pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2013.2.5.217

Indirect Branch Target Address Verification for Defense against Return-Oriented Programming Attacks

Soohyun Park† · Sunil Kim††

ABSTRACT

Return-Oriented Programming(ROP) is an advanced code-reuse attack like a return-to-libc attack. ROP attacks combine gadgets in program code area and make functions like a Turing-complete language. Some of previous defense methods against ROP attacks show high performance overhead because of dynamic execution flow analysis and can defend against only certain types of ROP attacks. In this paper, we propose Indirect Branch Target Address Verification (IBTAV). IBTAV detects ROP attacks by checking if target addresses of indirect branches are valid. IBTAV can defends against almost all ROP attacks because it verifies a target address of every indirect branch instruction. Since IBTAV does not require dynamic execution flow analysis, the performance overhead of IBTAV is relatively low. Our evaluation of IBTAV on SPEC CPU 2006 shows less than 15% performance overhead.

Keywords : Program Security, Return-Oriented Programming, Program Transformation

Return-Oriented Programming 공격 방어를 위한 간접 분기 목적 주소 검증 기법

박 수 현† · 김 선 일††

요 약

Return-Oriented Programming(ROP)는 기존 return-to-libc의 발전된 형태로 프로그램의 코드 영역에 있는 가젯을 조합하여 공격자가 원하는 모든 기능을 수행할 수 있는 코드 재사용 공격 기법이다. ROP 공격을 방어하는 기존 방어 기법들은 동적 실행 흐름 분석으로 인한 높은 성능 부하를 보이거나 ROP 공격에 대한 부분적인 방어만 가능하였다. 본 논문에서 제시하는 간접 분기 목적 주소 검증 기법(Indirect Branch Target Address Verification)은 간접 분기문의 목적 주소가 유효한지 검사해서 ROP 공격을 탐지하며, ROP 공격의 대부분을 방어할 수 있다. 또한 동적 실행 흐름 분석이 필요 없기 때문에 낮은 성능 부담을 보인다. SPEC CPU 2006 벤치마크를 대상으로 한 성능평가에서 15%보다 적은 성능 부하를 보였다.

키워드 : 프로그램 보안, Return-Oriented Programming, 프로그램 변환

1. 서 론

Return Oriented Programming(이하 ROP)[1]은 프로그램의 코드 영역 명령어를 사용하여 공격하는 또 다른 형태의 return-to-libc 공격 기법[2]이다. ROP 공격은 프로그램의 코드에서 사용 가능한 가젯(Gadget)들을 찾고 조합하여 공격자가 원하는 기능을 수행한다. 따라서 가젯으로 구성된 일종의 튜링-컴플리트(Turing-complete) 언어를 사용한다고 할 수 있다. 가젯은 메모리나 레지스터의 값들을 조작하여 필요한 연산을 수행할 수 있는 작은 코드 덩어리들이다. 가젯의 끝은 함수 반환 명령어(return)이나 간접 함수 호출(indirect call), 간접 점프 구문(indirect jump)과 같은 간접 분기문으로 구성되며, 간접 분기문의 목적 주소를 조작해서 다음 가젯으로 실행 흐름을 변경한다. ROP는 공격자가 코드를 직접 삽입할 필요가 없기 때문에 기존의 페이지 W⊕X 방어 기법[3]을 우회할 수 있다. 따라서 프로그램에서 버퍼 오버플로우[4]와 같은 취약점만 발견된다면 원하는 모든 기능을 수행할 수 있는 강력한 공격 기법이다.

ROP 공격을 방어하기 위해서 많은 연구가 선행되었다. 프로그램 실행 흐름 검증(Control-Flow Integrity, CFI)[5]에

† 준 회 원 : 홍익대학교 컴퓨터공학과 박사과정
†† 종신회원 : 홍익대학교 컴퓨터공학과 교수
논문접수 : 2013년 1월 14일
수 정 일 : 1차 2013년 3월 25일
심사완료 : 2013년 3월 28일
* Corresponding Author : Sunil Kim(sikim@cs.hongik.ac.kr)

