리눅스 커널 수준에서의 버퍼오버플로우 공격 방지 기법

김홍철* 송병욱 박인성 김상욱

경북대학교 컴퓨터과학과

Prevention of Buffer Overflow Attack on Linux Kernel Level

Hong-chul Kim* Byung-wook Song In-sung Park Sang-wook Kim

Department of Computer Science, Kyungpook National Univ.

요 약

시스템 침입을 위해서 사용할 수 있는 공격 기법은 그 종류가 매우 다양하다. 그러나 최종적인 시스템 침입의 목표는 버퍼오버플로우 공격을 통한 관리자 권한의 획득이다. 버퍼오버플로우 현상은 메모리 공간의 경계 영역에 대한 프로그래밍 언어 수준의 검사 도구를 제공하지 않는 C/C++ 의 언어적 특성으로 인해 발생한다. 본 논문에서는 리눅스 커널 수준에서 시스템 보안을 위한 참조 모니터를 제안하고 이를 이용하여 버퍼오버플로우 공격에 대응할 수 있는 보안 기법을 제시한다.

I. 서론

현재 컴퓨터 시스템을 시스템 해킹으로부터 막을 수 있는 방법으로 가장 많이 사용하는 형태는 운영 체제와 별도로 구성되는 보안 시스템을 구축하는 것이다. 그 예로는 침입 탐지 시스템, 침입 차단 시스템, 실시간 스캐너 탐지 도구, 파일 시스템 무결성 검사 도구 등이 있다.

그러나 이러한 별도의 보안 도구들 중 대부분은 시스템에 대한 해킹 과정이 어느 정도 진행되어 그 피해가 발견된 시점에서 침입에 대응하는 방법을 취하고 있다. 그 대표적인 예가 침입탐지 시스템이다. 침입 차단 시스템이나 실시간 스캐너 탐지 도구의 경우 침입에 대한 징후만을 판별하므로 실질적인 대응이 부족하다.

이와 같은 기존 보안 도구가 갖고 있는 한계의 근원은 실제로 피해를 당하는 쪽은 운영체제와 같은 대상 시스템 자체인데 반해, 그러한 침해 행위를 감시, 탐지 및 대응하는 도구는 운영체제와는 별개의 개체로서 존재하는 응용 프로그램 또는 별도로 운영되는 시스템이기 때문이다.

이와 같은 기존 보안 도구의 문제점을 해결하기 위해서 본 논문에서는 운영체제 내부에 보안 기능을 내장시킬 수 있는 기법을 제시한다. 이때, 운영체제가 가지고 있어야 하는 고유한 특성인 실행의 효율성 및 작동의 안정성을 보장하기 위해서 최소한의 기본적인 보안 모듈만을 추가한다. 이를 위해서 시스템 침입 행위에서 이용되는 공격 기법의 유형을 분석하고, 그 공격이 적용되는 근본적인 이유를 분석하여 그것을 보완할 수 있는 보안 기법을 제시한다.

커널 내부에 보안 기능 모듈을 직접 내장함으로써 별도의 보안 도구를 구축해야 하는 문제점이 해결된다. 또한, 시스템 침입 행위에 대한 탐지 및 대응 역시 운영체제 차원에서 수행되므로 가장 실질적인 보안 정책을 수립할 수 있다.

논문의 순서는 다음과 같다. 제 II장에서는 버퍼오버플로우 현상 및 버퍼 오버플로우 공격에 대해서 분석한다. 제 III장에서는 버퍼오버플로우 공격을 감지하고 대응할 수 있는 기능을 리눅스 커널 모듈의 형태로 구현하는 기법을 제시하고 제 IV장에서 결론을 맺는다.

II. 버퍼오버플로우 공격

1. 버퍼오버플로우

버퍼 오버플로우 공격의 원인을 분석하기 위해서는 운영체제에서의 프로세스 메모리 구조를 분석해야 한다. 프로그램은 물리적인 디스크 상에 텍스트, 초기화된 데이터, 심볼 테이블, 그리고 해당 운영체제 및 하드웨어에 종속적인 링커 헤더와 매직 넘버로 구성되어 있다. 이러한 형태의 프로그램이 실행되어 메모리로 확장될 때의 구조는 그림 1과 같다. 텍스트 영역은 실행될 기계어가 위치하고 있으므로 그 크기가 변하지 않는다. 데이터 영역은 초기화된 데이터 영역과 초기화되지 않은 영역(bss)으로 나누어진다. 스택 영역과 bss-data 영역은 프로그램을 실행함에 따라서 서로 상대방을 향해서 크기가 증가한다. 만일 스택과 bss-data 영역 사이에 충분한 공간이 존재하지 않으면 프로세스는 일시적으로 중지된 후 커널이 새로운 메모리 공간을 할당한다. 프로그램이 실행되면 제일 먼저 텍스트 영역이 메모리로 복사된 다음 데이터 영역이 적재된다. 스택은 맨 마지막 단계에서 동적으로 할당되어 존재하게 된다.

