시그너쳐 매칭에 기반한 버퍼넘침 공격 탐지의 한계 및 대응

김성수° 위규범
아주대학교 정보통신전문대학원
{pandergi°, kbwee}@ajou.ac.kr

Limits and Countermeasures on Buffer Overflow Attack Detection Based on Signature Matching

Sungsu Kim° Kyubum Wee
Graduate School of Information and Communication, Ajou University

요 약

C언어는 포인터형 변수를 제공하며 배열의 경계를 인식하지 않는다. 이러한 특성에서 기인한 버퍼넘침(buffer overflow)은 널리 알려진 취약점으로서 보안침해 수단으로 널리 악용되고 있다. 이 문제를 해결하기 위한 한 방법으로 오용탐지기술은 버퍼넘침에 공통적으로 사용되는 시그너쳐(Signature)를 가지고 클라이언트(client)가 전송한 패킷을 검사함으로서 고전적인 버퍼넘침을 탐지하고 있다. 본 논문에서는 이러한 탐지 방법을 우회할 수 있는 보다 위협적이고 지능적인 보안침해 가능성을 제시한다.

1. 서 론

인터넷 사용의 증가와 함께 국내 인터넷 사용자도 현재 2000여만 명을 넘어서고 있다. 이러한 인터넷 사용 환경의 증가는 정보화의 촉진이라는 긍적적인 측면뒤에 흔히 말하는 해킹, 바이러스 같은 역기능적인 측면을 내포하고 있어 현재 큰 사회적 문제로 부각되고 있다. 특히 시스템 내부에 대한 공격은 대부분 버퍼넘침을 이용한 공격이 주류를 이루고 있다. 이 문제를 해결하기 위해 많은 방법이 제시되고 상용화 되고 있지만 완벽한 예방과 탐지는 아직까지 어려운 실정이다. 그중 시그너쳐 매칭(signature matching)기술은 공격이 성과를 거두기 전에 탐지하고 제어할 수 있어 고전적인 버퍼넘침 공격을 무력화 시키고 있다 [1]. 본 연구에서는 시그너쳐 매칭에 기반한 탐지를 우회하여 공격할 수 있는 기법을 제시한다.
제 2장에서는 버퍼넘침을 설명하고 이것의 탐지 기법에 대해 분석한다. 3장에서는 시그너쳐 매칭을 우회할 수 있는 보안 침해 수법을 실증적 시나리오와 함께 제시한다. 마지막 4장에서는 결론 및 향후 연구 방향을 제시한다.

2. 관련연구

2.1 버퍼넘침공격

[그림 1] 스택 넘침을 위한 공격코드

버퍼넘침 기법은 다음의 세가지 특성을 이용한다. 첫째, C언어는 배열의 경계를 검사하지 않는다. 둘째, 이 기법은 운영체제의 시스템 호출특성을 이용하고, 마지막으로 함수의 호출과정이 관리되는 스택프레임(stack frame)을 조작한다. 공격과정은 세단계로 진행된다. 첫째, 루트 소유의 setuid가 적용된 프로그램(공격대상프로그램)을 실행시킨다. 둘째, 대상 프로그램이 입력되는 데이터의 크기를 검사하지 않는다면 쉘코드(shellcode : 쉘을 실행시킬 수 있는 기계어 코드)를 실행시키도록 복귀주소(return adderss)를 바꾼다. 마지막으로, 함수가 복귀 시 쉘코드가 실행된다. 그러나 변경된 복귀주소가 정확히 쉘코드를 가르킨다는 보장을 하기 어렵기 때문에 성공확률을 높이기 위해 쉘코드앞에 많은 수의 NOP코드를 삽입하여 성공확률을 높인다. 공격대상 프로그램에 삽입되는 최종적인 공격코드(attack code: NOP+쉘코드+복귀주소)는 [그림 1]과 같은 형태가 된다.

