스택메모리상의 명령 수행 탐지를 위한 시스템콜 모니터링 도구 설계

최양서* · 서동일* · 이상호**
*한국전자통신연구원 네트워크보안구조연구팀, **충북대학교 컴퓨터공학과

Design of System Call Monitoring System for Command Execution Detection in Stack Memory Area

Yang-seo Choi* · Dong-il Seo* · Sang-ho Lee**
*ETRI Network Security Infrastructure Research Team, **Chung-buk University
E-mail : yschoi92@etri.re.kr

요 약

지난 1988년 인터넷 웜 사건을 계기로 널리 알려진 버퍼 오버플로우 해킹 기법은 최근까지도 가장 널리 사용되고 있는 해킹 기법이다. 최근 이러한 버퍼 오버플로우 해킹 공격 기법을 방어하기 위한 연구가 계속되었고, Libsafe, Stack guard 등 많은 해결책이 제시되기도 하였다. 본 논문에서는 기존 호스트 기반 침입탐지 시스템에서 사용하는 시스템 콜 모니터링 기법을 이용하여 시스템 콜이 발생하는 메모리상의 위치를 확인함으로써 스택 버퍼 오버플로우 해킹 공격을 방지하기 위한 새로운 방어기법을 제시한다.

ABSTRACT

After Morris' Internet Worm in 1988, the stack buffer overflow hacking became generally known to hackers and it has been used to attack systems and servers very frequently. Recently, many researches tried to prevent it, and several solutions were developed such as Libsafe and StackGuard; however, these solutions have a few problems. In this paper we present a new stack buffer overflow attack prevention technique that uses the system call monitoring mechanism and memory address where the system call is made.

키워드
스택, 해킹, 시스템 콜, 보안, 정보보호

I. 서 론

1988년 인터넷 웜 사건[1]을 계기로 알려진 버퍼 오버플로우 공격은 최근까지도 가장 널리 사용되고 있는 해킹 기법중의 하나이다. 버퍼 오버플로우 기법은 프로그램의 흐름을 임의로 변경시키고 원하는 명령을 실행시킬 수 있다는 것이 알려지면서 본격적인 버퍼 오버플로우 해킹 공격의 시대가 도래하게 된다. 특히 1996년에 발표된 "Smashing the stack with fun"[1]이라는 글이 실리면서 초보자들도 버퍼 오버플로우 공격기법을 이해하고 사용하게 되었다. 이러한 버퍼 오버플로우 공격을 방지하기 위해 여러 기법들이 개발되었는데, 대부분 스택 상에서 복귀 주소 변경을 방지하기 위한 여러 기법을 사용한다. 그러나 본 논문에서는 호스트 기반의 침입탐지 시스템에서 사용하는 시스템 콜 모니터링 기법을 적용하여 메모리 상의 스택 영역 상에서 발생하는 버퍼 오버플로우 공격을 방지할 수 있는 기법을 제시한다.

본 논문에서는 2장에서 버퍼 오버플로우 공격기법에 대해 설명하고, 3장에서 관련 연구 및 제품들을 소개하고, 4장에서 시스템 콜 모니터링을 이용한 버퍼 오버플로우 방지기법을 설명한다. 5장에서는 프로토타입을 이용한 시뮬레이션 결과를 설명하고, 마지막으로 6장에서 결론을 내리도록 한다.

