안전한 시스템을 위한 Buffer Overflow 대응기법분석에 관한 연구

조 진 호* 황 현 욱** 박 종 백*
*조 선 대 학 교 전 자 정 보 통 신 공 학 부, **한 국 전 자 통 신 연 구 원(ETRI)
choco75@icsd.chosun.ac.kr, hhu62769@mail.etri.re.kr, pjb@icsd.chosun.ac.kr

A Study of Buffer Overflow Prevention Technique for Secure System

Jin-Ho Jo Hyun-Uk Hwang Jong-Baek Park
*Dept. of Electronic Information Communication, Chosun University, **ETRI

요 약

Buffer Overflow는 가장 핵심적인 해킹기법중의 하나이다. 현재 해킹피해의 대부분을 차지하며 local attack 뿐 아니라 remote attack까지 가능하므로 이를 이해함은 보안을 이해하는 차원에서 매우 중요하다. 본 논문에서는 Buffer Overflow의 원리와 최근 발생한 시스템의 해킹 형태를 살펴보고, 이를 방지할 수 있는 기법들을 분석하여 Buffer Overflow에 대응할 수 있는 모습을 그려보고자 한다.

1. 서론

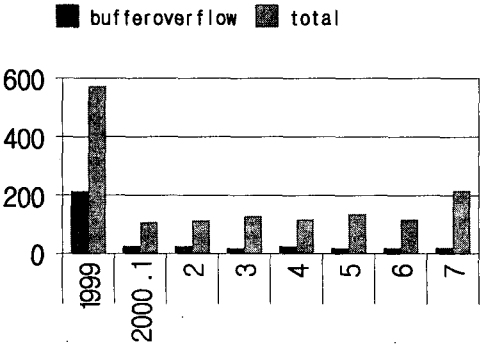
최근 들어 컴퓨터 기술의 발달과 인터넷의 확산으로 인하여 누구나 쉽게 인터넷을 접하게 되었다. 이로 인해 쉽게 해킹 정보를 접하게 되고 다른 시스템을 침입할 수 있는 크랙킹이 늘게 되었다. 이러한 이유로 컴퓨터의 보안 문제가 심각하게 대두되고 있다.

본 논문에서는 컴퓨터 시스템을 위협하는 여러 가지 해킹 기법 중 가장 최근 들어 급격한 증가와 심각한 위험성을 가져온 버퍼오버플로우 해킹 기법과 현재 유행하는 취약점을 분석하여 그 원리를 이해하고, 버퍼오버플로우 대응방법에 관한 방법론을 분석하여 보안시스템의 모습을 제시하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 버퍼오버플로우의 피해통계를 보고, 3장에서는 공격원리를 살펴보며, 4장에서는 현재 시스템의 해킹상황을 알아보고 5장에서 이의 대응기법을 살펴보며 6장에서 시스템에서 버퍼오버플로우의 대응영역을 살펴보고, 마지막에서 결론을 기술한다.

2. 버퍼오버플로우 상황 통계

1988년 Internet Worm 사건을 시작으로 하여, 인터넷이 급성장함에 따라 부적절한 방법으로 원격시스템에 접근하는 시도가 늘고 있다.



[그림 1] 전체 해킹피해상황에서 버퍼오버플로우의 빈도수 : CERTCC-KR 통계

그림 1에 CERTCC-KR의 통계에서 볼수 있는것처럼 1999년도에는 전체 해킹시도의 약 40%를 차지하며 올해들어서도 그 시도가 끊임없이 줄지 않고 있다.

