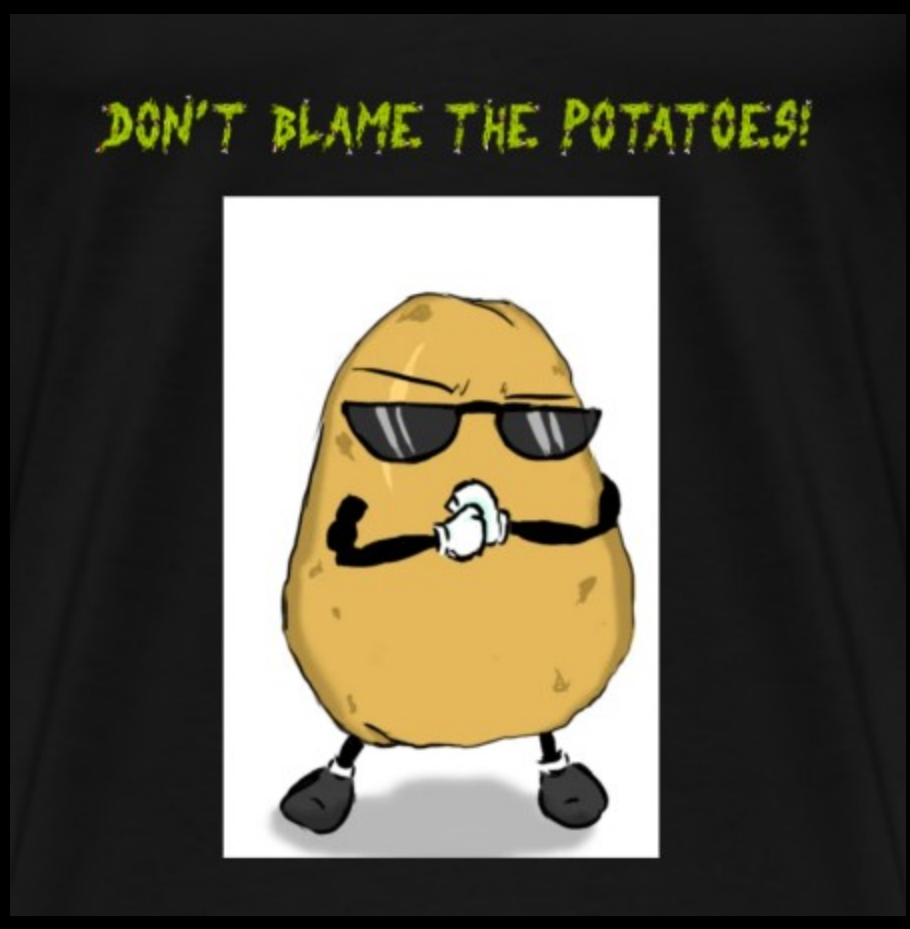
LocalPotato - When Swapping The Context Leads You To SYSTEM

10 February 2023 - Andrea Pierini & Antonio Cocomazzi



Intro

Here we are again with our new *potato flavor, the LocalPotato! This was a cool finding so we decided to create this dedicated website ;)

The journey to discovering the LocalPotato began with a hint from our friend Elad Shamir, who suggested examining the "Reserved" field in NTLM Challenge messages for potential exploitation opportunities.

After extensive research, it ended up with the “LocalPotato”, a not-so-common NTLM reflection attack in local authentication allowing for arbitrary file read/write. Combining this arbitrary file write primitive with code execution allowed us to achieve a full chain elevation of privilege from user to SYSTEM.

We reported our findings to the Microsoft Security Response Center (MSRC) on September 9, 2022, and it was resolved with the release of the January 2023 patch Tuesday and assigned the CVE number CVE-2023-21746.

