삽입 공격과는 방식이 다르지만 메모리 커럽션 취약점을 이용한 스택 내의 리턴주소나 점프주소의 변경이 선행된다는 점에서 궤를 같이한다.

이처럼 나날이 발전하고 있는 메모리 커럽션 취약점 기반의 공격들을 막기 위해서 많은 연구자들이 새로운 방어 기술의 개발에 매진하고 있으며 multi-variant execution environment (MVEE) 역시 보다 안정적으로 다양한 범위의 공격을 막고자 개발된 방어 기법 중 하나이다. MVEE는 하나의 프로그램을 서로 조금씩 다른 형태의 여러 개의 레플리카 (replica)들로 복제한 뒤 이들을 동시에 실행시키고 이들의 행동을 모니터링 함으로써 공격이 발생했는지를 탐지한다.

MVEE가 보다 다양한 공격 벡터를 신뢰성 있게 탐지하기 위해서는 레플리카들을 생성하는 기법을 신중하게 선택하는 것이 매우 중요하다. 따라서 본 논문에서는 MVEE의 핵심 요소 기술인 레플리카 생성 기법들에 대하여 조사하고, 이들을 비교·분석 하고자 한다.

2. MVEE란?

Multi-variant execution environment (MVEE)의 개념은 2006년 Cox에 의하여 최초로 고안되었다 [2]. Cox가 제시한 N-variant 시스템 프레임워크는 그림 1과 같이 여러 개의 레플리카들과 폴리그래퍼, 모니터로 이루어져 있다.



(그림 1) N-variant System Framework[2]

N-variant 시스템에서 폴리그래퍼는 외부로부터 받은 입력 값을 복사하여 각각의 레플리카들에게 전달한다. 시스템의 프로세스는 조금씩 다른 형태로 복제된 여러 개의 레플리카들로 대체되어 있으며 이들은 각자 프로그램을 수행한 뒤 그 출력을 모니터로 전달하게 된다. 그리고 모니터는 각각의 레플리카들의 출력 값 (행동)을 비교하여 이들이 일치하지 않을 경우 인터럽트를 발생시킨다.

최근 구현된 MVEE 시스템들에서는 하나의 모니터가 폴리그래퍼의 역할까지 수행하며 통제하는 입출력 값은 대게 시스템 콜 넘버와 아규먼트, 그리고 시스템 콜의 리턴 값을 사용한다.

3. 레플리카의 생성 기법

앞서 설명한 것과 같이 MVEE에서 실행되는 여러 개의 복제된 레플리카는 외부의 침입을 받지 않은 정상적인 상태일 때는 서로 같은 행동을 보이지만 공격 시도가 있었을 시에는 서로 불일치한 행동을 보이게 된다. 따라서 이러한 조건을 만족할 수 있도록 레플리카들을 생성하는 기법이 필요하며 이들이 어떠한 관점에서 다각화 되었는지에 따라 탐지할 수 있는 공격의 종류나 범위가 달라질 수 있으므로 올바른 레플리카 생성 기법의 선택이 MVEE 시스템의 완성도에 영향을 미칠 수 있다. 지금부터는 그동안 MVEE 시스템을 구현한 논문들에서 어떠한 방식의 레플리카 생성 기법을 사용했는지를 설명하고자 한다.