기반한 기법[6,7]들은 실행 흐름을 제어하고 검증할 수 있기 때문에, ROP 공격을 정확히 막을 수 있다. 하지만 프로그램의 동적 실행 흐름을 분석하고 검증하기 위해 가상 머신이나 동적 코드 분석 기술[8,9]을 사용해야 하고, 이로 인해 높은 성능 부하를 보인다. CFI에 기반한 기법 외에도, 프로그램의 함수 반환 구문인 “return” 명령어를 사용한 초기 ROP 공격을 방어하는 기법들이 제시되었다[10-12]. 이들 중 ROPDefender[11]와 같은 기법들은 올바른 함수 반환 주소를 알아내기 위해서, CFI 기반 기법과 유사하게 동적 프로그램 분석 도구를 사용한다. 최근 ROP 공격 기법은 간접 함수 호출 및 간접 점프 구문도 사용하기 때문에 함수 반환 구문만 방어하는 기법들은 한계를 보일 수 밖에 없다[13,14]. 프로그램의 코드 영역에서 ROP 공격이 사용할 수 있는 가젯을 제거하는 방어 기법[15,16] 또한 제시되었다. 이 기법들은 프로그램의 정적 분석을 통해 가젯을 찾고 제거하는데, 정렬된 명령어뿐 아니라 명령어의 중간에서 시작되는 가젯들도 처리하여야 한다. 예를 들어 프로그램이 “mov 0xc3, %eax”라는 인텔 x86 명령어를 사용하는 경우, 이 명령어의 중간인 “0xc3”으로 분기가 가능하다. “0xc3”은 “return”의 코드이기 때문에 가젯이 함수 반환 구문으로 사용할 수 있다. 따라서 가젯을 제거하는 기법은 가젯이 정렬되지 않은 명령어를 사용하지 못하도록 차단해야 한다[16]. 정렬되지 않은 명령어를 사용하지 못하도록 하기 위해 주로 명령어 주소 정렬 기법을 사용한다. 이 기법은 NOP(No Operation) 등의 명령어를 삽입해서 명령어를 정렬한다. NOP의 사용으로 인해 프로그램 코드가 매우 커지고 상당히 큰 성능 부하를 보인다.

본 논문에서는 실행 흐름의 분석 없이도, 모든 간접 분기문을 조작하는 ROP 공격을 탐지할 수 있는 간접 분기문 목적 주소 검증 기법(Indirect Branch Target Address Verification - IBTAV)을 제시한다. IBTAV는 프로그램이 간접 함수 호출이나 간접 점프, 함수 반환 구문과 같은 모든 종류의 간접 분기문을 사용할 때, 해당 구문의 목적 주소가 유효한지를 검사하여 ROP 공격을 탐지한다. 유효한 주소는 프로그램의 정적 분석을 통해 얻으며, 동적 실행 흐름 분석이 필요 없다. 따라서 가상 머신이나 동적 프로그램 분석 도구를 사용하지 않기 때문에 낮은 성능 부하를 보인다.

본 논문은 다음의 순서로 구성되어있다. 2장에서 본 논문에서 제시하는 IBTAV의 기본적인 동작 원리에 대해서 살펴본 뒤 3장에서 IBTAV기법의 구현 내용을 설명한다. 4장에서는 IBTAV 기법의 성능 및 방어 능력을 평가하고, 마지막으로 5장에서 결론을 맺는다.

2. 간접 분기 목적 주소 검증 기법 (Indirect Branch Target Address Verification)

ROP 공격은 공격자가 원하는 기능을 수행하기 위해서 연속적으로 가젯을 호출한다. 가젯은 함수 반환 및 간접 함수 호출, 간접 점프 구문과 같은 모든 종류의 간접 분기문을 사용해서 필요한 다음 가젯을 호출한다.

본 논문에서 제시하는 IBTAV 기법은 모든 간접 분기문의 목적 주소가 유효한 지 검사하여 ROP 공격을 탐지하고 방어한다. 유효한 목적 주소는 간접 분기문이 사용할 수 있는 올바른 목적 주소를 의미한다. 예를 들어 아래 Fig. 1의 프로그램에서 함수 포인터를 사용해서 foo 또는 bar 함수를 호출하고자 하는 경우, foo와 bar 함수 주소가 바로 유효한 목적 주소에 포함되는 것이다. 아래 Fig. 1에서 함수는 총 네 개 이지만, 이 중에서 함수 포인터의 값으로 사용되는 foo와 bar만 유효한 목적 주소에 포함된다.

