Return-Oriented Programming(ROP) 탐지 및 방지 기술 분석

배효빈*, 민재원**, 박민우**, 정태명***
*성균관대학교 컴퓨터공학과
**성균관대학교 전자전기컴퓨터공학과
***성균관대학교 정보통신대학
{hbbae, jwmin, mwpark}@imtl.skku.ac.kr, tmchung@ece.skku.ac.kr

Survey of Return-Oriented Programming(ROP) Detection and Prevention Methods

Hyo-Bin Bae*, Jae-Won Min**, Min-Woo Park**, Tai-Myoung Chung***
*Dept of Computer Engineering, SungKyunKwan University
**Dept. of Electrical and Computer Engineering, SungKyunKwan University
***College of Information and Communication Engineering, SungKyunKwan University

요 약

Return-Oriented Programming(ROP) 공격은 버퍼 오버플로우 공격, Return-into libc 공격의 계보를 이어 소프트웨어의 취약점을 공격하는 대표적인 기술 중 하나이다. 이 공격은 윈도우 운영체제 상에서의 Exploitation, iOS DEP 우회 및 코드 사이닝과 같은 기술을 무력화하기 위해 사용되고 있는 취약점 공격법이다. 그렇기 때문에 ROP 공격이 소개된 이후부터 현재까지 탐지법 및 방어법에 대한 논의가 활발하게 이루어지고 있다. 본 논문에서는 이러한 ROP 공격을 막기 위한 방법들을 특징에 따라 세 가지 종류로 분류하여 소개하고, 각각의 방법들의 장점과 단점을 비교 분석하여 향후 ROP 방어에 관한 연구에 기여를 하고자 한다.

1. 서론

급격한 인터넷 사용의 증가로 인해 해킹, 악성코드 등과 같은 소프트웨어 상의 취약점을 이용한 Exploitation 공격이 늘어나고 있다. 이러한 공격들은 프로그램이나 시스템의 가용성 뿐 만 아니라 데이터의 무결성, 기밀성 까지도 훼손시킬 수 있어 치명적이기 때문에 방어에 더욱 큰 노력을 기울여야 한다.

최근 Exploitation 공격 중에서 가장 빈번하게 사용되는 것이 Return-Oriented Programming(ROP)[1]이다. 이 공격법은 iPhone, Sequoia의 AVC 등 과거의 공격방식인 코드 삽입으로는 공격이 불가능했던 플랫폼의 손상과 운영체제의 익스플로잇 방어 기술을 우회하는 데 활발하게 이용되고 있다. 또한, ROP 공격을 쉽게 하는 다양한 도구들이 개발되어 나오고 있다. ROP를 발표했던 당시에는 자신이 원하는 ROP chain을 만들기 위하여 해커 자신이 직접 라이브러리를 분석하여 가젯을 찾아야 했으나, 현재는 이뮤니티 디버거의 Pycommand인 mona.py를 이용해 자동적으로 찾아낼 수 있다[2]. 또한 최근에는 ROP 용 컴파일러인 ROPC(ROP Compiler)도 선을 보였다[3]. 이런 ROP 공격 도구들은 쉽게 구할 수 있고, 이를 이용하면 많은 배경지식 없이도 정교한 공격이 가능하다.

ROP가 여러 플랫폼을 공격하는데 이용되고, ROP 공격 코드를 만들기 위한 진입장벽이 점점 낮아지고 있어 학계나 산업계에서는 그에 대한 방어법을 활발하게 연구하고 있다. 하지만 근본적인 ROP의 방어법이 아직 제안되지 않아 모두 우회가 가능하며, 실제로 도입하여 사용하기에는 부족함이 많다. 본 논문에서는 현재까지 제안된 여러 방어법들을 특징에 따라 분류한 뒤 각각의 장단점을 분석한다. 그리고 분석을 통해 ROP 공격의 위험성을 알리고 지금까지의 방어 방법들의 한계점을 드러내어, 새로운 근본적 패러다임이 제안되기를 기대한다.

본 논문의 구성은 다음과 같다. 1장의 서론에 이어, 2장에서는 Exploitation공격의 발전 방향과 ROP 공격에 대한 설명을 하고, 3장에서 방어 방법들을 분류하여 소개한 뒤 4장에서 분류된 그룹들을 비교·분석 한다. 마지막으로 5장에서 결론과 향후 연구를 제시한다.

