버퍼 오버플로우 공격 방어를 위한 스택 역위 탐지의 성능 평가

양한근[0*] 표창우[1] 이경호[2]
홍익대학교 컴퓨터공학과[01], 일리노이 대학(시카고) 전기 및 컴퓨터공학과[2]
(hkyang[0], pyo[1])@cs.hongik.ac.kr, ghlee[2]@ece.uic.edu

Performance Evaluation of Detecting Stack Inversion for Defending from Buffer Overflow Attack

Hankeun Yang[0] Changwoo Pyo[1] Gyungho Lee[2]
Dept of Computer Engineering, Hongik University[01],
Dept of Electrical and Computer Engineering, University of Illinois at Chicago[2]

요 약

실행 시간 스택 프레임의 하단과 상단을 가리키는 프레임 포인터와 스택 포인터는 항상 일정한 대소 관계를 유지한다. 선형 스택 공격이 진행되면, 이관계가 반전된다. 이때 스택이 역위되었다고 한다. 본 논문은 x86 프로세서 계열의 gcc 컴파일러에 스택 역위 탐지기능을 부여하여, 이 컴파일러를 사용하였을 때 실행 프로그램의 성능에 미치는 영향을 분석하였다.

1. 서 론

버퍼 오버플로우 공격은 버퍼에 크기를 넘어서는 데이터를 기록함으로써 메모리 내의 코드 포인터를 주입된 공격 코드를 가리키게 손상시킨다. 손상된 코드 포인터의 값이 프로그램 카운터에 적재되면 공격 코드가 실행된다[1].

최초의 버퍼 오버플로우 공격은 1988년 모리스 웜 사건으로써 finger 데몬 프로그램을 공격, 네트워크로 연결되어 있는 대부분의 유닉스 시스템을 마비시켰다.[10] 또한 2001년 7월에 발생한 코드레드 바이러스의 경우도 버퍼 오버플로우 공격에 의한 것인데 발생 6시간만에 25만대의 컴퓨터를 감염시켰다.[11] 버퍼 오버플로우와 관련된 공격 시도는 1997년부터 급격히 증가해 1998년에서는 취약점을 이용한 해킹 사고의 50% 이상을 차지하였다.[12]

버퍼 오버플로우 공격은 배열의 한계를 검사하지 않기 때문에 발생하는데, C 프로그램의 대부분이 이에 해당된다. 버퍼 오버플로우 실행시간 스택에 위치한 버퍼가 오버플로우 되면서 반환 주소를 덮어쓰게 되고 이 때 반환 주소를 스택 세그먼트에 적재된 임의의 코드의 시작 주소로 설정하면 공격자가 원하는 프로그램이 수행된다[2, 3, 4]. 이 경우 root 권한으로 수행되는 프로그램이 오버플로우를 일으켜 shell을 수행하였다면 공격자는 시스템의 관리자 권한을 획득한다.([그림 1] 참조)

본 눈문은 x86 프로세서 계열의 gcc 컴파일러에 스택 역위 탐지 기능을 부여하여, 이 컴파일러를 사용하였을 때 실행 프로그램의 성능에 미치는 영향을 분석하였다.

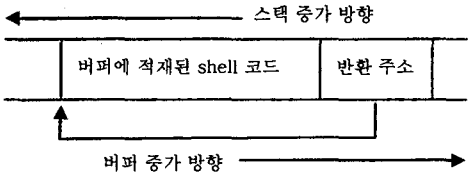

그림 1. 버퍼 오버플로우 공격 원리

2. 관련 연구

스택가드[3, 4]는 Immunix에서 gcc를 기반으로 구현되었다. 함수가 호출되었을 때 스택에 반환주소가 삽입되고 프레임포인터 값이 삽입되기 전 실행시간에 결정되는 임의의 정수 값(canary)을 프레임 포인터와 반환주소 사이에 삽입한다. 선형 스택 공격 시 버퍼의 시작지점부터 반환주소까지 순차적으로 데이터를 기록하게 되고 반환주소 사이에 위치한 프레임 포인터, canary 값을 함께 덮어쓰게 된다. 함수 종료 시 canary 값을 검사하여 버퍼 오버플로우 공격이 있었음을 판단하고 프로그램을 종료시킨다.

canary 값을 삽입하는 random canary버전 외에 버퍼에 문자열을 복사할 때 NULL, EOF 값 등을 삽입한 terminator canary[4]가 구현되어 있다. 이는 strcpy등 c언어의 문자열을 다루는 함수들이 NULL, EOF 값을 만나면 복사를 더 이상 수행하지 않는 점을 이용한 것인데 현재는 random canary 버전이 배포되어 있다.