간접 함수 호출 및 간접 점프 구문의 유효한 목적 주소와 함수 반환 구문의 유효한 목적 주소는 서로 다른 방법으로 찾는다. 아래 Fig. 1에서 함수 포인터 fp에 저장할 수 있는 foo나 bar함수의 주소는 간접 함수 호출 구문의 유효한 목적 주소에 해당한다. 즉 변수 또는 피 연산자, 레지스터가 명령어 주소를 참조하는 경우, 해당 명령어 주소가 간접 호출이나 간접 점프 구문의 유효한 목적 주소가 되는 것이다. 반면 함수 반환 구문의 유효한 목적 주소는 모든 함수 호출 구문의 다음 명령어 주소가 된다.

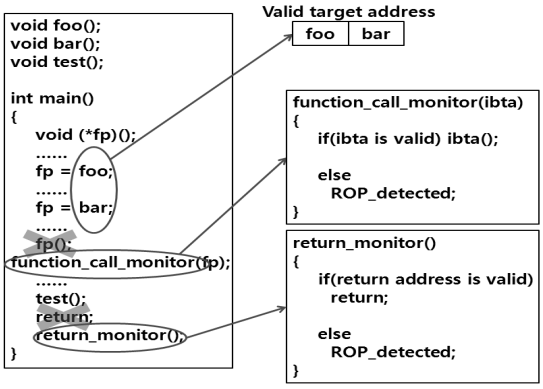

Fig. 1. Operation of IBTAV

IBTAV 기법이 적용되면 모든 간접 함수 호출 구문들은 위 Fig. 1과 같이 검사 함수function_call_monitor 호출 구문으로 대체된다. function_call_monitor 검사 함수는 인자로 전달된 목적 주소(ibta)가 유효한 목적 주소인지 검사한다. 함수 반환 구문 역시 제거 되고 위 Fig. 1과 같이 검사 함수인 return_monitor 함수 실행으로 대체된다. 검사 함수는 전달된 목적 주소로 실행 흐름을 변경하기 위해서 간접 분기문을 사용한다. 가젯이 검사 함수 내부의 간접 분기문을 사용할 수 없도록 모든 검사 함수를 유효한 목적 주소에서 제외한다. 가젯이 명령어 중간, 즉 정렬되지 않은 명령어로 분기해서 새로운 간접 분기문을 찾아서 사용할 수도 있다. 하지만 이러한 명령어 중간 주소는 유효한 목적 주소에 포함되지 않기 때문에, 검사 함수에 의해 해당 주소로 분기할 수 없다.

IBTAV 기법에서는 가젯의 주소가 유효한 목적 주소와 일치하지 못하는 경우 공격을 탐지하지 못하는 거짓 음성을 보인다. 하지만 4절의 방어 능력 평가에서도 알 수 있듯이, 가젯 주소가 유효한 목적 주소와 일치하는 경우는 극히 드

Table 1. Differences between IBTAV and previous defense techniques

Category	IBTAV	ROPDefender	CFI	G-Free
Defense against ROP attacks that use indirect jumps or indirect call instructions	○	X	○	△
Defense against ROP attacks that use branches to unaligned instructions	○	△	○	○
Static control flow analysis	X	○	○	○
Dynamic control flow analysis	X	○	○	X
Use of runtime tools such as virtual machines or binary instrument tools	X	○	○	X

물다. 정렬된 명령어의 경우, 가젯이 사용하는 간접 분기문은 검사 함수 내에서만 발견된다. 하지만 검사 함수는 유효한 목적 주소에서 제외되기 때문에, 가젯이 유효한 목적 주소와 일치하는 경우가 적다. 검사 함수 외의 명령어에서 간접 분기문을 사용하기 위해서는 정렬되지 않은 명령어로 분기해야 한다. 하지만 앞서 설명한 바와 같이 정렬되지 않은 명령어는 유효한 목적 주소에 포함되지 않기 때문에, 사용할 수 없다.

IBTAV는 이처럼 실행 흐름 분석 없이도 모든 간접 분기문의 목적 주소를 검사할 수 있기 때문에, 아래와 같은 세 가지 이점을 가진다.

