해킹의 유형과 대처 방법

조영현 reca@snags.org 김진욱 rein@snags.org

October 27, 2002

Contents

1 해킹과 보안

해킹과 보안은 창과 방패에 자주 비유된다. 해킹(hacking)은 보통 시스템이 무단으로 침입하는 행위를 일컫고, 보안은 시스템을 외부나 내부의 공격으로부터 안전하게 유지되도록 하는 것을 말한다. 이 문서의 목적은 시스템 안전하게 돌아가는 도움을 주기 위해서, 즉 보안에 대해서 도움을 주려고 작성된 것이다. 여기서는 해킹에 대해서 하나씩 알아보면서 그에 대한 해결책을 알아볼 것이다. 즉 손자병법에 나온 知彼知己 百戰不敗(지피지기 백전불패)의 정신을 보안에 적용하려는 것이다. 그러나 해킹기법 소개를 통해서 해커를 양성하고자 하는 것은 아니므로 자세한 구현은 생략하고, 그 원리와 대책에 집중할 것이다.

2 Hacking, Hacker

해커(hacker)는 일반적으로 해킹을 하는 사람을 일컷는다. 요즘은 흔히들 해커를 해커와 크래커(cracker)를 구분하는 경향이 있다. 이럴 때 해커는 보통 '그 능력이 뛰어난 프로그래머나 시스템전문가'를 뜻하고, 크래커는 이에 대비되게 '악의를 품고 시스템에 접근하거나 프로그램을 개조하는 사람'을 뜻하게 된다. 이 문서에서는 이렇게 나누어서 부르지 않고, 위에서의 크래커를 해커라고 칭해서 부르겠다. 그리고 해커의 일부중에 스크립트 키디(Script Kiddie)라고 불리는 이들도 있다. 이 사람들은 쉽게 시스템에 침입하려는 특징이 있다. 그래서 주로 다른 해커들이 만들어 놓은 툴을 이용해서 해킹을 한다. 스크립트 키디가 사실상 현재 시스템에 가장 큰 피해를 주고 있다. 해커들 중에 스크립트 키디가 절대 다수를 차지하는 것이 이유이다.[1]

3 해킹의 종류

해킹의 종류는 크게 해커가 어느 위치에서 해킹을 시도하는지에 따라서 나눌 수 있다. 시스템에 이미 접근할 권한을 가지고 다른 권한(root)를 가지려고 하거나 시스템을 피해를 주는 경우와 시스템 외부에서 네트워크를 통해서 시스템에 무허가의 접근 권한을 가지는 경우이다. 여기서는 기술적 해킹이외에 Social Engineering이라는 사람을 속여서 접근권한을 알아내는 기법도 소개할 것이다.

4 Social Engineering

4.1 Social Engineering이란?

다른 해킹 기법과는 달리 Social Engineering은 특별히 기술적 지식이 필요하지 않다.해커의 대부인 케빈 미트닉(Kevin Mitnick)도 Social Engineering의 위험함에 대해서 언급한 적이 있다. Social Engineering의 핵심은 시스템에의 중요한 위치에 있는 사람(예: superuser)를 속이는 것이다. 시스템 관리자에게 전화나 e-mail으로 root가 필요하다고 할때, 이럴때 root계정을 가르쳐주는 것이 이러한 것에 속한다.

4.2 해결책

별로 일어날 것 같지 않지만, root의 중요성을 모르는 관리자들이 많아서 예

[1] 이러한 Hacking Tool은 조금만 노력하면 쉽게 얻을 수 있기 때문에, Script Kiddie가 되기는 쉽다. 그렇지만 해커들사이에서는 실력의 부재로 제대로 취급받지 못한다.

상외로 쉽게 일어날 수 있다. 믿을 수 있는 사람이더라도 root계정이 필요하다면 직접 권한을 주지 말고 sudo로 간접적으로 주는 방법이 필요하다. 이렇게 하면, 권한의 일정부분을 제한 할 수 있다.시스템 관리자도 root권한이 필요할 때, 직접 root계정을 사용하지 않고 sudo를 사용하면 좋다. 대부분의 시스템에는 sudo가 기본적으로 설치되어 있지만, 그렇지 않다면 각 벤더(vendor)별 패키지[2]를 설치하거나 에서 소스[3]를 받아서 설치하면 된다.

5 Buffer Overflow

지난 몇년 간의 보안 사고 중에서 상당 부분을 차지하는 것 중에 buffer overflow 공격이 있다. 만약 cert.org나 securityfocus.org에서 *"overflow"*로 검색해보면 수많은 보안 취약점들이 검색되는 것을 볼 수 있을 것이다.

5.1 Buffer overflow란?

5.1.1 Computer Memory Organization

어떤 프로세스가 실행되었을 때 Computer의 virtual memory에는 몇 개의 구역이 메모리에 잡히게 된다.