Local NTLM Authentication

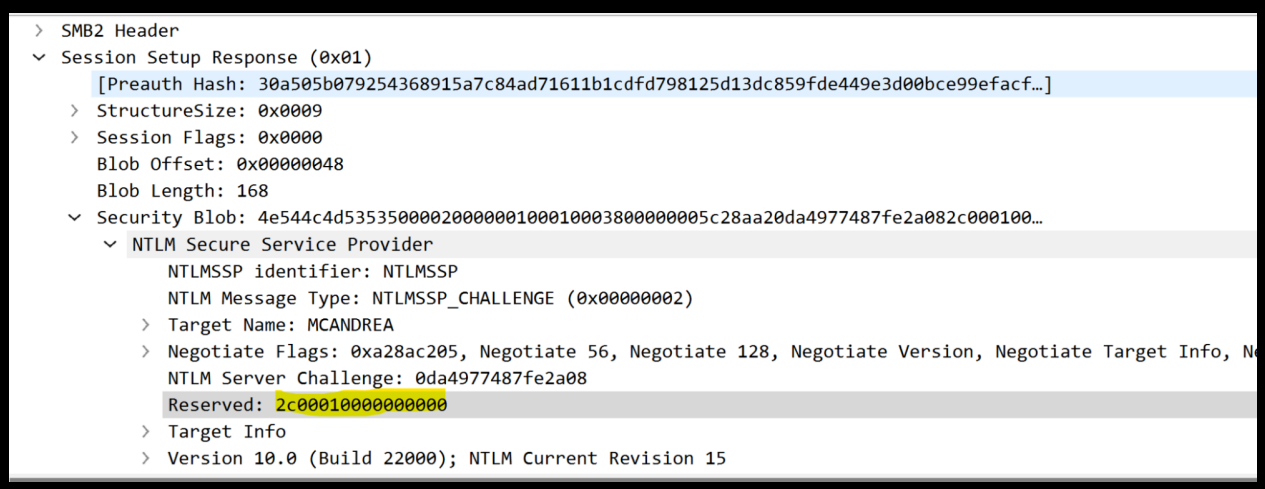
The NTLM authentication mechanism is part of the NTLMSSP (NTLM Security Support Provider), which is supported by the Windows security framework called SSPI (Security Support Provider Interface).
SSPI provides a flexible API for handling authentication tokens and supports several underlying providers, including NTLMSSP, SPNEGO, Kerberos, etc…

The NTLM authentication process involves the exchange of three types of messages (Type 1, Type 2, and Type 3) between the client and the server, processed by the NTLMSSP.
The SSPI authentication handshake abstracts away the details of NTLM and allows for a mechanism-independent means of applying authentication, integrity, and confidentiality primitives.

Local authentication is a special case of NTLM authentication in which the client and server are on the same machine.
The client acquires the credentials of the logged-in user and creates the Type 1 message,

which contains the workstation and domain name of the client.
The server examines the domain and workstation information and initiates local authentication if they match.
The client then receives the Type 2 message from the server and checks the presence of the "Negotiate Local Call" flag to determine if the security context handle is valid.
If it is, the default credentials are associated with the server context, and the resulting Type 3 message is empty.
The server then verifies that the security context is bound to a user, and if so, authentication is complete.

In summary, during local authentication, the “Reserved” field which is usually set to zero for non-local authentication in the NTLM type 2 message, will reference the local server context handle that the client should associate to.



In the above figure, we have highlighted the Reserved field containing the upper value of the context handle.

The Logic Bug

The NTLM authentication through SPPI is often misunderstood to involve direct mutual authentication between the client and server. However, in reality, the local authenticator (LSASS) is always involved, acting as the intermediary between the two.
It is responsible for creating the messages, checking the identity permissions, and generating the proper tokens.

