버퍼넘침(buffer overflow)을 이용한 해킹 공격기법 및 예방방안

이형봉[0] 차홍준
호남대학교 정보통신공학부 강원대학교 컴퓨터과학과
hblee@honam.ac.kr tchahj@cc.kangwon.ac.kr

A Study on Hacking Attack using Buffer Overflow and Strategy to Avoid the Attack

Hyung Bong Lee[0] Hong Joon Cha
Dept. of Info. and Comm. Engineering, Honam University
Dept. of Computer Science, Kangwon University

요 약

버퍼넘침(buffer overflow)은 특정 프로그램 언어에서 발생하는 배열의 경계파괴 현상을 말한다. 그 대표적인 언어로서 C/C++을 들 수 있는데, 이들 언어는 기본적으로 스트링(문자열)을 정의함에 있어서 크기속성을 배제하고 끝을 의미하는 종료문자(delimiter character, NULL)를 사용함으로써 배열(버퍼)의 경계침범 가능성을 허용하고 있다. 이 때 스택영역에 할당된 버퍼가 넘친다면, 주변에 위치한 지역변수, 레지스터 보관, 복귀주소 등의 값이 변질되어 원래의 의도된 제어흐름을 보장할 수 없게 된다. 특히 복귀주소 부분을 의도적으로 침범하여 특정 값을 덮어쓸 수 있다면 해당 프로그램의 동작을 인위적으로, 그리고 자유롭게 변경할 수 있게 된다. 본 논문에서는 이와 같은 스택영역에서의 버퍼넘침을 사용한 제어흐름 변경 해킹기법의 과정을 현존하는 UNIX 시스템 및 C/C++ 언어를 이용하여 살펴보고 대응방향을 모색한다.

1. 서론

기존에는 보안침해(해킹) 수단으로서 시스템의 관리나 설정상의 허점을 공격하는 것이 주류를 이루고 있었다. 그러나 이런 형태의 침해에 대한 대응기법이 어느 정도 정착되고 현실화되자, 최근에는 프로그램 언어의 구조, 프로세서의 특징, 그리고 운영체제의 특성을 복합적으로 이용하는 고도로 지능화된 기법들이 사용되고 있다.

이런 유형의 침해기법은 시스템을 파괴하는 것을 주 목적으로 하는 바이러스와는 달리 목적 시스템에 지속적으로 은밀하게 침투하여 오랜 기간동안 침해행위를 유지할 수 있을 뿐만 아니라, 특정 프로그램(원격 로깅 프로그램, 방화벽, 기간 업무 프로그램 등)의 허점을 철저하게 개인이 발굴하여 악용하게 되므로 예방이나 보완을 위해 널리 사용될 수 있는 툴이나 규칙 등이 존재하지 않는다는 점에서 더욱 심각하다고 할 수 있다[1,2].

위와 같은 침해기법 중 대표적인 것 중의 하나가 버퍼넘침(buffer overflow)을 사용하여 공격하는 것이다. 버퍼넘침이란 배열로 할당한 메모리 공간에 데이터(주로 문자열)를 저장하는 과정에서 할당된 배열 공간영역을 넘어서 다른 주변의 다른 저장공간을 침범하는 현상을 말한다. 프로그램 언어에 따라 런 타임에 배열경계 침범을 허용하거나 불허할 수가 있는데, C/C++ 프로그램 언어는 경계침범을 허용하는 대표적인 언어 중의 하나이다. 버퍼넘침이 보안침해를 위한 공격 수단이 될 수 있는 이유는 대부분의 버퍼(배열 공간)가 스택영역에 할당될 수 있고, 그 주변에는 프로그램 흐름을 결정하는 복귀주소 등의 제어정보가 존재하기 때문이다. 즉 버퍼를 벗어난 일부 데이터가 제어정보를 덮어쓰고, 그 데이터가 악성 코드가 존재하는 곳에 대한 주소를 가지고 있다면 해당 프로그램은 원래의 목적대로 실행되지 못하고, 공격자의 의도된 침해 기능을 수행하게 되는 것이다.

본 논문은 2 장에서 관련연구를 통하여 C/C++ 프로그램의 함수호출 과정 및 버퍼넘침이 공격에 악용되는 원리를 이해하고, 3 장에서 DEC Alpha 21164 프로세서에 'Digital UNIX' 운영체제를 탑재한 digital 워크스테이션 600au 시스템에서 C/C++로 작성된 프로그램의 버퍼넘침이 공격기법으로 사용되는 실증적 시나리오 몇 가지를 예시하고 이에 대한 예방방안을 제시한 후, 4 장의 결론으로 맺는다.

