Secureworks®

Dumping NTHashes from Azure AD

@DrAzureAD

https://linkedin.com/in/nestori

https://aadinternals.com

Dr Nestori Syynimaa

Senior Principal Security Researcher

Secureworks® CTU™

Twitter/*Mastodon/BlueSky:*

@DrAzureAD *@infosec.exchange*
.bsk.social

MVP
Microsoft®
Most Valuable
Professional

MICROSOFT
MOST VALUABLE
Security
Researcher
ESTD. 2022
VOLUME

AADInternals

- Admin & hacking toolkit for Azure AD & Microsoft 365
- Open source:
  - https://github.com/gerenios/aadinternals
  - https://aadinternals.com/aadinternals
- MITRE ATT&CK
  - https://attack.mitre.org/software/S0677/

Groups That Use This Software

ID	Name	References
G0016	APT29	[5]

Contents

- Azure Active Directory Domain Services (AADDS)
- Attacking AADDS to dump NTHashes
- Researching AADDS
- From cloud admin to on-prem admin

@DrAzureAD

Secureworks®

Azure Active Directory Passwords

Azure AD passwords 1/2

- “many companies believe that Microsoft may have access to users’ passwords”[1]
- “plain-text version of the password is never and can never be exposed to Microsoft” [1]
- “same [hash] process as the password hash sync from AD to AAD” [2]

1. https://www.microsoft.com/en-us/security/blog/2019/05/30/demystifying-password-hash-sync/
2. https://learn.microsoft.com/en-us/answers/questions/848370/salting-and-hashing

Azure AD passwords 2/2

- “Azure AD stores the last four password hashes in the password hash history” [1]
- Password hash sync :
  - “MD4+salt+PBKDF2+HMAC-SHA256” [2] x 1000
- “Azure AD Connect can be configured to synchronize the required NTLM or Kerberos password hashes” [3]

1. https://learn.microsoft.com/en-us/troubleshoot/azure/active-directory/pwd-hash-sync-auto-enable#resolution
2. https://learn.microsoft.com/en-us/azure/active-directory/hybrid/connect/how-to-connect-password-hash-synchronization#detailed-description-of-how-password-hash-synchronization-works
3. https://learn.microsoft.com/en-us/azure/active-directory-domain-services/tutorial-configure-password-hash-sync#password-hash-synchronization-using-azure-ad-connect

Password hash synchronisation (PHS) example

- Password: ”Password”
  - MD4 hash: ”a4f49c406510bdcab6824ee7c30fd852”

- What is sent to cloud:


```
v1;PPH1_MD4,b3916922bb03db814db8,1000,7fc9805111346520512d935ae4c390efdf27c983146bec1f9a035457bf4c7ecb;
v1;PPH1_MD4,86076dc313578089f936,1000,f4b6468f4d0634a5095d5f973d0cad61befb4a1f9f25fb95e66fdc9c1dee5784;
v1;PPH1_MD4,dfab3e8c150c507e3266,1000,9fb4a1c199ddc7c1f0bf44ae7247f8fba9dfb782b81674b78d7b44f800a802f7;
v1;PPH1_MD4,6deef55196ca5f68e7b0,1000,d4a055305cdc951584f6576edfe7bc98847e3dae6eb32686f212209b1de5b85e;
v1;PPH1_MD4,88f52495b66dcb6fcd04,1000,b05353568ff80c55310b968bc507b898748c716a19e12fb7[illegible]40b19;
```