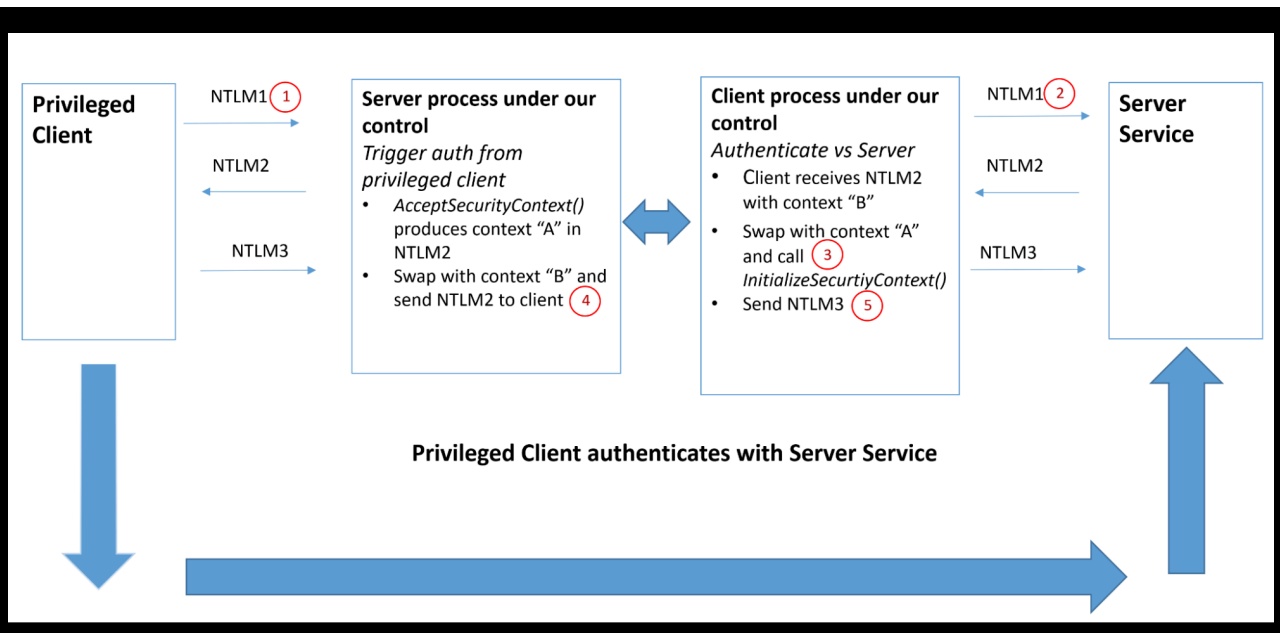
The objective of our research was to intercept a local NTLM authentication as a non-privileged local or domain user, and "swap" the contexts (NTLM "Reserved" field) with that of a privileged user (e.g. by coercing an authentication).

This would allow us to authenticate against a server service with these credentials, effectively exchanging the identity of our low-privileged user with a more privileged entity like SYSTEM. If successful, this would indicate that there are no checks in place to validate the Context exchanged between the two parties involved in the authentication.

The attack flow is as follows:

- Coerce the authentication of a privileged user against our server.
- Initiate an NTLM authentication of our client against a server service.
- Interception of the context "B" (Reserved bytes) of the NTLM Type 2 message coming from the server service where our unprivileged client is trying to authenticate.
- Retrieval of the context "A" (Reserved bytes) of the NTLM Type 2 message produced by our server when the privileged client tries to authenticate.
- Swap context A with B so that the privileged client will authenticate against the server service on behalf of the unprivileged client and vice versa.
- Retrieve both NTLM Type 3 empty response messages and forward them in the correct order to complete both authentication processes.
- As a result of the context swap, the Local Security Authority Subsystem (LSASS) will associate context B with the privileged identity and context A with the unprivileged identity. This results in the swap of contexts, allowing our malicious client to authenticate on behalf of the privileged user.

Below is a graphical representation of the attack flow:



To validate our assumptions about the context swap attack we did set up a custom scenario.
In our experiment, we used **two socket servers** and **two socket clients** to authenticate via NTLM with different users and exchange each other's "context".
Both parties were negotiating the NTLM authentication over a socket through SSPI.
In particular the clients with two calls to ***InitializeSecurityContext()*** and the servers with two calls to ***AcceptSecurityContext()***.