2. 관련연구

C/C++ 언어에서 버퍼넘침이 프로그램 흐름에 영향을 준다는 점과 이러한 사실이 시스템 불법침입의 수단으로 사용되는 과정은 이미 보편적으로 널리 알려져 있다[3]. 여기서는 C/C++의 함수호출 원리를 스택의 변화과정을 중심으로 살펴본 후 해킹의 수단으로 사용되는 과정을 진단한다.

2.1 UNIX(LINUX) a.out 포맷과 스택영역

UNIX 시스템에서 C/C++과 같은 고급언어가 기계어로 컴파일된 후 메모리에 적재되는 형태는 일반적으로 a.out 이라는 표준포맷을 따르고 있다. A.out 포맷은 프로세스 메모리 이미지를 text, data, stack 영역(region 혹은 section)으로 구분하고 각 영역은 고유의 특성(read, write, execute)을 가질 수 있도록 허용하고 있다. 이는 메모리 관리의 기법 중에서 segmented paging 기법에 해당된다[7].

여기서 중요한 사실은 스택영역은 프로그램이 처음 적재될 때, 프로세서,운영체제,컴파일러(언어)에 따라 각각 공유한 방법으로 설정되고 운영된다는 점이다. 스택은 초기크기에서 출발하여 제한된 범위까지 자동적으로 확장되는데, 대부분 함수가 호출되었을 때 지역변수를 위한 공간으로 사용된다.

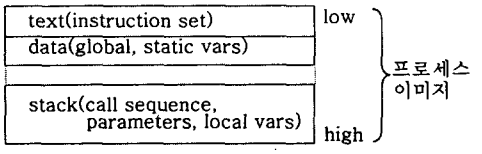

그림 1 UNIX a.out 포맷

2.2 C/C++ 함수호출 과정과 스택영역

C/C++언어의 함수호출 과정에서 스택에 저장되는 정보는 복귀주소, 일반 레지스터, 함수 파라미터 등을 포함하고, 호출된 후에는 지역변수 공간으로 사용된다. 컴파일러나 프로세서의 유형에 따라 복귀주소나 파라미터를 레지스터에 저장하는 경우가 있는데, 파라미터 수가 많거나, 함수호출 단계가 깊어지면 스택을 사용하지 않을 수 없다[5].

(그림 2)는 왼쪽 C 프로그램의 진행에 따른 스택의 생성과정을 보여주고 있다. 즉 함수가 호출될 때마다 호출된 서브함수를 위한 스택 프레임이 차곡차곡 쌓이게 되고, 특별한 경우(setjmp, longjmp 등)를 제외하고는 스택에 저장된 순서대로 복귀하면서 진행된다. 이때 복귀주소나 레지스터 저장부분을, 주 함수와 서브함수 중 어느 쪽의 스택프레임에 포함시킬 것인가 하는 것은 다분히 개념적인 선택의 문제인 것으로 생각하기 쉬우나, 다음에 설명될 버퍼넘침 문제를 다루는 시각에서는 상당히 심각한 차이점이 존재한다. 뿐만 아니라 이들 각 저장영역은 언어의 설계 및 프로세서의 특성에 따라 다르게 배치될 수도 있다.

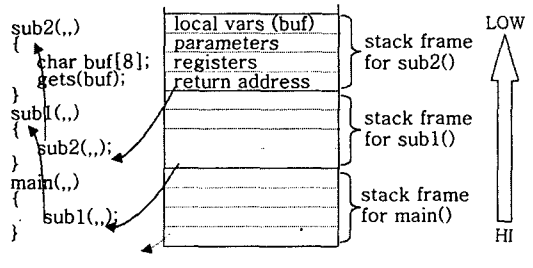

그림 2 C/C++ 함수호출과 스택변화

2.3 스택과 버퍼넘침(buffer overflow) 문제

버퍼넘침은 기본적으로 배열의 경계를 무시하는 언어의 특징에서 연유한다. (그림 2)의 예제 C 프로그램 진행과정에서 sub2() 함수의 지역변수인 buf가 경계를 벗어나 넘치면 파라미터, 레지스터, 복귀주소 등의 값 들을 덮어쓰게 된다. 즉 sub2()의 gets() 호출에서 사용자가 buf의 크기인 8보다 큰 문자열을 입력하더라도 buf의 경계를 조사하지 않기 때문에 위와 같은 상황을 피할 수 없다. 이 과정에서 복귀주소의 변화는 프로그램의 진행을 방해하므로 치명적이며, 더욱이 복귀주소 위치를 고려하여 의도적인 버퍼넘침을 유도하면 심각한 보안 침해를 당할 수 있다[3].