@DrAzureAD

Secureworks®

Azure Active Directory Domain Services

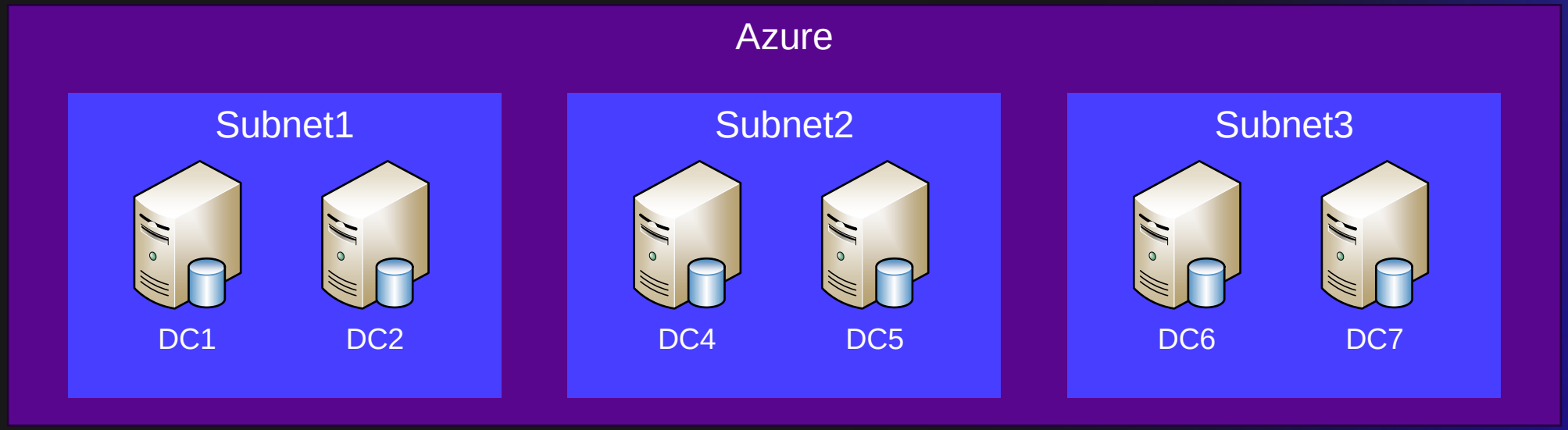
Azure AD Domain Services (AADDS)

- *enables you to use managed domain services—such as Windows Domain Join, group policy, LDAP, and Kerberos authentication—without having to deploy, manage, or patch domain controllers**
- Also known by technical name DC as-a-service (DCaaS)
- Allows using (legacy) software depending on DS in Azure
  - Users synchronised from Azure AD (needs at least one admin)
  - Extra users can be added manually
- Only one deployment per tenant!