스택은 프로그램이 실행됨에 따라서 여러 개의 스택 프레임으로 나누어진다. 각 스택 프레임은 해당 서브루틴에 대한 정보를 저장한다. 이는 서브루틴의 호출 인자 리스트, 복귀 주소, 자동 변수를 위한 메모리 공간 등을 포함한다. 스택 프레임에 위치하는 지역 변수는 스택의 맨 위를 가리키고 있는 레지스터를 이용해서 접근 가능하다. 만일 스택이 높은 메모리 영역에서 낮은 메모리 영역으로 성장한다면, 스택 프레임 내의 지역 변수는 스택의 맨 위를 가리키는 값에서 특정 값을 빼줌으로써 접근 가능하다. 하지만, 이와 같은 방법은 스택에 새로운 값이 계속 PUSH되거나 POP될 때마다 스택의 맨 위를 가리키는 값이 달라지므로 스택 프레임 내부에 위치하는 동일한 변수라도 시간에 따라서 오프셋 값이 달라지게 된다. 따라서, 대부분의 CPU에서는 스택의 맨 아래 부분을 가리키는 레지스터를 이용할 수 있도록 하고 있다. 이러한 용도로 사용되는 레지스터를 FP(Frame Pointer)라고 지칭한다.

스택의 맨 위 부분을 가리키는 레지스터는 SP(Stack Pointer)라고 지칭한다. 본 논문에서는 인텔 계열의 x86 CPU를 다루므로, 이들 레지스터는 각각 BP(EBP)와 SP라고 지칭된다. 즉, 서브루틴은 자신에게 할당된 로컬 스택 영역을 이용하기 위해서 BP(EBP)의 값에 특정 값을 빼 줌으로써 변수를 접근할 수 있다.

다음과 같은 C 프로그램이 32-bit Intel CPU 상에서 실행될 때 메모리의 스택 영역은 그림 1과 같다.

```
void function(char *str) {
        char buf[4];

        strcpy(buf, str);
}

int main(int argc, char* argv[]) {
        int i = 0;
        function(argv[i]);
}
```

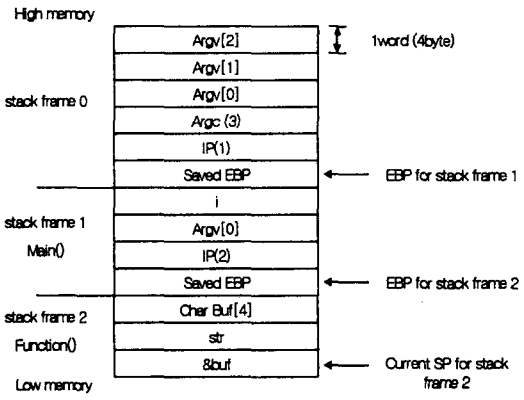

그림 1: 프로세스의 스택 프레임 구조

2. 버퍼오버플로우 공격

버퍼 오버플로우 공격은 이와 같은 프로그램의 버퍼에 오버플로우를 유발하여 의도적으로 특정 프로그램이 실행되도록 하는 것이다. 이것을 위해서는 특정 프로그램을 실행시키는 기계어가 필요한데, 이를 쉘 코드(Shell Code)라 지칭한다. 시스템을 공격하는 침입자는 자신이 실행하고자 하는 프로그램을 쉘 코드의 형태로 제작한다. 이렇게 생성한 쉘 코드는 버퍼 오버플로우가 발생할 때 그 내용이 메모리 상에 저장되도록 한다. 쉘 코드는 운영체제 및 하드웨어에 따라서 다르다. 쉘 코드는 필수적으로 시스템 호출을 유발시키는 기계어를 필요로 한다. 하드웨어에 따라서 기계어의 종류가 다르며, 하드웨어가 동일하더라도 운영체제가 다르면 시스템 호출을 위한 소프트웨어 인터럽트 절차가 다르다. 따라서, 쉘 코드는 특정 하드웨어 및 운영체제에 따라서 별도로 제작되어야 한다. 버퍼 오버플로우가 발생하면서 쉘 코드가 버퍼에 저장되고, 그것이 실행되는 절차는 그림 2와 같다. 이와 같이 버퍼 오버플로우 공격은 최종적으로 쉘 코드를 실행하는 기계어가 실행되도록 하

는데 그 목적이 있다.