Argv pointers	higher addresses
Argc	
Stack Data Area	
Heap Data Area	
Static Data Area (Uninitialized)	
Static Data Area (Initialized)	
Text(instructions)	lower addresses

5.1.2 Buffer overflow는 어디에서 일어나는가

문제가 되고 있는 buffer overflow를 일으키는 곳은 주로 stack과 heap 영역이다. C언어에서 어떤 배열을 잡거나 동적으로 할당된 메모리 공간에 그 공간의 크기보다 더 큰 것을 적으려고 하면 segmentation fault가 일어나거나, 상황에 따라서는 아무일 없다는 듯이 돌아가기도 한다. 그리고 이것처럼 원래의 buffer(C에서의 배열과 거의 동의어)의 크기보다 더 큰 크기의 공간에 내용을 적는 것을 buffer overflow라 하며, 동적으로 메모리가 할당될 때 사용되는 Heap이나 runtime때 함수 호출 등에 사용되는 stack에서 일어날 수 있다. 아래의 코드는 buffer overflow의 간단한 예이다.

```
#include<stdio.h>
#include<string.h>
void main()
{
    char* str = "string longer than buffer";
    char buf[10];
    char buf2[10];
```

[2]Solaris : http://www.sunfreeware.com, Redhat : http://www.rpmfind.net에서 구할 수 있다.

[3]ftp://ftp.sudo.ws/pub/sudo에서 구할 수 있다.

```
        strcpy(buf2, str);
        printf(buf);
    }
```

위 코드가 실행되고 나면 화면에 *r than buffer*라는 문장이 출력된다. 즉 stack에 buf와 buf2가 있는데, buf2의 공간을 다 쓰고도 더 복사가 이루어지기 때문에, 경계 밖으로 들어간 부분이 buf에 들어가버린 것이다[4]

5.2 Buffer overflow는 왜 문제인가?

5.2.1 Stack overflow

고급 프로그래밍 언어에는 함수의 개념이 있고, 이 함수가 호출될 때, 다음과 같은 과정이 이루어지게 된다. 우선 함수에 전달될 매개 변수가 stack에 push[5]되고, 함수가 종료되고 나서 되돌아올 위치(return address)가 push된다. Stack overflow는 stack에 push된 return address를 조작하는 방법으로 이루어 진다. 우선 suid가 걸려있는 프로그램 찾아서(즉 root 권한으로 실행 되는 프로그램), stack overflow를 일으킬 수 있는 곳을 찾고, buffer overflow로 이 return address 부분을 원하는 쉘 코드[6] 가 들어있는 곳의 주소로 덮어 씌워서 그 프로세스가 쉘 코드를 실행시키도록 만들면 된다.

그런데 이런 stack overvlow를 일으킬 수 있는 프로그램들은 매우 많다. 우선 buffer overflow를 일으킬 수 있는 함수들이 여러개 있다. 어떤 변수를 복사하는 등이 일을 할 때 buffer의 boundary check를 하지 않는 함수들은 buffer overflow를 일으킬 수 있기 때문에 이런 함수들이 사용된 프로그램은 잠재적인 위험이 있다.

초기에는 쉘 코드를 만드는 것이 복잡하고, Stack영역에서 조작할 return address의 위치를 알아내는 것이 힘들었으나 많은 해커들이 만들어낸 쉘 코드의 소스가 인터넷에 널리 퍼져있고, 내부의 공격자가 있는 경우 쉽게 return address를 추측할 수 있고, 외부에서 공격하는 경우에도 어느 정도의 시행착오 후에는 가능하기 때문에 stack overflow로 인한 각종 보안 문제가 생기고 있다.

5.2.2 Heap overflow

Buffer overflow를 이용하여 heap[7] overflow를 일으킬 수도 있다. Stack overflow는 알려진지 상대적으로 오래되었고 컴파일러 레벨 [8]로, 혹은 StackGuard 같은 툴의 도움으로 어느 정도 안전해 졌다. 그렇지만 Heap을 이용한 buffer overflow 공격은 아직 별다른 대책이 마련되어 있지 않다.

Heap overflow의 공격 목표는 주로 heap영역에 있는 function pointer나 파일 이름, 암호, saved uid등을 덮어 써서 원하는 효과를 얻어내는 것이다. stack overflow의 경우 주로 return address를 조작해서 원하는 코드를 실행하도록 하

[4]좀 생각해보면 출력문이 좀 이상할 것이다. 아마 ge가 왜 안찍혔나 할텐데, 메모리에는 CPU word 단위(대부분 4바이트일 것이다)로 할당이 이루어지기 때문에 실제로는 12바이트가 할당되어 있어서이다

[5]Stack은 push와 pop의 두개의 연산으로 구현되어 있다. push는 원소 하나를 현재까지 사용된 stack 맨 위에 넣고, pop은 stack맨 위의 원소를 하나 꺼낸다

[6]Shell을 띄워주는 코드 : 주로 execve() system call로 /bin/sh를 실행시킨다

[7]Heap은 malloc()이나 free()같은 동적 메모리 할당 및 해제 함수로 조작되는 영역이다

[8]boundary check를 하게 되어있으나 일정 수준 이상 복잡한 것은 compile이 이루어지지 않는다고 한다

는 것이지만 heap overflow의 경우에는 한 buffer를 overflow 시켜서 인접한 혹은 근처의 buffer를 공략하여 function pointer등의 조작을 노린다.

5.3 해결책

5.3.1 Stack overflow

Stack overflow를 막으려면 stack overflow를 위해서 필요한 선결 조건들을 제거해주면 된다.