After some adjustments, we were successful in swapping identities and we were able to trick LSASS by associating the context with the "wrong" server.

To exploit this in a real-world scenario, we then had to find a useful trigger for coercing a privileged client and an appropriate server service.

The Triggers for coercing a privileged client

Based on our previous research, we identified two key triggers for coercing a privileged client: the BITS service attempting to authenticate as the SYSTEM user via HTTP on port 5985 (WinRM), and authenticated RPC/DCOM privileged user calls.

RogueWinRM is a technique that takes advantage of the BITS service's attempt to authenticate as the SYSTEM user via HTTP on port 5985. Since this port is not enabled by default on Windows 10/11, it provides an opportunity to implement a custom HTTP server that can capture the authentication flow. This allows us to obtain SYSTEM-level authentication.

RemotePotato0 is a method for coercing privileged authentication on a target machine by taking advantage of standard COM marshaling. In our scenario, we discovered three interesting default CLSIDs that authenticate as SYSTEM:

1. CLSID: {90F18417-F0F1-484E-9D3C-59DCEEE5DBD8}
   The ActiveX Installer Service "AxInstSv" is available only on Windows 10/11.
2. CLSID: {854A20FB-2D44-457D-992F-EF13785D2B51}
   The Printer Extensions and Notifications Service "PrintNotify" is available on Windows 10/11 and Server 2016/2019/2022.
3. CLSID: {A9819296-E5B3-4E67-8226-5E72CE9E1FB7}
   The Universal Print Management Service "McpManagementService" is available on Windows 11 and Server 2022.

By leveraging one of these triggers we could have the proper privileged identity to abuse.

Exploiting a server service

Initially, we tried to find a privileged candidate for our server service by examining the exposed RPC services, such as the Service Control Manager. However, we encountered a problem with local authentication to RPC services, as it is not possible to perform any reflection or relay attacks due to mitigations in the RPC runtime library (rpcrt4.dll).

As explained in this **blog post** by **James Forshaw**, Microsoft has added a mitigation in the RPC runtime to prevent authentication relay attacks from being successful.
This is done in "*SSECURITY_CONTEXT::ValidateUpgradeCriteria()*" by checking if the authentication for an RPC connection was from the local system, and if so, setting a flag in the security context. The server will then reject the RPC call if this flag is set, before any code is called in the server. The only way to bypass this check is to either have authentication from a non-local system or have an authentication level of RPC_C_AUTHN_LEVEL_PKT_INTEGRITY or higher, which requires knowledge of the session key for signing or encryption which of course mitigate effectively any relaying attempts.

Next, we turned our attention to the SMB server, with the goal of performing an arbitrary file write with elevated privileges.

The only requirement was that the SMB server should not require signing, which is the default for servers that are not Domain Controllers.

However, we found that the SMB protocol also has some mitigations in place to prevent cross-protocol reflection attacks.
This mitigation, also referred as **CVE-2016-3225**, has been released to address the **WebDAV->SMB** relaying attack scenario.
Basically, it requires the use of the SPN "cifs/127.0.0.1" when initializing local authentication through InitializeSecurityContext() for connecting to the SMB server, even for authentication protocols other than Kerberos, such as NTLM.
The main idea behind this mitigation is to prevent relaying local authentication between two different protocols, which would result in an SPN mismatch in the authenticator and ultimately lead to an access denied error.

According to **James Forshaw** article "**Windows Exploitation Tricks: Relaying DCOM Authentication**", it is possible to trick a privileged DCOM client into using an arbitrary Service Principal Name (SPN) to forge an arbitrary Kerberos ticket.
While this applies for Kerberos, it turns out that it can also affect the SPN setting in an NTLM authentication.
For this reason we chose to use the RPC/DCOM trigger for coercing a privileged client because we could return an arbitrary SPN in the binding strings of the Oxid resolver, thus bypassing the SMB anti-reflection mechanism.
All we needed to do was to set an SPN of "cifs/127.0.0.1" in the originating privileged client, which was not a problem thanks to our trigger:

```
hRes=CoInitialize(NULL);
WCHAR spnInfo[] = L"cifs/127.0.0.1";
authInfo.dwAuthnSvc = RPC_C_AUTHN_WINNT;
authInfo.pPrincipalName = spnInfo;  // this is important for relaying to SMB locally
hRes=CoInitializeSecurity(NULL, 1, &authInfo, NULL, RPC_C_AUTHN_LEVEL_CONNECT, RPC_C_IMP_LEVEL_IMPERSONATE, NULL, EOAC_DYNAMIC_CLOAKING, NULL);
```