A) 실행 흐름에 대한 분석이 필요 없다. 간접 분기문의 목적 주소는 대개 실행 시간에 결정되기 때문에, 올바른 간접 분기문의 목적 주소를 알아내기 위해서 동적 실행 흐름 분석이 필요하다. 하지만 IBTAV는 정적 시간에 분석한 유효한 목적 주소만 사용하기 때문에, 동적 실행 흐름 분석이 필요 없다. 따라서 가상 머신이나 동적 프로그램 분석 도구를 사용할 필요가 없으며, 성능 상의 이점을 보일 수 있다.

B) 프로그램에서 발견되는 가젯이 최소화 된다. 기존의 프로그램에 있는 모든 간접 분기문이 제거되고, 검사 함수 내부에서만 사용한다. 따라서 간접 분기문을 사용하는 가젯은 검사 함수 내에서만 발견될 수 있다. 가젯이 검사 함수 내의 간접 분기문을 사용할 수 없도록 검사 함수를 유효한 목적 주소에서 제외한다.

C) 정렬되지 않은 명령어 주소로의 분기를 차단한다. 정렬되지 않은 명령어 주소는 유효한 목적 주소에 포함되지 않으며, 검사 함수에 의해서 항상 탐지된다. 따라서 NOP 등을 사용해서 명령어 주소를 정렬하지 않아도, 정렬되지 않은 명령어 주소로의 분기를 차단할 수 있다 이로 인해 프로그램의 명령어 코드 크기와 성능 상의 이점을 볼 수 있다.

이러한 세 가지 이점을 통해, IBTAV는 기존 ROP 공격 방어 기법과 위 Table 1과 같은 차이점을 보인다.

Table 1의 ROPDefender[11]는 함수 반환 구문에 기반한 방어 기법으로서, 그 외의 간접 분기문을 사용하는 ROP 공격은 방어할 수 없다. 또한 간접 점프 및 간접 함수 호출 구문을 사용하여 정렬되지 않은 명령어로 분기하는 경우 역시 방어할 수 없다. 함수 호출 구문을 이용하여 정렬되지 않은 명령어로 분기하는 것은 방어할 수 있다. ROPDefender는 가상 머신이나 동적 프로그램 분석 도구를 사용하여 동적 실행 흐름 분석을 수행하고, 올바른 함수 반환 주소를 알아낸다. Table 1의 CFI는 CFI 기법에 기반한 방어 기법[6,7]이다. 프로그램의 정적 및 동적 실행 흐름 분석이 필요하며, 이로 인해 가상 머신이나 동적 프로그램 분석 도구를 사용해야 한다. Table 1의 G-Free[16]는 프로그램에서 사용 가능한 가젯을 제거하는 기법이다. 가젯 제거 기법은 프로그램이 정렬되지 않은 명령어로 분기하는 것을 막기 위해, 정적 실행 흐름 분석을 통해 명령어들을 정렬한다. 하지만 프로그램의 모든 가젯을 제거할 수는 없기 때문에, 간접 점프나 간접 함수 호출 구문을 사용하는 ROP 공격의 일부만 방어할 수 있다.

다음 3절에서는 IBTAV 기법의 구현 내용을 살펴본다.

3. 구 현

프로그램에 IBTAV 기법을 적용하기 위해서 프로그램 변환 및 분석 도구를 개발하였다. 프로그램 변환 도구는 프로그램의 모든 간접 분기문을 제거하고, 대신 검사 함수를 실행하도록 프로그램을 수정한다. 프로그램 분석 도구는 함수 반환 구문에 대한 유효한 목적 주소를 추출한다. 아울러 링커를 수정하여, 간접 함수 호출과 간접 점프 구문의 유효한 목적 주소를 추출하였다.

3.1 프로그램 변환

프로그램 변환 도구는 간접 점프/호출 구문, 함수 반환 구문을 각각 function_call_monitor과 return_monitor 검사 함수 실행으로 변경한다. 이 도구는 POSIX 운영체제에서 사용하는 ABI(Application Binary Interface)인 ELF(Executable and Linkable Format)[17]을 사용한다. 프로그램을 변환하는 과정에서 간접 점프/호출 구문은 함수 반환 구문과 다른 방법으로 처리된다. 이는 프로그램이 실행 시간 동안 사용하는 스택 메모리를 그대로 유지하기 위한 목적이다. 예를 들어, 간접 함수 호출 구문의 경우 아래 Table 2와 같이 검사 함수인 function_call_monitor 함수 호출 구문으로 변경된다.