프로세스가 정상적인 control flow를 따라서 진행되는 경우에는 문제가 없을 것이다. Stack overflow는 정상적인 프로세스의 흐름을 return address등의 조작을 통해 악의성 코드로 돌려놓는 것이다. 이런 return address의 조작을 막을 수 있다면 악의적인 코드가 실행되는 것을 막을 수 있을 것이다. 이러한 기능을 해주는 툴로 StackGuard [2] 같은 툴들이 있다. StackGuard는 stack에 push된 return address의 값이 "변경"되는 것을 감지하고 이런 조작된 return address의 경우 조작된 주소로 돌아가는 것이 아니라 실행을 중단한다. 그 방법은 다음과 같다. 우선 함수가 호출될 때 return address뒤에 Canary value를 넣는다. 이 canary로는 어떤 random값이나 null character, 혹은 CR, LF같은 terminator를 사용하고 함수 호출이 끝난 다음 이 Canary 값을 검사하여 조작이 있을 경우 stack overflow를 시도했다는 것을 알 수 있게 된다. 이 canary value로 사용되는 값들은 예측할 수 없거나, buffer overflow를 일으킬 수 없게하는[9] 값이 사용되기 때문에 아주 약간의 성능 감소가 있긴 하지만 stack overflow를 효과적으로 막아낼 수 있게 된다.

또한 stack overflow를 일으키려면 바꿔놓은 return address로 갔을 때, 그 곳의 코드가 "executable"이어야 한다. 이런 경우는 stack 영역을 "unexecutable"하게 바꿔주면 된다. stack에 있는 코드들을 unexecutable하게 만들어도 실제로 stack에 쌓이는 것들은 return address나 local variable들이기 때문에 문제는 없다. 많은 수의 UNIX 그리고 Linux[10]에서 stack을 unexecutable상태로 만들어주는 패치가 있다.

혹은 source code scanning툴[11]을 이용해서 source code level에서 검사하거나, 프로그램을 작성할 때 boundary check를 하는 library[12]를 사용하여 buffer overflow가 일어나지 않도록 하는 방법도 있다.

5.3.2 Heap overflow

Heap overflow의 경우 stack overflow의 경우와는 달리 heap을 보호해주는 방법[13]이 없다. 따라서 프로그램이 작성될 때 buffer overflow가 일어나지 않는 "secure"한 코드로 작성이 되어야 한다. System 관리자들은 계속적으로 OS에 대한 patch를 해주는 방법이 최선이다. 이것은 끝도 없이 계속되야하는 작업이지만 이 방법으로 buffer overflow뿐만 아니라 다른 보안 문제들도 해결되어질 것이다[14].

[9] 예를 들어서 null chararcter가 canary value로 사용되었다면 strcpy()같은 함수로는 overflow가 일어나지 않을 것이다

[10] Linux의 경우 http://www.openwall.com/에서 찾아 볼 수 있다

[11] memory leak을 검사하는 툴로 유명한 Rational의 purify(plus)는 boundary check가 없는 buffer에 대한 검사도 해준다

[12] 예를 들어서 strcpy()대신에 strncpy(), gets()대신에 fgets()를 사용하는 방법이다

[13] Stack이 경우 StackGuard나 StackShield같은 것이 있다

[14] 물론 완벽히 안전해질 수는 없다

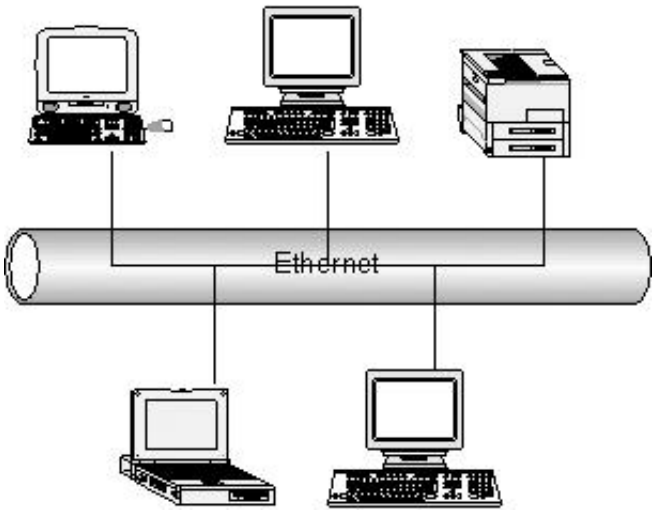

Figure 1: Ethernet 구성도

6 Sniffing

6.1 Sniffing이란?

Sniff라는 단어를 사전에서 찾아보면 그뜻이 '코를 킁킁거리다, 냄새를 맡다'이라는 것을 알 수 있다. 해킹에서의 Sniffing은 냄새를 '맡는' 것이 아니라 네트워크의 패킷을 중간에서 '맡고' 중요한 정보를 알아내려는 것이다. 일종의 도청이라고 할 수 있다. 그리고 스니퍼(Sniffer)는 스니핑에 사용되는 도구이다. 스니핑은 해킹에도 쓰일 수도 있지만, 네트워크 관리자가 네트워크를 관리하기 위한 툴으로 쓰이기도 한다. 그러므로 스니핑은 양날의 칼이라고도 할 수 있겠다. 스니핑은 기본적으로 Ethernet의 성질을 이용하는 것이다. 그러므로 먼저 Ethernet에 대해서 간략하게 알아보도록 하겠다.

