스택가드를 개선한 버퍼 오버플로우 공격으로부터의 대응책

정길균, 권영미
목원대학교 컴퓨터공학과

Improved Protecting Method based on Stackguard against Buffer Overflow Attack

Kil-Gyoon Jeong, Youngmi Kwon
Dept. of Computer Engineering, Mokwon University
E-mail : secuboy@mokwon.ac.kr, ymkwon@mokwon.ac.kr

요 약

버퍼 오버플로우는 최근에 나온 개념은 아니다. 1996년 버퍼 오버플로우가 Phrack 문서를 통해 세상에 알려지자 유닉스 계열의 운영체제들은 보안 문제로 많은 어려움을 겪게 되었다. 이 공격은 내부에서 쉘을 실행해 원격지에서 루트권한을 획득할 수 있기 때문에 보안상 위협 중에 가장 심각한 부류에 속한다. 아직까지도 공격 방법 중에서는 가장 많은 부분을 차지하고 있다. 본 논문에서는 버퍼 오버플로우의 원리 및 관련 연구에 관해서 살펴보고, 기존 대응책인 stackguard를 기본으로 하여 좀 더 개선시킨 대응책을 제안하고자 한다.

1. 서론

연구와 군사 목적으로 발전한 인터넷은 현재 많은 기업들과 일반인들이 사용하게 되었고 현대 사회에서 중요한 기반으로 자리잡게 되었다. 더불어 보안문제가 인터넷 사회에 끼치는 영향이 커짐에 따라 보다 많은 침입자들이 현재의 보안모델의 취약점을 공격하려 하고 있으며, 공격용 프로그램 또한 보다 복잡하고 정교해 지고 있다.

Buffer overflow 공격은 컴퓨터가 프로그램을 실행시키는 과정과 그 때 일어나는 메모리 관련 일 등에 대해서 자세하게 알고 있어야 그 동작 방식을 이해할 수 있는 아주 어려운 기술 중의 하나이다. 한번에 호스트를 장악할 수 있으므로, 많은 해커들이 즐겨 이용하는 방법이다.

Phrack 문서를 통해 세상에 알려진 buffer overflow는 최근 들어서도 해킹 유형 중에 대표적으로 알려져 있다. 한 호스트의 루트권한을 획득할 수 있으므로 보안상에 가장 위험한 부류로 속한다. 2001년 1월부터 9월까지 CERT 침해사고현황을 보면 <표 1>과 같이 buffer overflow가 많은 부분을 차지하고 있다[1].

<표 1> 2001년 1월부터 9월까지 국내 해킹통계

구분	건수	비고
사용자 도용	216	개인사용자계정 도용 등
s/w보안 오류	143	IIS Unicode 관련 오류
Buffer overflow 취약점	275	named/bind, ftpd, rpc.statd 취약점이용한 공격
서비스거부공격	49	smurf, TFN2K 공격 등
E-mail관련공격	58	스팸메일 관련 공격

Buffer overflow에 대한 완벽한 방어책이 나오지 않았기 때문에, buffer overflow 공격은 끊임없이 시도되고 있다.

본 논문에서는 buffer overflow의 원리와 대응책들에 관해서 살펴보고 이를 바탕으로 개선된 대응책을 제안하고자 한다.

2. Buffer overflow

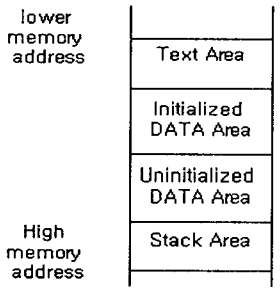
Buffer overflow 공격이란 말 그대로 지정된 버퍼의 크기보다 더 많은 데이터를 입력하여 오버플로우 에러가 일어나게 한 후, 프로그램이 비정상적으로 동작하도록 만드는 것을 의미한다. C 언어에서는 데이터가 지정된 버퍼의 크기보다 더 많이 입력되었는지를 체크하지 않고 사용하는 것이 일반적인데, 이것은 이를 하나하나 체크하다 보면 프로그램의 수행 성능이 크게 저하되기 때문이다[2].

특별한 관심을 끌지 못하던 buffer overflow는 1988년 fingerd의 취약점을 이용한 Morris Worm사건으로 인해 일반인에게도 알려졌다. 그 뒤 1996년 Phrack잡지 49호에 Aleph One이 "Smashing The Stack For Fun And Profit"을 발표하였는데[3], 이 문서는 그 동안 어렵게만 느껴지던, buffer overflow 공격을 기본적인 메모리 구조부터 공격방법까지 상세히 설명하고 있다. 그 이후로 buffer overflow는 가장 강력한 해킹 방법이 되었으며, 현재까지도 가장 많은 부분이 buffer overflow에 의해 일어나고 있다.