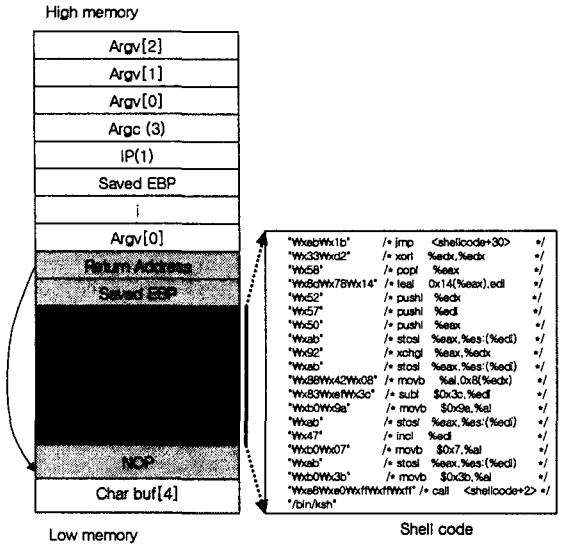

그림 2: 버퍼 오버플로우에 의한 쉘 코드 실행

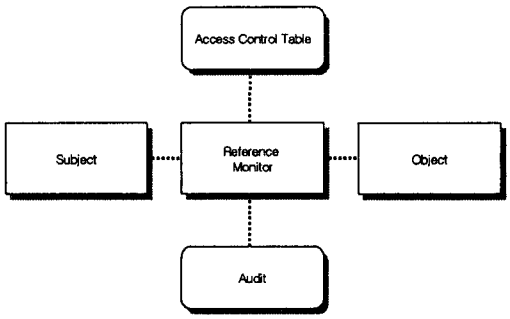

그림 3: 참조 모니터 구성 요소

버퍼오버플로우 공격을 방지하기 위한 참조 모니터의 접근 제어 테이블은 execve와 같은 주요 시스템 콜에 대한 보안 정책을 가지고 있다. execve 시스템 콜에 대한 테이블 엔트리는 연결 리스트로 구현되어 execve 시스템 콜에 대한 접근 제어 보안 정책을 표현한다. execve 시스템 콜이 호출될 경우 참조 모니터는 전달된 인자 및 현재 프로세스의 정보를 이용해서 execve 시스템 콜 요청을 수락하거나 거부할 수 있다.

III. 버퍼오버플로우 공격 대응 기법

1. 참조 모니터

참조 모니터(Reference Monitor)는 시스템에 대한 어떠한 행위가 요청될 때 이를 사전에 수립되어 있는 정책을 참조하여 그 요청을 허용할 것인지 여부를 결정하는데 사용되는 방법이다. 그림 3은 컴퓨터 보안에 적용되는 참조 모니터의 개념적인 모델을 나타낸다.[1]

컴퓨터 시스템에 적용되는 참조 모니터 모델은 주체(Subjects), 대상(Objects), 접근 제어 테이블(Access Control Table), 감사(Audit)로 구성된다.

본 논문에서는 버퍼오버플로우 공격에 대응하기 위해서 참조 모니터 모델을 응용한다. 버퍼 오버플로우 공격은 최종적으로 execve 계열의 시스템 콜을 호출하여 쉘 프로세스가 생성되도록 한다는 특성을 가지고 있다. 따라서 버퍼오버플로우 공격을 방지하기 위한 참조 모니터 모델에서 통제의 대상이 되는 행위 주체는 응용 프로그램이 되며, 대상은 프로세스를 생성하는 execve 시스템 콜이다.

2. 리눅스 커널의 수정

1) 프로세스 식별

버퍼오버플로우 공격의 대상은 모든 프로세스가 아니다. 공격의 대상이 되는 프로세스는 루트 권한으로 실행되는 프로세스이다. 이와 같은 공격 대상 프로세스는 크게 다음과 같이 분류할 수 있다:

- 백그라운드 데몬 프로세스
- SUID 프로세스
- 기타 서비스 프로세스

백그라운드 데몬 프로세스는 어떠한 서비스를 제공하기 위해서 시스템의 백그라운드에서 실행되는 프로세스이다. **sendmail**, **httpd** 등이 그 예이다. SUID 프로세스는 일반 사용자가 시스템의 자원에 제한적으로 접근할 필요가 있는 경우에 사용된다. SUID 프로세스는 사용자가 원하는 시기에 프로그램을 직접 실행할 수 있기 때문에 시스템 침입시 가장 많이 이용되는 부류의 프로세스이다. 기타 서비스 프로세스는 로컬 또는 리모트 서비스를 하기 위해서 일시적으로 실행되는 루트 프로세스로서 **xinetd(inetd)**에 의해서 제어되는 **pop3**, **in.telnetd**, **in.ftpd** 등과 같은 네트웍 관련 서비스 데몬 프로세스이다.