2. 관련연구

2.1 Exploitation 기술의 발전

가장 먼저 나온 Exploitation 공격은 버퍼 오버플로우 공격이다[4]. 이 공격은 스택 상의 리턴 주소 값을 덮어씀으로서 프로그램이 정상적인 흐름에서 벗어나게 하고 프로세스 이미지에 코드를 삽입함으로서 자신의 목적을 달성한다. 이 공격을 방어하기 위해 Visual C++에서는 GS옵션을 제공하고 있고[5], 인터넷 익스플로러 8 이후의 버전에서는 메모리 보호 기법으로서 DEP(Data Execution Protection)/NX(No Execution)이 개발되어 이용되고 있다[6]. 이 외에도 StackGuard[7], PointGuardTM[8]등의 방어

방법들이 학계에서 활발하게 발표되었다. 많은 방어법들이 쇄도 하였고, 특히 스택 메모리의 실행 권한을 없애 코드 삽입이 불가능해 지면서 버퍼 오버플로우 공격이 제 기능을 할 수 없게 되었고, 새로운 공격법이 필요하게 되었다.

1997년 Solar Designer는 버퍼 오버플로우를 발전시킨 Return-into libc(RTL)이라는 새로운 공격법을 제안하였다[9]. 이 공격법은 코드를 직접 삽입하는 것이 아니라, 오버플로우 시킨 스택의 리턴 주소에 해당 소프트웨어가 사용하는 C표준 라이브러리인 libc의 함수 주소를 연속적으로 집어넣음으로서 자신이 원하는 행동을 수행하게 만든다. 하지만 이런 RTL 공격은 버퍼 오버플로우 공격에 비해 제한적이다. 첫째, 버퍼 오버플로우는 자신이 원하는 코드를 직접 삽입할 수 있는 반면 RTL은 이미 주어져 있는 libc함수만을 조합해 사용해야 한다. 둘째, 자신이 원하는 코드가 libc 내에 존재하지 않는 경우 공격이 불가능하다. 셋째, 특정 함수를 이용하여 libc공격이 성공하더라도 해당 공격을 알아챈 libc관리자가 공격에 관계된 함수를 삭제하거나 수정하는 경우 공격은 실패로 끝나게 된다.

2.2 Return-Oriented Programming(ROP)

2007년 Shacham은 이러한 RTL공격의 단점을 보완하기 위해 Return-Oriented Programming(ROP)를 발표하였다[1]. RTL이 함수 첫 부분으로 점프를 해 함수 전체를 읽어오는 방식이었던데 반해 ROP는 자신에게 필요한 부분으로 점프하여 가젯이라고 불리는 어셈블리어 조각들을 순차적으로 불러들이는 방법으로 동작한다. 이러한 이유로 ROP는 RTL에 비해 유연하고 정교한 코드의 작성이 가능하다.

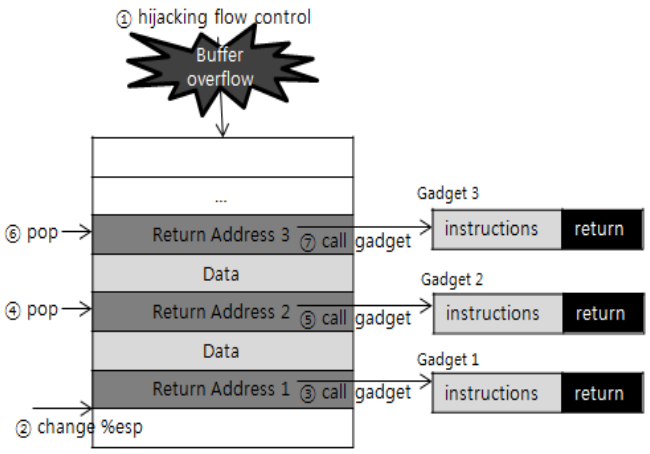
```
Gadget 1
C7  07 00 00 00 0f      movl $0x0f000000, (%edi)
95                      xchg %ebp, %eax
45                      inc %ebp
c3                      ret

Gadget 2
81 c4 88 00 00 00       add $0x00000088, %esp
5f                      pop %edi
5d                      pop %edp
c3                      ret
```

(그림 1) 가젯 [1]

ROP에서 가장 중요한 개념은 취약한 프로그램 내부의 어셈블리 코드의 조각들인 가젯이다. 가젯은 실행 권한이 있는 모든 메모리 내에 존재하는 코드의 조각으로서, (그림 1)과 같이 return 명령어로 끝나는 어셈블리어 리스트로 이루어져 있다. ROP에서는 이러한 가젯들을 연속적으로 불러와 의도하는 작업을 수행하도록 한다. 충분히 많은 수의 가젯 만 존재 한다면 악의적 사용자는 자신이 원하는 임의의 코드를 제한 없이 생성해 낼 수 있다.

