Applied Application Security

Julian Cohen
HockeyInJune@isis.poly.edu
THOTCON 0x2

How To Protect Your Boxes From Jon Oberheide

Julian Cohen
HockeyInJune@isis.poly.edu
THOTCON 0x2

Exploitation Mitigation Techniques

Julian Cohen
HockeyInJune@isis.poly.edu
THOTCON 0x2

Memory Corruption: An Arms Race

Julian Cohen
HockeyInJune@isis.poly.edu
THOTCON 0x2

Cyber Security Awareness Week

- Capture The Flag competition
- Application Security
- Worldwide

- http://csawctf.poly.edu/
- http://www.poly.edu/csaw

Motivation

- Insecure platform
- Too much attack surface
- Outdated software
- We don’t want to get owned!

Assumptions

- Linux
- Bugs exist in code
- Anything that has attack surface is vulnerable
- Exploitation cost can be manipulated

Goals

- Make it really fucking hard to land an exploit

VULNERABILITIES AND EXPLOITS

Memory Corruption	Smashing the Stack
Integer Manipulation	Return To Library
Uncontrolled Format String	Return Oriented Programming
Memory Mismanagement	Global Offset Table Overwrite
Race Condition	Use After Free

Buffer Overflow

```
void vuln(void) {
        char buffer[8];
        gets(buffer);
        printf(buffer);
}
```

Compile-Time Hardening

- -D_FORTIFY_SOURCE=2
  - Bounds checking on dangerous function calls
  - http://gcc.gnu.org/ml/gcc-patches/2004-09/msg02055.html
  - http://isisblogs.poly.edu/?p=205
- -Wformat -Wformat-security
- -Werror=format-security
  - Uncontrolled format string detection
  - http://gcc.gnu.org/onlinedocs/gcc-4.6.0/gcc/Warning-Options.html

Default

```
push   ebp
mov    ebp,esp
push   ebx
sub    esp,0x24
lea    ebx,[ebp-0x10]
mov    DWORD PTR [esp],ebx
call   8048304 <gets@plt>
mov    DWORD PTR [esp],ebx
call   8048324 <printf@plt>
add    esp,0x24
pop    ebx
pop    ebp
ret
```

Fortify

```
push   ebp
mov    ebp,esp
push   ebx
sub    esp,0x24
mov    DWORD PTR [esp+0x4],0x8
lea    ebx,[ebp-0x10]
mov    DWORD PTR [esp],ebx
call   804833c <__gets_chk@plt>
mov    DWORD PTR [esp+0x4],ebx
mov    DWORD PTR [esp],0x1
call   804832c <__printf_chk@plt>
add    esp,0x24
pop    ebx
pop    ebp
ret
```

```
hake@ubuntu:~/code/thotcon$ python -c "print('A'*32)" | ./code
Segmentation fault
```

```
hake@ubuntu:~/code/thotcon$ python -c "print('A'*32)" | ./codefortify
*** buffer overflow detected ***: ./codefortify terminated
======= Backtrace: =========
/lib/libc.so.6(__fortify_fail+0x50)[0xb4a970]
/lib/libc.so.6(+0xe486a)[0xb4986a]
/lib/libc.so.6(__gets_chk+0x16a)[0xb497fa]
./codefortify[0x804842e]
./codefortify[0x804844f]
/lib/libc.so.6(__libc_start_main+0xe7)[0xa7bce7]
./codefortify[0x8048381]
```

Smashing The Stack

```
hake@ubuntu:~/code/thotcon$ python -c "print('A'*32)" > data
hake@ubuntu:~/code/thotcon$ gdb -q ./code
Reading symbols from /home/hake/code/thotcon/code...done.
(gdb) r < data
Starting program: /home/hake/code/thotcon/code < data

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

Stack Smashing Protection

- Stack canaries
  - -fstack-protector-all
  - Push word onto stack before return address

- http://gcc.gnu.org/onlinedocs/gcc-4.6.0/gcc/Optimize-Options.html

Default

```
push   ebp
mov    ebp,esp
push   ebx
sub    esp,0x24
lea    ebx,[ebp-0x10]
mov    DWORD PTR [esp],ebx
call   8048304 <gets@plt>
mov    DWORD PTR [esp],ebx
call   8048324 <printf@plt>
add    esp,0x24
pop    ebx
pop    ebp
ret
```

Stack Protector

```
push   ebp
mov    ebp,esp
push   ebx
sub    esp,0x24
mov    eax,gs:0x14
mov    DWORD PTR [ebp-0xc],eax
xor    eax,eax
lea    ebx,[ebp-0x14]
mov    DWORD PTR [esp],ebx
call   8048348 <gets@plt>
mov    DWORD PTR [esp],ebx
call   8048368 <printf@plt>
mov    eax,DWORD PTR [ebp-0xc]
xor    eax,DWORD PTR gs:0x14
je     8048486 <vuln+0x36>
call   8048378 <__stack_chk_fail@plt>
add    esp,0x24
pop    ebx
pop    ebp
ret
```

```
hake@ubuntu:~/code/thotcon$ python -c "print('A'*32)" | ./bo
Segmentation fault
```

```
hake@ubuntu:~/code/thotcon$ python -c "print('A'*32)" | ./boprotector
*** stack smashing detected ***: ./boprotector terminated
======= Backtrace: =========
/lib/libc.so.6(__fortify_fail+0x50)[0xf14970]
/lib/libc.so.6(+0xe591a)[0xf1491a]
./boprotector[0x8048486]
[0x41414141]
```

No-eXecute Bit

- Prevents data from being treated as code
- Hardware and kernel support

```
flags           : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 syscall nx mmxext fxsr_opt rdtscp lm 3dnowext 3d
now constant_tsc tsc_reliable nonstop_tsc pni cx16 hypervisor lahf_lm extapic 3d
nowprefetch
```

- http://people.redhat.com/mingo/exec-shield/ANNOUNCE-exec-shield
- http://pax.grsecurity.net/

Default Stack