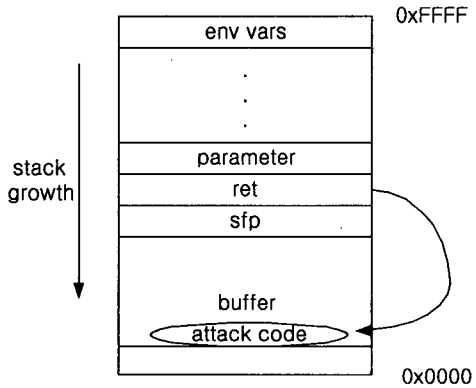
3. 버퍼오버플로우의 공격원리

일반적으로 해킹 공격의 원리는 크게 두가지를 지향한다. 첫째는, 공격 코드를 삽입하여 명령어의 순서를 변경함으로써 악성 코드가 쉘을 실행시키는 방법과, 둘째는, 동작되는 프로세스의 path를 변경함으로써 공격코드를 실행시키는 방법이다. 이 두가지 모두를 지향하는 공격이 버퍼오버플로우 공격이다. 그 중에서 일반적으로 말하는 기법은 "stack smashing attack"이라고 하는 버퍼오버플로우 기법을 말하는 것이고, 또 다른 오버플로우 기법은 heap overflow 가 있다. 이는 응용프로그램에 의해 heap 이란 곳에 동적으로 할당되는 메모리영역을 이용해 exploit 하므로 버퍼오버플로우 기법보다 더 어렵고 일반적으로 유행하지는 않는다

```
char shellcode[] =
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
"\x80\xe8\xdc\xff\xff\xff/bin/sh"; /* 쉘코드 생성, x86*/
char large_string[128];
int main() {
  char buffer[96];
  int i;
  long *long_ptr = (long *)  large_string1; /*버퍼의 주소를 채움*/

  for (i = 0; i < 32; i++)
    *(long_ptr + i) = (int) buffer;
  for (i = 0; i < strlen(shellcode); i++)
  large_string[i] = shellcode[i]; /*쉘코드를 large_string 배열에 처음에 채움*/
  strcpy(buffer,large_string); /*large_string을 buffer에 복사.*/
}
```

<표1> 스택버퍼오버플로우의 샘플코드



<그림2 > 표1 프로그램 실행 후 스택

표 1의 코드는 버퍼오버플로우의 샘플코드를 실행한 모습을 보여주는 스택의 모형이다. 이를 실행하면 그림 1에서 보는것처럼 return address가 buffer의 주소로 덮어 쒸어짐으로서 버퍼의 제일 앞의 주소를 가리키고 여기에 쉘코드가 위치함으로써 쉘을 생성하게 된다.

4. 버퍼오버플로우를 이용한 시스템 해킹

리눅스와 같이 소스가 공개된 운영체제에서부터 UNIX, WINDOWS 시스템에서도 buffer overflow의 위험은 상당수 존재하고 있으며 이들 프로그램의 사용에 주의를 기울여야할 것이다.

취약 프로그램	설명
/usr/bin/man	redhat 6.1 (and others), setgid획득
wuftpd250	heap overflow를 통한 remote attack root 획득
DNS/BIND	ADM named 8.2/8.2.1 NXT remote overflow, root 획득
rpc.statd	remote root획득, redhat 6.x & other
amd	remote root획득, redhat 6.0
qpopper	qpopper2.5 이전버전
MDBMS	MDBMS V0.96b6 remote root획득

<표2>최근 버퍼오버플로우 취약성을 가진 프로그램

위 제시된 목록은 최근 buffer overflow를 이용한 취약성을 가진 프로그램을 나열했다. buffer overflow는 local attack 뿐 아니라 remote 에서도 공격으로 쉽게 초보들도 exploit 코드를 실행함으로써 root의 권한을 획득하고 있다.

5. 버퍼오버플로우 대응기법

버퍼오버플로우를 대응 기법은 크게 두 가지가 있다. 첫째는 C컴파일러가 Boundary 검사를 하도록 수정함으로써 buffer overflow를 막는것이고, 둘째는 커널 자체를 수정함으로써 stack 영역의 실행영역을 non-excutable 하도록 수정해주는 것이다. 다음은 버퍼오버플로우를 방지하기 위한 기법에 대한 설명이다.

● secure linux kernel patch

kernel patch는 user stack에서 코드실행을 막아줌으로써 가장 근본적인 영역에서 버퍼오버플로우를 해결하는데 역점을 두었다.

```
#ifdef CONFIG_1GB
        .quad 0x00cbfa000000f7ff     /* 0x23 user   3GB-8MB code at 0 */
#elif defined(CONFIG_2GB)
        .quad 0x00c7fa000000f7ff     /* 0x23 user   2GB-8MB code at 0 */
#elif defined(CONFIG_3GB)
        .quad 0x00c3fa000000f7ff     /* 0x23 user   1GB-8MB code at 0 */
#else
#error Unknown max physical memory size requested
#endif
#else
        .quad 0x00cffa000000ffff  /* 0x23 user   4GB code at 0x00000000 */
```

<표3> head.S에서 user 공간을 정의한 kernel patch

```
#define MAGIC_SIGRETURN      (PAGE_OFFSET + 0xDE0000)
#define MAGIC_RT_SIGRETURN (PAGE_OFFSET + 0xDE0001)
```

● StackGuard

Stackguard는 컴파일러를 이용한 기술이다. 이는 stack의 return address 옆에 "canary" 워드를 넣고, 함수가 return 될 때 canary value 가 바뀌었는지를 살펴봄으로써 stack smashing attack이 실행된 것으로 간주하고 syslog에 기록을 남기고 프로그램을 정지한다.