Table 2. Transformation of an indirect function call

Before transformation		After transformation	
jle	42 <main+0x2e>	jle	42 <main+0x2e>
movl	foo,0x8(%esp)	movl	foo,0x8(%esp)
jmp	4a <main+0x36>	jmp	4a <main+0x36>
movl	bar,0x8(%esp)	movl	bar,0x8(%esp)
mov	0x8(%esp),%eax	mov	0x8(%esp),%eax
call	***%eax**	**call**	**function_call_monitor**

함수 반환 구문은 다른 방식으로 처리한다. 함수 반환 구문이 사용되는 시점에는 프로그램의 스택에 함수 반환 값 등이 저장되어 있기 때문에, 반드시 스택의 형태를 보존해야 한다. 하지만 위와 같이 검사 함수를 호출하게 되면 스택의 형태가 변하기 때문에, 아래 Table 3과 같이 함수 호출 구문이 아닌 직접 분기문(jmp)을 사용해서 검사 함수인 return_monitor 함수를 실행한다.

Table 3. Transformation of a function return

Before transformation		After transformation	
sub	$0x16, %esp	sub	$0x16, %esp
return		**jmp**	**return_monitor**

3.2 유효한 목적 주소 추출

간접 점프 구문이나 간접 호출 구문에서 사용될 수 있는 유효한 목적 주소는 링커에 의해서 처리된다. 예를 들어 함수 포인터를 사용해서 foo 함수를 호출한다면, 프로그램에 함수 foo의 주소가 기록되어야 한다. 함수 주소가 기록되는 곳은 명령어 내의 피연산자 혹은 데이터 메모리 영역이다. 함수의 주소가 기록되어야 하는 위치는 재배치 정보를 통해 관리하며, 각 재배치 정보는 기록해야 할 심볼 정보를 참조한다. 즉 foo 함수의 주소가 명령어 피연산자로 사용되는 경우, 피연산자에 대한 재배치 정보가 foo 함수 심볼을 참조하는 것이다. 링커는 이러한 재배치 및 심볼 정보를 통해 간접 점프 혹은 간접 함수 호출 구문의 유효한 목적 주소를 추출한다.

이와는 달리, 함수 반환 구문의 목적 주소는 함수 호출 구문에 의해서 스택에 자동으로 저장된다. 이 경우 스택에 저장되는 함수 반환 구문의 목적 주소는 해당 함수 호출 구문의 바로 다음 명령어 주소가 된다. 예를 들어 “call foo”라는 함수 호출 구문의 주소가 10번지이고 그 다음 명령어 주소가 14번지라면, 스택에 저장되는 함수 반환 구문의 목적 주소는 14번지가 된다. 따라서 프로그램을 분석하여 모든 함수 호출 구문의 다음 명령어 주소를 함수 반환 구문의 유효한 목적 주소로 추출한다.

IBTAV는 이와 같이 프로그램 분석 도구와 수정된 링커를 통해 유효한 목적 주소를 추출한다. IBTAV가 적용된 프로그램은 간접 분기문 대신 적절한 검사 함수를 실행하여 목적 주소가 올바른지 검사한다. 다음 절에서는 IBTAV 기법의 성능과 방어 능력을 평가한다.

4. 성능 및 방어 능력 평가

본 논문에서는 IBTAV 기법을 적용하였을 때 프로그램 성능에 어떤 영향을 미치는 지 알아보기 위해 SPEC CPU 2006 벤치마크[18]를 사용하였다. 벤치마크는 Intel 펜티엄 D CPU 3.0 GHz, 4.0 GByte RAM, 그리고 Ubuntu 12.04 운영체제 상에서 수행되었다. 아래 Fig. 2는 IBTAV를 적용하지 않은 프로그램과 IBTAV를 적용한 프로그램의 실행 시간을 비교한 내용이다.