* https://azure.microsoft.com/en-us/products/active-directory/ds

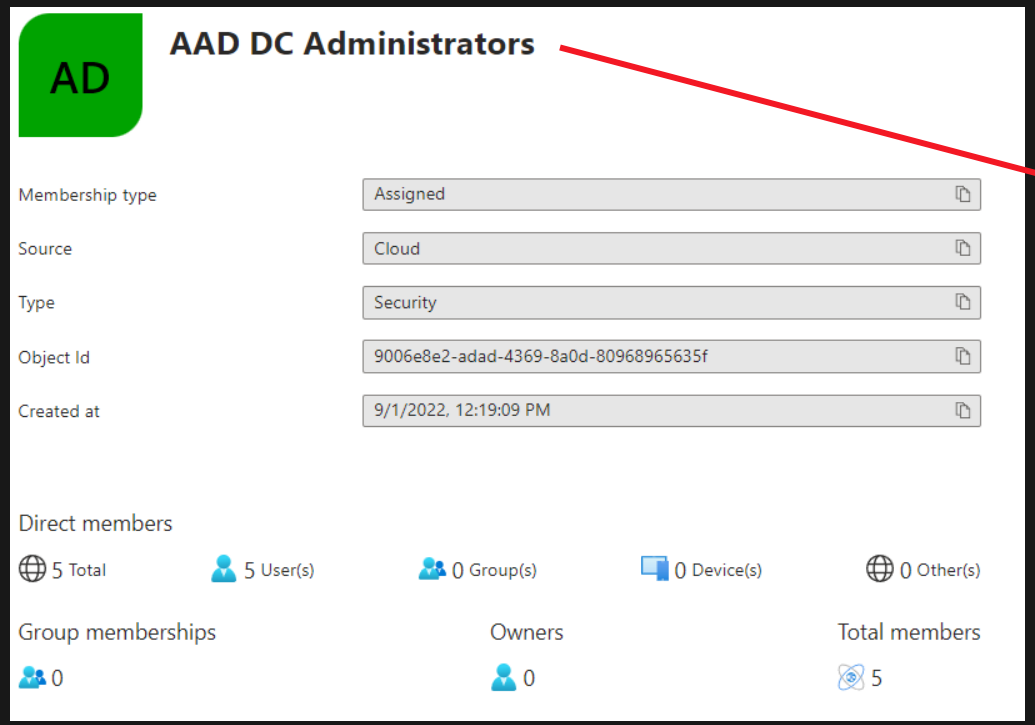
AADDS topology

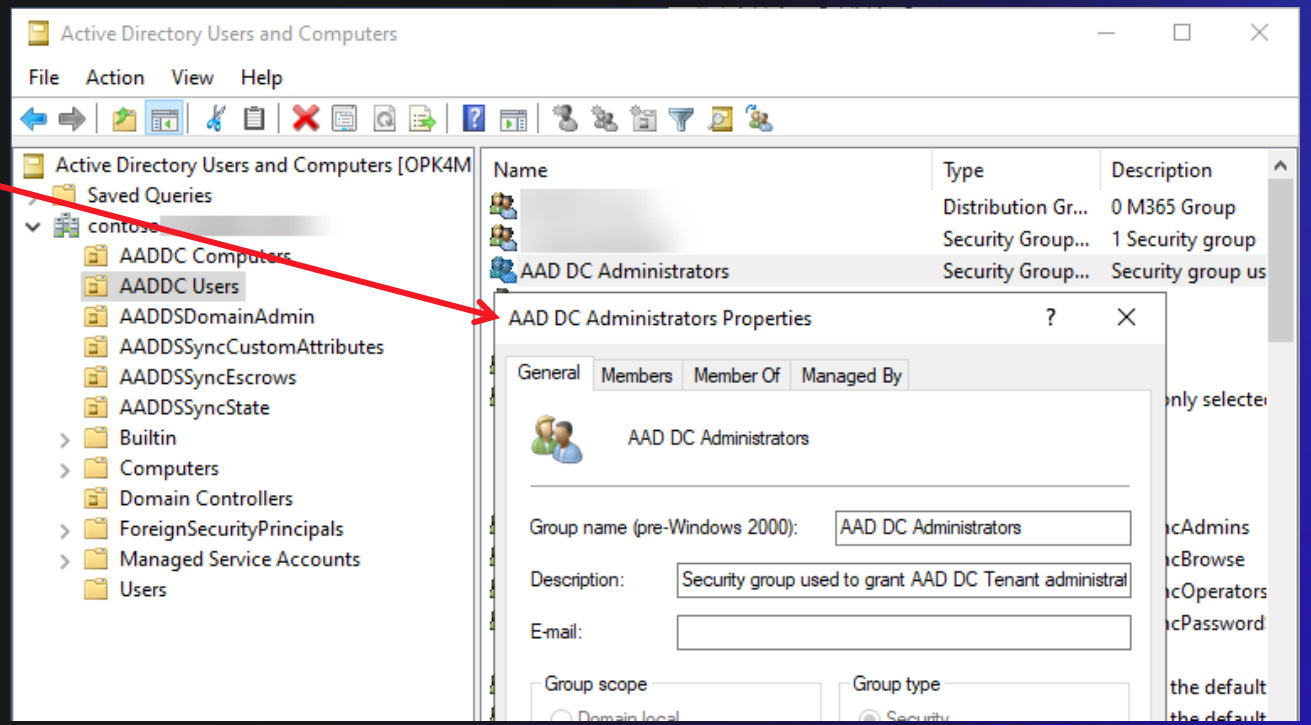
- 2 DCs deployed to chosen subnet
- Extra DCs via replica sets
  - One set per subnet
  - Replication via MS backbone network