```
08048000-08049000 r-xp 00000000 08:01 403630     /home/hake/code/thotcon/nx
08049000-0804a000 r--p 00000000 08:01 403630     /home/hake/code/thotcon/nx
0804a000-0804b000 rw-p 00001000 08:01 403630     /home/hake/code/thotcon/nx
b78de000-b78df000 rw-p 00000000 00:00 0
b78ec000-b78ef000 rw-p 00000000 00:00 0
bfebe000-bfedf000 rw-p 00000000 00:00 0          [stack]
```

Executable Stack

```
08048000-08049000 r-xp 00000000 08:01 419141     /home/hake/code/thotcon/nxxstack
08049000-0804a000 r-xp 00000000 08:01 419141     /home/hake/code/thotcon/nxxstack
0804a000-0804b000 rwxp 00001000 08:01 419141     /home/hake/code/thotcon/nxxstack
bfc04000-bfc25000 rwxp 00000000 00:00 0          [stack]
```

Default Heap

```
218ba000-2191e000 rw-p 00000000 00:00 0          [heap]
b7864000-b7867000 rw-p 00000000 00:00 0
b7875000-b7877000 rw-p 00000000 00:00 0
bf922000-bf943000 rw-p 00000000 00:00 0          [stack]
```

Non-Executable Stack

```
(gdb) r `python -c "print('A'*20+'\xb8\xf3\xff\xbf')"`
Starting program: /home/hake/code/thotcon/nx `python -c "print('A'*20+'\xb8\xf3\
xff\xbf')"`

Program received signal SIGSEGV, Segmentation fault.
0xbffff3b8 in ?? ()
(gdb) info registers eip
eip            0xbffff3b8       0xbffff3b8
(gdb) x/i 0xbffff3b8
=> 0xbffff3b8:  inc    %ecx
```

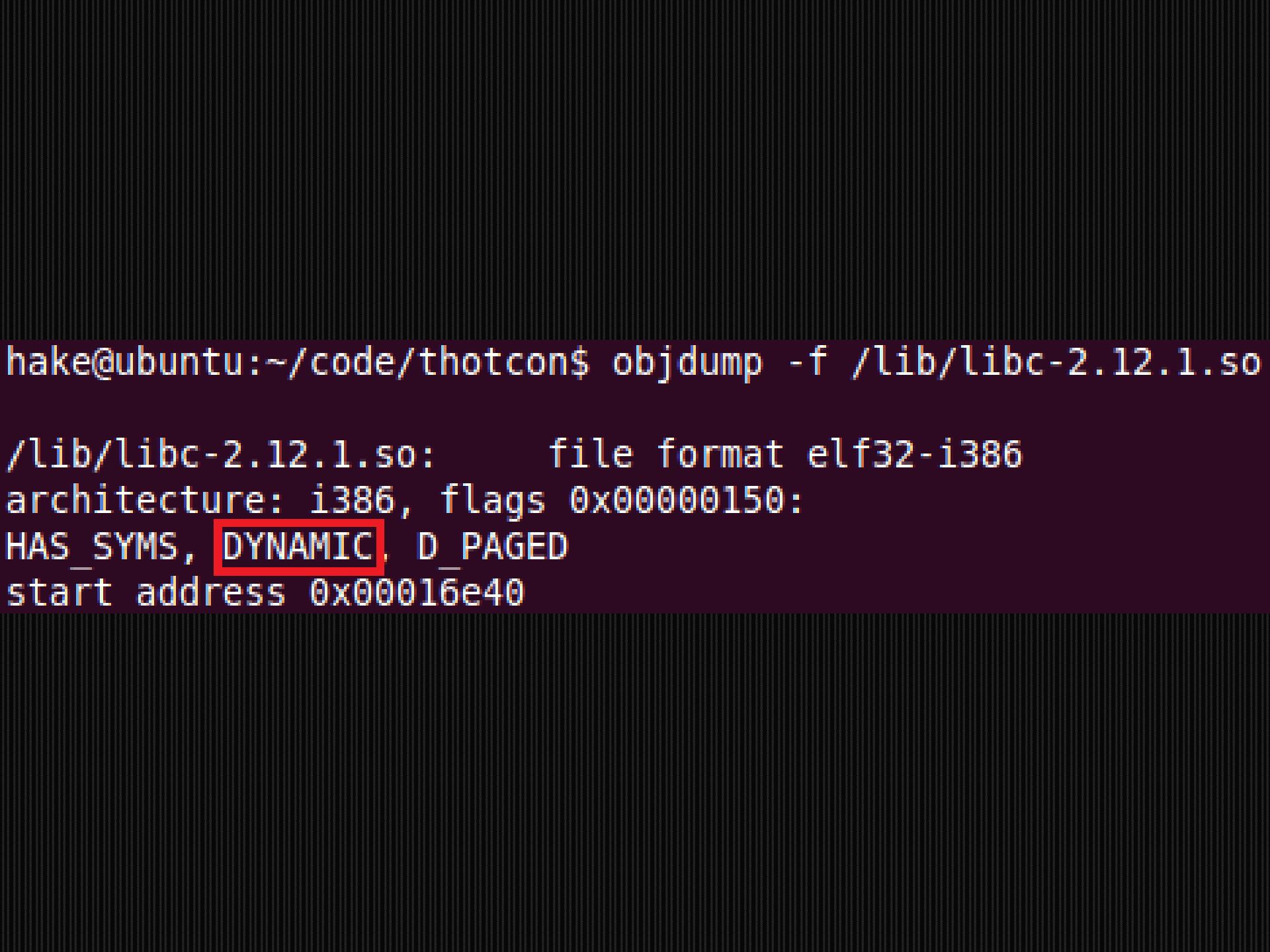
Return To Library