3. 버퍼넘침을 사용한 실증적 보안침해 시나리오 및 대응 방안

버퍼넘침을 수단으로 보안침해가 이루어질 수 있는 시나리오가 현존하는 UNIX 및 LINUX 시스템에 엄연히 존재하고 있다. 여기서는 1장에서 설명한 UNIX 시스템상에서 실증한 보안침해 시나리오를 제시한다.

3.1 관련된 UNIX 특성

■ 권한(privilege)위임 방법

UNIX시스템은 사용자가 아닌 프로그램에 권한을 부여하는 방법이 있는데 이른바 setuid/setgid 모드가 그 것이다. 즉 실행파일에 위의 모드를 설정해 놓으면 그 것이 수행될 때 해당 프로세스는 사용자가 아닌 실행파일의 소유자권한을 발휘한다.

■ 프로그램 실행 방법

UNIX시스템에서 새로운 프로그램을 실행시키는 유일한 방법은 exec()을 사용하는 것이다.

3.2 보안침해 환경 및 시나리오

Setuid 모드가 관리자로 설정된 (그림 2)와 같은 프로세스가, 버퍼크기를 고려하지 않는 gets() 함수를 호출하여 키보드로부터 사용자 입력을 받아들이는 상황에서, 적당한 문자열을 입력함으로써 현재 프로세스를 관리자 권한을 보유한 쉘 프로세스로 대체하는 과정을 실증적으로 제시한다.

■ 시나리오 1[3]

① 스택영역을 참조하여 buf의 주소를 예측한다.
② buf 에 입력할 문자열 내용으로 exec(" /bin/sh" , argv, NULL) 문장을 처리하는 기계어 코드를 준비한다.
③ 위의 문자열 뒤에는 buf의 주소를 반복하여 추가하여 buf가 넘쳐 복귀주소를 덮어쓸 수 있도록 길게 한다.
④ 위의 문자열을 입력한다.

문자열 입력 후 스택은 (그림 3)과 같이 변하고, sub2()에서 복귀할 때 복귀주소가 buf로 설정되어 buf에 들어있는 코드를 수행한다. 그러나 이 시스템에서는 이 시나리오는 동작하지 않는다. 왜냐하면 스택영역에는 실행모드가 설정되지 않았기 때문에[5,6], SIGSEGV 신호를 유발하고 멈춘다.

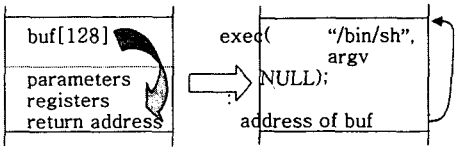

그림 3 sub2() 스택 프레임의 변화

■ 시나리오 2

① 해당 프로그램의 **실행파일을** 탐색하여 exec (" /bin/sh" , argv, NULL) **형태의** 코드가 있는 곳의 주소를 찾는다.

② 위 주소를 여러 번 **중복하여 버퍼넘침으로** 복귀주소를 덮어쓸 수 있을 만큼 **긴 입력문자로** 만든다.

③ 입력문자를 입력한다.

이 시나리오는 테스트 **시스템에서** 동작하는 것은 물론 다른 어느 시스템에서도 **동작할 것이다.**

■ 시나리오 3

① 버퍼가 shared memory(**혹은 mapped memory**) 와 스택 두 영역에 존재하는 **경우를 탐색한다.**

② shared memory에 있는 **버퍼 입력에** 시나리오 1에서 만든 입력 문자열을 **입력한다.**

③ 스택에 있는 버퍼 입력**시 shared memory**주소를 여러 번 반복하여 만든 문**자열을 입력하여** 버퍼넘침을 유도한다.

테스트 시스템에서 shared **memory 영역은** 실행모드가 가능하기 때문에 이 **시나리오 역시** 동작한다.

3.3 버퍼넘침에 대한 예방방안

3장의 몇 가지 실증적 **시나리오를 고려하면** 다음과 같은 몇 가지 대응 방안을 **제시할 수 있다.**

■ 버퍼넘침 경우 배제

버퍼크기를 고려하는 **함수만을 사용하여**, 언어 자체가 제공하지 않는 버퍼침범 **여부 검사를** 사용자가 실시한다.