3.1 주소 공간 파티셔닝

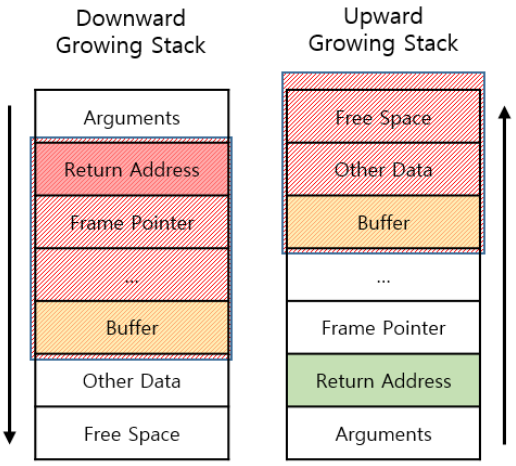
상술한 [2] 에서는 두 가지의 레플리카 생성 기법을 제시하고 있다. 첫 번째는 주소 공간 파티셔닝 (address space partitioning)을 기반으로 한 레플리카 생성 기법이다. 각 레플리카가 사용할 수 있는 유효한 주소 공간을 서로 겹치지 않도록 함으로써 절대 주소를 기반으로 하는 모든 메모리 커럽션 기반 공격을 피할 수 있다. 공격자가 format string [3], integer overflow, double-free [4]등의 취약점을 사용하여 프로그램의 제어 흐름 자신이 의도한 절대 주소로 점프하도록 하더라도 이는 다른 레플리카에서는 유효하지 않은 주소가 될 것이므로 행동 불일치가 발생하게 되고 모니터가 탐지할 수 있게 된다. 또한 두 레플리카의 주소 공간 불일치로 인하여 코드 재사용 공격 역시 매우 어려워지게 된다.

3.2 명령어 세트 태깅

[2]에서 제시한 두 번째 레플리카 생성 기법은 시스템의 명령어 세트를 태깅하는 방법을 사용하였다. 만약 두 레플리카가 x86과 PowerPC과 같이 서로 다른 종류의 프로세서 위에서 실행된다면 공격자가 두 레플리카가 같은 행동을 하도록 하면서 동시에 공격하는 것은 불가능에 가까울 것이다. 하지만 이러한 시스템을 구현하는 데에는 상당한 어려움이 따르므로 보다 간단하면서도 비슷한 효과를 보기 위해 명령어 세트를 태깅하는 방법을 생각해볼 수 있다[6]. 예를 들어 두 개의 레플리카의 명령어 세트에 각각 하나의 비트를 추가하고 이를 0, 1로 다르게 설정할 수 있다. 즉, 추가된 비트가 0인 명령어는 레플리카1에서는 존재하지 않는 명령어가 되는 것이다. 이렇게 할 경우 공격자가 코드 삽입 공격을 시도할 때 모든 레플리카에서 실행이 가능한 공격 코드를 따로 삽입하는 것이 불가능

하므로 레플리카간의 행동 불일치가 생기고 이를 모니터가 탐지할 수 있게 된다. 하지만 이와 같은 레플리카 생성 기법을 사용할 시에는 생성된 레플리카들의 실행 코드 주소는 여전히 동일하므로 코드 재사용 공격은 유효하다.

3.3 스택의 확장 방향성



(그림 2) Reverse Stack Execution[3]

B. Salamat가 제시한 MVEE 시스템인 "Orchestra"에서는 두 개의 레플리카가 사용하는 스택이 자라나는 방향을 서로 다르게 함으로써 레플리카의 생성 조건을 만족시켰다[5]. 이렇게 할 경우 그림 2와 같이 공격자가 버퍼 오버플로 취약점을 사용하여 어떠한 레플리카의 리턴 주소를 덮어 쓰더라도 다른 레플리카의 스택에서는 해당 리턴 주소가 덮어 쓰이지 않은 채로 남아있기 때문에 두 레플리카간의 행동의 차이가 생기고 모니터가 이를 감지하게 되는 것이다.

3.4 겹치지 않는 오프셋 공간

2016년에 K. Koning이 발표한 MvArmor에서는 3.1에서 설명한 것과 같이 주소 공간을 겹치지 않도록 함으로써 absolute spatial attack을 방지하는 것에 더하여 relative spatial attack을 막을 수 있도록 레플리카 생성 기법을 발전시켰다[8]. 이를 위해서 레플리카 간의 메모리 오브젝트들 사이에 충분한 공간을 둠으로써 한 어떠한 포인터에 오프셋을 더한 것이 하나의 레플리카에서만 유효하도록 한다. 하지만 MvArmor에서 채택한 레플리카 생성 기법도 use-after-free와 같은 취약점을 사용한 temporal attack을 완벽하게 막기에는 불충분하다.

3.5 레플리카 생성 기법들의 비교

위에서 설명한 레플리카 생성 기법들 외에도 "파일 재명명 (file renaming)", "로컬 메모리 재배치 (local memory rearrangement), "시스템 콜 번호 랜덤화[7]", "유저 ID 랜덤화" [2]등의 다양한 기술을 사용하여 레플리카들을 생성할 수도 있을 것이다.