```
hake@ubuntu:~/code/thotcon$ ls -al /tmp | grep PWNED
hake@ubuntu:~/code/thotcon$ python -c "print('/usr/bin/touch /tmp/PWNED\n' + 'A'
*20 + '\x80\x86\x16\x00' + 'A'*4 + '\xe0\xf3\xff\xbf')" | ./retlib
Segmentation fault
hake@ubuntu:~/code/thotcon$ ls -al /tmp | grep PWNED
-rw-r--r--  1 hake hake    0 2011-04-11 11:17 PWNED
```

Position Independent Code

- Shared libraries are compiled PIC
  - -fpic -fPIC -shared

- http://gcc.gnu.org/onlinedocs/gcc-4.6.0/gcc/Code-Gen-Options.html#Code-Gen-Options
- http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html
- http://www.gentoo.org/proj/en/hardened/pic-internals.xml
- http://www.gentoo.org/proj/en/hardened/pic-guide.xml


hake@ubuntu:~/code/thotcon$ objdump -f /lib/libc-2.12.1.so
/lib/libc-2.12.1.so: file format elf32-i386
architecture: i386, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x00016e40

Address Space Layout Randomization

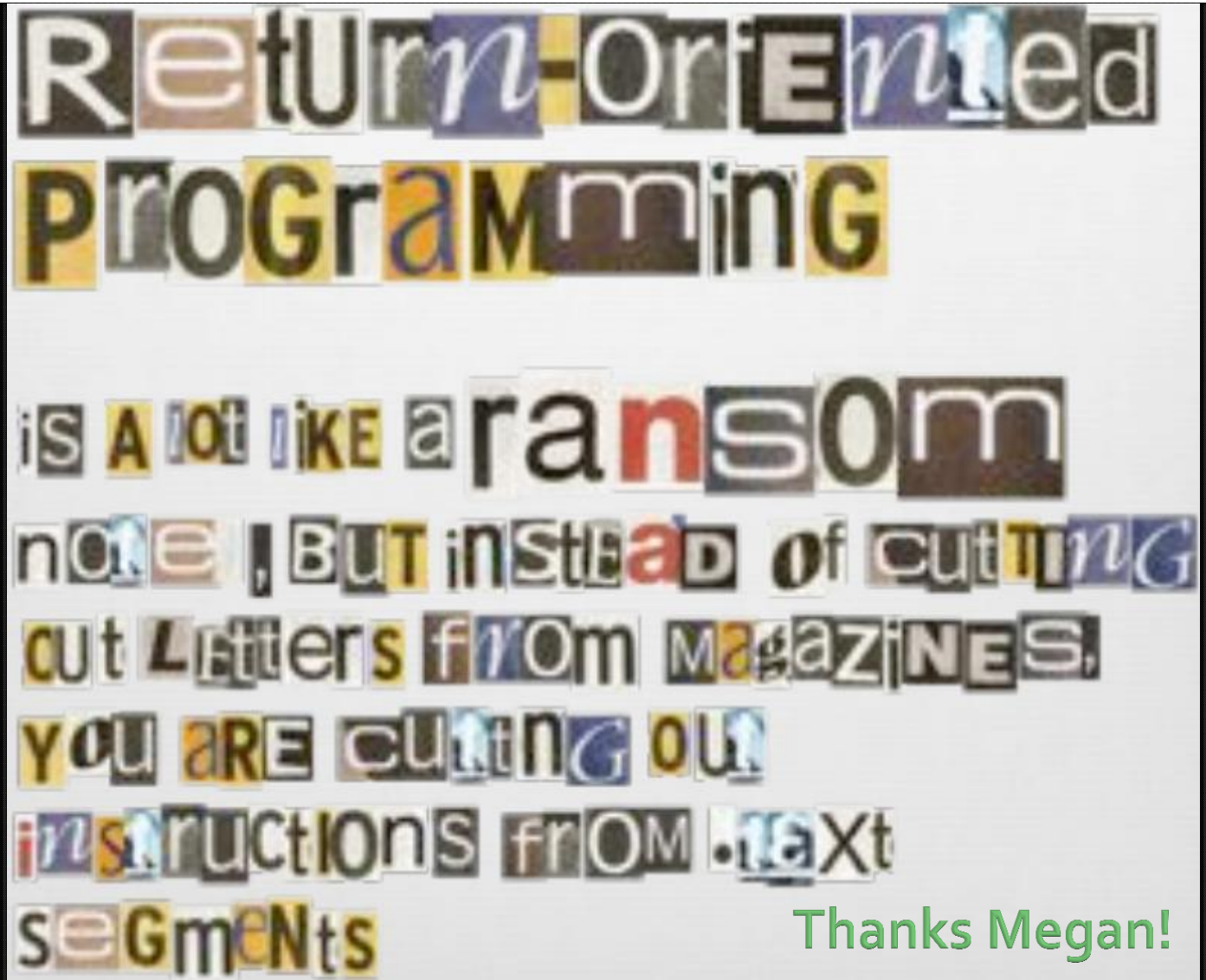
- Stack, Heap, Libraries
- Loaded dynamically into random at runtime
  - kernel.randomize_va_space = 2

  - http://people.redhat.com/mingo/exec-shield/ANNOUNCE-exec-shield
  - http://pax.grsecurity.net/

```c
void vuln(void) {
        char stack[8];
        char* heap = malloc(8);
        void* lib = &system;
        void* bin = &vuln;
        printf("Stack:\t\t%8x\n", &stack);
        printf("Heap:\t\t%8x\n", heap);
        printf("Library:\t%8x\n", lib);
        printf("Binary:\t\t%8x\n", bin);
}
```