AADDS administration 1/2

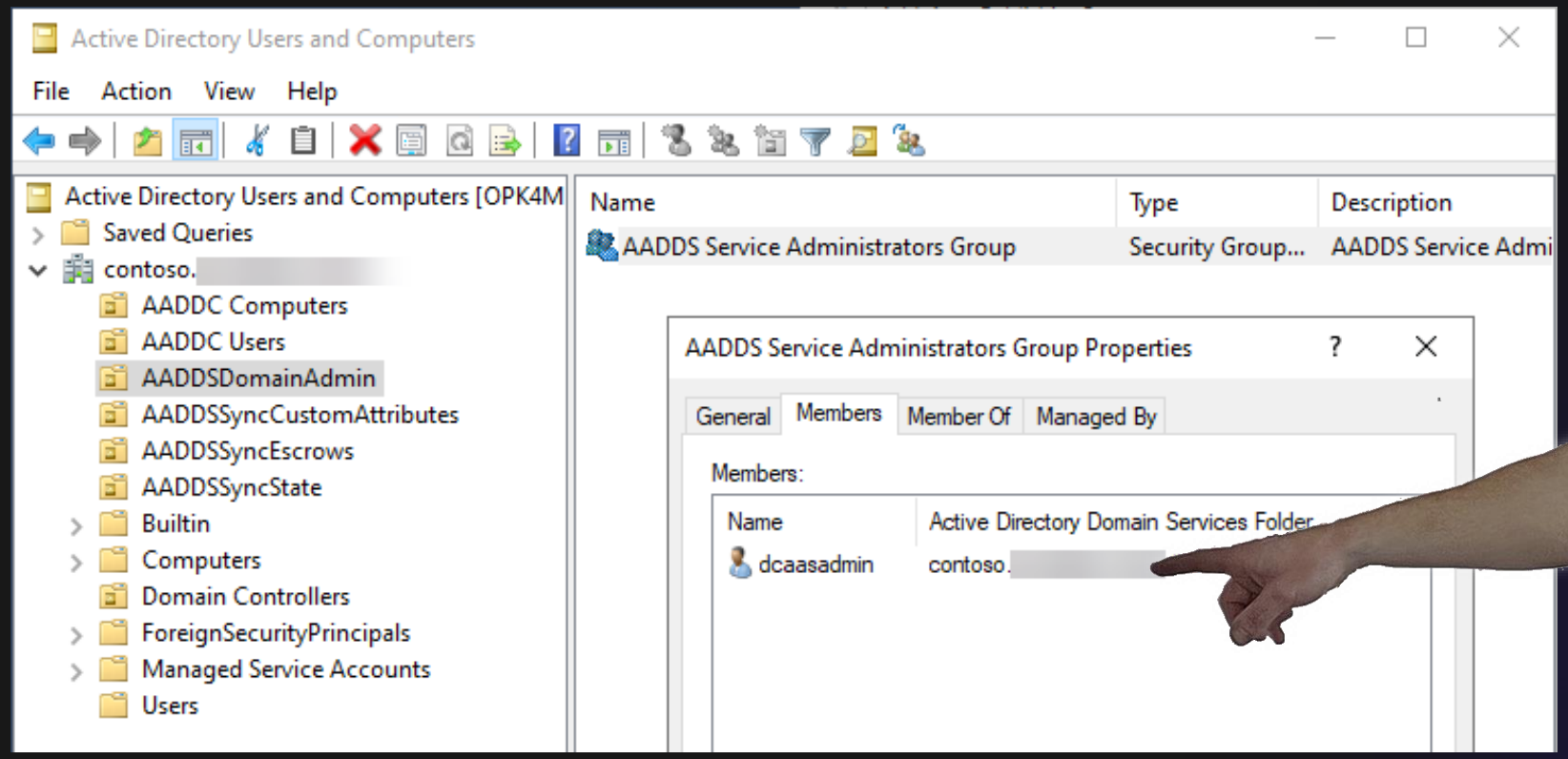
- Normal RSAT tools / PowerShell (from a domain joined computer)
- Admin permissions via `AAD DC Administrators` group