여기서는 buffer overflow를 이해하기 위한 메모리구조에 대해서 설명하고, 3장에서 버퍼오버플로우 공격에 대해 설명하고자 한다.

2.1 프로세스 메모리(process memory) 구조

프로세스 메모리구조는 아래의 <그림 1>과 같다.



<그림 1> 프로세스 메모리 구조

Text area는 프로그램의 기본이라고 할 수 있는 명령어와 데이터를 가지고 있다. 이는 읽기만 가능한 메모리 영역이다. DATA area는 두 부분으로 나뉘어진다. Initialized DATA area는 프로세스 정적변수가 들어가는 부분이다. Uninitialized DATA area는 동적으로 메모리가 할당되는 부분으로 heap이라고 불린다.

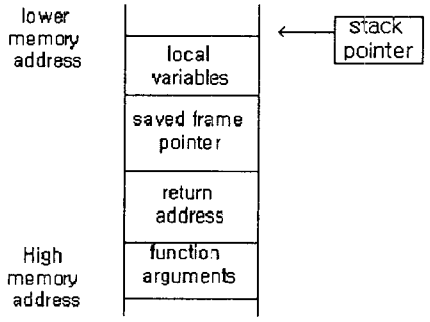
Stack area는 C 언어에서 한 함수를 호출할 때, 그 함수 내부에서 사용되는 지역변수나 함수의 반환주소 등을 저장하게 된다. 어떤 C로 만들어진 프로그램이 있다고 하자. 그 프로그램이 수행되는 과정에서 함수가 호출되었을 경우 그 호출된 함수의 인자등을 저장하기 위한 공간이 stack이다. 함수가 종료 되던, 수행중인 함수를 끝내고 돌아갈 return address를 저장된 곳에서 가져와야 하는데, 이 때 return address가 이미 해킹 가능한 코드부분을 수행하도록 변조되어 있는 경우 시스템이 공격당하게 된다. 따라서 이 곳이 buffer overflow의 공격 대상이다[3].

2.2 Stack 구조

본 절에서는 프로세스 메모리 구조의 stack 부분을 좀 더 상세히 설명한다. Stack 영역은 메모리의 연속된 블록 영역으로서 데이터를 포함하며 <그림 2>와 같은 구조를 갖는다.

Stack에서 할 수 있는 동작은 PUSH 또는 POP으로서, PUSH는 stack의 가장 위쪽에 데이터를 집어 넣는 일이고, POP은 stack의 가장 위쪽에서 데이터를 빼내는 일이다.

운영체제의 메모리 구조에서 stack은 뒤집힌 구조를 하고 있다. 즉, stack의 꼭대기는 stack에 데이터가 채워질 때마다 낮은 메모리 주소를 가리키게 된다.



<그림 2> stack 구조

Stack Pointer라고 불리는 레지스터는 stack에 제일 마지막으로 들어간 주소를 가리킨다.

Stack Pointer는 함수가 호출되고 종료되면서 변하

기 때문에, 한 함수 내에서 유일한 위치를 가리키는 Frame Pointer(FP)가 필요하게 되었다. Frame Pointer는 변수에 접근을 용이하게 하기 위해서 사용하는 포인터이다. 많은 컴파일러들은 local 변수와 파라미터와의 거리가 변하지 않는 FP라는 레지스터를 사용하고, 인텔 CPU에서는 BP(EBP)가 이 레지스터로 이용된다[2-3].

어떤 함수가 실행될 때, 그 함수에서 stack이 반드시 해야 하는 일은 다음과 같다.

(1) 이전의 Frame Pointer를 stack에 저장해 놓는다.
(2) SP를 FP에 복사한다.
(3) SP를 증가시켜서, 로컬변수를 위한 공간을 할당한다.

Return address는 어떤 함수가 실행된 뒤 실행될 코드의 주소가 있는 곳이다. 그런데 만약 이 함수 내에서 메모리를 잘못 조작하여 return address영역에 임의의 값을 넣는다면, 코드는 원래 실행될 곳이 아닌 엉뚱한 곳의 코드를 실행하게 된다. 이 곳에 공격(shell을 실행시키는 등)을 할 수 있게 코드를 채워 넣으면 버퍼 오버플로우가 되는 것이다.

3. Buffer overflow 분류

3.1 Stack 기반 오버플로우

Stack 기반 buffer overflow 공격이란 stack area에 정해진 버퍼보다 큰 코드 등을 삽입하여 루트쉘을 실행시키기 위해 시도되는 공격을 의미한다.

