공격 수행 방식 관점에서의 코드 재사용 공격 기법 분석

이정민*, 권순홍**, 손우영*, 이종혁**

*세종대학교 프로토콜공학연구실 (학부생)
**정보보호학과 & 지능형드론 융합전공, 세종대학교 (대학원생, 교수)

Analysis of Code Reuse Attack Techniques from Attack Performance Methods Perspective

Jeongmin Lee*, Soonhong Kwon**, Wooyoung Son*, Jong-Hyouk Lee**

*Protocol Engineering Lab., Sejong University (Undergraduate student)
**Dept. of Computer and Information Security & Convergence Engineering for Intelligent Drone, Sejong University (Graduate student, Professor)

요 약

최근 발생한 러-우 전쟁을 통해, 국가 주요정보통신기반시설을 타겟으로 한 사이버 공격이 우선적으로 진행됨을 확인할 수 있다. 이는 주요정보통신기반시설을 시스템 및 네트워크 관점에서의 보안 위협으로부터 보호할 수 있어야 함을 시사한다. 이미 수많은 소프트웨어 공격자들은 메모리상의 코드를 재사용하여 임의의 코드를 실행하는 방식인 코드 재사용 공격을 수행함으로써, 방대한 피해를 입힌 경우가 다수 존재한다. 이에 최근에는 해당 공격에 대한 소프트웨어 및 하드웨어 기반의 보안 메커니즘이 적용되고 있으나, 고도화된 공격 역시 빠르게 연구/개발되고 있어 사전에 공격 기술의 발전 방향성을 파악하고 이를 대응하기 위한 노력이 요구되고 있다. 이에 본 논문에서는 코드 재사용 공격 기법들의 특징을 비교/분석함으로써 코드 재사용 공격 기법의 발전 방향성을 예상하여 보호 메커니즘을 연구/개발하는 데 있어 기틀을 제공하고자 한다.

I. 서론

최근 러시아와 우크라이나의 전쟁은 국가 주요정보통신기반시설이 사이버 공격의 우선적 목표가 될 수 있음을 보여주었다. 이러한 사례를 통해 시스템 및 네트워크 관점에서 소프트웨어 보안의 중요성을 확인할 수 있다. 소프트웨어 공격자들은 일반적으로 메모리 상의 코드를 재사용하여 임의의 코드를 실행하는 공격 방식인 코드 재사용 공격(CRA, Code Reuse Attack)을 통해 목적을 달성한다. 최근에는 FineIBT, Shadow Stack 등 소프트웨어 및 하드웨어 기반의 제어 흐름 무결성(CFI, Control Flow Integrity) 보안 메커니즘이 적용되고 있으나, 고도화된 공격 역시 빠르게 연구/개발 되고 있는 실정이다. 이에 본 논문에서는 CRA 기술을 비교/분석함으로써 사전에 공격 발전 방향성을 파악하고, 보호 메커니즘을 연구/개발하는데 기틀을 제공하고자 한다.

본 논문의 2장에서는 주요 CRA 기법의 공격 수행 방식을 분석하며, 3장에서는 각 기법의 특징들을 비교함으로써 CRA의 발전 방향성을 예상한다. 4장에서는 본 논문의 결론을 맺는다.

II. 코드 재사용 공격 기법

국가 주요정보통신기반시설을 사전에 보호하기 위해서는 하드웨어 뿐만 아니라 소프트웨어 보안에 대해서도 고려되어야 한다. 이에 본 장에서는 공격자들이 시스템 제어권을 탈취하는 등의 목적을 달성하기 위해 지속적으로 고도화되어 등장하는 코드 재사용 공격 기법에 대해 분석한다.

CRA는 시스템 메모리 상에 존재하는 명령어를 재구성하여 수행하는 공격이다. CRA 등장 이전에는 코드 주입(Code Injection)을 통해 프로그램의 제어 흐름을 이동시켰으나, 이는 코드와 데이터 영역이 분리되지 않아 수행이 가능하였다. 예를 들어, 쉘 코드를 스택(Stack) 영역에 주입하고 PC(Program Counter)를 해당 주소공간으로 변조함으로써 임의의 코드를 실행, 제어 흐름을 탈취할 수 있다. 그러나 리눅스의 NX(Non-eXecutable), 윈도우의 DEP(Data Execution Prevention) 등의 메커니즘이 도입되며, 코드 주입 방식의 공격이 차단되었다.

ROP(Return-Oriented Programming) 공격은 메모리에 이미 존재하는 코드 조각인 가젯(gadget)을 활용함으로써 프로그램의 정상적인 실행 흐름을 변경하는 공격 기법을 말한다. 해당 공격이 수행되기 위해서는 스택 버퍼 오버플로우와 같이 메모리 조작 취약점을 통해 발생하는 메모리 손상 취약점이 존재해야 하며, 해당 취약점을 기반으로 공격자는 메모리 내 특정 포인터나 반환 주소를 조작할 수 있다. 'ret' 명령으로 종료되어 다른 가젯으로 넘어갈 수 있도록 설계된 가젯의 특징으로 인해, 공격자는 스택 내 반환 주소의 조작을 통해 여러 가젯을 연속적으로 연결함으로써 공격자의 의도에 따라 나열된 가젯들이 수행되게 한다. 이를 통해 최종적으로 공격자가 실행하고자 하는 공격 코드가 수행된다. 이와 유사하게 JOP 공격은 ROP 공격과 공격 수행 방식은 동일하나 'ret' 명령 대신 'jmp' 명령을 사용한다는 특징이 있다 [1][2].