II. 버퍼오버플로우 공격 기법

스택 버퍼 오버플로우는 함수의 복귀주소를 변경함으로써 프로그램의 흐름을 해커가 원하는 명령이 수행되도록 변경하는 해킹 기법이다. 버퍼 오

버플로우 해킹 기법이 스택영역을 이용하는 이유는 사용자의 입력이 저장되는 함수 내부의 지역 변수들과 함수의 복귀주소가 스택 영역에 저장되기 때문이다. 함수가 호출되는 경우, 호출된 함수 종료 후 계속적인 프로그램의 수행을 위해 함수 호출 시의 환경을 스택 상에 저장하게 되는데 이때 복귀주소, 프레임 포인터, 스택 포인터, 함수 호출 시 사용된 각종 인자, 호출된 함수에서 사용하는 지역변수 등이 저장된다. 이때, 함수 호출 시 함수의 복귀 주소가 일반 지역변수보다 먼저 저장되고, 스택 영역은 LIFO 형태로 운영되므로 특정 함수의 지역변수는 복귀주소가 저장되어 있는 위치보다 상대적으로 낮은 메모리 주소에 저장되게 된다. 또한 스택 영역의 확장 방향과 지역 변수의 확장 방향이 서로 반대 방향이기 때문에 지역변수를 오버플로우 시키게 되면 복귀주소를 덮어 쓸 수 있게 된다 [1].

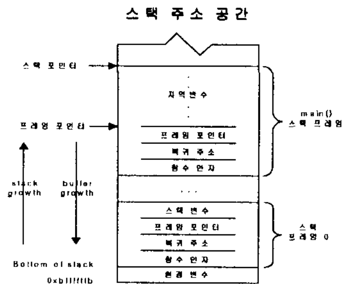

그림 4 : 스택 영역

버퍼 오버플로우로 인해 복귀주소가 변경되는 경우, 대부분 "Segmentation Fault", 혹은 "Bus error"와 같은 메시지를 출력하고 프로그램이 종료된다. 그러나 기계어 형태의 적절한 수행코드를 지역 변수에 저장하고 복귀주소의 위치에 입력한 수행코드의 메모리 주소를 입력하면 에러 메시지 없이 해당 수행코드를 실행시킬 수 있게 된다.

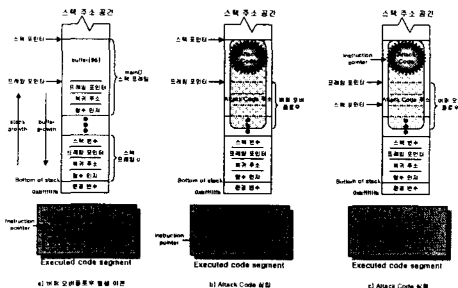

그림 5 : 버퍼 오버플로우를 이용한 프로그램 흐름 변경 과정

그림 2의 a)는 버퍼 오버플로우가 일어나기 전의 스택 영역과 프로그램 흐름을 보여주고 있다. b)는 strcpy() 함수를 통해 버퍼 오버플로우가 일어난 상황의 스택 영역을 보여주고 있는데, 이때 96바이트 크기로 이루어진 buffer[96]에 128바이트 문자열 large_string[128]이 덮어 씌어지게 되기 때문에 오버플로우가 일어나게 된다. 아직까지 프로그램의 흐름은 변경되지 않았으나, 함수의 복귀 명령인 return; 명령을 수행할 때 참조해야할 복귀주소가 Attack Code가 저장되어 있는 위치로 변경되었기 때문에 c)와 같이 Instruction Pointer는 Attack Code로 이동하게 된다.

III. 관련 연구

1. StackGuard[4]

StackGuard는 Immunix Project[5]의 일환으로 개발된 일종의 컴파일러 확장 기능으로서, 스택 상에서 발생하는 버퍼 오버플로우 공격을 통한 복귀주소 변경을 방지한다. StackGuard는 2가지 방법을 이용하여 복귀주소를 보호한다. 첫 번째 방법으로 기존 컴파일러를 패치 시켜 프로그램 소스를 컴파일할 때 카나리(canary)라는 특정 워드를 호출되는 함수의 복귀주소 바로 앞(낮은 주소)에 삽입하여 프로그램이 실행되고 복귀되는 시점에 해당 카나리 워드의 변경 여부를 확인한다. 확인 결과 카나리 워드가 변경되지 않은 경우에만 함수의 복귀를 실시한다. 이는 스택상의 버퍼를 오버플로우 시켜 복귀주소를 변경하는 경우 카나리 워드 역시 변경될 것이라는 가정 하에 진행된다. 두 번째 방법은 MemGuard[6]라는 디버깅 도구를 적용하여 복귀주소가 저장된 워드에 쓰기 방지를 수행하여 변경이 불가능하도록 만든다. MemGuard라는 것은 "quasi-invariant"[3]라고 불리는 개념을 이용하는데, 이는 가상 메모리 표시 방법을 이용하여 일정 기간동안 특정 메모리 페이지를 읽기 전용으로 설정함으로써 쓰기를 금지시키는 기법이다.