```
hake@ubuntu:~/code/thotcon$ ./aslr
Stack:          bf902834
Heap:           8357008
Library:        8048370
Binary:         8048488
hake@ubuntu:~/code/thotcon$ ./aslr
Stack:          bf942404
Heap:           8a40008
Library:        8048370
Binary:         8048488
hake@ubuntu:~/code/thotcon$ ./aslr
Stack:          bfddb444
Heap:           9955008
Library:        8048370
Binary:         8048488
hake@ubuntu:~/code/thotcon$ ./aslr
Stack:          bfc9ec14
Heap:           845b008
Library:        8048370
Binary:         8048488
```

Return Oriented Programming

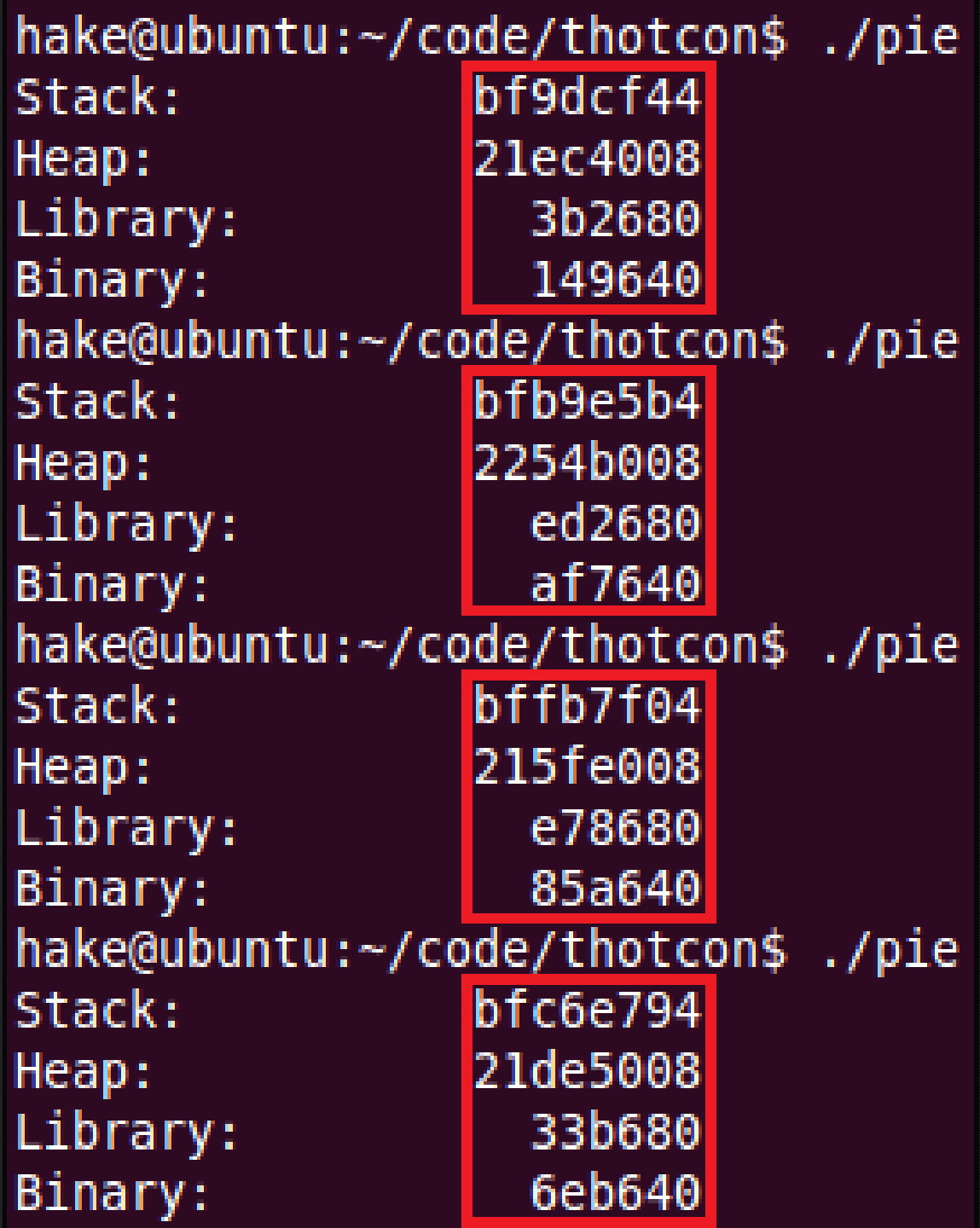


Position Independent Executable

- Binaries can be compiled PIE
  - -pie -fpie

- http://gcc.gnu.org/onlinedocs/gcc-4.6.0/gcc/Code-Gen-Options.html#Code-Gen-Options
- http://gcc.gnu.org/ml/gcc-patches/2003-06/msg00140.html

```
hake@ubuntu:~/code/thotcon$ objdump -f nopie

nopie:     file format elf32-i386
architecture: i386, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x080483c0