주 공격 대상이 되는 프로세스는 SETUID로 실행되는 rlogin등의 프로그램과 strcpy(), sprintf(), gets()등 실행시 경계값을 체크하지 않는 함수를 포함하는 프로그램 등이다.

스택기반 buffer overflow 공격에서 SETUID를 공격 대상으로 할 때의 과정은 다음과 같다. 첫 째, 루트소유의 SETUID 프로그램을 실행시킨다. 둘 째, 쉘코드를 담고 있는 버퍼를 SETUID 프로그램에 적용시켜 버퍼를 오버플로우 시킨다. 마지막으로 버퍼 오버플로우된 프로그램이 실행한 루트쉘을 얻는다.

3.2 힙 기반 오버플로우

Heap이란 응용 application에 의해 동적으로 할당되는 메모리 영역이다. Heap 영역의 함수 포인터 등이 원하는 코드(쉘코드)를 실행하도록 오버플로우 시키는 것이 heap 기반 오버플로우이다. 주로 heap의 하위 버퍼를 오버플로우시켜 상위 heap의 동적 변수에 원하는 데이터를 덮어쓰는 기법을 사용한다[4].

3.3 대응책

다음은 일반적으로 적용될 수 있는 buffer overflow 공격에 대한 대응책들이다.

(1) 경계검사를 하지 않는 함수들 사용을 자제하고, 다른 권장 함수들을 이용한다. 예를 들어서 strcpy() 경우는 strncpy()을 이용한다[5]. strcpy() 함수는 버퍼의 크기를 검사하지 않아 문제가 발생하므로 복사할 데이터의 크기를 미리 검사하는 strncpy() 함수를 대신 사용할 수 있다. strncpy() 함수는 NULL 문자로 끝내야 하는데 소스의 버퍼 크기가 복사할 버퍼보다 크거나 같으면 NULL로 끝나지 않을 수 있기 때문이다. strcpy()와 strncpy()의 사용 예를 <표 2>에 보였다.

<표 2> strcpy() 와 strncpy()의 사용예 비교

strcpy()	strncpy()

void func(char *str)
{
char buffer[256];
strcpy(buffer, str);
return;
}
	void func(char *str)
	{
	char buffer[256];
	strncpy(buffer,str, sizeof(buffer)-1);
	buffer[sizeof(buffer)-1] = 0;
	return;
	}

(2) setuid 루트프로그램이나 서버프로그램이 주로 루트 권한으로 실행이 되므로 루트권한인 파일들을 탐지 대상으로 설정하여 사용을 최소화한다.

(3) 많은 OS회사들이 buffer overflow를 막기 위해 stack area에서의 실행을 불가능하게 만드는 커널 패치를 제공한다. 이것을 사용해 buffer overflow 공격을 어느 정도 막도록 한다[2].

(4) 최신 보안 패치가 나오면 즉각 패치한다.

(5) 경계검사를 하는 Stackguard로 컴파일한다.

Stackguard의 기본 개념은 해당프로그램 구동시 stack상에 일정주소번지에 프로그래머가 유도하는 canary를 심어두고 stack이 붕괴되거나 변조되었을 경우 유도된 canary를 체크하여 프로그램을 강제 종료시키는 방법이다[6]. 이는 함수가 호출되면 return address뒤에 canary value를 set해 주는데 함수가 return될 때 canary value가 바뀌어 있으면 공격 프로그램이 실행된 것으로 생각하고 syslog에 기록을 남기고 프로그램을 종료시킨다. Stackguard는 컴파일할 때 적용되기 때문에 문제가 발생할 수 있다. 시스템의 모든 프로그램이 모두 stackguard를 통해 컴파일 되어야 한다는 것인데, 이로 인해 stackguard가 잘못 설치되었을 때에는 시스템이 부팅되지 않는 경우도 생긴다.

4. 제안모델

4.1 구조

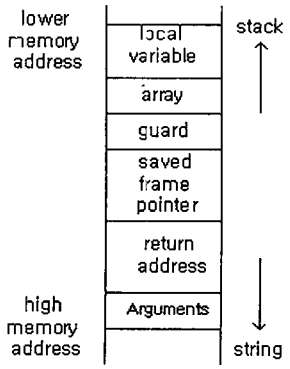
Stack 구조에서 알 수 있듯이 네 개의 지역들은 오버플로우 공격으로부터 보호되어야 한다.

Stackguard는 return address가 정해진 후에 바로 guard(시스템이 재시작 할 때마다 변경되는 값)를 설정하도록 설계되어 있다.