StackGuard에서는 공격자의 canary spoofing을 막기 위해 크게 2가지 기술을 사용한다. 첫째, Random canary 기술이다. 이는 프로그램이 실행되는 동안에 random으로 선택되어져, 이전에 실행한 실행 이미지를 통해 얻은 canary 값을 다시 사용하지 못하게 된다. 둘째, termiantor canary 기술이다. 이는 NULL(0x00), CR(Ox0d), LF(0x0a)와 EOF(0xff)을 포함해 canary 값을 만들게 된다. 이를 이용함으로 문자열 작동을 멈추게 하는 것이다.

● libsafe

libsafe는 buffer overflow를 발견하고 다루는데 새로운 방법을 쓰고 있다. libsafe는 취약성이 알려진 모든 library function call을 도중에서 잡는다. libsafe는 동적 적재 가능한 library로 프로그램상의 버퍼오버플로우 취약성을 갖는 system call에 대해 효과적이다.

함수 원형	문제점
strcpy(char *dest, const char *src)	dest buffer overflow
strcat(char *dest, const char *src)	dest buffer overflow
getwd(char *buf)	buf overflow
gets(char *s)	s buffer overflow
fscanf(FILE *stream, const char *format, ...)	argument overflow
scanf(const char *format, ...)	argument overflow
realpath(char *path, char resolved_path[])	path buffer overflow
sprintf(char *str, const char *format,..)	str buffer overflow

<표4> 표준 C library에서 안전하지 못한 함수들

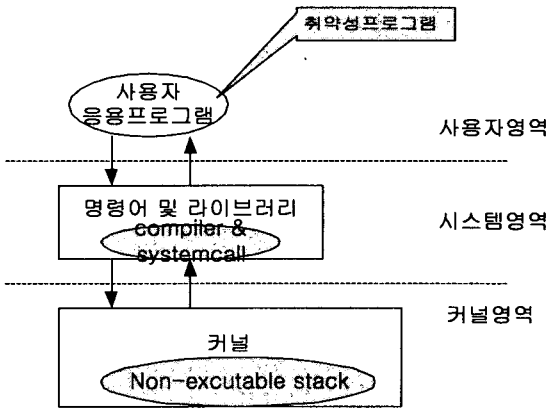
그림은 위에서 밝힌(표2) 취약성 프로그램을 대상으로 libsafe와 stackguard, non-stack kernel patch를 대상으로 적용시켜본 결과이다.

취약 프로그램	None	libsafe	StackGuard	Non-Executable Stack
/usr/bin/man	setgid획득	중지	중지	중지
rpc.statd	root획득	중지	중지	중지
DNS/BIND	root획득	중지	중지	중지

<표5> 취약프로그램에 대한 각 기법 적용

6. 시스템에서 buffer overflow의 대응 영역

시스템의 안정성을 위해서는 해킹 대응에 대한 시스템의 설계 또한 중요하다. buffer overflow는 root를 획득할수 있는 해킹기법의 가장 핵심적인 기술로 많은 주의를 요하지 않을 수 없다. buffer overflow를 막는 것은 시스템의 보안을 한층 강화시키기 위한 요소일뿐이다.



<그림3> 시스템에서 buffer overflow의 대응 영역

7. 결론

본 논문에서는 버퍼오버플로우의 위험성과 원리를 알아보고 그에 따른 대처방안에 관한 방법에 대해 분석해보았으며 이의 대응 영역을 그려보았다. 버퍼오버플로우 문제의 시초는 프로그래머의 코딩자체에서 시작된 문제라고 하더라도 안전한 시스템을 위해서는 버퍼오버플로우의 방지는 필수적인 것이고 이를 이해하는 것은 매우 중요한 일이다. 버퍼오버플로우를 막기위해 여러 가지 기법들을 분석해보았지만, 안전한 시스템을 위해서는 꾸준한 관리와 노력이 필수적이다. 항상 보안의 측면에서 관심과 주의를 기울여야 할 것이다.

[참고문헌]

[1] CERTCC-KR advisory, http://www.certcc.or.kr
[2] Aleph One. http://www.phrack.com Smashing the stack for fun and profit(Phrack Magazine, 49-14, 1998)
[3] Linux Kernel patch, Openwall project http://www.openwall.com/linux,
[4] Libsafe: Protecting Critical Elements of Stacks http://www.bell-labs.com/org/11356/libsafe.html
[5] StackGuard : Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks, In Proceedings of the 7th USENIX Security Conference
[6] http://www.attrition.org
[7] linux exploit code, http://www.self-evident.com