2. Libsafe 및 Libverify[7]

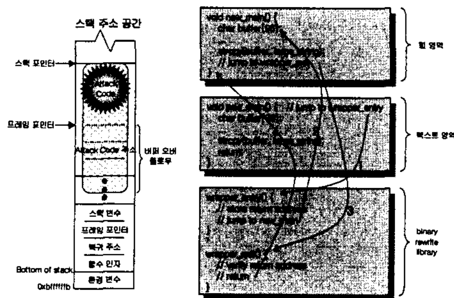

그림 6 : Libverify 동작 방식

Libsafe는 bell-labs에서 개발한 버퍼 오버플로우 공격 방지 기법으로 StackGuard와는 달리 소스 코드가 필요하지 않다. 이 기법은 버퍼 오버플로우

취약점을 가진 것으로 알려져 있는 함수들이 호출되는 경우 Libsafe가 제공하는 취약한 함수와 동일한 이름의 함수를 사용하여 버퍼 오버플로우 취약점을 극복한다.

Libverify[7]는 StackGuard와 비슷한 기법을 사용하는데, StackGuard에서 사용하는 카나리 워드를 Libverify에서는 복귀주소를 사용한다. 즉, 복귀주소 자체의 변경여부를 확인하여 복귀주소가 변경되지 않은 경우에만 정상적인 함수 복귀가 진행된다.

3. Non-executable Stack

Non-executable Stack[8]이라는 것은 말 그대로 스택 영역에서 프로그램이 수행되지 못하도록 커널을 패치 시키는 방법이다. 거의 대부분의 버퍼 오버플로우 해킹 기법은 스택 영역 상에서 수행되므로 메모리 상의 스택 영역 자체를 실행 불가능하도록 만들게 되면 대부분의 버퍼 오버플로우 공격을 방지할 수 있게 된다. 그러나 스택 영역 자체를 실행 불가능하도록 만드는 경우에는 executable stack을 이용하는 몇몇 프로그램들이 정상적으로 수행되지 못하게 된다.

IV. 시스템 콜 모니터링을 이용한 버퍼오버플로우 방지기법

1. 버퍼 오버플로우 발생 위치

2장에서 살펴본 대로 버퍼 오버플로우 공격은 스택 상에 쉘 코드를 입력하고 함수의 복귀주소를 입력한 쉘 코드의 주소로 변경시켜 입력한 쉘 코드를 수행하게 된다. 이때 사용되는 쉘 코드는 스택 영역 상에 위치하게 되므로, 공격자에 의해 입력된 쉘 코드에서 호출하는 모든 시스템 콜은 스택 영역 상에서 발생하게 된다. 따라서 호출되는 시스템 콜의 메모리 주소가 스택 영역이고, 일반적인 호스트 기반 침입탐지 시스템에서 정의하여 사용하고 있는 해킹으로 판단되는 각종 명령들인 경우 스택 오버플로우가 발생한 것으로 판단할 수 있다.

2. 버퍼 오버플로우 방지를 위한 시스템 구성

본 논문에서 제안하는 시스템 콜 모니터링을 이용한 버퍼 오버플로우 방지 시스템은 취약하다고 판단되어 감시하여야 하는 프로그램 목록 P와 목록 P에 해당되는 프로그램의 메모리 영역 그리고 해킹이라고 판단되는 시스템 콜 목록 S를 이용하여 공격 결정 클래스를 구성하고, 구성된 클래스와 시스템 콜 탐지 엔진 구조를 통합하여 시스템 콜 모니터링 엔진을 구성한다.