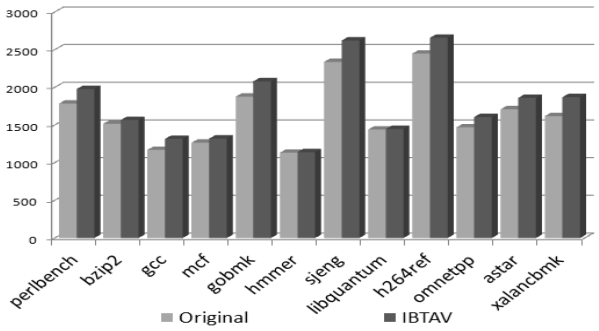

Fig. 2. SPEC CPU 2006 results: run time comparisons of original SPEC benchmarks vs. IBTAV applied SPEC benchmarks

위 Fig. 2에서 x 축은 벤치마크에서 사용한 각각의 프로그램을, 그리고 y 축은 각 프로그램의 실행 시간(단위 : 초)을 나타낸다. IBTAV이 적용된 경우 측정되는 성능 부담은 세 가지 요소에 의해서 결정된다. 첫째는 함수 호출 횟수이다. 모든 함수 호출에 대해서 함수 반환 구문이 발생한다고 가정하는 경우, 함수 호출이 발생할 때 마다 함수 반환 구문을 검사하는 성능 부담을 가진다. 둘째는 간접 함수 호출이나 간접 점프 구문의 호출 회수이다. 이 경우에도 목적 주소가 올바른지 검사하는 검사 함수가 호출되고 이 과정에서 성능 부하를 보인다. 셋째는 유효한 목적 주소 목록의 크기이다. 목록의 크기가 클수록 검사 시간이 증가한다.

IBTAV이 적용된 프로그램은 최대 약 15% 이내, 평균 약 8%의 성능 부담을 보였다. 가장 적은 성능 부담을 보인 hmmer 프로그램의 경우 전체 함수 호출 회수가 8300만 번이었으며 간접 함수 호출이 만 번 미만이었다. 반면 가장 높은 성능 부담을 보인 xalancbmk 프로그램의 경우 전체 함수 호출 횟수가 약 300억회에 달했으며, 또한 간접 함수 호출 구문도 약 1.5억번 호출되었다. 성능 부담에서 가장 큰 비중을 차지하는 것은 함수 반환 구문의 목적 주소에 대한 검사 함수 사용이다. 함수 호출 횟수만큼 함수 반환 구문에 대한 검사 함수가 호출된다고 볼 수 있기 때문이다. 또한 함수 반환 구문의 유효한 목적 주소는 간접 함수 또는 간접 점프 구문의 유효한 목적 주소에 비해 그 목록의 크기가 크다. 그럼에도 불구하고 최대 15% 이내의 성능 부하를 보였기 때문에, 실제 컴퓨팅 환경에서 IBTAV를 적용하고 ROP 공격을 방어하는 데 있어서 납득할 만한 성능 부하를 보인다고 판단된다.

또한 IBTAV 기법의 방어 능력을 평가하기 위해, ROPgadget 도구[19]를 사용해서 벤치마크에 사용된 프로그램이 각각 원본 상태일 때와 IBTAV 기법이 적용되었을 때 몇 개의 가젯이 검색되는 지를 분석하였다. 또한 2절에서 설명한 바와 같이 가젯의 주소가 유효한 목적 주소와 일치하는 경우 공격을 탐지하지 못하기 때문에, 검색된 가젯 주소가 유효한 목적 주소와 일치하는 지 여부를 분석하였다. 아래 Table 4은 원본 프로그램과 IBTAV가 적용된 프로그램에서 발견된 가젯의 수, 그리고 유효한 목적 주소와 일치하는 가젯이 몇 개가 있는지 보여주고 있다.

Table 4. Number of gadgets

Program	Number of gadgets in original program	Number of gadgets in IBTAV program	Number of gadgets that have valid addresses
perlbench	230	175	0
bzip2	42	37	0
gcc	503	390	0
mcf	19	18	0
gobmk	228	199	0
hmmer	116	108	0
sjeng	52	49	0
libquantum	57	48	0
h264ref	118	110	0
omnetpp	286	232	0
astar	50	40	0
xalancbmk	744	528	0

IBTAV가 적용된 경우에도 가젯의 수가 0이 되지 않는 것은, 사용한 ROPgadget 도구가 명령어 중간에서도 발견될 수 있는 모든 가젯을 사용하기 때문이다. 예를 들어 코드 영역에서 “C3”이라는 값이 발견되는 경우, 이 값이 실제로는 피 연산자 값이라도 인텔에서 사용되는 함수 반환 구문 코드와 동일하기 때문에 가젯의 일부로 사용될 수 있다. 하지만 그 외에 프로그램의 모든 간접 분기문을 제거하였기 때문에 IBTAV에서는 관찰되는 가젯의 수가 줄어들었음을 알 수 있다. 정렬되지 않은 명령어를 사용하는 가젯은 IBTAV 기법이 적용된 경우 사용할 수 없기 때문에 고려 대상이 되지 않는다. 아울러 유효한 목적 주소와 일치하는 가젯 수는 모두 0으로 측정되었다. 즉 벤치마크 프로그램에 IBTAV 기법이 적용되는 경우 ROP 공격이 불가능하다는 점을 알 수 있다.