본 논문에서는 guard를 값이 아닌 지역으로 설정하여 침입 흔적을 미리 알아낼 수 있게 하는 방법을 제안한다. <그림 3>에 보인바와 같이 guard지역은 sfp(saved frame pointer)다음에 설정된다. Guard가 설정된 곳은 stack을 파괴하기 위한 공격이 시작되는 곳이다.

Guard에 설정될 값은 공격자가 알 수 없는 값이어야 한다. 만약, 설정된 값을 공격자가 알고 있다면 frame pointer와 buffer 사이 부분에 값을 가득 채울 수 있다. 그리고 guard의 검증테스트를 통과한 후, return address를 바꿀 것이다.

Guard값은 random한 값을 선택한다. random 값은 프로그램이 실행될 때마다 새로이 계산된다. 그리고 비인증된 사용자는 그 값을 찾을 수가 없다. Linux에



<그림 3> 제안 모델 구조

는 random number generator가 있으므로 /dev/random이나 /dev/urandom등을 이용하여 설계한다. 이 디바이스들은 읽힐 때 random number를 반환하며 응용 소프트웨어가 암호화를 위한 보안키를 선택하는데 도움을 주도록 설계되어 있다.

4.2 특징

제안하는 모델이 가정하고 있는 것은 다음과 같은 것들이다.

(1) Argument부분은 배열이나 pointer변수가 위치하지 않는다. 즉, 어떤 값도 넘치게끔 대입 못하고 고정된 크기의 argument area에 대해서는 값이 넘쳐날 수 없다.

(2) Array부분은 배열을 포함한 구조나 배열이 위치한다. 유일한 약점이 되는 위치로서 실제크기보다 큰 값을 쓰면 시스템이 종료된다.

(3) Local variable부분은 배열이 위치하지 않는다. 이 부분은 버퍼가 없고, 단순 int i 같은 변수가 존재한다.

제안된 모델은 다음과 같은 특징을 갖는다.

첫 째, 메모리 area는 함수가 리턴 될 때 손상되지 않는다. Array area는 stack을 공격할 때 파괴될 수 있는 취약한 지역이다. frame 바깥쪽의 피해는 guard 값을 살펴봄으로써 대비할 수 있다. 만약 frame의 바깥쪽이 피해가 발생했다면 프로그램은 실행을 멈춘다.

둘 째, pointer 변수를 공격하는 것은 성공하지 못한

다. 공격이 성공을 하기 위해서는 공격자가 function pointer값을 변경하고, function pointer를 사용하고 있는 함수를 호출해야 한다.

Stackguard는 단순히 canary값을 이용하여 return address만 보호하는 기능을 가지고 있으나, 여기서 제안한 모델은 guard area를 설정함으로써 saved frame pointer와 argument도 보호 할 수 있다. 또한 함수의 return address를 guard area에 복사함으로써 공격에 의해 return address가 변경되면, 그걸 감지하여 다시 원래의 주소로 되돌려 놓을 수 있고, 강력한 공격일 경우에는 시스템을 종료시킨다.

5. 결론 및 향후 연구과제

본 논문에서는 stackguard가 단순히 return adress만을 검사하여 보호하는것을 좀 더 개선시킨 모델을 제안하였다. Canary 값만을 이용하여 return address만 보호하는 방법을 guard지역을 설정하여 시스템만 아는 난수 값을 저장해 둔 후에 그 값의 변경 유무를 검사함으로써 saved frame부분과 argument부분도 보호하게 하였다.

아직까지 heap 오버플로우에 관한 대책은 뚜렷이 나온 게 없으므로, 이에 대한 대비가 이루어져야겠다. 또한 해킹으로부터 프로그램을 보호하는 방법과 시스템 오류및 취약점을 찾아내는 방법들도 더욱 정교하게 개발되어야 할 것이다. 해킹 기술이 끊임없이 발전되고 있는한 대응 기술 또한 끊임없이 발전되어야 한다.

[참고문헌]

[1] CERT TCC-KR, "Hacking 9월 통계자료", http://www.certcc.or.kr/

[2] Taeho Oh, "Bufferoverflow attack", Security PLUS for UNIX, 영진.com, pp59-176, 2000

[3] Aleph One, "Smashing The Stack For Fun And Profit", Phrack Magazine issue49, vol. 7., 1996

[4] Matt Conover, "w00w00 on Heap Overflows", http://www.w00w00.org/files/articles

[5] 박현미, "안전한 유닉스 프로그래밍을 위한 지침서 V.0.7", CERT TC-KR Technical Report CERTCC-KR-TR-2001-02, 2001

[6]"Stackguard Mechanism", http://www.immunix.org