In the end, we were able to write an arbitrary file with SYSTEM privileges and arbitrary contents.
The network capture of the SMB packets shows us successfully authenticating to the C$ share as the SYSTEM user and overwriting the file PrintConfig.dll:

smb || smb2

	Time	Source	Destination	Protocol	Length	Info
33	1.012105	127.0.0.1	127.0.0.1	SMB	117	Negotiate Protocol Request
35	1.012429	127.0.0.1	127.0.0.1	SMB2	296	Negotiate Protocol Response
37	1.012518	127.0.0.1	127.0.0.1	SMB2	152	Negotiate Protocol Request
39	1.012708	127.0.0.1	127.0.0.1	SMB2	296	Negotiate Protocol Response
41	1.012874	127.0.0.1	127.0.0.1	SMB2	186	Session Setup Request, NTLMSSP_NEGOTIATE
43	1.013029	127.0.0.1	127.0.0.1	SMB2	332	Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALLENGE
51	1.014371	127.0.0.1	127.0.0.1	SMB2	224	Session Setup Request, NTLMSSP_AUTH, User: \
53	1.014625	127.0.0.1	127.0.0.1	SMB2	120	Session Setup Response
55	1.014981	127.0.0.1	127.0.0.1	SMB2	148	Tree Connect Request Tree: \\127.0.0.1\c$
57	1.015171	127.0.0.1	127.0.0.1	SMB2	128	Tree Connect Response
63	1.015455	127.0.0.1	127.0.0.1	SMB2	272	Create Request File: windows\System32\spool\drivers\x64\3\PrintConfig.dll
67	1.015718	127.0.0.1	127.0.0.1	SMB2	200	Create Response File: windows\System32\spool\drivers\x64\3\PrintConfig.dll
69	1.016053	127.0.0.1	127.0.0.1	SMB2	101…	Write Request Len:10000 Off:0 File: windows\System32\spool\drivers\x64\3\PrintConfig.dll
71	1.016236	127.0.0.1	127.0.0.1	SMB2	128	Write Response
73	1.016306	127.0.0.1	127.0.0.1	SMB2	101…	Write Request Len:10000 Off:10000 File: windows\System32\spool\drivers\x64\3\PrintConfig.dll
75	1.016507	127.0.0.1	127.0.0.1	SMB2	128	Write Response
77	1.016580	127.0.0.1	127.0.0.1	SMB2	101…	Write Request Len:10000 Off:20000 File: windows\System32\spool\drivers\x64\3\PrintConfig.dll
79	1.016650	127.0.0.1	127.0.0.1	SMB2	128	Write Response
81	1.016702	127.0.0.1	127.0.0.1	SMB2	101…	Write Request Len:10000 Off:30000 File: windows\System32\spool\drivers\x64\3\PrintConfig.dll
83	1.016805	127.0.0.1	127.0.0.1	SMB2	128	Write Response
85	1.016844	127.0.0.1	127.0.0.1	SMB2	101…	Write Request Len:10000 Off:40000 File: windows\System32\spool\drivers\x64\3\PrintConfig.dll
87	1.016885	127.0.0.1	127.0.0.1	SMB2	128	Write Response
89	1.016924	127.0.0.1	127.0.0.1	SMB2	101…	Write Request Len:10000 Off:50000 File: windows\System32\spool\drivers\x64\3\PrintConfig.dll
91	1.017015	127.0.0.1	127.0.0.1	SMB2	128	Write Response
93	1.017058	127.0.0.1	127.0.0.1	SMB2	101…	Write Request Len:10000 Off:60000 File: windows\System32\spool\drivers\x64\3\PrintConfig.dll