IBTAV가 기존의 ROP 공격 방어 기법에 비해 어느 정도의 성능 향상을 보이는 지 알아보기 위해, IBTAV와 ROPdefender[11]의 성능을 비교하였다. ROPdefender는 실행 시간 동안 사용하는 함수 반환 구문의 목적 주소를 검사하며, 올바른 함수 반환 주소를 알아내기 위해 정적 및 동적 실행 흐름 분석을 일부 수행한다. 따라서 동적 프로그램 분석 및 변환 도구인 PIN[9]를 사용한다. ROPDefender는 IBTAV와 마찬가지로 표준 벤치마크 프로그램인 SPEC CPU 2006을 사용하여 성능을 평가하였기 때문에, ROPDefender 논문에 개재된 실험 데이터를 직접 사용해서 성능을 비교하였다(Fig. 3).

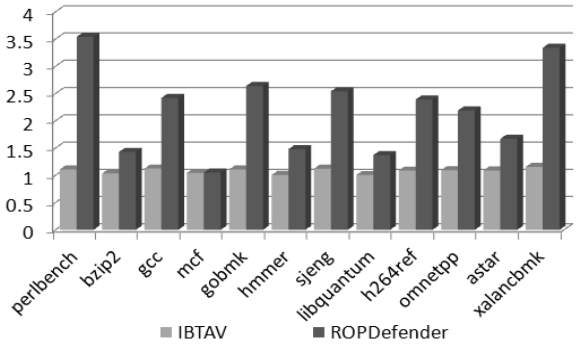

Fig. 3. Performance overhead comparisons of IBTAV and ROPDefender

위 Fig. 3에서 x 축은 벤치마크에서 사용하는 프로그램을, 그리고 y축은 원래 프로그램이 실행되는 시간과 방어 기법이 적용된 프로그램이 실행되는 시간간의 비율을 나타낸다. ROPdefender는 평균 117%의 성능 부하를 보인 반면, IBTAV는 평균 8%의 성능 부하를 보였다. 이는 ROP defender가 동적 프로그램 분석 및 변환 도구인 PIN을 사용하였기 때문이며, IBTAV는 이러한 동적 프로그램 분석 도구나 가상 머신을 사용하지 않기 때문에 성능상 큰 이점을 보인다.

5. 결 론

본 논문에서는 ROP 공격을 탐지하고 방어하기 위한 간접 분기문 목적 주소 검증(Indirect Branch Target Address Verification) 기법을 제시하였다. ROP 공격 기법은 프로그램 코드 영역에서 사용 가능한 가젯을 찾고, 이들을 조합하여 공격자가 원하는 기능을 수행하는 공격 기법이다. 기존 return-to-libc와 같은 코드 재사용 공격 기법에서 한 단계 발전한 형태라 할 수 있다. IBTAV는 모든 간접 분기문의 목적 주소가 유효한지 검사하여 가젯이 호출되는 것을 막는다. 동적 실행 흐름 분석 없이도 간접 함수 호출이나 간접 점프 구문을 사용하는 ROP 공격을 방어할 수 있기 때문에, 가상 머신이나 동적 코드 분석 도구를 사용하지 않아 성능 부담이 적다. IBTAV 기법이 적용되면 프로그램의 모든 간접 분기문이 제거되고 검사 함수 내부에서만 간접 분기문을 사용하기 때문에 가젯이 최소화 된다. 아울러 유효한 목적 주소로의 분기만 허용하기 때문에, 유효한 목적 주소에 포함될 수 없는 정렬되지 않은 명령어로의 분기가 차단된다.

참 고 문 헌

[1] Roemer, R.; Buchanan, E.; Shacham, H. and Savage, S. “Return-oriented programming: Systems, languages, and applications”, ACM Transactions on Information and System Security (TISSEC), ACM, 2012.