6.2 Ethernet소개

Ethernet은 10Mbps에서 100Mbps까지의 속도를 내는 LAN(Local Area Network) 기술을 말한다. LAN을 구현하는 기술은 Ethernet외에 Token-ring, FDDI, ATM 등이 있지만, 가장 널리쓰이는 기술은 Ethernet이다. 기본적으로 공유된 회선을 이용하기 때문에, Ethernet을 통하는 패킷들은 모든 컴퓨터에 전송된다. 이때 각 컴퓨터는 자신에게 온 패킷 중에 자신의 것인지 탐지하고 받아들이게 된다. 자신의 것인지 판별하는 것은 Ethernet Inteface[15]에 고유하게 부여된 MAC(Media Access Control) Address를 이용한다. Ethernet에 연결되어 있는 여러 시스템에서 동시에 패킷을 보낼 때에는 CSMA/CD 프로토콜[16]을 이용하여서 한번에 한 시스템에서만 패킷을 보내도록 유도한다.

[15]Network Interface Card(NIC)의 일종, 일반적으로 LAN card라고 불리는 것

[16]Ethernet의 핵심은 이 CSMA/CD 프로토콜이라고 할 수 있기 때문에, CDMA/CD 프로토콜과 Ethernet과 동일하게 취급할 수도 있다

6.3 Sniffing의 원리와 실제

스니핑은 이러한 Ethernet의 데이터가 모든 컴퓨터에 전송되는 것을 이용하는 것이다. 일반적으로 Ethernet Interface는 자신의 MAC Address가 없는 패킷은 모두 버리지만, "promiscuous mode"에 진입하게 되면 모든 패킷을 다 받아들이게 된다.

```
0000  00 00 0c 07 ac 00 00 a0  c9 32 05 4e 08 00 45 00  ........ .2.N..E.
0010  00 a9 52 89 40 00 40 06  a2 3c 93 2e 6a 24 d3 20  ..R.@.@. .<..j$.
0020  75 16 05 e8 00 50 74 ac  64 52 73 71 a2 ef 80 18  u....Pt. dRsq....
0030  16 d0 39 0c 00 00 01 01  08 0a 00 21 30 89 40 f6  ..9..... ...!0.@.
0040  5a da 43 6f 6e 74 65 6e  74 2d 54 79 70 65 3a 20  Z.Conten t-Type:
0050  61 70 70 6c 69 63 61 74  69 6f 6e 2f 78 2d 77 77  applicat ion/x-ww
0060  77 2d 66 6f 72 6d 2d 75  72 6c 65 6e 63 6f 64 65  w-form-u rlencode
0070  64 0d 0a 43 6f 6e 74 65  6e 74 2d 4c 65 6e 67 74  d..Conte nt-Lengt
0080  68 3a 20 34 36 0d 0a 0d  0a 69 64 3d 74 61 72 67  h: 46... .id=targ
0090  65 74 26 70 77 3d 31 32  33 34 35 36 37 26 78 3d  et&pw=12 34567&x=
00a0  33 36 26 79 3d 37 26 6c  6f 67 69 6e 6d 6f 64 65  36&y=7&l oginmode
00b0  3d 6e 6f 72 6d 61 6c                              =normal
```

위의 예제는 웹브라우저 상에서 특정 사이트에 로그인 하는 것을 "promicuous mode"로 스니핑을 한 예이다. "id=target&pw=1234567" 부분에서 아이디가 "target"이고 패스워드가 "1234567"임을 쉽게 알아낼 수 있다. 이렇듯 쉽게 Sniffing이 되지만 일반적인 Sniffing에도 한계가 있는데, 바로 같은 허브(Hub)상에 위치한 시스템의 패킷에 대해서만 가능하다는 것이다. 즉, 같은 최하위 "공유" 네트워크에서만 가능하다는 뜻이다. 이러한 제약점을 극복한 SNMP sniffing도 존재한다.

6.4 해결책

Sniffer는 다른 해킹툴에 비해서 비교적 손쉽게 구할 수 있기 때문에, 다른 보안상의 위험에 비해서 자주 노출되어 있을 가능성이 높다. 그러기 때문에 이에 대한 대책을 취하는 것이 다른 것에 비해 중요하다고 할 수 있다.

6.4.1 sniffing을 어렵게 하는 방법

일반적인 Sniffing은 네트워크 자원이 공유되어 있다는 상황에서 이루어진다. 이러한 공유 방식을 바꾸면 좀 더 Sniffing을 어렵게 하는 환경으로 바뀔 수 있다. 이더넷을 제어하는 허브를 Repeater Hub(일반적으로 허브Hub를 지칭한다)를 사용하지 않고, Switching Hub(일반적으로 스위치라고 불리는 것)을 사용하면 된다. 이렇게 환경을 바꾸면, 모든 패킷이 모든 컴퓨터에 가지 않고, 스위치가 패킷을 보고 특정 호스트에만 보내게 된다.(Switching이라고 한다) 하지만, 이런 환경에 놓여도 ARP를 이용한 Sniffing이 가능하다.

6.4.2 sniffing을 피하는 방법 : 암호화

Sniffing을 피해서 안전하게 네트워크를 이용하기 위한 가장 좋은 방법은 암호화를 이용해 데이터를 교환하는 것이다. Sniffing을 통해서 취약성이 있는 서비스는 TELNET, HTTP, SNMP, POP, FTP, IMAP 등이 있다. 이러한 서비스들은 대부분 별다른 과정없이 패킷에서 바로 비밀번호를 읽어낼 수 있다. 암호화를 이용해서 데이터통신을 하는 대표적인 툴이 Secure Shell[17]이다. Secure

[17]Secure Shell은 SSH Communications Security에서 만들었지만, 그 폐쇄성으로 다른 Secure Shell 구현들이 존재하는데 Linux,BSD계열에서 쓰이는 OpenSSH이 대표적이다.