2) 시스템 콜 접근 제어

버퍼 오버플로우 공격이 발생했을 때 최종적인 과정에서 발생하는 시스템 콜은 execve이다. 따라서 버퍼 오버플로우 공격을 감지하고 대응하기 위해서는 execve 시스템 콜을 가로채어 커널의 실행 흐름을 참조 모니터 제어 모듈로 전이시켜야 한다.

리눅스 커널에서 시스템 콜 테이블은 다음과 같이 표현된다.

```
extern void *sys_call_table[];
```

시스템 콜 테이블 sys_call_table[] 중에서 특정 시스템 콜에 해당하는 커널 함수를 찾기 위해서는 커널에서 정의되어 있는 인덱스 매크로를 이용한다. 각 시스템 콜의 인덱스는 다음과 같은 형태로 정의되어 있다.

```
#define __NR_exit          1
#define __NR_fork          2
.....
#define __NR_execve        11
```

execve 시스템 콜은 커널 내부에서 sys_execve 함수로 구현되어 있다. C 표준 함수가 제공하는 execve 함수는 실행 프로그램의 파일 경로, 명령 행 리스트, 환경변수 리스트의 시작을 가리키는 주소 값을 특정 레지스터에 저장하며, sys_execve 커널 함수는 이들을 pt_regs 자료 구조 타입으로 받아들인다. 참조 모니터 제어 모듈을 구동시키기 위해서 sys_execve 커널 함수는 그림 4와 같은 형태로 변경된다.

sys_execve 커널 함수는 인자로 전달받은 pt_regs 자료구조로부터 새로운 프로그램을 실행하기 위해서 필요한 정보를 얻는다. pt_regs 자료구조는 다음과 같다.

```
struct pt_regs {
        long ebx, ecx, edx;
        long esi, edi;
        long esp, ebp;
        long eax, orig_eax;
        int  xds, xes, xss;
        int  xes, xcs;
        long eip;
        long eflags;
};
```

현재 리눅스는 인텔 x86 CPU를 포함한 다양한 하드웨어에서 운영된다. 본 논문에서는 인텔 x86 CPU에서 운영되는 버전만을 다룬다. execve 시스템 콜의 인자 filename, args[], envp[]는 각각 pt_regs의 ebx, ecx, edx 필드를 통해서 sys_execve 커널 함수로 전달된다. x86 하드웨어에서 운영되는 리눅스 커널의 경우 쉘 코드는 execve 시스템 콜을 발생시키기 위해서 0x80 소프트웨어 인터럽트를 이용한다

.

```
asmlinkage int sys_execve(struct pt_regs regs)
{
  int error;
  char * filename;
  check_result_t result;

  filename = getname((char *) regs.ebx);

  if (vulnerable_process(current)) {
    result = check_bof_attack(current, filename,
                              (char **)regs.ecx,
                              (char **)regs.edx);
    if (result == ENAUT)
      return;
  }

  error = PTR_ERR(filename);
  if (IS_ERR(filename)) goto out;
  error = do_execve(filename, (char **) regs.ecx, (char **) regs.edx, &regs);
  if (error == 0)
    current->ptrace &= ~PT_DTRACE;
  putname(filename);
out:
  return error;
}
```

그림 4: sys_execve 커널 함수의 수정

IV. 결론

본 논문에서는 시스템 침입 과정에서 필수적으로 사용되고 있는 버퍼오버플로우 공격을 분석하고 이를 리눅스 커널 수준에서 대응할 수 있는 기법을 제시하였다. 버퍼오버플로우는 모든 프로그램이 잠재적으로 가지고 있는 특성이므로 이것 자체를 방지하는 것은 사실상 불가능하다. 본 논문에서는 버퍼오버플로우가 발생하더라도 그것이 불법적인 루트 쉘을 생성하지 않는 방법을 사용하였다. 이는 커널 수준에서 구현되므로 사용자로부터의 투명성을 유지하며, 특정 시스템 콜에 대해서만 참조 모니터를 통해서 접근 제어 정책을 적용하므로 심각한 시스템의 성능저하를 유발하지 않는다. 향후 과제로는 false negative 및 false positive 오류를 줄이는 기법을 고안하는 것이다.

참고문헌

[1] Aleph One, "Smashing The Stack For Fun and Profit," Phrack Magazine, Vol. 7, N. 49, 1996

[2] Bellovin S.M., Cheswick W. R, Firewalls and Internet Security, Addison Wesley, 1994

[3] Alessandro Rubini, Jonathan Corbet, LINUX DEVICE DRIVERS, Oreilly, 2001

[4] Ames, S.R., Gasser, M., Schell, R.R., "Security Kernel Design and Implementation: An Introduction," IEEE Computer, Vol. 16, N. 7, 14-22, 1983.