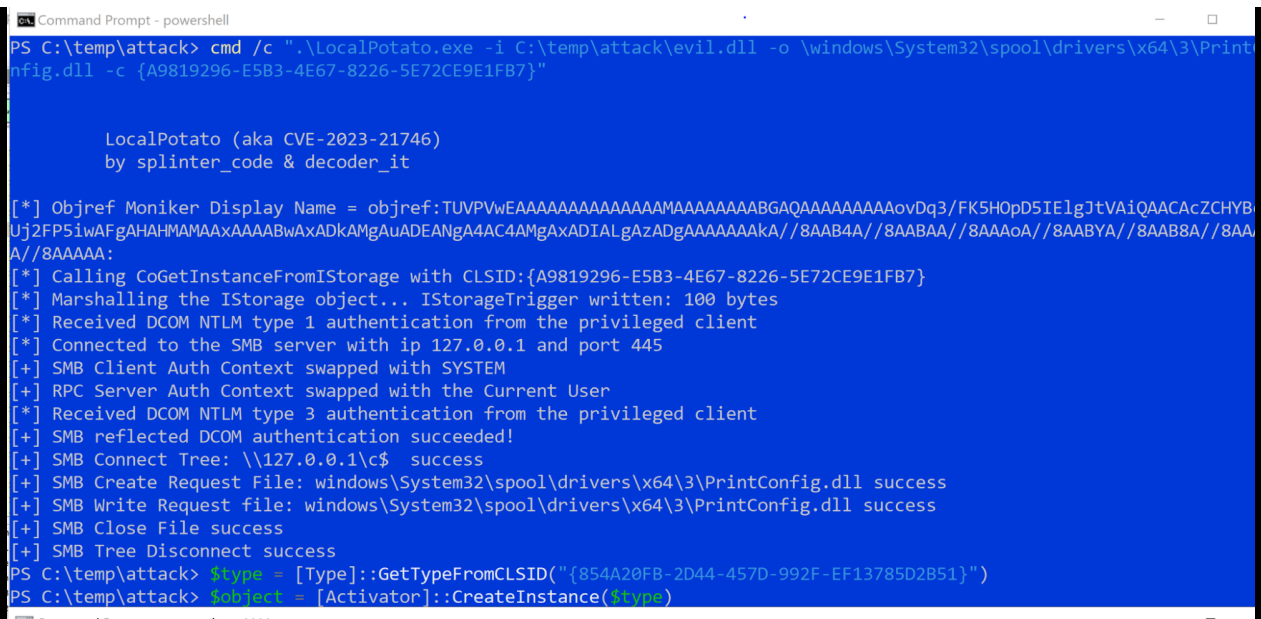
The POC

Creating a proof of concept for LocalPotato was a challenging task as it required writing SMB packets and sending them through the loopback interface for low-level NTLM authentication, accessing the local share, and finally writing a file.
We relied on **Wireshark** captures and Microsoft's **MS-SMB2** protocol specifications to complete the process. After multiple tests and code adjustments, we were finally successful.

To simplify the attack chain, we opted to eliminate the redirection to a non-Windows machine listening on port 135 and instead have the fake oxid resolver running on the Windows victim machine, so that the Potato trigger is local and the whole attack chain is fully local.

Just like we did in **JuicyPotatoNG**, we leveraged the **SPPI hooks** to manipulate NTLM messages coming to our COM server from the privileged client, enabling the Context Swapping.