Shell은 공개키 교환 방식으로 서버와의 터미널 접속에 사용된다. Telnet을 사용하고 있다면, 서비스를 중지하고 Secure Shell로 바구는 것이 좋다. 그리고 Secure Shell을 이용하면 위에서 열기한 서비스들을 터널링[18]을 이용해 안전하게 사용할 수 있다.
전자 상거래를 할때, 브라우저의 창에 "HTTP://" 대신에 "HTTPS://"를 볼 수 있는 경우가 있다. 이 것이 HTTP를 암호화해서 통신하는 것이다. 이 암호화에는 Secure Socket Layer(SSL)이 이용되는 것이다. SSL을 이용해서 HTTP 데이터를 모두 암호화해서 전송한다면 보안에 가장 좋은 방법이겠지만 이는 속도가 매우 느리므로, 인증과 같은 중요한 부분에만 이용하는 것이 권장된다. SSL은 HTTP를 암호화하는 이외에도 다양한 서비스에 대해서도 암호화를 제공한다.
E-mail은 자주 쓰인다는 점에 비해서 평문(암호화된지 않은 문자열)으로 보내지므로 취약성을 가지고 있다. 그러므로 중요한 E-mail의 경우에는 공개키 암호화 방식을 이용한 PGP(Pretty Good Privacy)[19]를 이용해서 보내도록 한다.

7 Denial of Service

DoS는 Deinal of Service의 약자로 한명 혹은 다수의 사용자가 다른 사용자들이 시스템 자원이나 network 자원을 사용할 수 없게 독점해버리는 상황을 말한다. 또한 DoS는 보통의 해킹 방법과는 달리 root권한을 획득하는 것이 목적이 아니라 시스템이 정상적인 서비스를 제공할 수 없게 만드는 것 그 목적이 있다. 여기에서는 DoS와 DDoS그리고 DRDoS에 대한 간단한 언급과 각각에 대한 해결책을 살펴본다.

7.1 DoS

7.1.1 DoS란?

DoS 공격을 이해하려면 TCP에 대해서 약간의 지식이 필요하다. TCP는 network의 server와 client에 대해서 가상적인[20] connection을 만들어 준다. 이런 연결을 만들기 위해서 3-way hand shaking이란 방법을 사용하는데 이 과정은 다음의 순서로 이루어진다.

1. Client가 server에게 SYN packet을 보낸다.
2. Server가 client에게 SYN/ACK packet을 보낸다.
3. Client가 server에게 ACK packet을 보내고 연결이 생성된다.

이 과정의 첫번째 단계가 지나고 나면 server쪽에서는 client와 통신하기 위해 데이터를 위한 버퍼를 할당하고, 통신에 필요한 여러가지 정보[21]를 위한 공간을 확보한다. 즉, SYN packet을 받은 것만으로 server의 일정량의 "자원"이 할당되는 것이다.

만약 어떤 악의를 가진 사용자가 자신의 IP를 속이고[22] 계속해서 SYN packet을 보내면 server는 계속해서 연결을 위한 자원을 할당되고, 되돌아 오지 않는 ACK

[18] tunneling, ssh접속위에서 다른 서비스가 이루어지게 하는 것이다.
[19] 참고 http://www.pgpi.org
[20] 물리적으로 이어져 있지 않다면 연결 자체가 성립하지 않는다
[21] remote host의 IP, port 등등
[22] IP를 속이지 않고 해도 가능은 하지만, 해당 IP를 막는 것만으로 DoS를 막을 수 있기 때문에 별 의미없는 공격이 된다

packet에 대해서는 중간에 SYN/ACK packet이 lost된 것으로 생각하고 다시 SYN/ACK을 보낸다[23]. 이 것이 SYN flood[24]라고 불리우는 현상으로, 별로 빠르지 않은 한 개의 컴퓨터에서도 대용량의 서버가 서비스를 제공할 수 없게 만들 수 있는 방법이다.

7.1.2 DoS는 어떻게 막을 것인가?

현재는 SYN cookies라는 방법이 사용되고 있다. SYN cookie는 현재 FreeBSD와 Linux[25]에 기본적으로 포함되어 있다. 이 툴에서는 SYN이 들어온 것을 별도로 관리하고(즉 자원을 할당하지 않고), ACK까지 받게 되었을 때야 비로서 활성화 시킨다. 그리고 악의적인 사용자가 ACK마저 조작해서 보내는 것을 막기 위해 SYN/ACK을 보낼 때의 sequence number를 SYN의 IP, port, 시간에 대한 일방향 hash로 만들어서 악의적인 사용자가 유효한 ACK을 추측할 수 없게 만든다.

7.2 DDoS

DDoS[26]에는 DoS와는 다른 기법이 사용된다. 위에서 설명된 DoS가 network의 대역폭과는 상관없이 SYN flood를 이용해서 서버의 자원을 고갈시켰다면, DDoS는 미리 준비된 zombie[27]를 이용하여, 여러 개의 network 노드에서 "대량"의 packet을 보내어 목표 server/network의 대역폭을 대부분 점유해서 일반적인 사용자가 서비스를 제공받을 수 없게 만드는 것이다.

