Access violation when executing [41414141]

Buffer Overflow Exploiting Basics
CCC Göttingen
Sebastian Brabetz – 02.06.2015

1. Introduction

2. Buffer Overflow Theory

3. Live Demo

4. More Information

5. Questions

whoami

- Worked 4 years as Security- / Firewall-Administrator

- 1 year as IT Security Specialist

- Currently IT Security Engineer

- OSCP since 06/2014

- Blog: http://itunsecurity.wordpress.com

- Email: sebastian.brabetz@web.de

- IRC: irc.hackint.org / #ccc-ks

- Twitter: @teh_warriar

- Feel free to contact me with questions!
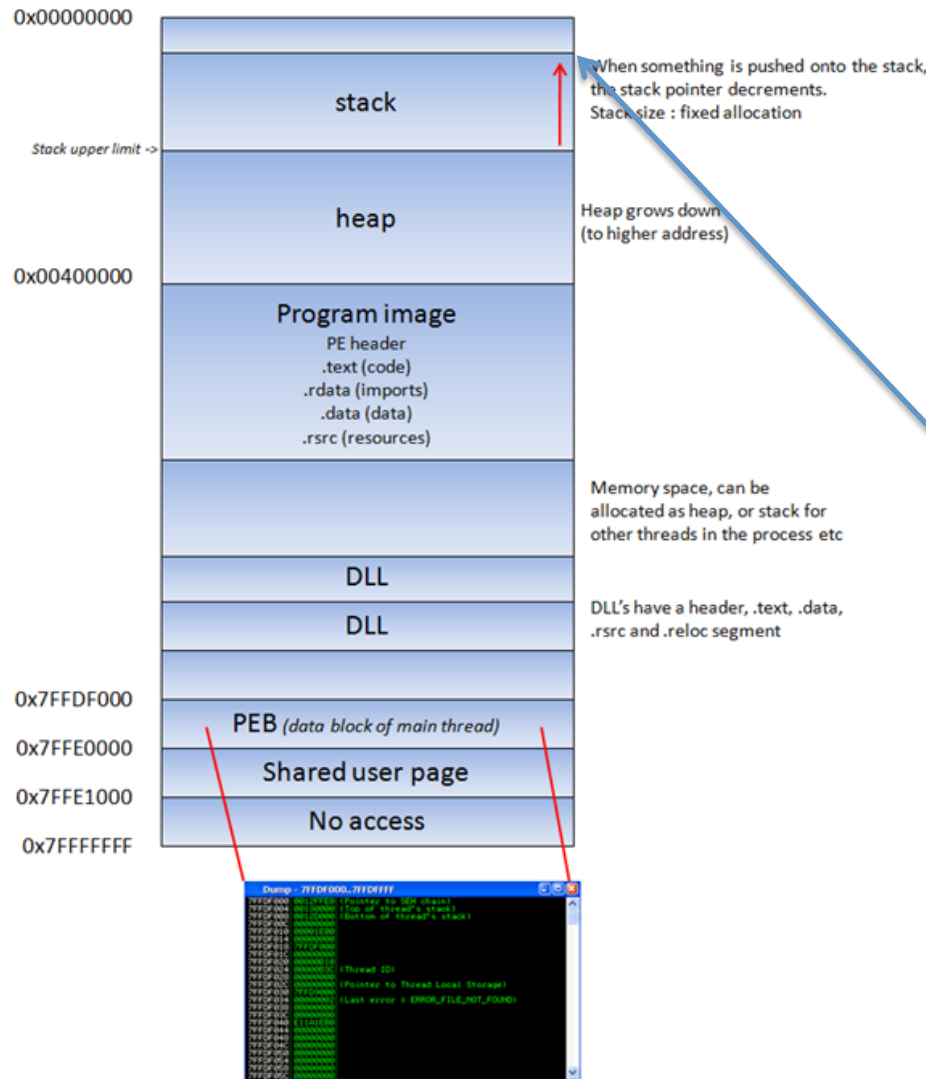
17AtwQtkVxYKReTDGiomhFFjFKL9J3G3P5
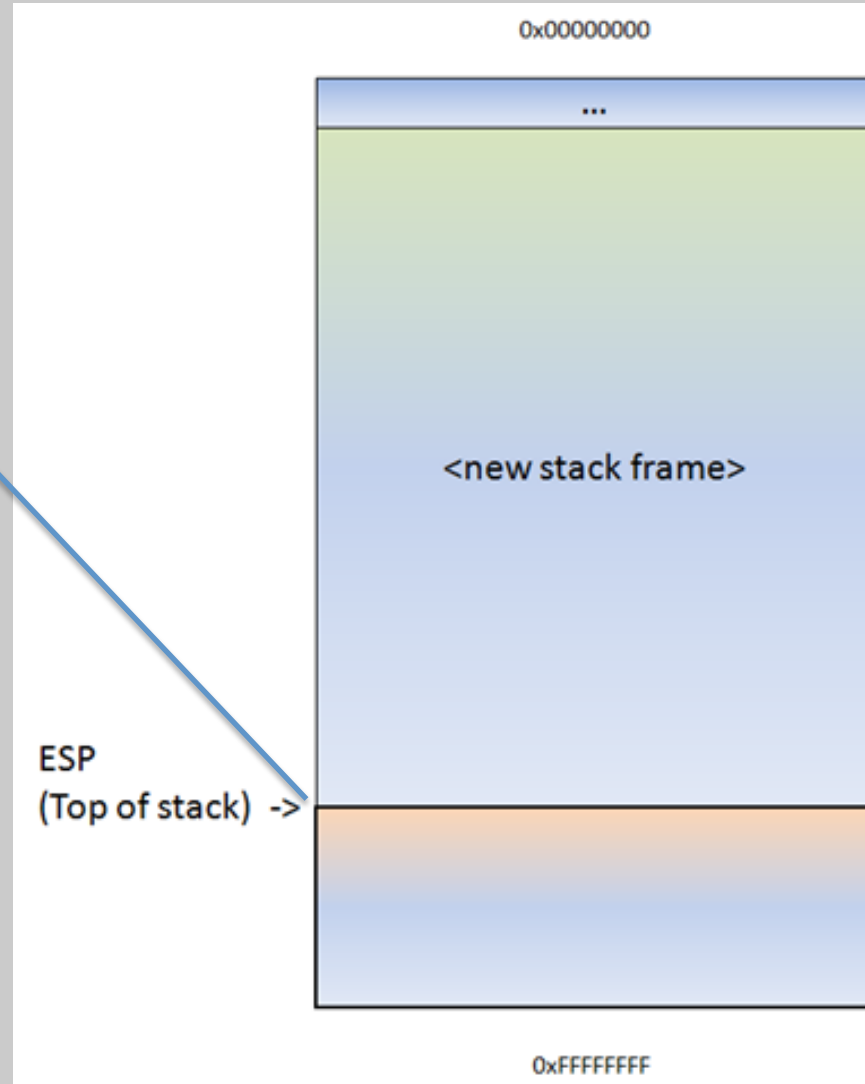
What is a (stack) Buffer Overflow?