ROP의 수행과정은 (그림 2)와 같다. 우선 첫 번째로 버퍼 오버플로우를 이용하여 프로그램의 메모리 제어 권



(그림 2) Return-Oriented Programming 수행과정

한을 얻어 온다. 그 이후 스택 포인터인 esp 레지스터 값을 첫 번째 리턴 주소 영역으로 옮겨 준다. 해당 주소 영역에 자신이 원하는 가젯의 주소를 넣어 연속적인 명령어들을 불러오고 return을 만나면 pop하여 다음 리턴 주소 영역으로 스택 포인터 값을 옮긴다. 그리고 이 리턴 주소의 값에 다음으로 자신이 불러와야 하는 가젯의 주소를 넣고 해당되는 가젯을 불러온다. 자신이 필요로 하는 모든 가젯을 다 불러 올 때 까지 이러한 과정을 반복하며 가젯을 실행해 ROP chain을 만들어 공격에 이용한다

3. 감지 및 방어법 분류 및 분석

ROP를 막는 방법은 방어하는 시점 따라서 크게 컴파일러 기반의 방어, 바이너리 패칭, 런타임 방어 이렇게 세 가지로 크게 나눠진다. 3장에서는 현재까지 제안된 방어방법들을 시점 별로 분류하여 각각에 대한 설명을 한다.

3.1 Compiler-based mitigation

컴파일러 기반 방어는 ROP가 발생하기 이전에 사전조치를 하는 방법이다. 이 방법의 예로는 우선 G-Free[10]가 있는데, 이것은 컴파일 시간동안 공격자의 가젯으로 쓰일 수 있는 ret, jmp*/call* 명령어를 삭제하거나 무력하게 만들어 ROP 공격에서 필요로 하는 가젯들을 제공하지 않는 방법이다. 하지만 이러한 동작을 하게 하는 명령어 재작성 기술과 새로 삽입되는 코드로 인하여 일반적인 경우에 비해 G-Free로 컴파일 한 경우에는 약 25.9%의 라이브러리 사이즈 오버헤드가 발생하고 실행시간에도 약 3.1%의 오버헤드가 발생한다. 또한 중요한 G-Free의 단점은 모든 라이브러리들이 해당 기술을 이용해 컴파일 되어 있지 않으면 사실상 제대로 된 방어를 할 수 없기 때문에 이미 컴파일 한 것들도 이 방법을 이용해 재 컴파일 해야 할 필요성이 있다는 것이다. 하지만 G-Free는 return이외의 방법을 이용해 가젯을 만드는 ROP의 향상된 버전인 indirect jump 또한 방어가 가능한 장점이 있다.

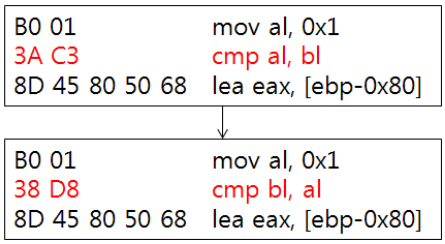
그리고 또 다른 예로 컴파일러를 기반으로 커널 레벨

에서 Return-Oriented 루트킷을 방어하는 방법이 존재한다[11]. 이 방어법 역시 return 명령어를 삭제하거나 무력하게 만들어 ROP를 저지한다. 또한 이것 역시 컴파일러를 기반으로 한 방어법이기 때문에 약 10%의 사이즈 오버헤드와 20%이하의 실행 오버헤드가 발생한다.

3.2 Binary level mitigation

이미 컴파일이 된 바이너리 코드 수준에서 ROP를 방어하는 방법으로서, 크게 프로그램 실행 시 마다 가상주소 공간의 위치를 바꾸는 메모리 랜덤화방법과, in-place code randomization을 이용하는 Binary Patching방법이 있다.

메모리 랜덤화는 여러 가지의 ROP 방어법 중 가장 널리 쓰이고 있다. 이 방법의 예로는 윈도우즈, 리눅스, 맥 OS X 등 여러 OS에서 사용되고 있는 Address Space Layout Randomization (ASLR)이 있다[12]. 이 방법은 프로세스의 가상주소공간에 객체가 매핑 될 때, 유저 어플리케이션의 힙, 스택, 데이터 및 공유 라이브러리의 가상주소를 프로그램 실행 시 마다 랜덤하게 변경하여 ROP 공격이 필요로 하는 가젯을 제대로 불러 올 수 없게 한다. 하지만 랜덤화 개수가 256개뿐이라는 단점을 가지고 있다.