7.3 DRDoS

DRDoS[28]는 더욱 발전된 형태의 DDoS로 TCP의 3-way hand shaking을 이용한다. 우선 공격을 하려는 host에서는 network에 잘 연결되어 있는 server에 자신의 주소를 공격하려는 목표 서버의 주소로 속여서 SYN을 보낸다. SYN을 받은 server는 목표서버에게 SYN/ACK을 보내게 된다. 그런데 만약 이런 SYN/ACK이 엄청나게 많은 수가 오게 된다면, network 자원을 다 소모해버려서, 정상적인 서비스를 수행할 수 없게 된다.

또한 이런 형태의 공격은 SYN을 network에 잘 연결되어 있는 internet의 core router들이나 yahoo 등의 잘 알려져 있는 host들에게 계속적으로 보낼 경우, 그 server들에서 빠른 속도로 SYN/ACK을 보내오기 때문에 목표가 된 server는 심각한 장애에 빠질 수 밖에 없다. 그리고 공격의 대상이 될 수 있는 port는 일반적으로 사용되는 모든 TCP port일 수 있기 때문에[29], 간단하게 막을 수 있는 방법도 없다.

---

[23]물론 위조된 IP가 실존하는 host의 것이었다면 자기 것이 아니라는 회신을 보내오겠지만 random하게 생성된 IP주소가 실존하는 host의 것일 확률은 매우 작다

[24]이 외에도 ping이나 대용량의 메일을 이용해서 flood를 발생시키는 방법도 있다

[25]Linux에서는 echo 1 > /proc/sys/net/ipv4/tcp_syncookies 를 해줘야 활성화된다.

[26]Distributed Deinal of Service

[27]악의적인 사용자의 조정에 따라 지정된 서버나 network으로 대량의 packet을 보내기 위한 프로그램

[28]Distributed Reflection DoS

[29]Reflection이 일어나는 server(즉 SYN을 받은 server)에 열려있는 모든 port를 사용할 수 있다

이런 종류의 공격은 어떻게 막을 수 있을까? 우선 server측에서는 높은 번호대의 port를 전부 무시하게 만든다[30]. 단 server에 따라서는 server의 역할만 수행하는 것이 아니라 client가 되는 경우[31]도 있으므로 이런 경우에 따른 설정은 따로 해줘야할 것이다.

8 Spoofing

Spoofing은 그 단어가 뜻하는 바처럼, 공격자가 공격대상에 자신이 신뢰하는 시스템인 것처럼 속이는 것을 말한다. 이는 TCP/IP의 구조적인 취약점을 이용한 것으로 Kevin Mitnick이 처음으로 사용했다. Spoofing을 성공적이게 수행하려면 그 기술만큼이나 운이 따라줘야하는 해킹 기법이다. Spoofing에는 IP Spoofing, ARP Spoofing, DNS Spoofing이 있는데, 여기서는 가장 광범위하게 쓰일 수 있는 IP Spoofing에만 집중을 하도록 하겠다.

8.1 TCP/IP Connection

Spoofing은 TCP/IP의 취약점을 이용한 것이므로, 이를 이해하려면 먼저 TCP/IP에 대한 이해가 선행되어야 한다. TCP/IP는 현재 네트워크 프로토콜 중에 표준으로 자리 잡혀있다. 5개의 계층(Application,Transport, Network, Link, Physical)으로 데이터 교환의 추상화를 제공하여 손쉽게 개발자들이 네트워크를 이용할 수 있게 해준다.

<table>
<tr><td colspan="3">source port</td><td>destination port</td></tr>
<tr><td colspan="4">sequence number</td></tr>
<tr><td colspan="4">acknowledgement number</td></tr>
<tr><td>header length</td><td>reserved</td><td>code bits</td><td>window size</td></tr>
<tr><td colspan="3">checksum</td><td>urgent offset</td></tr>
<tr><td colspan="4">options (if any)</td></tr>
<tr><td colspan="4">data (if any)</td></tr>
<tr><td colspan="4">...</td></tr>
</table>

TCP에서 두 시스템간에 연결이 이루어질 경우에 앞에서 알아본 것과 같이 3-way hand shaking이라는 방법을 사용하는데 좀 더 자세하게 알아보겠다.

A	⇒	SYN	Seq:1500	ack:–	⇒	B
A	⇐	SYN + ACK	Seq:1300	ack:1501	⇐	B
A	⇒	ACK	Seq:1501	ack:1301	⇒	B

여기서 “A”가 접속을 시도하는 호스트이고. “B”가 접속을 받아들이는 호스트이다. 먼저 A가 B에게 접속을 시도한다는 “SYN” bit와 현재 보내는 패킷의 순서를 나타내는 Seqence Number를 같이 보낸다. 이를 받은 “B”는 알아들었다는 뜻의 “ACK”과 “SYN”, 그리고 Sequence Number, “A”로부터 다음에 받을 패킷의 Sequence Number인 Acknowledgement Number를 포함하는 패킷으로 응답한다. 이에 “A”는 “ACK”으로 응답하고 접속과정이 끝나는 것이다. Initial Sequence Number(ISN)라는 것이 있는데 이는 각 시스템이 처음으로 보낸 Sequence Number이다. 여기에서는 “A”의 ISN은 첫번째 과정에서 ‘1500’으로 나타나고 , “B”의 ISN은 두번째 과정에서 ‘1300’으로 나타난다.