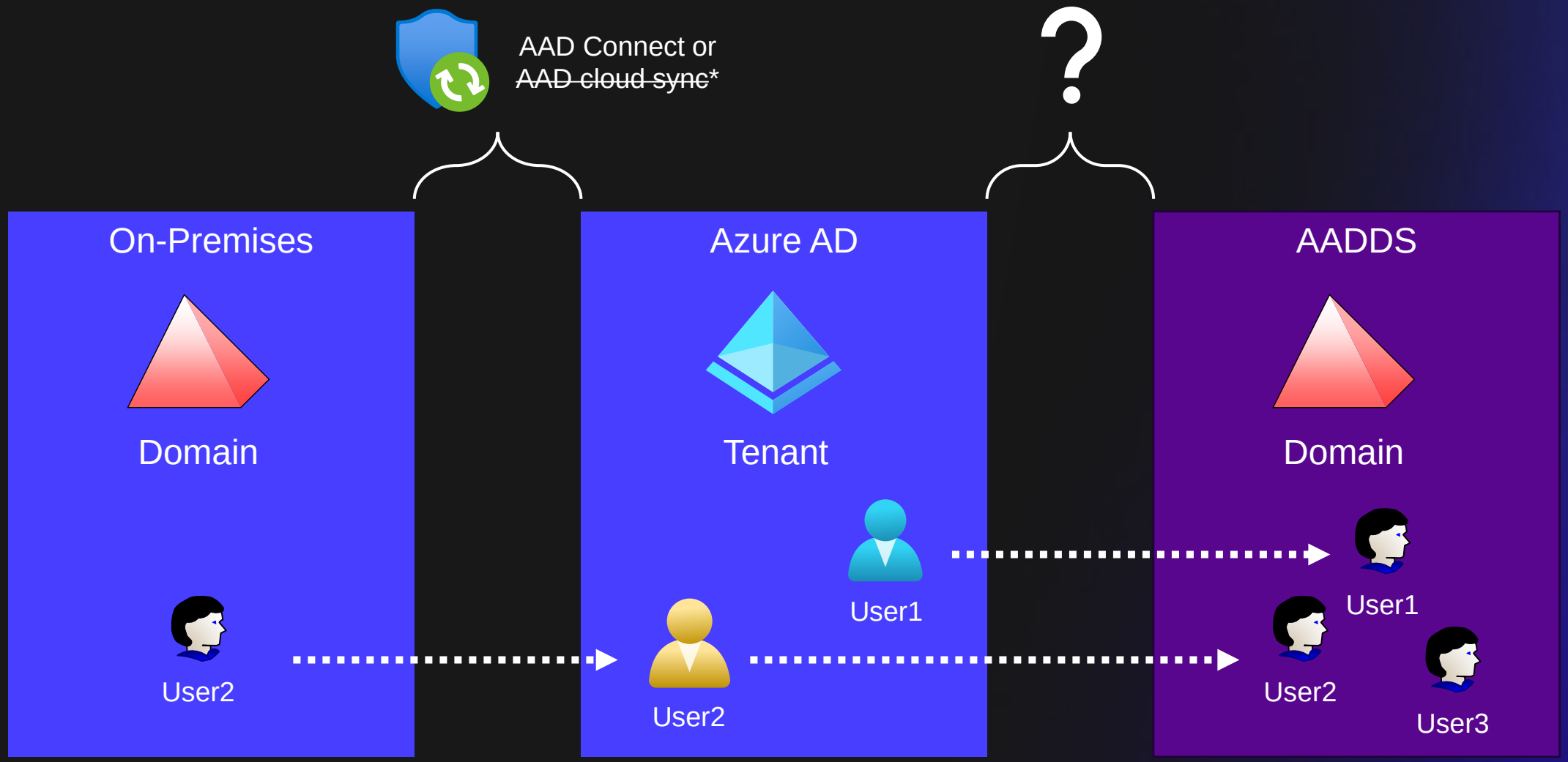


AADDS administration 2/2

- No Domain Admin permissions
- Built-in admin account: `dcaasadmin`



User deployment 1/2



* https://learn.microsoft.com/en-us/azure/active-directory-domain-services/tutorial-create-instance#enable-user-accounts-for-azure-ad-ds

User deployment 2/2

- AADDS user is a *copy* of Azure AD user
- If hybrid (on-prem originated):
  - Azure AD user is a *copy* of on-prem user
  - AADDS user is *not a replica* of on-prem user
    - Different SID,
    - (Possibly) different NetBIOS domain
- On-prem → Azure AD synced with AAD Connect
- Azure AD → AADDS sync technique unknown

Secureworks®

Attacking AADDS

Bring in the big guns!