Wikipedia:

In software, a stack buffer overflow or stack buffer overrun occurs when a program writes to a memory address on the program's call stack outside of the intended data structure; usually a fixed length buffer.

What is a (stack) Buffer Overflow?
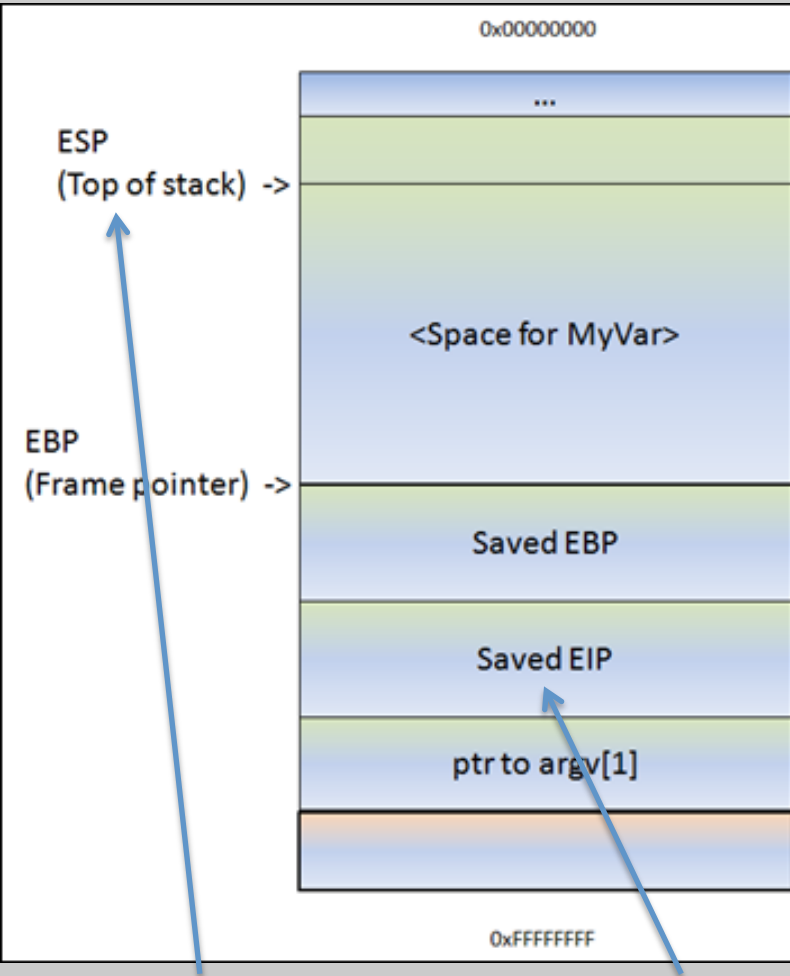
win32 process memory map

new stack frame will be placed on top of the stack

win32 process memory map:

Address	Region
0x00000000	
	stack
Stack upper limit ->	
	heap
0x00400000	
	Program image PE header .text (code) .rdata (imports) .data (data) .rsrc (resources)
	DLL
	DLL
0x7FFDF000	
	PEB (data block of main thread)
0x7FFE0000	
0x7FFE1000	Shared user page
0x7FFFFFFF	No access

When something is pushed onto the stack, the stack pointer decrements. Stack size : fixed allocation

Heap grows down (to higher address)

Memory space, can be allocated as heap, or stack for other threads in the process etc

DLL's have a header, .text, .data, .rsrc and .reloc segment

Dump - 7FFDF000..7FFDFFFF

new stack frame:

0x00000000

...

<new stack frame>
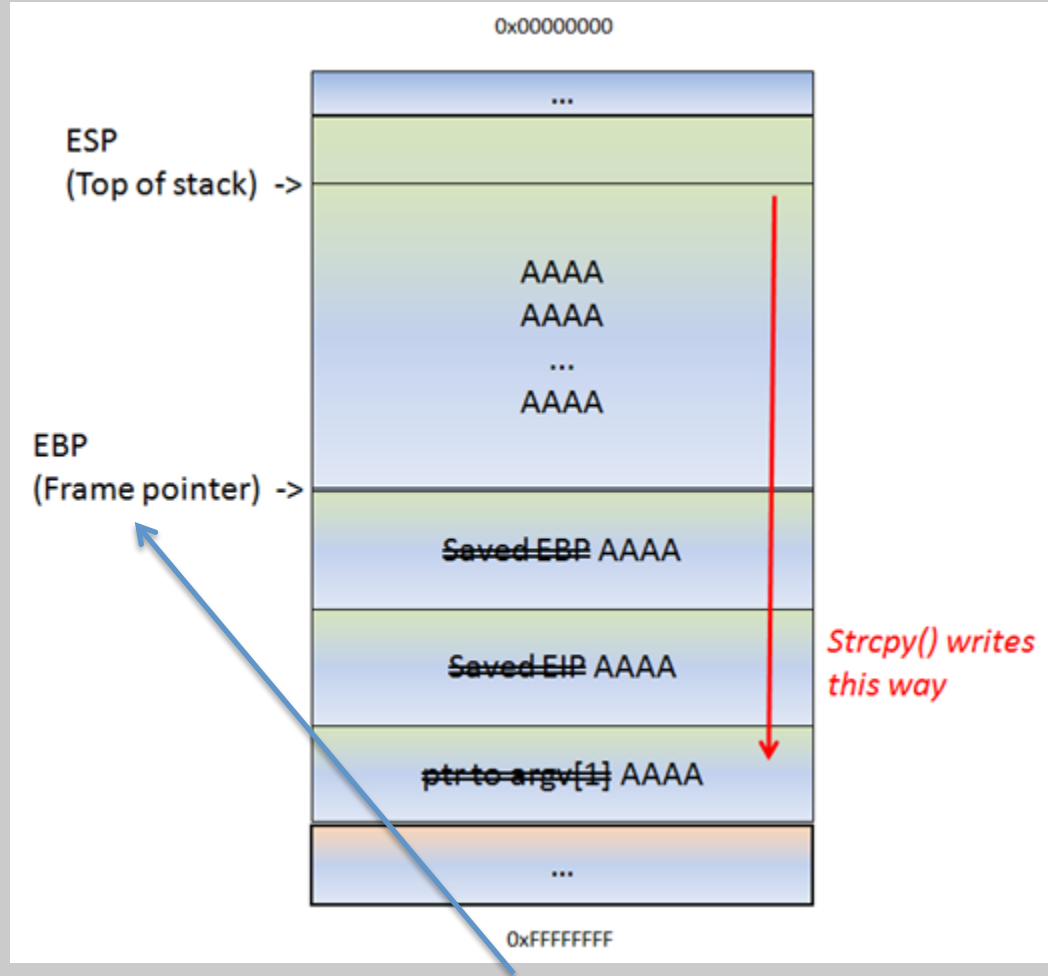
ESP
(Top of stack) ->

0xFFFFFFFF

What is a (stack) Buffer Overflow?

stack after function prologue

buffer overflows into EIP

ESP always points to the top of the current stack context

The EIP (saved during the function prologue) tells the CPU where to return after the function (gets loaded into the CPU during the function epilogue)