[30] 악의적인 사용자가 SYN을 보낼 때에는 높은 번호 대의 port를 사용하기 때문에(client이므로), SYN/ACK이 들어오는 port는 높은 번호이다

[31] 예 : SMTP server에서 메일을 가져오는 웹 서버

8.2 Spoofing의 원리

Spoofing이 일어나기 위한 가정은 먼저 속이기 위한 두 시스템이 필요하고, 그 중 한 시스템이 다른 시스템을 특별히 신뢰하고 있어야 한다. 예를 들면 rsh, rlogin과 같은 Berkeley r-Utility들이 설정되어 있는 것이다.[32] 먼저 속여서 되고자 하는 대상을 "bleu"라고 하고, 속이는 대상을 "rei"라고 하겠다. 먼저 "bleu"를 DoS(Denial of Service)로 그 기능은 마비시키고, "rei"에 TCP Connection의 첫번째 단계인 "SYN"패킷을 보내서 연결을 요청한다. 이때 IP는 "bleu"의 것으로 바꿔서 보낸다. "rei"는 "bleu"에서 패킷이 온줄 알고 "SYN + ACK" 패킷을 "rei"에 보내지만 DoS로 마비되어서 받지 못한다. 이제 "rei"에 다시 "ACK"패킷을 보내서 속이게 된다. 그런데 여기서 Spoofing이 쉽게 일어날 수 없는 점을 발견하게 된다. 마지막에 "rei"에 패킷을 보낼때, 패킷의 Acknowledgement Number를 손쉽게 적을 수 없다는 점이다. 즉. "rei"가 "bleu"에 보낸 패킷의 Sequence Number를 알아내기 힘들다는 점이다. Sqeunce Number를 알아내는 방법은 Network를 관찰하면서 예측하는 것이 주로 쓰인다.[33] 이후에 "bleu"와 "rei"의 신뢰관계를 이용해서 시스템에 침입하는 것이다.[34]

8.3 대책

Spoofing은 근본적으로 TCP/IP의 취약점을 이용하기 때문에 우리가 TCP/IP 이외의 프로토콜을 사용하는 한 계속 위험에 노출되어 있을 수 밖에 없다. 완벽하게 피할 수는 없지만 Spoofing을 어렵게 할 수는 있다.

8.3.1 외부에서 오는 패킷막기

내부망을 연결하고 있는 라우터나 스위치 레벨에서 외부에서 오는 패킷이 내부의 IP를 가지고 있는 경우 통과시키지 않는 방법도 있다. 이럴 경우 신뢰하는 호스트는 서로 같은 내부망안에서 위치하도록 하는 것이 좋다. 이러한 방식은 대부분의 Spoofing을 방지할 수 있지만, 해커가 내부망 안에 존재한다면 무용지물이 될 것이다.

8.3.2 공개키 방식을 이용한 서비스

내부망에서도 Spoofing을 막기위한 방법이 필요하다. 이는 앞서 sniffing부분에서도 언급한 것과 같이 암호화된 데이터 교환이 필요한다. 그러나 우리는 특별히 Berkeley r-Utility와 같은 암호없이 다른 시스템에 접근해야할 상황에 처할 수도 있다.[35] 그러나 Berkeley r-Utility는 취약점이 분명해서 절대 쓰지 말아야 할 서비스 중에 하나이다. 이때는 Secure Shell의 키인증방식이 하나의 대안이 될 수 있을 것이다.[36]

bash$ ssh-keygen -t dsa

[32]요즘은 Berkeley r-Utility와 같은 것들은 보안상의 문제로 잘 쓰이지 않는다. 그러므로 다른 해킹기법과 복합적으로 쓰이는 경우가 많다.

[33]Sequence Number는 OS에 따라 SYN 패킷을 받을 때마다 64000씩 증가하는 경우, 시간에 비례해서 증가하는 경우, 무작위로 생성하는 경우가 있다.

[34]Berkeley r-Utility가 가능한 경우, *echo '+ +' >/.rhosts*과 같이 모든 호스트로부터 rlogin이 가능하게 설정하는 방법이 있다.

[35]Clustering과 같이, 한 시스템에서 전체 시스템을 관리할 때 암호없이 접근하는 방법이 필요하다

[36]Berkeley r-Utility와 거의 같이 호스트 기반의 인증방식도 존재한다. 하지만 OpenSSH에서 실제로 구현 가능이 불확실해서 생략하였다. 참조 : http://www.ssh.com/support/documentation/online/ssh/adminguide/31/Host-Based_Authentication.html

Enter file in which to save the key (/home/reca/.ssh/id_dsa): 키가 저장될 경로를 묻는 것인데 <ENTER>.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
암호를 입력하지 않고 <ENTER>를 입력한다.
bash$ scp id_dsa.pub rei:.ssh/authorized_keys2
인증키를 복사한다. 'rei'는 접속하고자 하는 대상
bash$ ssh rei
이제 암호없이 로그인된다.

간단하게 ssh2 프로토콜을 이용한 설정을 알아보았다. 암호를 공백으로 두면 묻지 않는 점을 이용하여 Berkeley r-Utility의 기능을 내고 있다.[37] 이렇게 하면 Spoofing에는 단순히 ISN(Initial Sequence Number)만 추측해야 하는 것이 아니라 Secure Shell의 통신상의 데이터도 알아야 하므로 Spoofing의 확률은 현저하게 떨어진다. 다른 방식을 원한다면 Kerberos[38]를 알아보도록 한다.