Nevada Romsdahl

@nevadaromsdahl

Technical Lead

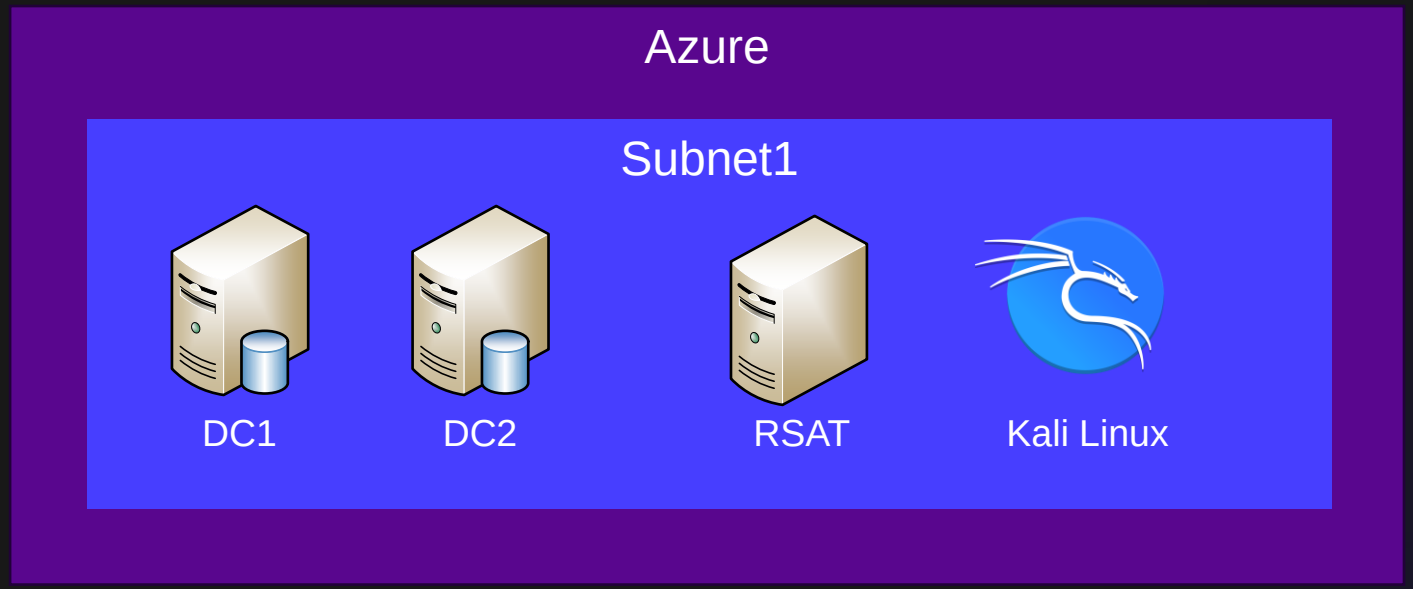
Tony Gore

@nullg0re

Security Researcher

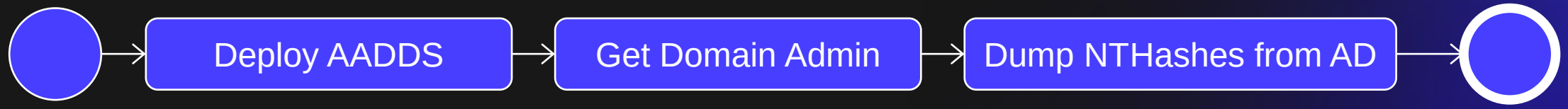
Attack topology

- Domain joined computer with RSAT tools
- Kali Linux



Attack results

- Used [redacted] and [redacted] techniques to get Domain Admin* 😈
- Dump NTHashes from AADDS DCs 🔥



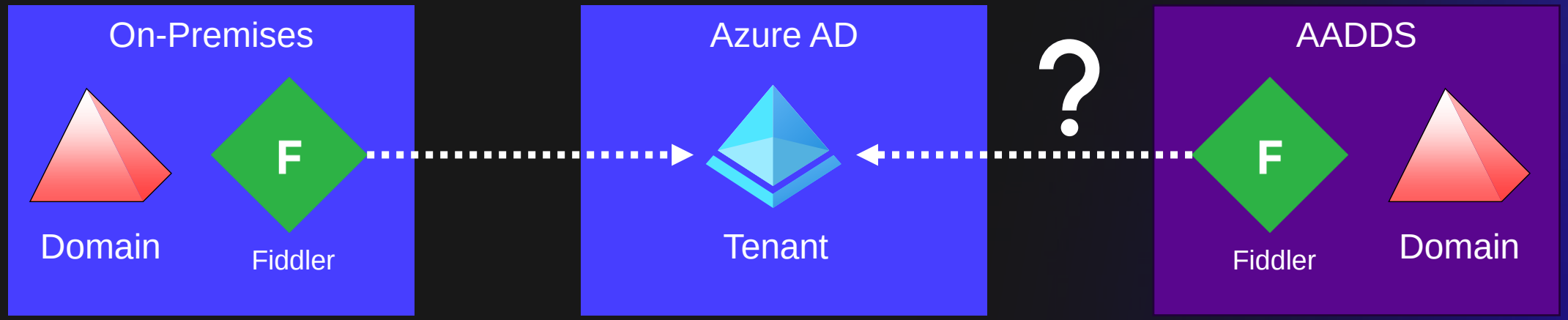
* Fixed by Microsoft on Mar 7th 2023

Secureworks®

Researching AADDS

Research topology

- Installed Telerik **Fiddler** web debugging proxy to on-prem and Azure
- Configured on-prem AAD Connect to use Fiddler
- Configured AADDS DCs to use Fiddler



On-prem → AzureAD 1/2

- Azure AD Connect `GetWindowsCredentialsSyncConfig`:
  - `EnableWindowsLegacyCredentials`
  - `EnableWindowsSupplementalCredentials`
  - `SecretEncryptionCertificate`