DOP(Data-Oriented Programming) 공격은 실행되는 명령어가 공격자에 의해 의도된 명령어로 실행될 수 있도록 비제어 데이터(e.g., 상태 플래그, 포인터, 등)를 조작하는데 주 목적을 두고 있으며, 기존 공격 기법과 비교하여 제어 흐름을 변경하지 않는다는 특징이 존재한다. 해당 공격이 성공하기 위해서 메모리 손상 취약점, 데이터 지향 가젯, 가젯 디스패치의 세 가지 조건이 요구된다. 첫 번째로는 메모리의 특정 영역을 조작하기 위해서 프로그램 내 힙(Heap) 혹은 버퍼 오버플로우 취약점이 존재해야 한다. 두 번째로는 프로그램 내 공격자가 의도한 명령어가 실행될 수 있도록 가젯들이 요구되며, 본 공격에서는 가상 명령어 체계인 MINDOP을 기반으로 하여 데이터 지향 가젯을 구성함으로써 제어 흐름을 변경하지 않고 메모리 데이터를 변조할 수 있도록 하였다. 세 번째로는 프로그램 내에서 가젯을 반복적으로 사용할 수 있도록 하는 가젯 디스패치가 요구된다. 단일 가젯으로는 공격자가 요구하는 바를 달성할 수 없음에 따라 MINDOP 가젯 체인을 구성하여 복잡한 연산이 가능하도록 함으로써 공격자가 의도한 명령어를 실행할 수 있다 [3].

BOP(Block-Oriented Programming 공격은 앞서 설명한 DOP와 동일하게 제어 흐름을 변경하지 않는 것을 기반으로 하여 공격자가 의도한 명령어 시퀀스를 실행할 수 있도록 하는 공격이다. BOP 공격에 대한 이름을 통해 유추할 수 있듯이 프로그램 내부의 기본 블록을 가젯으로 활용하여 DOP와 동일하게 비제어 데이터를 변조하여 목표를 달성한다. BOP 공격의 특징으로는 현 시점에서 일반적으로 적용되어 있는 SSP(Stack Smashing Protector), NX Bit, ASLR(Address Space Layout Randomization) 이 활성화되어 있을 뿐만 아니라, 최근 화두가 되고 있는 CFI 및 Shadow Stack이 적용되어 있는 상황에서 공격이 수행된다는 특징이 있다. BOP 공격 역시 메모리 손상 취약점을 기반으로 하고 있으며, 튜링 완전 언어인 SPL(SPloit Language)을 기반으로 공격 페이로드를 작성한다. 이때, SPL 기반 공격 페이로드는 프로그램 기본 블록으로 변환하고, 체이닝 과정을 통해 프로그램의 CFG(Control Flow Graph)를 따르면서 CFI를 위반하지 않도록 한다. 이를 통해 제어 흐름을 벗어나지 않으면서 공격자가 의도한 명령어 시퀀스가 실행될 수 있다 [4].

POP(Page-Oriented Programming) 공격은 임의 커널 메모리 읽기/쓰기 취약점을 통해 공격자가 커널 내 민감 함수로의 제어 흐름을 생성하는데 주 목적을 둔다. 이는 기존 공격 기법들과 비교하여 직접 분기를 활용하여 제어 흐름 무결성 기법을 우회한다는 특징이 존재한다. 해당 공격은 페이지 카빙, 페이지 스티칭, 페이지 플러싱 세 단계로 공격이 수행된다. 첫 번째로 페이지 카빙 단계에서는 커널 바이너리에서 민감 함수와 호출 및 NOP(No-OPeration) 가젯을 식별한다. 해당 가젯을 통해 공격이 시작된 함수로 돌아가는 Round-trip 전략과 그렇지 않은 One-way 전략을 수행할 수 있다. 두 번째로 페이지 스티칭은 페이지 테이블을 재매핑함으로써 커널 내 민감 함수와 가젯을 결합한다. 이를 통해 기존 페이지와 동일한 페이지 오프셋을 가지는 가젯으로 대체함으로써 새로운 제어 흐름을 생성한다. 마지막으로 페이지 플러싱은 TLB(Translation Lookaside Buffer)를 플러시하여 공격 성공률을 높인다 [5].

III. 공격 수행 방식 관점에서의 코드 재사용 공격 기법 분석

본 장에서는 2장에서 언급한 CRA 기법에 대해 공격 수행 방식 관점에서 비교/분석을 수행하였으며, 이는 [표 1]을 통해 확인할 수 있다. 이를 통해, 향후 제시될 CRA 기법이 어떠한 특징을 가질지 예상하고자 한다.