hake@ubuntu:~/code/thotcon$ objdump -f pie

pie:     file format elf32-i386
architecture: i386, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x00000530
```


hake@ubuntu:~/code/thotcon$ ./pie
Stack: bf9dcf44
Heap: 21ec4008
Library: 3b2680
Binary: 149640
hake@ubuntu:~/code/thotcon$ ./pie
Stack: bfb9e5b4
Heap: 2254b008
Library: ed2680
Binary: af7640
hake@ubuntu:~/code/thotcon$ ./pie
Stack: bffb7f04
Heap: 215fe008
Library: e78680
Binary: 85a640
hake@ubuntu:~/code/thotcon$ ./pie
Stack: bfc6e794
Heap: 21de5008
Library: 33b680
Binary: 6eb640

Global Offset Table Overwrite

- .got always mapped to same memory location
- GOT dynamically updated during runtime
- An attacker can overwrite GOT entries

Relocation Read-Only

- Internal sections mapped before other sections
- Internal sections mapped read-only
  - -Wl,-z,relro,-z,now
  - .got .dtors .dynamic
- Dynamic linker resolves all symbols at start
- Colloquially known as RELRO

  - http://tk-blog.blogspot.com/2009/02/relro-not-so-well-known-memory.html

Segments Marked Read-Only

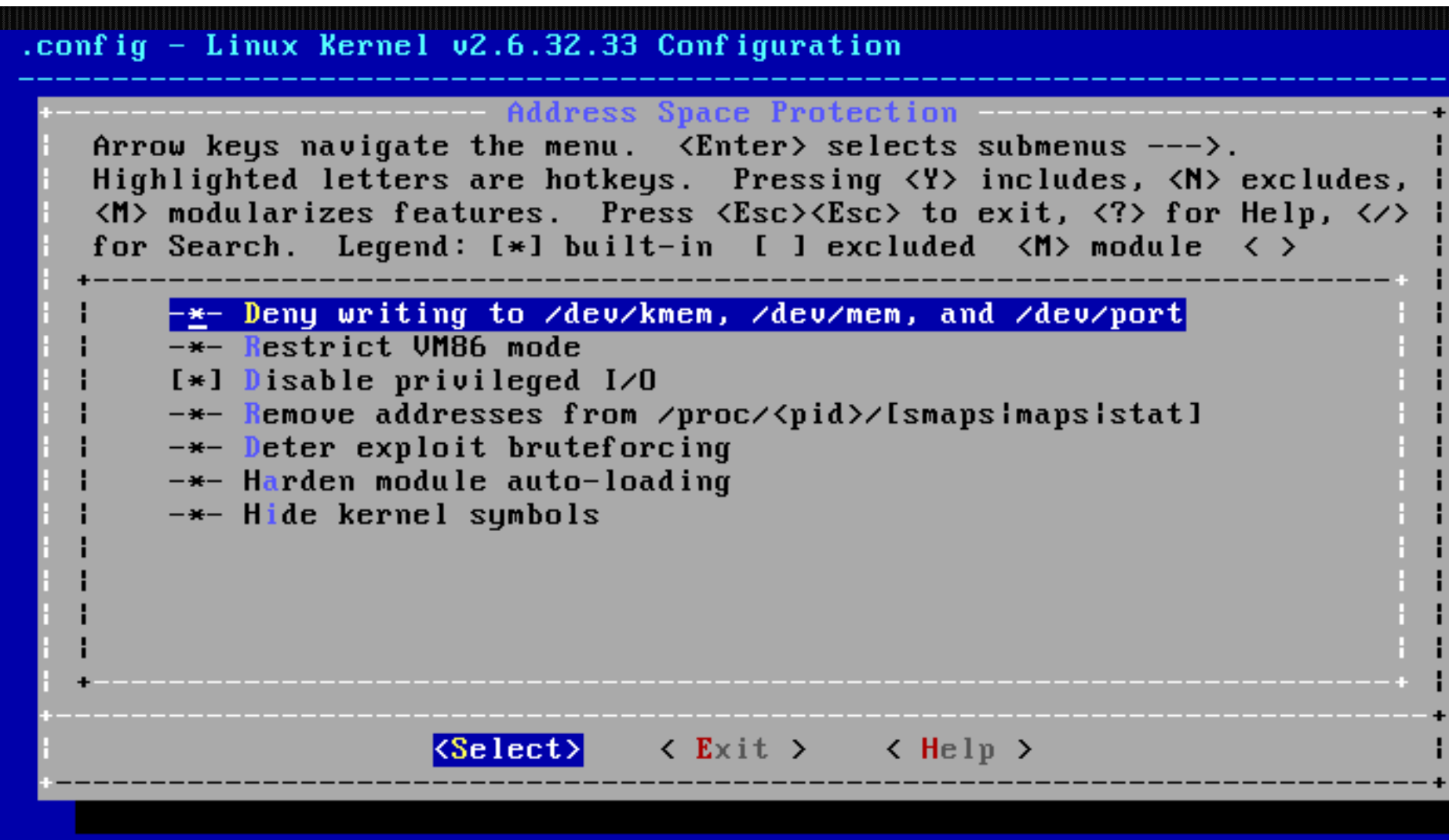
```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00100 0x00100 R E 0x4
  INTERP         0x000134 0x08048134 0x08048134 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x005c4 0x005c4 R E 0x1000
  LOAD           0x000ef0 0x08049ef0 0x08049ef0 0x00118 0x00120 RW  0x1000
  DYNAMIC        0x000f04 0x08049f04 0x08049f04 0x000d8 0x000d8 RW  0x4
  NOTE           0x000148 0x08048148 0x08048148 0x00044 0x00044 R   0x4
  GNU STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4
  GNU RELRO      0x000ef0 0x08049ef0 0x08049ef0 0x00110 0x00110 R   0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym
on_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame
   03     .ctors .dtors .jcr .dynamic .got .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06
   07     .ctors .dtors .jcr .dynamic .got
```

Kernel Hardening

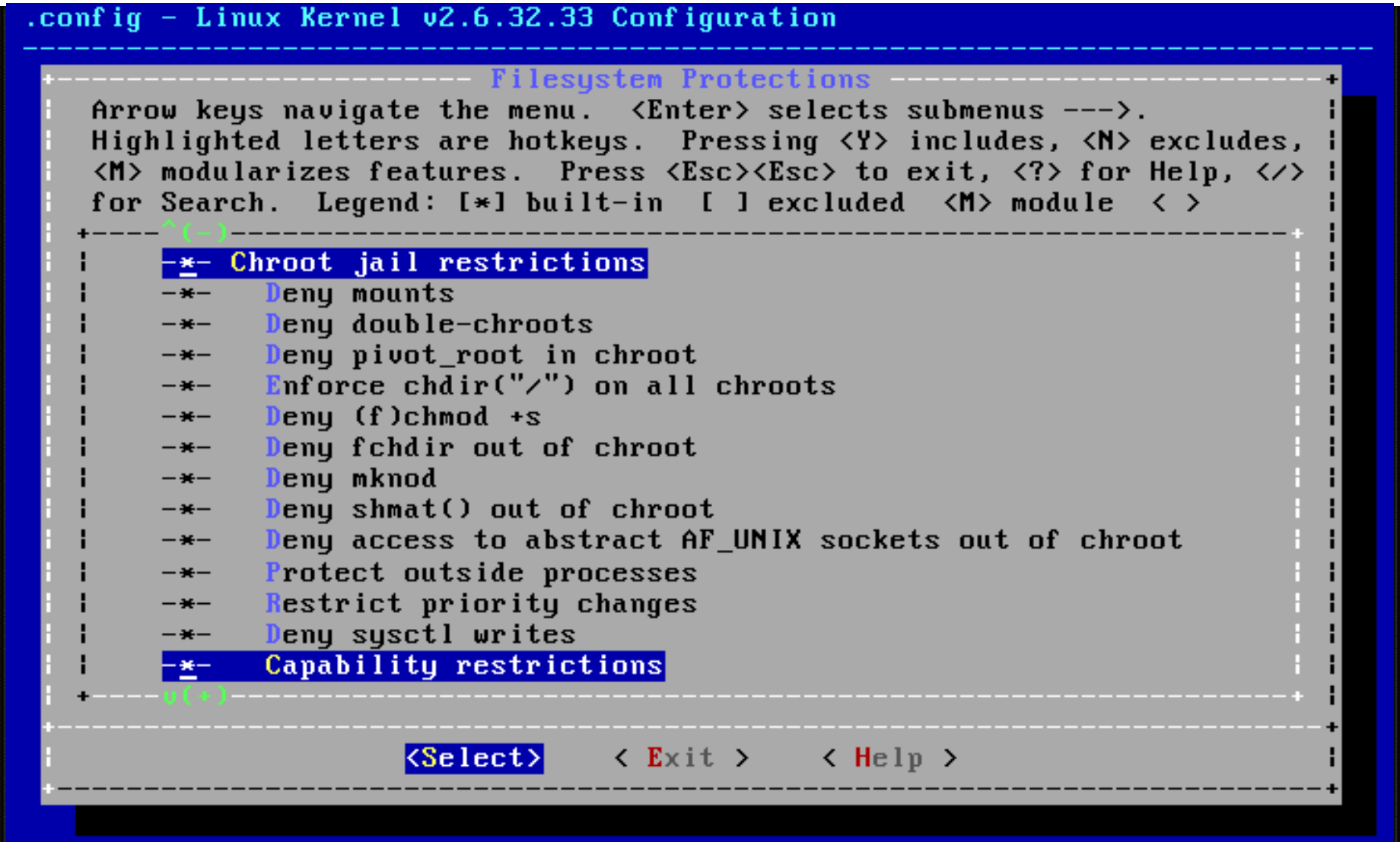
- grsecurity kernel patch