프로그램 목록 P는 해당 시스템을 관리하는 시스템 관리자에 의해 결정되는데, 이 목록에 포함되는 프로그램으로는 취약점을 가지고 있을 것으로 판단되는 일반 프로그램과 외부 네트워크로부터 패킷을 받아들이고 각종 서비스를 제공하는 데몬 프로그램들 등이 포함된다.

프로그램 목록 P에 해당되는 프로그램들이 사용하는 메모리 영역 M은 각 시스템에 따라 변경될 수 있으나 본 논문에서 실험한 Linux 시스템에서는 쉽게 확인할 수 있었다. 시스템 콜 목록 S에는 해킹 여부를 판단할 수 있는 여러 시스템 콜들이 포함되게 된다[2].

3. 동작 시나리오

목록 P에 의해 지정된 프로그램에서 시스템 콜이 발생하게 되면 시스템 콜은 시스템 콜 해석기로 전송되고, 시스템 콜 해석기는 시스템 콜 모니터링 엔진으로 수신된 시스템 콜 정보를 전송한다. 시스템 콜 모니터링 엔진은 수신된 시스템 콜을 이미 정의되어 있는 P, M, S를 이용하여 해킹 여부를 판단한다. 이때 호출된 시스템 콜이 정상적인 시스템 콜로 판단되는 경우에는 추가적인 작업이 진행되지 않고, 요청된 시스템 콜의 진행에 아무런 영향을 미치지 않는다. 만약 해킹으로 판단되는 시스템 콜이 발생하는 경우에는 해당 시스템 콜을 요청한 프로세스를 정지시키고 로그 기록을 남기며 관리자에게 경보 메시지를 전송한다.

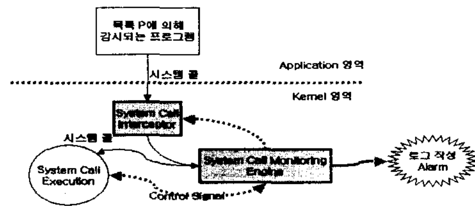

그림 7 : 시스템 동작 시나리오

4. 시스템 특징

본 시스템은 스택 상에서 수행되는 기존의 동작, 즉 몇몇 시스템에서 사용하는 signal handling, 컴파일 등을 정상적으로 수행시킬 수 있고, 스택 영역 상에서 발생하는 시스템 호출만을 대상으로 하기 때문에 Fault Positive, Fault Negative가 발생할 확률이 매우 적으며, 기존의 시스템 콜 모니터링을 이용한 침입탐지 시스템에서 필요한 시스템 콜의 지속적인 관리가 필요가 없으며 취약하다고 판단되는 프로그램의 소스 코드가 필요하지 않은 장점이 있다. 본 시스템은 알려지지 않은 해킹 패턴에 대한 탐지에도 활용될 수 있을 것으로 판단된다. 본 시스템의 단점으로는 시스템 커널의 변경이 필요하다는 것이 있다.

IV. 프로토타입을 이용한 시뮬레이션

본 논문에서 제작한 프로토타입은 기존에 제공되고 있는 각종 시스템 파일 및 프로그램들을 이용하여 시스템 콜이 발생하는 메모리 주소를 확인하여 버퍼 오버플로우가 발생한 프로세스를 정지시

키는 프로그램이다. 주로 사용된 프로그램 및 파일은 strace 명령과 /proc/<process id>/maps 파일이다.

1. 프로세스 메모리 사용 영역 확인

실행중인 프로세스의 maps 파일을 살펴보면 각 프로세스들이 사용하는 메모리의 영역과 그 위치를 확인할 수 있다. 그림 5는 리눅스 시스템에서 동작중인 X윈도우 파일서버의 메모리 영역을 확인한 것으로 스택영역의 위치는 0xbfffc000 - 0xc0000000입을 확인할 수 있다.