2.2 버퍼넘침을 탐지하기 위한 기법들

버퍼넘침을 성공적으로 수행하기 위해 대개 700byte정도의 NOP코드와 80byte정도의 shellcode, 그리고 200byte정도의 새로운 복귀 주소가 필요하다. 즉 공격을 성공하기 위한 공격코드는 약 1000byte정도가 필요하다는 것이다. 상용하는 N-IDS는 공격자가 성공확률을 높이기 위해 많은 NOP코드가 필요하다는 것과, 쉘코드에 "/bin/sh"문자열이 필요하다는 것을 이용하여 버퍼넘침 공격을 탐지하고 있다 [2].
버퍼넘침을 탐지하고 방지하기 위한 여러 가지의 방법들이 있다 하지만 본 논문에서는 네트웍 기반의 탐지방법에 대해 언급하고자 한다. 시그너쳐 매칭기법을 사용하는 N-IDS(예를 들면 snort같은)는 버퍼넘침을 탐지하기 위해 네크웍 트레픽을 감시하고, 패킷을 검사한다 [2]. 알려진 공격에 대한 패턴을 가지고 패킷을 검사하고, 쉘코드에 필요한 "/bin/sh"문자열이 있는지를 검사하여 침입을 판단하게 된다. 그러나 재배치(reordering)나 삽입(insertion)과 같은 기술로 공격코드를 변형한다면 오용탐지기법으로는 변형된 공격코드를 탐지할 수 없는 문제점이 있다. 게다가 어떤 침입자들은 공격코드를 암호화 하기에 더욱더 탐지가 어렵다. 이런 문제점을 해결하기 위해 sledge(연속된 NOP코드 또는 연속된 NOP대치코드)와,

MEL(Maximum Execution Length)기법을 이용하여 암호화되고 변형된 형태의 버퍼넘침을 탐지할 수 있는 방법이 제시되었다 [3]. 이 방법은 sledge탐지 방법이란 명칭으로 본 논문에서 도용한다. 위에서 언급한 시그너쳐 매칭기법과 sledge탐지 방법을 조합하면 네트웍상에서 버퍼넘침을 성공하기는 매우 어렵다.

3. 우회방법

고전적인 시그너쳐 매칭기법과 sledge탐지 방법을 우회하기 위해서는 다음의 조건을 만족해야 한다. 첫째, sledge는 매우 작은 크기이거나 없어야 한다. 또한 0x90이 아닌 다른 코드로 대치되어야 한다. 둘째, 쉘코드는 매우 작은 크기를 가지고 있어야 하며 암호화 되어야 한다. 물론 복호화 엔진도 가지고 있어야 한다. 마지막으로, 쉘코드를 가리키는 복귀주소는 정확한 쉘코드의 주소를 알아야 하며, 알지 못한다 해도 오차가 매우 적어야 한다. 이와 같은 조건을 만족하기 위해 본 논문에서는 세가지 방법을 사용하고 이 방법을 조합하여 새로운 형태의 보안침해 방법을 제시하고자 한다.

3.1 다형성 쉘코드(Polymorphic Shellcode)

2001년 4월에 K2가 시그니쳐매칭기법을 우회하기 위해 ADMutate를 발표했다 [4]. 이 방법은 버퍼넘침을 위해 사용되는 공격코드의 형태를 바꿈으로서 탐지를 우회할 수 있다. 이 기법은 다형성(polymorphism)에 기본 이론을 둔다. 다형성은 다양한 형태로 존재할 수 있는 능력이다. 이것은 같은 함수를 수행하는데 다양한 방법이 있을 수 있다는 것이다. 즉 공격코드 전체를 같은 기능을 수행하되 다른 형태로 만들어 탐지를 피할수 있다는 것이다. 공격코드를 대치하거나 암호화 하기위해서는 크게 3부분을 바꾸어야 한다. 첫째, sledge부분이다. Intel IA32환경에서 NOP(0x90)은 50개 이상의 코드로 대치될 수 있다. 대치될 수 있는 코드의 일부 예는 아래의 표와 같다.

<표 1> IA32에 따른 단일 바이트 NOP대체코드

mnemonic	Explanation	Opcode
AAA	ASCII Adjust After Addition	0x37
AAS	ASCII Adjust After Subtraction	0x3f
CWDE	Convert Word To Doubleword	0x98
CLC	Clear Carry Flag	0xf8
...	...	...

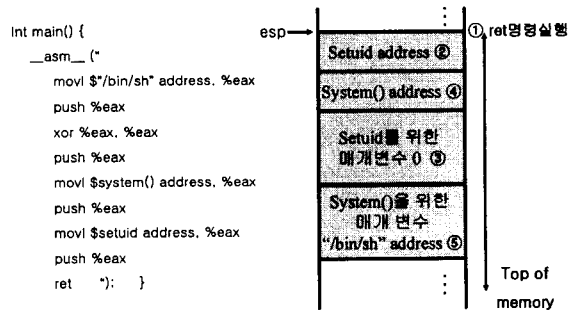
<표 2> IA32에 따른 다중 바이트 NOP대체코드

Instruction	Bytecode
adc $0x70, %cl	80 d1 70
adc $0x70d18070, %eax	81 d1 70 80 d1 70
and $0x55120125, %eax	25 01 12 44
...	...