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope" xmlns:a="http://www.w3.org/2005/08/addressing">
    <s:Header>
        <a:Action s:mustUnderstand="1">http://schemas.microsoft.com/online/aws/change/2010/01/IProvisioningWebSe
        <a:RelatesTo>urn:uuid:87de2c3e-cdfc-4608-84cf-7dfac9973375</a:RelatesTo>
        <ActivityId CorrelationId="c4934d6a-d93f-4c70-820e-285b5bb119e2" xmlns="http://schemas.microsoft.com/200
    </s:Header>
    <s:Body>
        <GetWindowsCredentialsSyncConfigResponse xmlns="http://schemas.microsoft.com/online/aws/change/2010/01">
            <GetWindowsCredentialsSyncConfigResult xmlns:b="http://schemas.datacontract.org/2004/07/Microsoft.On
                <b:EnableWindowsLegacyCredentials>true</b:EnableWindowsLegacyCredentials>
                <b:EnableWindowsSupplementalCredentials>true</b:EnableWindowsSupplementalCredentials>
                <b:SecretEncryptionCertificate>MIIDGjCCAgKgAwIBAgIQHMqYdlWtfO/9a/lruRoD6DANBgkqhkiG9w0BAQsFADA9M
            </GetWindowsCredentialsSyncConfigResult>
        </GetWindowsCredentialsSyncConfigResponse>
    </s:Body>
</s:Envelope>
```

On-prem → AzureAD

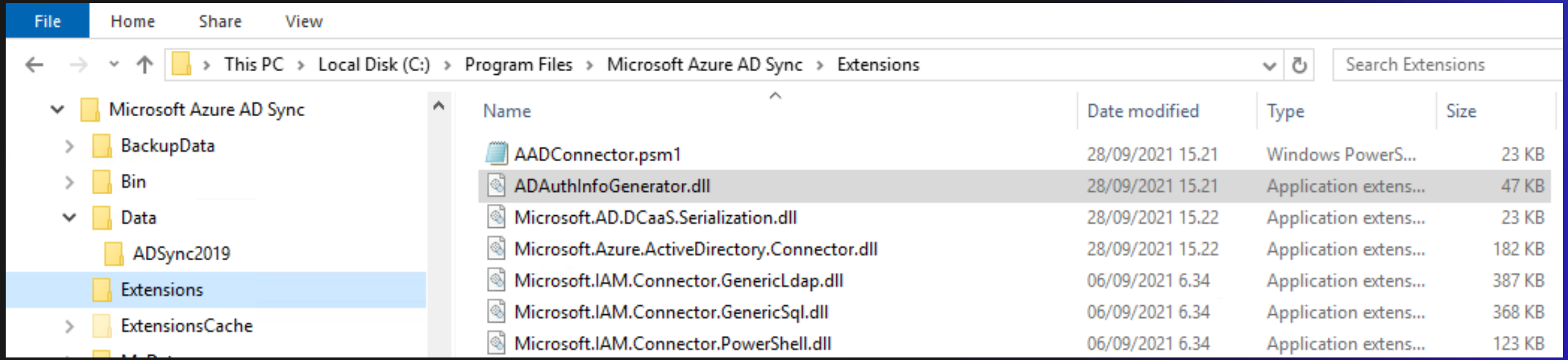
- `WindowsLegacyCredentials`
- `WindowsSupplementalCredentials`

Encrypted NTHash blob

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope" xmlns:a="http://www.w3.org/2005/08/addressing">
    <s:Header>
    <s:Body>
        <ProvisionCredentials xmlns="http://schemas.microsoft.com/online/aws/change/2010/01">
            <request xmlns:b="http://schemas.datacontract.org/2004/07/Microsoft.Online.Coexistence.Schema" xml
                <b:RequestItems>
                    <b:SyncCredentialsChangeItem>
                        <b:ChangeDate>2023-06-18T12:13:30.3462651Z</b:ChangeDate>
                        <b:CloudAnchor i:nil="true"/>
                        <b:CredentialData>v1;PPH1_MD4,cb22d8cb841e7ca2d026,1000,dc1cb9e72e7648586f9aa8838f37bb
                        <b:ForcePasswordChangeOnLogon>false</b:ForcePasswordChangeOnLogon>
                        <b:SourceAnchor>Eo+jOAQeqUi6rEv8+Yu1Rq==</b:SourceAnchor>
                        <b:WindowsLegacyCredentials>AQAAAAAAAABAAAAAQAAABQAAAAAQAAEAAAACAAAAB5z79RLF+VNDq6ky
                        <b:WindowsSupplementalCredentials>AQAAAAAAAABAAAAAQAAABQAAAAAQAAEAAAADAFAAB5z79RLF+V
                    </b:SyncCredentialsChangeItem>
                </b:RequestItems>
            </request>
        </ProvisionCredentials>
    </s:Body>
</s:Envelope>
```

What's inside the blobs?