Converting an arbitrary file write into EoP is relatively straightforward.
In our case, we utilized the McpManagementService CLSID on a Windows 2022 server, overwrote the printconfig.dll library, and instantiated the PrintNotify object.
This forced the service to load our malicious PrintConfig.dll, granting us a SYSTEM shell:

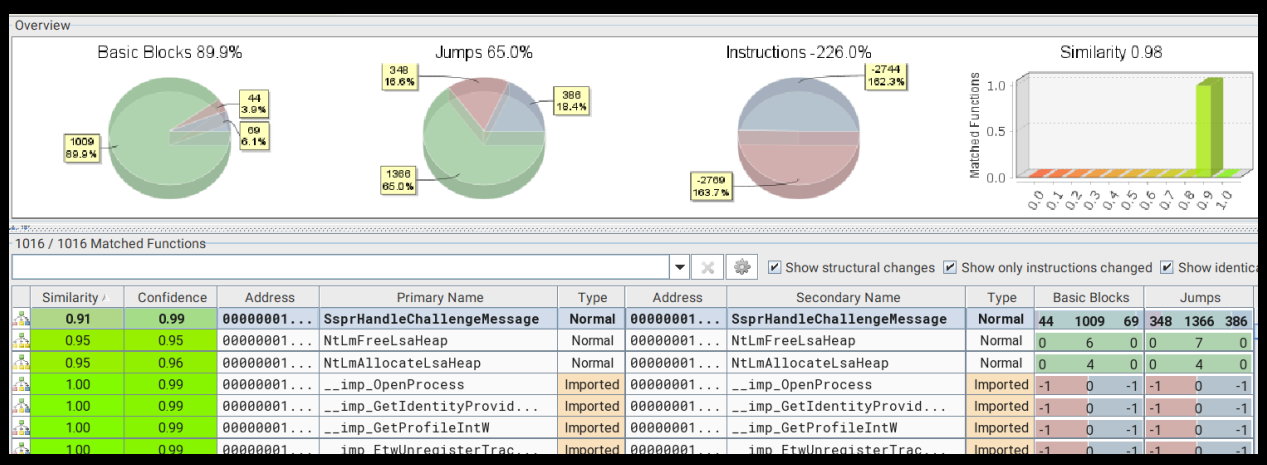


There are various methods to weaponize an arbitrary file write into code execution as SYSTEM, such as using an XPS Print Job or NetMan DLL Hijacking. So you are free to combine the LocalPotato primitive with what you prefer ;)

The LocalPotato POC is available at → https://github.com/decoder-it/LocalPotato

The Patch

The LocalPotato vulnerability was found in the NTLM authentication scheme. To locate the source of the vulnerability, we conducted a binary diff analysis of msv1_0.dll, the security package loaded into LSASS to handle all NTLM-related operations:



Similarity	Confidence	Address	Primary Name	Type	Address	Secondary Name	Type	Basic Blocks			Jumps		
0.91	0.99	00000001...	SsprHandleChallengeMessage	Normal	00000001...	SsprHandleChallengeMessage	Normal	44	1009	69	348	1366	386
0.95	0.95	00000001...	NtLmFreeLsaHeap	Normal	00000001...	NtLmFreeLsaHeap	Normal	0	6	0	0	7	0
0.95	0.96	00000001...	NtLmAllocateLsaHeap	Normal	00000001...	NtLmAllocateLsaHeap	Normal	0	4	0	0	4	0
1.00	0.99	00000001...	__imp_OpenProcess	Imported	00000001...	__imp_OpenProcess	Imported	-1	0	-1	-1	0	-1
1.00	0.99	00000001...	__imp_GetIdentityProvid...	Imported	00000001...	__imp_GetIdentityProvid...	Imported	-1	0	-1	-1	0	-1
1.00	0.99	00000001...	__imp_GetProfileIntW	Imported	00000001...	__imp_GetProfileIntW	Imported	-1	0	-1	-1	0	-1
1.00	0.99	00000001...	__imp_EtwUnregisterTrac...	Imported	00000001...	__imp_EtwUnregisterTrac...	Imported	-1	0	-1	-1	0	-1

The main focus of the analysis was the function SsprHandleChallengeMessage(), which handles NTLM challenges.