■ 프로세스 스택주소의 **다양화**

스택영역에 위치한 버퍼**의 시작주소**는 프로세스가 수행될 때마다 동일하기 **때문에 추측하여** 알아내기가 비교적 용이하다. 그러나 **프로세스가** 시작할 때마다 다르게 설정된다면 **훨씬 어려워질** 것이다. 이는 crt 루틴에 대한 역간의 **수정으로 가능하다.**

인하여 없어질 가능성이 **매우 높기 때문이다.**

■ 힙(heap) 영역의 버퍼**사용**

스택버퍼 보다는 동적 **메모리할당을 이용한** 힙 영역의 버퍼를 사용하면 **버퍼넘침으로 인한** 복귀주소에 대한 공격을 최소화할 수 **있다.**

■ 언어의 스택 프레임 운용 **최적화**

(그림 2)에서 저장영역 **순서를 변경하여** 복귀주소를 높은 주소에서 **낮은 주소로 옮기면 버퍼넘침** 공격의 성공률을 낮출 수 **있다. 즉 덮어쓴** 복귀주소가, 현재의 서브함수가 **복귀할 곳이 아니고** 그 다음 함수가 복귀할 주소이기 **때문에(그림 4 참조)**, 버퍼에 저장된 코드 내용이 다른 **서브함수의** 호출로 인하여 없어질 가능성이 **매우 높기 때문이다.**

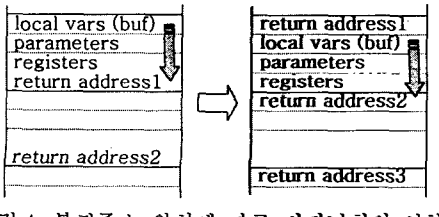

그림 4 복귀주소 위치에 **따른 버퍼넘침의 영향**

■ 스택영역에 대한 실행모드 **제거**

버퍼넘침 공격은 스택영역에 악성 프로그램 코드를 옮겨놓고 그것을 실행시킬 수 있을 때 더욱 강력한 침해수단이 된다. 이를 방지하기 위해서는 스택영역에 존재하는 프로그램 코드는 실행될 수 없도록 하는 프로세서 및 운영체제가 필요하다.

4. 결론

3장에서 살펴본 버퍼넘침에 대한 취약점은 현존하는 UNIX 및 LINUX 시스템에 엄연히 존재하고 있다. 그렇다고 버퍼넘침을 불허하는 언어만을 사용하여 모든 프로그램을 개발할 수는 없을 것이다. 프로그램 설계상 버퍼넘침을 오히려 활용하는 경우도 있다는 사실을 상기하면 더욱 그렇다. 따라서 버퍼넘침을 허용하는 프로그램 언어를 사용하는 한 버퍼넘침에 대한 공격을 기계적이고 자동적으로 봉쇄할 수는 없다. 이는 버퍼넘침을 허용하되 공격을 방어할 수 있는 프로그램 작성법이 무엇보다도 중요함을 의미하고 있다.

지금까지의 소프트웨어공학 이론은 읽기나 이해하기가 쉬고 오류가 최소인 소프트웨어 생산에 주요 관심을 두고 있었다. 그러나 지금에 와서는 이러한 추구만으로는 부족하다. 방화벽 소프트웨어와 같은 거대한 보안 프로그램 자체가 앞에서 살펴본 바와 같은 공격대상을 내포하고 있다면 다른 오류보다 더욱 심각한 문제가 아닐 수 없기 때문이다.

위와 같이 보안 측면에서의 고품질을 보장하기 위해서는 소프트웨어 설계 및 구현 단계에서 적용될 새로운 소프트웨어공학적 패러다임과, 생산된 소프트웨어에 대한 안전성 테스트를 위한 보안검증 도구들이 연구되고 개발되어야 할 것이다.

5. 참고 문헌

[1]. David A. Wheeler , " Secure Programming for Linux and Unix HOWTO" , GFDL, 1999.

[2]. McClure, Scambray , Kurtz " Hacking Exposed " , OSBORNE, 1998.

[3]. Aleph One , " Smashing The Stack For Fun And Profit" , http://www.shmoo.com/phrack/Phrack49/p49-14,1996.

[4]. Uresh Vahalla , " UNIX Internals " , PRENTICE - HALL,1996.

[5]. Digital, " Digital UNIX Reference Pages " , Digital Press, 1996.

[6]. MAURICE J. BACH , " The Design of the UNIX Operating System" , PRENTICE-HALL,1986.

[7]. H. M. DEITEL , " An Introduction t o Operating Systems" , ADDISON-WESLEY,1983.