(그림 3) Binary patching[13]

Binary Patching[13]은 기본 코드 블록 자체의 위치를 임의로 바꾸는 Randomization과 달리 블록의 위치는 옮기지 않은 상태로 위의 (그림 3)와 같이 코드의 의미는 해치지 않게 재배치하여 기계어 코드를 바꿈으로서, ROP가 의도된 동작을 수행할 수 없도록 만든다.

3.3 Runtime detection

이 방법은 실제 공격이 발생하는 실행 시간에 ROP 공격을 감지하여 해당하는 악의적 프로그램을 방어하겠다는 것으로서 공격에 대한 후속조취에 해당한다. 이러한 실행 시간 감지의 방법으로 가장 흔하게 쓰이는 것이 빈번한 return의 반복을 감지하여 ROP를 특정하고 방어하는 방법, 그리고 컨트롤 플로우을 감시하여 비정상적인 경우 ROP로 간주하고 탐지해 내는 방법이다.

ROP 공격은 탐지되기 쉬운 몇 가지의 자신만의 특징을 가지는데 그 중 가장 눈에 띄는 것이 바로 return 명령문의 빈번한 사용이다. ROP가 이용하는 가젯은 보통 return으로 끝이 나는 2, 3줄의 명령어 조각이며, 공격의 특성 상 가젯을 조립하여 자신이 의도한 공격 코드를 만들어야 하므로 return 명령문의 사용빈도가 높다. 이러한 ROP의 특징을 이용한 탐지 및 방지법에는 DROP[14], [15]이 있으며 이것들은 두 개의 return 명령문 사이에 자리한 명령어의 개수를 세고, ROP라 판단될 정도의 간격이 여러 번 반복되면 ROP 공격임을 확신한다. 이러한 공격을 회피하는 방법에는, 가젯의 길이를 늘이는 방법, indirect jump를 이용하는 방법이 있다..

Control flow 감시를 통한 방어 방법에는 크게 Control Flow Integrity(CFI)[16]와 Control Flow를 계속 감시하다가 흐름이 유저 모드에서 커널 모드로 넘어갈 때 적합성을 판단하는 kBouncer[17]가 있다.

CFI는 프로그램이 미리 정해진 Control Flow Graph(CFG)를 따라 진행되도록 보장하는 방법이다[16]. 컴파일 타임에 어셈블리 코드에 쓰이는 레지스터가 가리킬 수 있는 가능한 위치에 레이블을 심어놓고, 두 값을 비교한 후 jump해 무결성을 지킨다. 하지만 실행시간 이전에 미리 CFG를 정하기 때문에 동적 라이브러리와 같이 실행시간에 변경될 수 있는 프로그램에 대한 보장이 어렵다.

kBouncer는 시스템 콜을 통해 권한의 흐름이 유저 모드에서 커널 모드로 넘어가는 순간을 체크해 해당 시스템 콜을 불러낸 것이 정상적인 프로그램인지 ROP 공격인지를 판별한다. 이 때 사용하는 것이 최신 인텔 CPU에서만 제공되는 LBR 레지스터인데, 여타의 방법과 달리 하드웨어의 기능을 사용해 성능을 높였다. 하지만 이 방법으로는 indirect jump를 방어할 수 없고, 시스템 콜을 하기 직전에 LBR레지스터의 값을 합법적인 call이라 덮어쓰면 공격을 탐지할 수 없는 단점이 존재한다.

4. 탐지 및 방어법 비교분석

앞 장에서 종류별 방어법에 대한 소개를 하였고, 이번 장에서는 <표 1>과 함께 각각의 장단점을 정리 하였다.

사이즈 오버헤드의 경우는 컴파일러 기반 방어법에서만 발생한다. 그 이유는 컴파일러 기반 방식가 기존의 명령어를 재작성해 ROP에 가젯을 제공하지 않는 방법이기 때문이다. 바이너리 코드를 재생성해야 하기 때문에 그 과정에서 바이너리 크기가 증가하지만, 이외의 방법들은 바이너리 코드에 관여하지 않아 사이즈에 변화가 없다.

컴파일 기반의 방어법과 Binary level 방어법에서는 실행 전에 미리 방어를 수행하기 때문에 실행시간 오버헤드가 크지 않다. 하지만 Runtime detection 방법은 실행시간에 VM을 이용해 계속된 감시를 하면서 분석해 방어를 해야 하기 때문에 실행시간 오버헤드가 클 수밖에 없다. 반면에 kBouncer는 실행시간 오버헤드를 줄이기 위해 고안된 방법으로서, 계속 감시하고 있는 것이 아니라 시스템 콜이 발생 했을 때만 적합성 검사를 실시하고 LBR 레지스터라는 하드웨어 기능을 사용하기 때문에 일반적인 런타임 탐지에 비해 적은 오버헤드를 갖는다.