[2] c0ntex, “Bypassing non-executable-stack during exploitation using return-to-libc”, “http://www.infosecwriters.com/text_resources/pdf/return-to-libc.pdf”

[3] Theo de Raadt, “i386 W^X”, “http://marc.info/ ?l=openbsd-misc&m=105056000801065”

[4] Aleph One, “Smashing The Stack For Fun And Profit”, Phrack, 7(49), 1996.

[5] Abadi, M.; Budiu, M.; Erlingsson, &U. and Ligatti, J., “Control-flow integrity principles, implementations, and applications”, ACM Trans. Inf. Syst. Secur., ACM, 2009.

[6] Bletsch, T.; Jiang, X. and Freeh, V., “Mitigating code-reuse attacks with control-flow locking.”, Proceedings of the 27th Annual Computer Security Applications Conference, pp.353-362, 2011.

[7] Zeng, B.; Tan, G. and Morrisett, G., “Combining control-flow integrity and static analysis for efficient and validated data sandboxing”, Proceedings of the 18th ACM conference on Computer and communications security, pp.29-40, 2011.

[8] Visual Studio .Net 2003, “Introduction to Instrumentation and Tracing”, http://msdn.microsoft.com/en-us/library/aa983649(VS.71).aspx

[9] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood.., “Pin: Building customized program analysis tools with dynamic instrumentation.”, Proceedings of PLDI 2005, pp.190 - 200, ACM, 2005.

[10] Francillon, A., Perito D. and Castelluccia, C.,“Defending embedded systems against control flow attacks.”, Proceedings of the First ACM Workshop on Secure Execution of Untrusted Code, SecuCode 2009, pp.19 - 26, ACM, 2009.

[11] Davi, L.; Sadeghi, A. and Winandy, M., “ROPdefender: A detection tool to defend against return-oriented programming attacks.”, Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, pp.40-51, ACM, 2011.

[12] Chen, P., Xiao, H., Shen, X., Yin, X., Mao, B. and Xie, L., “DROP: Detecting return-oriented programming malicious code”, Information Systems Security, pp.163-177, Springer, 2009.

[13] Checkoway, S.; Davi, L.; Dmitrienko, A.; Sadeghi, A.; Shacham, H. and Winandy, M., “Return-oriented programming without returns”, Proceedings of the 17th ACM conference on Computer and communications security, pp.559-572, ACM, 2010.

[14] Bletsch, T., Jiang, X., Freeh, V. and Liang, Z., “Jump-oriented programming: A new class of code-reuse attack”, Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security, pp.30-40, ACM, 2011.

[15] Pappas, V., Polychronakis, M. and Keromytis, A., “Smashing the gadgets: Hindering return-oriented programming using in-place code randomization.”, Security and Privacy (SP), 2012 IEEE Symposium on, pp.601-615, IEEE, 2012.

[16] Onarlioglu, K., Bilge, L., Lanzi, A., Balzarotti, D. and Kirda, E., “G-Free: defeating return-oriented programming through gadget-less binaries.”, Proceedings of the 26th Annual Computer Security Applications Conference, pp.49-68, USENIX, 2010.

[17] Tools Interface Standards - TIS: Executable and Linkable Format(ELF), version 1.2. Portable formats specications (1995)

[18] Standard performance Evaluation Corporation. http://www.spec.org/benchmarks.html

[19] Jonathan Salwan, “ROPgadget Tool”, http://shell-storm.org/project/ROPgadget

박 수 현

e-mail : ardeness@gmail.com

2007년 홍익대학교 컴퓨터공학부(학사)

2009년 홍익대학교 컴퓨터공학과(석사)

2010년~현 재 홍익대학교 컴퓨터공학과 박사과정

관심분야 : 시스템소프트웨어,시스템보안

김 선 일

e-mail : sikim@cs.hongik.ac.kr

1985년 서울대학교 컴퓨터공학과(학사)

1987년 서울대학교 컴퓨터공학과(석사)

1995년 University of Illinois at Urbana-Champagin(전산학박사)

1995년~1999년 미국 IBM 연구원

1999년~현 재 홍익대학교 컴퓨터공학과 교수

관심분야 : 시스템 소프트웨어, 임베디드 시스템, 시스템 보안