EBP always points to the base of the current stack context

Illustrations from http://www.corelan.be 6

Live Demo: The Vulnerable Program

SeattleLab Mailserver: SLMail Version 5.5.0.4433

SLMail POP3 Server Remote Buffer Overrun Vulnerability

www.securityfocus.com/bid/7519/info

| info | discussion | exploit | solution | references |

SLMail POP3 Server Remote Buffer Overrun Vulnerability

Bugtraq ID:	7519
Class:	Boundary Condition Error
CVE:	
Remote:	Yes
Local:	No
Published:	May 07 2003 12:00AM
Updated:	May 07 2003 12:00AM
Credit:	Discovery credited to "NGSSoftware Insight Security Research" <nisr@nextgenss.com>.
Vulnerable:	BVRP Software SLMail 5.1 .0.4420
Not Vulnerable:	BVRP Software SLMail 5.5

fail!

SLMail POP3 Server Remote Buffer Overrun Vulnerability

www.securityfocus.com/bid/7519/discuss

| info | discussion | exploit | solution | references |

SLMail POP3 Server Remote Buffer Overrun Vulnerability

It has been reported that a boundary condition error exists in SLMail POP3 Server. A remote attacker sending a password of excessive length, during the authentication process to the POP3 server, may cause a buffer overrun that could result in execution of malicious instructions and system compromise.

Great for demonstrations because:

- POP3 is an easy Protocoll
- Cleartext Protocoll
- Simple Stack Overflow vulnerability

„A remote attacker sending a password of exessive length, during the authentication process to the POP3 server, may cause a buffer overrun that could result in execution of malicious instructions and system compromise."

What could possibly go wrong?!

More information / Links

What you need to reproduce this:

- Windows XP SP2 German
  (other versions might have different jmp esp adresses!)
- Immunity Debugger (see next slide)
- Mona.py (see next slide)
- SLMail 5.5.0.04433
  http://slmail.software.informer.com/5.5/
- Kali Linux http://www.kali.org
- **This slides and all scripts are available for download on https://github.com/warriar/vortraege**

More information / Links

Python Socket Programming:

- http://openbook.galileocomputing.de/python/python_kapitel_20_001.htm

Learning Program Vulnserver:

- http://www.thegreycorner.com/2010/12/introducing-vulnserver.html
- http://rockfishsec.blogspot.de/2014/01/fuzzing-vulnserver-with-peach-3.html
- http://resources.infosecinstitute.com/fuzzing-vulnserver-discovering-vulnerable-commands-part-1/

Free Exploit Writing Tutorials from Corelan:

- https://www.corelan.be/index.php/articles/
- https://www.corelan.be/index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflows/

11 Part Tutorial including SEH, DEP and ASLR Bypassing, Egg Hunting, ROP and more!

Immunity Debugger and monay.py Plugin from Corelan:

- http://debugger.immunityinc.com
- https://www.corelan.be/index.php/2011/07/14/mona-py-the-manual/

German Book about Buffer Overflows on Amazon:

- http://www.amazon.de/Buffer-Overflows-Format-String-Schwachstellen-Funktionsweisen-Gegenmaßnahmen/dp/3898641929/ref=sr_1_1?ie=UTF8&qid=1410008572&sr=8-1&keywords=buffer+overflow+und+format+string+schwachstellen
- Free Look inside 1st Chapter (explains Stack Memory Management!): https://www.dpunkt.de/leseproben/2003/Auszug_aus_Kapitel_1.pdf

Questions?

Backup

CPU Register

8 General purpose registers for arythmetic operations and string operations as well as stack and base pointer to mark the stack context

Extended Instruction Pointer points to the next instruction the CPU will exectue
Not writeable from the program!

6 Segment registers to identify segments in the memory

EFLAGS Register can store and provide status informations about the running program

```
Registers (FPU)                              <    <    <    <    <
EAX 00000000
ECX 019A9EF0 ASCII "14/09/06 16:08:41 P3-0001: Illegal command
EDX 77C31B78 msvcrt.77C31B78
EBX 00000004
ESP 019AA154 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 41414141
ESI 00000000
EDI 00000001

EIP 42424242

C 0   ES 0023 32bit 0(FFFFFFFF)
P 1   CS 001B 32bit 0(FFFFFFFF)
A 1   SS 0023 32bit 0(FFFFFFFF)
Z 0   DS 0023 32bit 0(FFFFFFFF)
S 1   FS 003B 32bit 7FFAD000(FFF)
T 0   GS 0000 NULL
D 0
O 0   LastErr ERROR_SUCCESS (00000000)

EFL 00010296 (NO,NB,NE,A,S,PE,L,LE)
```

Stack

Abb. 1–1

Typisches Prozess-speicher-Layout eines C-Programms