- PaX

- http://grsecurity.net/
- http://pax.grsecurity.net/

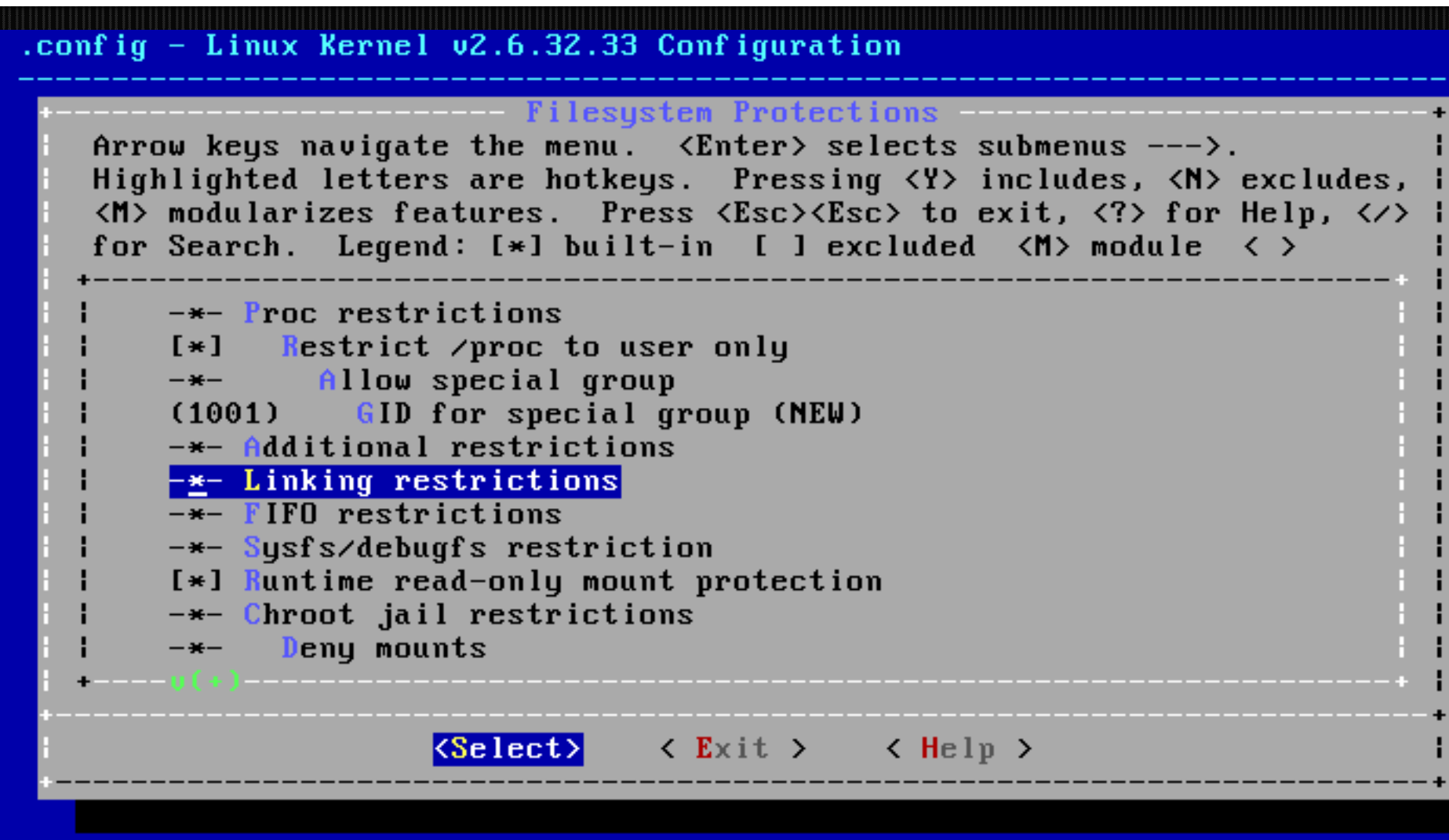
Address Space Protection



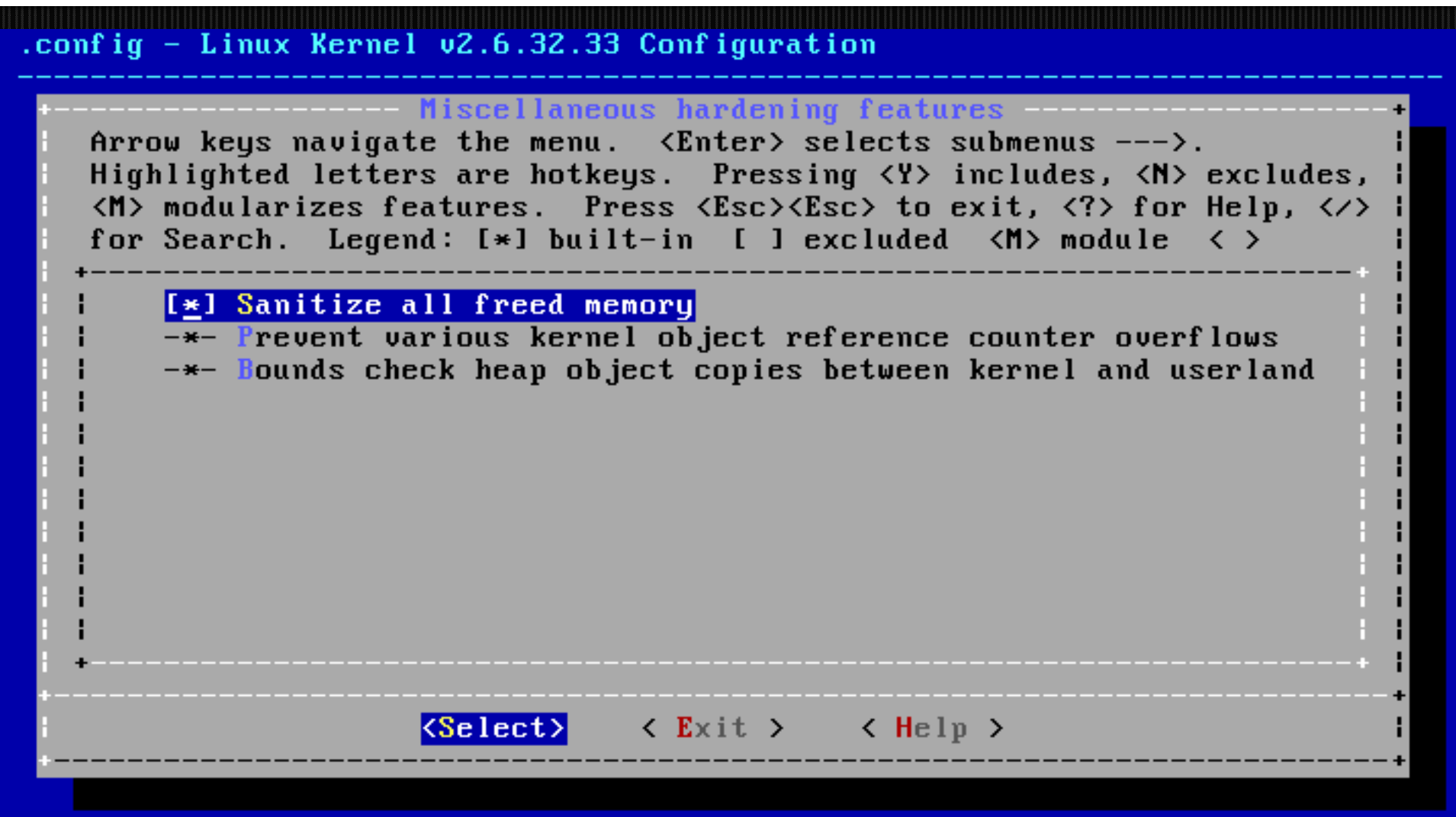
Chroot Hardening



Race Protection



Use After Free Protection



Verification and Testing

- checksec.sh
- paxtest

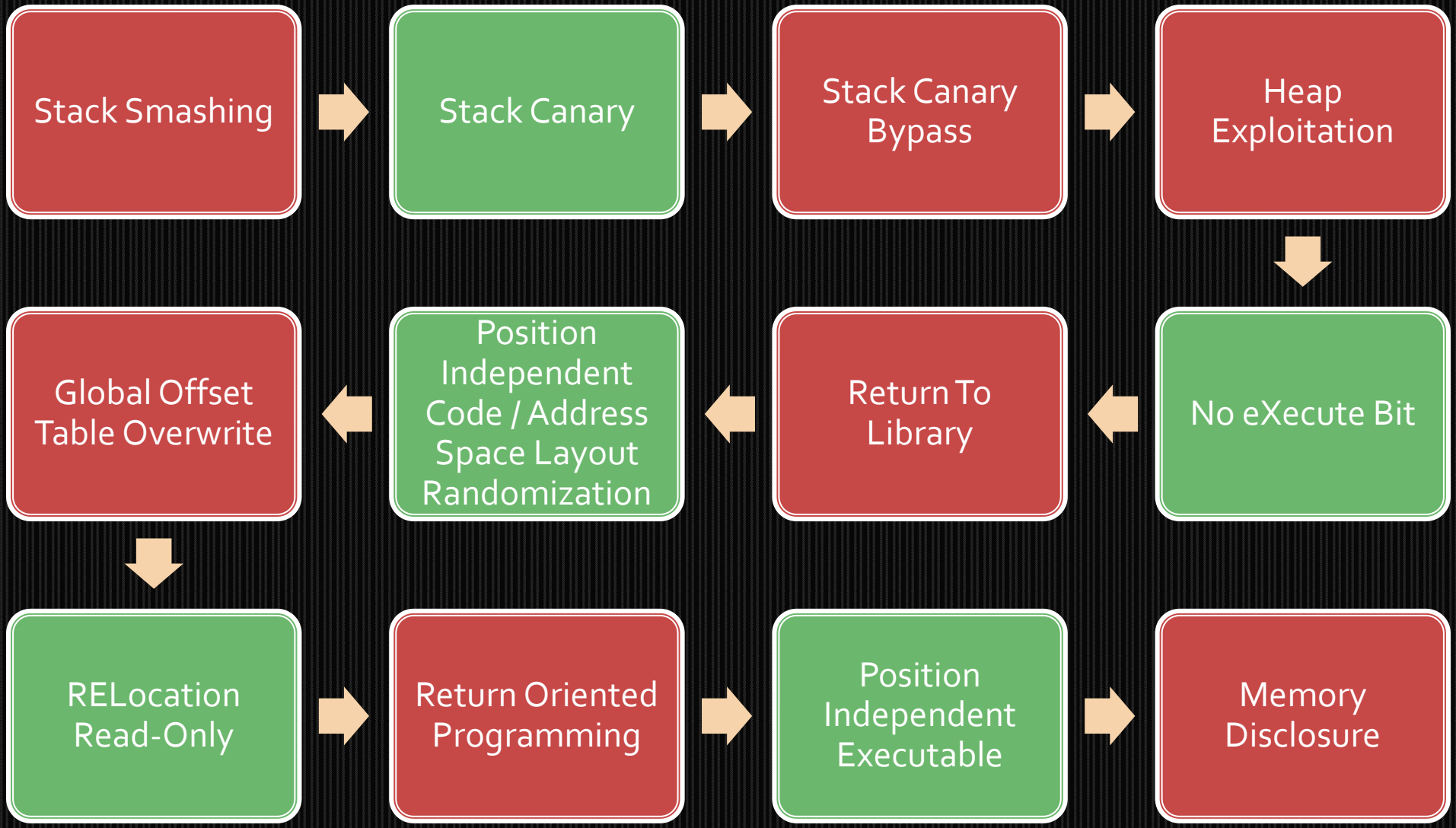
```
hake@ubuntu:~/tools$ ./checksec.sh --file ../code/thotcon/thotcon
RELRO           STACK CANARY      NX              PIE
Full RELRO      Canary found      NX enabled      PIE enabled
```

  - http://www.trapkit.de/tools/checksec.html

Memory Disclosure

- Locations of randomized sections are secret
- If an attacker can obtain a piece of memory
  - An attacker can calculate the random offset

The Arms Race



I Killed All The Bugs, Bro

Actually, no.

- Our protections are based on exploits that leverage known memory locations

But we achieved something

- A weaponized vulnerability for a release version of software we used would not land

Goals