둘째, 쉘코드 부분은 임의적으로 생성된 (srand())키를 XOR연산을 통해서 암호화 한다. 물론 암호화 한 쉘코드는 복호화 코드가 필요하며 복호화 코드는 암호화된 쉘코드의 앞에 위치하게 된다. 그러면 쉘코드에 포함된 "/bin/sh"문자열로 탐지하는 IDS를 무력화 시킬 수 있다. 이 기법은 sledge와 쉘코드의 시그너쳐를 가지고 탐지하는 오용기법을 무력화 시킬 수 있으나, sledge탐지 방법은 피할 수가 없다 왜냐하면 코드를 변형하고 암호화 해도 전체 공격코드의 길이를 줄일 수는 없기 때문이다.

3.2 return-into-libc기법

이 기법은 다음의 네가지 사실에 기본 개념을 둔다. 첫째, 대부분의 시스템이 공유라이브러리의 경우 공격대상 프로그램이나 공격프로그램이나 모두 system()과 setuid()의 주소가 동일하게 나타난다 [5]. 그리고 그 주소를 알아낼수가 있다. 둘째, ret instruction이다. ret은 현제 esp레지스터가 가르키는 곳에서 값을 꺼내서 eip값으로 설정한다. 만일 esp레지스터가 가르키는 곳의 값이 0xbfffff4a라면 ret실행시 eip값이 0xbfffff4a로 설정된다. 그렇다면 eip값에 따라 0xbfffff4a주소의 instruction이 실행된다. 만일 esp레지스터가 가르키는 곳이 system()의 주소라면 system()함수가 실행된다. 셋째, ret에 의해 호출되어지는 system()과 call에 의해 호출되어 지는 system()과는 차이가 있다. call은 호출될 때 복귀주소를 삽입하고 ret은 그렇지 않다. 따라서 ret을 통해 호출됨으로서 복귀주소가 스택에 없는 상황에서 system()함수 내부에서 ret명령이 수행되면 이전에 esp레지스터가 가르키던 곳에서 +4byte공간이 다음수행할 복귀주소가 된다. 마지막으로, 함수에 필요한 매개변수는 함수가 실행된 주소에서 +4byte떨어진 곳에서 매개변수값을 찾느다. 이와 같은 사실을 바탕으로 [그림 2]와 같은 형태의 쉘코드를 작성할 수 있다.



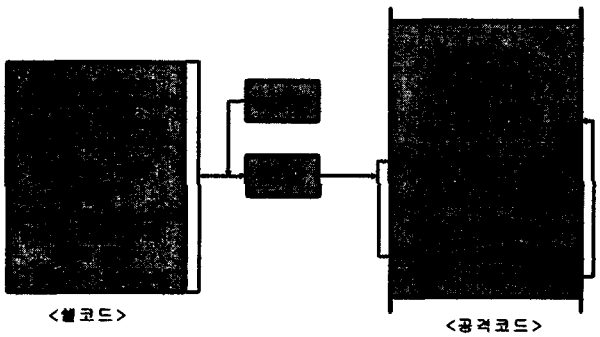
[그림 2] setuid(0),system("/bin/sh")을 실행할수 있는 쉘코드

①번에 ret명령이 실행되면 복귀주소를 스택에 삽입하지 않고 ②setuid()가 실행되고 실행이 끝난 주소에서 +4byte떨어진 ③setuid()의 매개변수를 입력받는다. setuid() 실행 후 복귀주소가 저장되지 않았으므로 esp레지스터가 가르키던 곳에서 +4byte에 위치한 ④system()으로 복귀한다. ⑤물론 같은 원리로 system()에 필요한 매개변수를 입력받는다. 그러면 결과적으로 setuid(0)와 system("/bin/sh")이 동시에 실행되는 결과를 가져온다. 이 기법으로 쉘코드를 23byte로 줄일 수 있다.