3. 스택 역위 탐지의 구현

스택 역위 탐지[9] 역시 컴파일러를 통해 생성된 어셈블리 코드가 버퍼 오버플로우 공격을 방어한다. gcc 2.95.3 버전을 기반으로 구현되었으며 Intel 계열의 리눅스 운영체제에서 작동한다. 공격에 대한 방어는 함수 종료시 프레임 포인터와 스택 포인터와의 위치 비교를 통해 공격 여부를 판단한다.

실행 시간 스택 프레임의 하단과 상단을 가리키는 프레임 포인터와 스택 포인터는 항상 일정한 대소 관계를 유지한다. 함수 호출이 발생하면 반환 주소가 스택에 push 된다. 이어서 피호출 함수는 호출 함수의 프레임 포인터 값을 push하고 스택 포인터 레지스터 값을 프레임 포인터 레지스터에 복사한다. 피호출 함수의 버퍼를 포함한 지역 변수들이 스택에 할당되면 스택 프레임의 모습은 그림 2의 형태를 가진다.
함수 종료 시 프레임 포인터 값이 스택 포인터 값으로 복사되고 호출 함수의 프레임 포인터 값이 pop된 후 반환 주소로 복귀한다.

선형 스택 공격이 진행되면 이 관계가 반전되는데 이 때 스택이 역위되었다고 한다. 정상적으로 함수가 종료한다면 pop된 프레임 포인터는 호출 함수의 프레임 포인터가 되어 스택 포인터보다 하위 주소를 가리켜야 하지만 선형 스택 공격은 프레임 포인터를 포함하여 반환주소 값을 공격 받은 버퍼의 시작 주소 값으로 덮어쓴다. 이 경우 함수 종료 시 스택 상의 호출 함수의 프레임 포인터 값은 호출 함수의 프레임 포인터 값이 피호출 함수 스택 프레임 상에 존재하는 공격받은 버퍼의 주소 값으로 대체 되어 있기 때문에 pop이 이루어지면 스택 포인터는 반환 주소를 가리키고 프레임 포인터 레지스터 는 피호출 함수의 버퍼를 가리키는 스택 역위가 발생한다.(그림 3 참조)

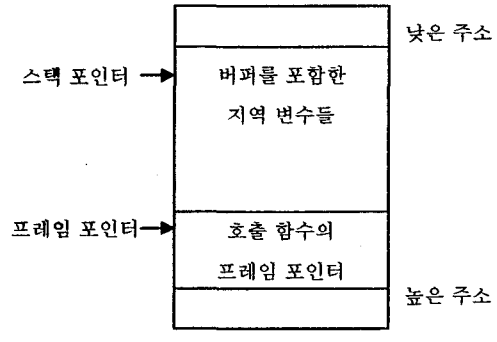

그림 2. 정상적인 스택 프레임의 모습

스택 역위 현상이 발생하여 버퍼 오버플로우 공격으로 판단되면 컴파일러에 내장되어 구현된 핸들러가 실행되어 메시지를 출력하고 프로그램을 정상 종료시킨다.

또한 스택 프레임의 모습을 변화시키지 않으며 스택가드에 비해 컴파일 시 생성되는 명령어가 적다.

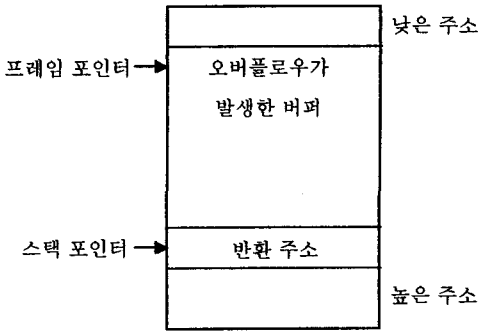

그림 3. 버퍼 오버플로우 발생시 스택 프레임 모습

4. 성능 평가

스택 가드와 스택 역위 탐지 기법은 모두 GNU의 gcc를 기반으로 구현되었다는 점에서 공통점을 가진다. 각각의 컴파일러가 생성하는 기계어에서 차이점을 가져오게 되고 이는 직접적으로 프로그램의 성능에 영향을 미친다. 비교를 위해 스택 가드 발표 당시 측정되었던 마이크로 벤치마크[1]를 비롯하여 추가적으로 SPEC95[5]의 compress와 피보나치 수열, 하노이 탑, 퀵 소트 알고리즘에 대하여 성능을 비교하였다.

스택가드의 마이크로 벤치마크는 4가지에 대한 함수 호출을 구현하여 5억 회 호출을 통한 성능을 평가한다.([표 1] 참조) 성능 평가는 Intel Pentium 2 프로세서에 128MB 램을 장착한 리눅스 PC에서 수행되었으며 실행 시간을 기준으로 측정되었다.

" i++" 는 base case로써 지역 변수를 loop 구문을 통해 증가시킨다. " void inc()" 는 전역 변수 i를 inc() 함수를 호출하여 증가시키며 " void inc(int *)" 는 변수를 포인터 변수로 전달하여 증가시킨다.
" int inc(int)" 는 변수를 전달, 증가된 인자 값을 반환하여 반복 호출한다.