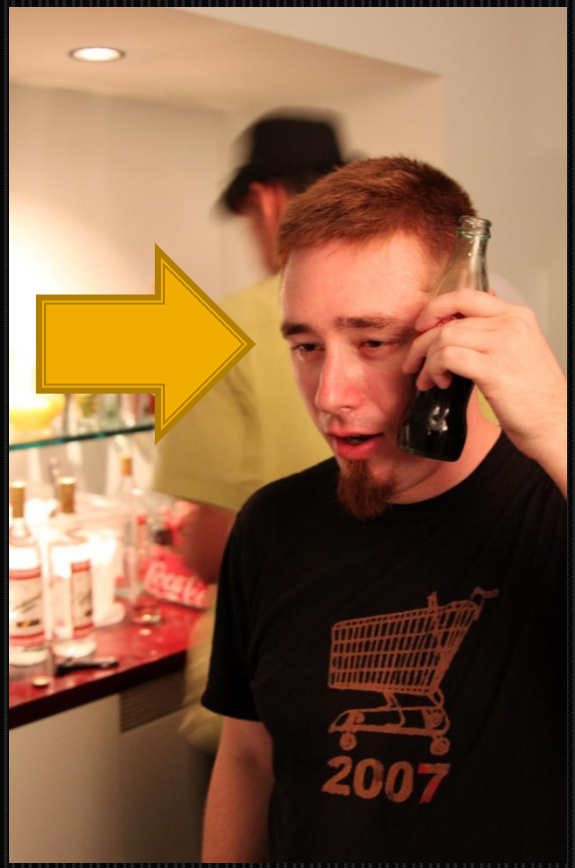
- Make it really fucking hard to land an exploit

Protected from
Jon Oberheide

Goals

- Make it really fucking hard to land an exploit

Protected from @SecureTips



Thanks

Jon Oberheide
Alex Sotirov
Mike Zusman
Amber Baldet
Chris Wysopal
Boris Kochergin
Jon Tomek
Beckie Mossman
Jeff Jarmoc
Nicholas Percoco
Zach Lanier
Hugo Fortier
Shawn Moyer
NECCDC Red Team
SophSec

Dan Guido
Erik Cabetas
Kelly Lum
Shyama Rose
Zane Lackey
Luis Garcia
Ben Nell
Apneet Jolly
Mario Heiderich
Dug Song
Spencer Pratt
Leigh Honeywell
Barnaby Jack
Dr. Raid
PainSec

Brandon Edwards
Paolo Soto
Marcin W.
Mark Dowd
Brian Holyfield
Efstratios Gavas
Zack Fasel
Leigh Hollowell
Kevin Nassery
Dave Goldsmith
Colin Ames
Dean De Beer
Ron Gutierrez
Dual Core
MOBiLEDiSCO

Dino Dai Zovi
Stephen Ridley
Aaron Portnoy
Peter Silberman
Joe Hemler
Michael Aiello
M. Jakubowski
Phil Da Silva
Justin Clarke
Matthieu Suiche
Raphael Mudge
Chris Valasek
Rafal Los
Tamari Kirtadze
busticati.org