CRA는 NX, DEP 등 데이터 실행 방지 정책을 우회하기 위한 배경으로 등장하였다. 이에 따라, CRA는 공통적으로 스택이나 BSS(Block Starting Symbol) 영역과 같이, 데

[표 1] 주요 CRA 기법 특징 비교/분석

CRA 기법	코드 주입 방식 배제	제어 흐름 탈취 수행 여부	DOA수행 여부	직접 분기 이용	페이지 테이블 수정
POP [5]	O	X	O	O	O
BOP [4]	O	X	O	X	X
DOP [3]	O	X	O	X	X
JOP [2]	O	O	X	X	X
ROP [1]	O	O	X	X	X

이터가 저장되는 영역에 쉘 코드 등을 주입하고 이를 실행함으로써 사용자 쉘을 탈취하는 코드 주입 방식을 사용하지 않고 공격을 수행하는 특징을 가진다.

초창기 CRA 기법은 대표적으로 ROP, JOP가 있다. 해당 공격 기법들은 제어 흐름 탈취(Control-flow Hijacking) 방식으로 프로그램의 제어 흐름을 전환한다. 제어 흐름 탈취 방식은 권한을 상승시키거나 정보를 유출하는 등의 목적은 수행하나, CFI 기반 보호 기법에 효과적으로 탐지되고 차단된다는 한계점을 가진다.

CFI 기반 보호 기법은 간접 분기 명령어 보안 정책을 적용함으로써 런타임(Runtime) 프로그램이 유효한 제어 흐름을 따르도록 강제한다. 이를 우회하기 위해, 제어 흐름과 관련 없는 비제어 데이터를 조작함으로써 유효한 제어 흐름 내에서 권한 상승, 정보 유출 등의 목표를 수행하는 DOA(Data Only Attack) 방식이 등장하였다. 이에 따라 DOP, BOP, POP 공격 기법은 DOA 방식을 채택함으로써 CFI 기반 보호 기법에 쉽게 탐지되는 기존 공격 방식의 한계점을 보완하였다.

가장 최근에 등장한 POP 공격 기법의 경우, 페이지 테이블을 수정함으로써 직접 분기를 통한 공격을 수행한다. 따라서, 해당 공격 기법은 간접 분기 명령어 보안 정책을 적용하는 CFI 기반 보호 기법과 달리 직접 분기를 사용함으로써 비교될 수 있는 DOP, BOP 기법에 비해 CFI 기반 정책에 자유롭고, 공격이 덜 복잡하다는 특징을 가진다.

앞서 수행한 공격 수행 방식 관점에서의 CRA 분석을 통해, 최신 CRA 기법들은 CFI 기반의 보호 기법이 널리 사용되는 추세에 맞추어 DOA 기법을 채택함을 확인하였다. 또한, CFI 기반 보호 기법에 보다 자유로운 공격을 수행하기 위해서 페이지 테이블을 수정하는 방식을 통해 직접 분기를 활용하는 공격 방식을 채택함을 파악하였다. 이를 통해, 향후 직접 분기를 활용하는 DOA 기반 CRA 기법이 등장할 것으로 예상된다.

IV. 결론

본 논문에서는 대표적인 CRA 기법에 대해 동작 과정 및 특징을 기반으로 분석함으로써, CRA 기법의 발전 방향성을 예상하였다. 해당 내용을 기반으로 향후 연구에서는 DOA 기법 및 직접 분기 방식을 채택하는 공격 루트를 파악하고, 이를 효과적으로 보호할 수 있는 방안을 연구하고자 한다.

Acknowledgement

이 논문은 2024년도 한국과학기술정보연구원(KISTI)의 자체사업으로 수행된 연구입니다 (과제번호: (KISTI)J24JR007241). 본 연구는 과학기술정보통신부 및 정보통신기획평가원의 정보통신방송혁신인재양성(메타버스융합대학원)사업 연구 결과로 수행되었음(IITP-2023-RS-2023-00254529).

[참고문헌]

[1] H. Shacham, "The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86)," *in Proc. 14th ACM Conf. on Comput. and Commun. Secur.*, pp. 552-561, 2007.

[2] T. Bletsch et al., "Jump-oriented programming: a new class of code-reuse attack," *in Proc. 6th ACM Symp. on Inform., Comput. and Commun. Secur.*, pp. 30-40, 2011.

[3] H. Hu et al., "Data-oriented programming: On the expressiveness of non-control data attacks," *in Proc. 2016 IEEE Symp. on Secur. and Privacy (SP)*, pp. 969-986, 2016.

[4] K. K. Ispoglou et al., "Block oriented programming: Automating data-only attacks," *in Proc. 2018 ACM SIGSAC Conf. on Comput. and Commun. Secur.*, pp. 1868-1882, 2018.

[5] S. Han et al., "Page-Oriented Programming: Subverting Control-Flow Integrity of Commodity Operating System Kernels with Non-Writable Code Pages."