표 1. 마이크로 벤치마크 결과

단위 : 초

측정 치	스택가드	스택역위탐지	Speed up
i++	7.00	7.65	0.92
void inc()	20.60	17.26	1.19
void inc(int *)	27.07	24.59	1.10
int inc(int)	26.13	25.59	1.02

[표 2]는 피보나치 값을 40, 하노이 탑의 원판을 30, 퀵 소트는 정수 값을 1,000,000로 설정, SPEC95의 compress까지 총 4가지 항목을 비교한 자료이다. 마이크로 벤치 마크와는 달리 함수 호출 회수가 많으면서도 다양한 알고리즘을 적용하였고 벤치마크의 표준인 SPEC95 벤치마크를 함께 사용하여 성능을 측정하였다.

표 2. 피보나치, 하노이 탑, 퀵 소트, SPEC95 벤치 마크 결과

단위 : 초

	스택가드	스택역위탐지	Speed up
피보나치	15.41	15.00	1.03
하노이 탑	81.70	78.65	1.04
퀵 소트	1.70	1.69	1.01
SPEC95	8.88	8.53	1.04

[표 1]과 [표 2]에서와 같이 base case를 제외하면 모두 스택 역위 탐지 기법이 우수한 것으로 나타났다. 이는 스택가드가 생성하는 명령어가 비교적으로 많다는 것에 기인한다.

스택 역위 탐지 기법은 스택가드와 비교하여 최소 1%, 최대 16%의 성능 향상을 가져옴을 확인할 수 있다. 두 방법 모두 함수의 호출마다 공격 방어를 위한 코드를 생성하므로 함수의 호출 회수가 많을수록 성능 차이는 커진다.

5. 결론 및 향후 과제

역위 탐지 기법은 스택 프레임의 모양을 유지하며 스택가드에 비해 비교적 작은 크기의 코드를 생성, 공격을 판단 함으로써 버퍼 오버플로우 공격에 대해 방어뿐만 아니라 프로그램의 성능 면에서도 이점을 얻을 수 있다.

오픈 소스 성격의 프로그램이 개발되면서 일반인도 프로그램 소스의 분석이 가능하게 되었다. 또한 보안에 대한 주제를 연구하는 기관[6, 7]이 늘어나면서 해킹 기술의 빠른 보급이 이루어졌다. 버퍼 오버플로우 공격은 프로그래밍 습관에서 취약점이 발생하는데 개발 단계에서 근본적으로 문제점을 차단하는 것이 개발 후의 보안에 투입되는 시간과 비용을 절약할 수 있다.

최근에는 버퍼 오버플로우 공격에 대한 방어를 무력화 시키기 위해 프레임 포인터나 함수 포인터를 오버플로우 시키는 기법들이 소개되고 있다[8]. 함수의 버퍼 외에 버퍼 오버플로우 공격이 발생할 수 있는 프로그래밍 언어의 요소들에 대해서도 방어를 수행할 수 있도록 스택 역위 탐지를 발전시키는 방향으로 연구가 이루어져야 한다.

6. 참고 문헌

[1] Aleph One, "*Smashing stack for fun and profit*", Phrack 49-14, 1996

[2] Mudge, "*How to write Buffer Overflows*", http://www.insecure.org/stf/mudge_buffer_overflow_tutorial.html, 1995

[3] Crispin Cowan and Calton Pu and David Maier etal., " *StackGuard: Automatic Detection and Prevention of Buffer-Overflow Attacks*", the 7th USENIX Security Symposium, 1998

[4] Crispin Cowan and Perry Wagle and Calton Pu and Steve Beattie and Jonathan Walpole, "*Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade*", the {DARPA} Information Survivability Conference and Expo DISCEX, 2000

[5] SPEC, http://www.spec.org

[6] SecurityFocus, http://www.securityfocus.com

[7] Phrack, http://www.phrack.org

[8] Bulba and Kil3r, "*Bypassing Stackguard and Stackshield*", Phrack 56-5, 2000

[9] Changwoo Pyo, Gyungho Lee, "*Encoding Function Pointers and Memory Arrangement Checking against Buffer Overflow Attack*", 4th International Conference on Information and Communications Security

[10] M. W. Eichin and J. A. Rochlis. With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988. In Proc. *IEEE Symposium on Security and Privacy*, pages 326-343, 1989

[11] Roman Danyliw and Allen Householder. CERT Advisory CA-2001-23: *Continued Threat of the CodeRed Worm*. http://www.cert.org/advisories/CA-2001-23/html, Jul 2001.

[12] David Wagner, Jeffrey S. Foster, Eric A. Brewer, and Alexander Aiken. *A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities*. In Network and Distributed System Security Symposium, Feb 2000