8.3.3 안전한 OS사용하기

여기서 안전한 OS를 사용한다는 것은 Initial Sequnce Number(ISN)를 추측하기 어렵게 생성하는 OS를 말한다. 추측하기 어려운 ISN이라는 것은 무작위로 ISN를 생성하는 것이라고 할 수 있다. OS를 만드는 측에서도 이의 중요성을 인식해서 점점 'SYN'을 받을 때마다 64000씩 증가시키거나 시간에 비례해서 증가시키는 옛날의 방식에서 무작위로 생성하는 방식[39]으로 바뀌고 있다. 이렇게 하면 ISN를 추측하기 어려우져서 Snoofing에 보다 안전해진다. 다음은 주요 OS별 ISN 생성 방법이다.[40]

FreeBSD FreeBSD 4.3-RELEASE이후에 OpenBSD 2.8의 알고리즘을 채택함으로서 임의로 ISN을 생성한다.

AIX AIX 4이후 버전부터 임의로 ISN을 생성한다.

Linux Linux Kernel 2.x대는 RFC 1948을 구현한다.

IRIX IRIX 6.5.3 이후 버전부터 RFC 1948(ISN임의생성표준)과 MD5를 구현하였다. 그러나 기본으로 이용하지 않는다. 이용하기 위해서는 root로 다음을 실행시킨다.
/usr/sbin/systune -b tcpiss_md5 1

Solaris SunOS 2.6부터 RFC 1948을 적용하고 있지만 기본으로 적용되지 않는다(기본은 자체 알고리즘을 이용한 ISN 임의 생성이다)./*etc/default/inetinit*의 TCP_STRONG_ISS가 기본적으로 1로 설정되어 있지만 2로 바꾸면 RFC 1948식 ISN 임의생성을 사용할 수 있다.

---

[37] 암호를 공백으로 두는 것은 조심해야 한다. 다른 취약점을 이용해 해커가 시스템에 침입했을 때, 다른 시스템으로 쉽게 접근하는 발판을 제공해 주기 때문이다.

[38] 인증서버를 두고 한번 받은 인증으로 일정시간동안 재로그인 없이 Kerberos에 포함된 시스템들에 접근할 수 있다. 참조:http://web.mit.edu/kerberos/www/

[39] pseudo-random number generators(PRNGs)

[40] 임의로 ISN을 생성할 경우 각 OS별 구현이 존재하고 RFC 1948를 따른 구현이 존재하는데, 후자가 일반적으로 좀 더 완전한 임의 생성에 가깝다. 자세한 사항은 RFC문서를 참조하기를 바란다.

9 맺음말

지금까지 해킹 기법과 거기에 대응하는 보안 방안에 대해서 알아보았다. 알아본 해킹 기법은 Social Engineering, Buffer Overflow, Sniffing, Denial of Service, Spoofing이다. 사실 해킹 기법은 여기에서 소개한 방법보다 많지만, 여기서 소개한 해킹 기법이 현재 가장 널리 쓰이는 것이라고 할 수 있다. 시스템 관리자들은 해커들의 기술을 이해함으로서 어떻게 자신의 시스템이 더욱 안전하게 작동할 수 있는지에 대해서 생각해 볼 수 있을 것이다. 사실 이 문서에서 소개한 보안대책을 사용하려면 귀찮을 수도 있고, 또 추상적인 대책도 또한 있어 쉽게 적용할 수 없을 수도 있지만, 하나씩 알아보고 하면 점점 더 단단한 시스템을 만들 수 있을 것이다. 혹자는 이 문서에서 해킹 기법을 소개함으로서 보안에 해가 된다고 생각할지도 모르나, 글을 마치면서 처음 글을 썼을 때의 마음은 변함이 없다. 이 문서가 시스템 보안에 어느 정도라도 도움이 되었다면 필자는 그것으로 만족할 것이다.

References

[1] Aleph One, *Smashing The Stack For Fun And Profit*, Phrack 49-14(http://www.phrack.org/phrack/49/P49-14)

[2] *StackGuard*(http://www.immunix.org/stackguard.html)

[3] Mark E. Donaldson, *Inside the Buffer Overflow Attack: Mechanism, Method, & Prevention*, SANS institute(http://rr.sans.org/code/inside_buffer.php)

[4] W. Richard Stevens, *TCP/IP Illustrated, Volume 2*, Addison-Wesley Publishing Company, 1994

[5] 박현미, 신은경, 이현우, *네트워크 스니핑 기술 및 방지대책*, CERTCC-KR 기술문서(http://www.certcc.or.kr/paper/tr2000/2000-07/tr2000-07.htm)

[6] Charles E. Spurgeon, *Ethernet,The Definitive Guide*, O'Reilly

[7] CERTCC-KR, *IP spoofing 공격과 대책*, CERTCC-KR 기술문서(http://www.certcc.or.kr/advisory/tr/IPspoof.html)

[8] *Denial of Service : Investigation & Exploration Pages*, Gibson Research Corporation(http://grc.com/dos)

[9] D.J Bernstein, *SYN cookies*(http://cr.yp.to/syncookies.html)

[10] Michael Schiffman, *IP-spoofing Demystified*, Phrack 48-14(http://www.phrack.org/phrack/48/P48-14)

[11] Bill Woodcock, *Ask Woody about Spoofing Attacks*, Zocalo Engineering(http://www.netsurf.com/nsf/v01/01/local/spoof.html)

[12] Jeffrey S. Havrilla, *Statistical Weaknesses in TCP/IP Initial Sequence Numbers*, CERT Advisory CA-2001-09(http://www.cert.org/advisories/CA-2001-09.html)