We observed the addition of a new check for the enabled feature “Feature_MSRC74246_Servicing_NTLM_ServiceBinding_ContextSwapping” when authentication occurs:

```
if ( wil::details::FeatureImpl<__WilFeatureTraits_Feature_MSRC74246_Servicing_NTLM_ServiceBinding_ContextSwapping>::
__private_IsEnabled((volatile signed __int32 *)&`wil::Feature<__WilFeatureTraits_Feature_MSRC74246_Servicing_NTLM_ServiceBinding_ContextSwapping>::GetImpl'::`2'::impl) )
{
  if ( v53 && (v110 = v53->Length, (_WORD)v110) ) //SPN is set
  {
    if ( v292 & 0x20000000 ) // ISC_REQ_UNVERIFIED_TARGET_NAME
    {
      v112 = 2;
      v113 = (unsigned __int64)(v110 + 2) >> 1;
      v111 = &word_1800767D8; // SPN is set NULL
      v114 = (wchar_t *)NtLmAllocateLsaHeap((unsigned int)(v110 + 2));
      v115 = v114;
      if ( !v114 )
      {
        v23 = 0;
        v22 = -1073741801;
        *((_DWORD *)v294 + 86) = 0;
        v116 = WPP_GLOBAL_Control;
        if ( WPP_GLOBAL_Control == &WPP_GLOBAL_Control || !(*((_BYTE *)WPP_GLOBAL_Control + 28) & 1) )
          goto LABEL_696;
        v117 = (_WORD)v114 + 93;
        goto LABEL_386;
      }
      memcpy_0(v114, v293->Buffer, v293->Length);
      v115[v113 - 1] = 0;
      if ( WPP_GLOBAL_Control != &WPP_GLOBAL_Control && *((_BYTE *)WPP_GLOBAL_Control + 28) & 2 )
        WPP_SF_S(
          *((_QWORD *)WPP_GLOBAL_Control + 2),
          0x5Eu,
          (__int64)&WPP_7d5a2267b9f23575f5c3e348859abb1f_Traceguids,
          v115);
      NtLmFreeLsaHeap(v115);
    }
    else
    {
      v111 = v53->Buffer;
      v112 = v53->Length;
    }
  }
  else
  {
    v111 = &word_1800767D8;
    v112 = 2;
  }
```

The check introduced by Microsoft ensures that if the ISC_REQ_UNVERIFIED_TARGET_NAME flag is set and an SPN is present, the SPN is

set to NULL.
This change effectively addresses the vulnerability by disrupting this specific exploitation scenario.
The SMB anti-reflection mechanism checks for the presence of a specific SPN, such as "cifs/127.0.0.1", to determine whether to allow or deny access. With the patch in place, a NULL value will be found, thus denying the authentication.
It's important to note that the ISC_REQ_UNVERIFIED_TARGET_NAME flag is passed and used by the DCOM privileged client, but prior to this patch, it was not taken into consideration for NTLM authentication.

Microsoft has released patches for supported versions of Windows, but don't worry if you have an older version. **0patch** provides fixes for LocalPotato for unsupported versions as well!

Conclusion

In conclusion, the LocalPotato vulnerability highlights the weaknesses of the NTLM authentication scheme in local authentication.

Microsoft has resolved the issue with the release of the patch **CVE-2023-21746**, but this fix may just be a workaround as detecting forged context handles in the NTLM protocol may be difficult.

It is important to note that this type of attack is not specific to the SMB or RPC protocols, but rather a general weakness in the authentication flow.
Other protocols that use NTLM as authentication method may still be vulnerable, provided exploitable services can be found.

What's next?

Well, to be honest, we ran out of ideas. But for sure, if we'll find something new it will be the "Golden Potato"!

Acknowledgments

Our thanks go to these two top security researchers:

- Elad Shamir (**@elad_shamir**) who gave us the initial idea and with whom we constantly discussed and debated this topic
- James Forshaw (**@tiraniddo**) who gave us useful hints when everything seemed to be lost