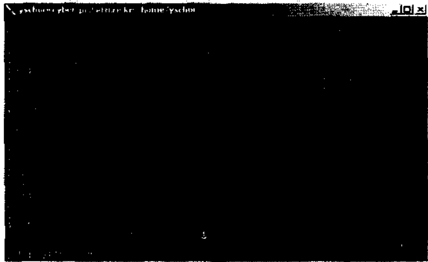

그림 8 : 프로세스 메모리 영역 확인

2. 호출되는 시스템 콜의 주소 확인

그림 6은 해킹 시도 시 호출되는 시스템 콜들을 확인하는 것으로 "/bin/sh" 명령이 다른 명령들과는 달리 스택영역(메모리 주소 0xbffffa52)에서 실행되는 것을 볼 수 있다.

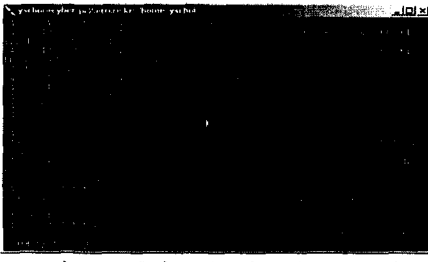

그림 9 : 호출되는 시스템 콜의 주소 확인

3. 프로세스 정지

간단한 프로토타입을 이용하여 발생하는 시스템콜을 입력받아 스택 영역 상에서 발생하는 시스템콜을 찾고, 해당 명령을 출력해 보았다. 그림 7에서 스택 상에서 수행된 명령을 정지시키고 해당 명령을 출력한 것을 볼 수 있다.

그림 10 : 프로세스 메모리 영역 확인

V. 결 론

본 논문에서는 버퍼 오버플로우 해킹 기법을 방지하기 위한 새로운 기법을 제안하였다. 본 기법은 기존의 여러 버퍼 오버플로우 해킹 기법의 문제점인 프로그램 소스를 요구하지 않으면서, 일반 침입탐지 시스템에서 탐지 할 수 없는 공격을 방어할 수 있다.

본 해킹 방지 시스템의 가장 중요한 부분은 해킹을 판단하기 위해 사용하는 시스템 콜 목록 S이다. 시스템 콜 목록 S를 얼마나 정확히 정의할 수 있는가가 시스템 전체의 성능을 좌우하게 된다. 따라서 이 부분은 해킹 기법에 대한 풍부한 지식과 실제 해킹 경험이 있는 해커들로부터 얻어내야 한다. 또한 각 시스템에서 사용하는 메모리 영역의 정확한 위치 계산 및 추출을 위해 추가적인 연구가 필요하다. 본 논문에서는 Intel x86 CPU를 사용하는 RedHat Linux를 대상으로 테스트 하였으며, Linux 시스템의 경우 각 프로세스들에 대한 메모리 영역을 maps 파일을 통해 쉽게 확인할 수 있어, 스택영역의 위치를 정확히 얻어낼 수 있었다.

참고문헌

[1] Aleph One, "Smashing The Stack For Fun And Profit", Phrack 49th Ed. File 14th of 16, Phrack.org, Nov. 1996

[2] Rebecca Gurley Bace, Intrusion Detection , Macmillan Technical Publishing, 2000

[3] Crispin Cowan, Tito Autrey, Charles Krasic, Cal-ton Pu, and Jonathan Walpole. Fast Concurrent Dynamic Linking for an Adaptive Operating System. In International Conference on Configurable Distributed Systems (ICCD S96), Annapolis, MD, May 1996.

[4] Immunix project, http://immunix.org/

[5] Qian Zhang, "The Synthetix MemGuard Kernel Programmer's Interface", June 1997

[6] Bulba, Kil3r, "Bypassing Stackguard and Stackshield", Phrack

[7] Arash Baratloo, Timothy Tsai, and Navjot Singh, "Transparent Run-Time Defense Against Stack Smashing Attacks", Proceedings of the USENIX Annual Technical Conference, June 2000.

[8] Solar Designer, "Non-Executable User Stack." http://www.false.com/security/lin ux -stack/.