	Compiler-based		Binary level mitigation		Runtime detection		
	[11]	[12]	Randomization	Binary patching	Return의 반복	Flow control 감시	
						CFI	[18]
사이즈 오버헤드	O		X	X	X	X	
실행 시간 오버헤드	Low		Low	Low	High	High	Low
타겟 레벨	유저	커널	유저	유저	유저	유저	유저/커널
기타 단점	재컴파일 필요		한정된 랜덤화 개수	한정된 랜덤화 범위	indirect jump 방어 불가	정상 프로그램에 대한 오탐 가능성	LBR 덧쓰기를 통한 회피

<표 1> 비교 분석

타겟 레벨의 경우는 해당 방어법들이 유저모드와 커널 모드 중 어느 것을 타겟으로 설계되었는지에 대한 항목이다. kBouncer는 유저모드에서 커널모드로 넘어가는 순간에 적합성 검사를 하기 때문에 유저모드와 커널모드의 중간으로 표현했다.

마지막으로 그룹별로 비교하지 않은 기타 단점들은 표의 가장 아래 행에 기재되어 있다.

5. 결론

Return-Oriented Programming(ROP) 공격법은 최근 학회 및 해커들의 많은 관심을 받고 있으며, 이것을 주제로 한 많은 논문들이 제출되고 있다. 이 논문에서는 ROP와 관련된 논문 중 탐지 및 방어 방법에 대한 것들을 모아 방어 시점에 따라 분류하여 소개하고 비교 분석해 보았다. 이 방법들은 ROP 공격법에 대한 근본적 해결법이 아니기 때문에 우회가 가능하며, 실제로 이것들만을 가지고 방어하기에는 한계가 존재한다. 이러한 기술들의 장단점을 살피고 단점을 보완한 새로운 기술에 대한 연구가 필요할 것이다.

참고문헌

[1] Hovav Shacham. “The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls(on the x86)”. 14th ACM Conference on CSS, 2007

[2] mona.py, http://redmine.corelan.be/projects/mona, Oct, 2012

[3] ROPC, https://github.com/pakt/ropc, Sep, 2012

[4] Aleph One. “Smashing the stack for fun and profit”. Phrack Magazine, 49(14), Nov. 1996

[5] MSDN, http://msdn.microsoft.com/ko-kr/library/8dbf701c(d=lightweight).aspx, Sep, 2012

[6] A Sotirov, M Dowd “Bypassing Browser Memory Protections: Setting Back browser sercurity by 10 years”, blackhat, 2008

[7] C Cowan, C Pu, D Maier, J Walpole, P Bakke, S Beattie, A Grier, P Wagle, Q Zhang “StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks“, 7th USENIX Security Symposium, Jan, 1998

[8] C Cowan, S Beattie, J Johansen, P Wagle, “PointGuardTM: Protecting Pointers From Buffer Overflow Vulnerabilities", Security Symposium, Aug, 2003

[9] Solar Designer. “return-to-libc” attack, Bugtraq, Aug. 1997

[10] K Onarlioglu, L Bilge, A Lanzi, D Balzarotti, E Kirda “G-Free: Defeating Return-Oriented Programming through Gadget-less Binaries”, 26th ACSAC '10, 2010

[11] J Li, Z Wang, X Jiang, M Grace, S Bahram "Defeating Return-Oriented Rootkits With Return-less Kernels", EuroSys '10, April, 2010

[12] ASLR, http://pax.grsecurity.net/docs/aslr.txt, Oct, 2012

[13] V Pappas, M Polychronakis, A Keromytis, “Smashing the Gadgets : Hindering Return-Oriented Programming Using In-Place", IEEE Symposium on Security and Privacy, 2012

[14] P Chen, H Xiao, X Shen, X Yin, B Mao, L Xie "DROP: Detecting return-oriented programming malicious code" ICISS 2009, Dec, 2009

[15] L Davi, A Sadeghi, M Winandy "Dynamic integrity measurement and attestation: Towards defense against return-oriented programming attacks", STC2009, Nov. 2009

[16] M Abadi, M Budiu, U Erlingsson, J Ligatti "Control-flow integrity: Principles, implementations, and applications", CCS '05, 2005

[17] V Pappas, "kBouncer: Efficient and Transparent ROP Mitigation", Apr, 2012