3.3 다형성 쉘코드 + return-into-libc + 복귀주소 산출

쉘코드를 가르키는 복귀주소는 다음 두 가지 가능성이 있다. 첫째, 공격대상 프로그램이 디버그 루틴(debug routine)을 포함하고 있을 경우이다. 이 경우에는 디버그를 통해 정확한 스택주소(stack address)를 계산해 낼 수 있다. 그렇기 때문에 sledge가 없는 공격코드를 생성해 낼 수가 있다. 둘째, 디버그 루틴이 없다면 계산을 통해 산출해 내야 한다. 공격코드는 sledge+쉘코드+쉘코드를 가르키는 복귀주소의 형태로 존재한다. 이것은 각각의 코드가 모두 붙어있다는 것이다. 쉘코드의

길이는 차이가 있겠지만 30byte를 넘어가지 않게 만들 수 있다. 즉 복귀주소와 쉘코드와 얼마 떨어지지 않았음을 알 수 있다. 따라서 상대 번지 값을 복귀주소에 계산해서 삽입함으로써 적은 바이트의 길이를 가지는 sledge를 가지고도 충분히 공격을 성공할 수 있는 가능성이 있다. 세가지 조건을 만족하는 공격코드의 형태는 아래 [그림 3]과 같은 형태가 된다.



[그림3] 최종공격코드

위의 그림은 최종적인 공격코드의 형태이다. IDS를 우회하기 위해 최종적으로 다음의 과정을 따른다. 첫째, setuid(0)와 system("/bin/sh")를 실행시킬수 있는 쉘코드를 작성한다. 작성한 쉘코드는 30byte를 초과하지 않는 크기이며 이 코드를 가지고 srand()함수로 랜덤하게 생성된 32bit의 키(key)를 가지고 XOR연산을 통해서 암호화된 쉘코드를 생성한다. 둘째, 암호화된 쉘코드를 복호화 할 수 있는 코드를 작성한다. 이 복호화 코드는 생성된 키 값을 바탕으로 역연산을 통해 원래의 쉘코드로 복원하는 기능을 한다. 이렇게 생성된 코드는 "/bin/sh"문자열을 은폐할 수 있으며 toupper()나 tolower()와 같은 필터링에 유동성 있게 반응할 수 있다. 셋째, NOP코드를 대치 할 수 있는 코드를 삽입한다. 물론 sledge는 매우 작고 없을 수도 있다. 마지막으로 복귀주소를 계산해 공격을 시도한다.

4. 결론

본 논문에서는 N-IDS의 시그너쳐 매칭기법과 sledge탐지 방법을 우회할수 있는 보안침해방법을 제시하고 심층적 분석을 통해 보다 위협적이고 지능적인 보안 침해 가능성을 제시하였다. 이러한 기법들은 더욱 다양한 조합을 통해 시스템을 공격할 수 있으며 점점 바이러스 형태의 단계적이고 지속적인 침해 형태로 발전하고 있다. 본 연구를 바탕으로 이러한 형태에 버퍼넘침을 미연에 방지하고, 탐지할 수 있는 대응방안에 대한 연구가 이루어 질수 있다.

본 연구를 바탕으로 고전적인 버퍼넘침과 이것의 변형된 형태의 침해방법에 대해 실시간으로 탐지하고 예방할 수 있는 기법을 연구중에 있다. Sledge탐지 방법에 기본을 두고 있으며 변형된 형태의 다른 버퍼넘침 기법도 적용될 수 있는 패턴(pattern)를 선택해 추가할 계획이다. 또한 sledge탐지 방법에서 버퍼넘침이라 판단하는 기준이 되는 임계값(threshold)에 변화를 주어 실험하여 그에 따른 공정 오류(false positive)측면에서의 최적화된 값을 찾는 연구가 진행중이다.

5. 참고문헌

[1] AlephOne, "Smashing the stack for fun and profit", *Phrack Magazine*, 49(14), 1996.

[2] Snort team, *http://www.snort.org*.

[3] Thomas Toth, Christopher Kruegel, "Accurate Buffer Overflow Detection via Abstract Payload Execution", *RAID2002 Symposium*, p274-291, 2002.

[4] Home of K2, *http://www.ktwo.ca*

[5] Lamagra, "The OMEGA Project finished", *Corezine Magazine*, vol. 3, 1999.

[6] C. Cowan, C. Pu, D. Maier, H. Hinton, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang, "Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks", *http://www.immunix.org/documentation.html*, 1998.

[7] Intel, "IA-32 Intel Architecture Software Developer's Manual Volume 1-3", 2002.

[8] SecurityFocus Corporate Site, *http://www.securityfocus.com*