하지만 이와 같은 레플리카 생성 기법은 각각 막을 수 있는 공격의 종류와 범위가 다르다. 이를 <표 2>와 같이 정리해볼 수 있다. 명령어 세트 태깅 같은 경우는 막을 수 있는 공격이 한정되어 있지만 단순히 실행되는 명령어에

<표 2> 레플리카 생성 기법에 따른 탐지 공격

Rank	주소 공간 파티셔닝	명령어 세트 태깅	스택의 확장 방향성	겹치지 않는 오프셋 공간
코드 삽입 공격	✓	✓	✓	✓
코드 재사용 공격	✓			✓
absolute spatial attack	✓		✓	✓
relative spatial attack				✓
temporal attack				

추가되는 비트만 확인하면 되기 때문에 시스템의 성능 오버헤드가 크게 발생하지 않으므로 가벼운 시스템에 적합할 것이다. 그와 반면에 현재 개발되어 있는 레플리카 생성 기법 중에서는 MvArmor에서 사용된 겹치지 않는 오프셋 공간을 활용한 레플리카 생성이 가장 다양한 공격 벡터에 대처할 수 있는 것을 알 수 있으며 따라서 보안이 중시 되는 은행 서버와 같은 시스템에 적용되면 좋을 것이다. 하지만 이 역시 temporal 공격을 완전히 막는 것은 불가능 하다. 따라서 향후에 temporal attack을 막을 수 있는 레플리카 생성 기법을 개발하고 이를 기존에 개발된 생성 기법과 적절히 조합하여 레플리카를 생성한다면 보다 안전한 MVEE 시스템을 만들 수 있을 것이다[9].

4. 결론

메모리 커럽션 취약점을 사용한 공격을 막기 위한 기술 중 하나인 multi-variant execution environment (MVEE)는 하나의 프로그램을 여러 개의 다각화된 레플리카로 복제하고 이들 사이의 행동 비교를 통해 공격을 탐지한다. MVEE의 중요한 요소기술인 레플리카 생성에는 다양한 기법이 활용되며 필요에 따라 시스템에 맞는 방식을 채택할 수 있다. 기존에 개발된 레플리카 생성 기법들을 분석한 결과 현재로서는 MVEE를 사용하여 temporal attack을 탐지하는 것이 어렵다는 사실을 알 수 있었으며 이에 대응할 수 있는 신규 레플리카 생성 기법의 개발이 필요할 것으로 생각된다. 개발된 레플리카 생성 기법을 기존의 기법과 조합하여 레플리카를 생성한다면 MVEE의 보안성 개선에 도움이 될 것으로 예상된다.

5. ACKNOWLEDGEMENT

이 논문은 2017년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.2017-0-00213, 능동적 사전보안을 위한 사이버 자가변이 기술 개발)

참고문헌

[1] http://www.tiobe.com/tiobe-index/

[2] B. Cox et al., “N-variant systems: A secretless framework for security through diversity,” in USENIX Security, 2006.

[3] U. Shankar et al., “Detecting Format String Vulnerabilities with Type Qualifiers,” in USENIX Security, 2001.

[4] J. Erickson et al., “Hacking: The Art of Exploitation,” by No Starch Press, 2003

[5] B. Salamat et al., “Orchestra: Intrusion detection using parallel execution and monitoring of program variants in user-space,” in Eurosys, 2009.

[6] G. S. Kc et al., “Countering code-injection attacks with instruction-set randomization,” in Conference on Computer and Communications Security (CCS), 2003.

[7] M. Chew et al., “Mitigating buffer overflows by operating system randomization,” Technical report, Department of Computer Science, Carnegie Mellon University, 2002.

[8] K. Koning et al., “Secure and efficient multi-variant execution using hardware-assisted process virtualization,” in Conference on Dependable Systems and Networks (DSN), 2016.

[9] S. Volckaert et al., “Secure and efficient application monitoring and replication,” in USENIX Annual Technical Conference (ATC), 2016.