- AAD Connect uses `GetADAuthInfo` to encrypt legacy credential blobs
- Located at `<Program Files>\Microsoft Azure AD Sync\Extensions\ADAuthInfoGenerator.dll`



```
52      // Token: 0x06000059 RID: 89 RVA: 0x000013D8 File Offset: 0x000007D8
53      private unsafe byte[] GetADAuthInfo(byte[] properties, uint flags, X509Certificate2 certificate)
54      {
55          byte[] array = null;
56          byte[] array2 = null;
57          byte[] array3 = null;
58          byte* ptr = null;
59          uint num = 0;
60          byte[] array4 = null;
61          byte[] result;
62          try
63          {
64              _BYTEARRAY bytearray;
65              initblk(ref bytearray, 0, 16L);
66              _BYTEARRAY bytearray2;
67              initblk(ref bytearray2, 0, 16L);
68              _BYTEARRAY bytearray3;
69              initblk(ref bytearray3, 0, 16L);
70              _BYTEARRAY bytearray4;
71              initblk(ref bytearray4, 0, 16L);
72              array = null;
73              array2 = null;
74              array3 = null;
75              EncryptionKeyAlgorithm encryptionKeyAlgorithm;
76              PayloadEncryptionKeyAlgorithm payloadEncryptionKeyAlgorithm;
77              array4 = this.Encrypt(properties, certificate, ref array3, ref array, ref array2, &encryptionKeyAlgorithm, &payloadEncryptionKeyAlgorithm);
78              ref byte byte& = ref array4[0];
79              bytearray4 = ref byte&;
80              *(ref bytearray4 + 8) = array4.Length;
81              ref byte byte&2 = (0 >= array.Length) ? 0L : ref array[0];
82              bytearray = ref byte&2;
83              *(ref bytearray + 8) = array.Length;
84              ref byte byte&3 = (0 >= array2.Length) ? 0L : ref array2[0];
85              bytearray2 = ref byte&3;
86              *(ref bytearray2 + 8) = array2.Length;
87              ref byte byte&4 = (0 >= array3.Length) ? 0L : ref array3[0];
88              bytearray3 = ref byte&4;
89              *(ref bytearray3 + 8) = array3.Length;
90              int num2 = <Module>.WriteADAuthInfo(flags, &bytearray3, &bytearray, &bytearray2, encryptionKeyAlgorithm, &bytearray4, payloadEncryptionKeyAlgorithm, nu
```

Encrypt the pwd

Create the blob

```
195     // Token: 0x0600005B RID: 91 RVA: 0x000010EC File Offset: 0x000004EC
196     private unsafe byte[] Encrypt(byte[] data, X509Certificate2 certificate, ref byte[] thumbPrint, ref byte[] key, ref byte[] iv, EncryptionKeyAlgorithm* encryptionKeyAlgorithm, PayloadEncryptionKeyAlgorithm*
          payloadEncryptionKeyAlgorithm)
197     {
198         if (null == certificate)
199         {
200             *encryptionKeyAlgorithm = (EncryptionKeyAlgorithm)0;
201             *payloadEncryptionKeyAlgorithm = (PayloadEncryptionKeyAlgorithm)0;
202             key = new byte[0];
203             iv = new byte[0];
204             thumbPrint = new byte[0];
205             int num = data.Length;
206             byte[] array = new byte[num];
207             Array.Copy(data, array, num);
208             return array;
209         }
210         MemoryStream memoryStream = null;
211         ICryptoTransform cryptoTransform = null;
212         CryptoStream cryptoStream = null;
213         byte[] result;
214         try
215         {
216             AesCryptoServiceProvider aesCryptoServiceProvider = new AesCryptoServiceProvider();
217             aesCryptoServiceProvider.KeySize = 256;
218             aesCryptoServiceProvider.BlockSize = 128;
219             aesCryptoServiceProvider.Mode = CipherMode.CBC;
220             aesCryptoServiceProvider.Padding = PaddingMode.PKCS7;
221             aesCryptoServiceProvider.GenerateKey();
222             aesCryptoServiceProvider.GenerateIV();
223             byte[] array2 = (certificate.PublicKey.Key as RSACryptoServiceProvider).Encrypt(aesCryptoServiceProvider.Key, true);
224             Array.Reverse(array2);
225             memoryStream = new MemoryStream();
226             cryptoTransform = aesCryptoServiceProvider.CreateEncryptor();
227             cryptoStream = new CryptoStream(memoryStream, cryptoTransform, CryptoStreamMode.Write);
228             cryptoStream.Write(data, 0, data.Length);
229             cryptoStream.FlushFinalBlock();
230             thumbPrint = new byte[certificate.Thumbprint.Length / 2];
231             for (int i = 0; i < thumbPrint.Length; i++)
232             {
233                 byte b = byte.Parse(certificate.Thumbprint.Substring(i * 2, 2), NumberStyles.AllowHexSpecifier);
234                 thumbPrint[i] = b;
235             }
236             *encryptionKeyAlgorithm = (EncryptionKeyAlgorithm)1;
237             *payloadEncryptionKeyAlgorithm = (PayloadEncryptionKeyAlgorithm)1;
238             iv = aesCryptoServiceProvider.IV;
239             key = array2;
240             result = memoryStream.ToArray();
```

Encryption details

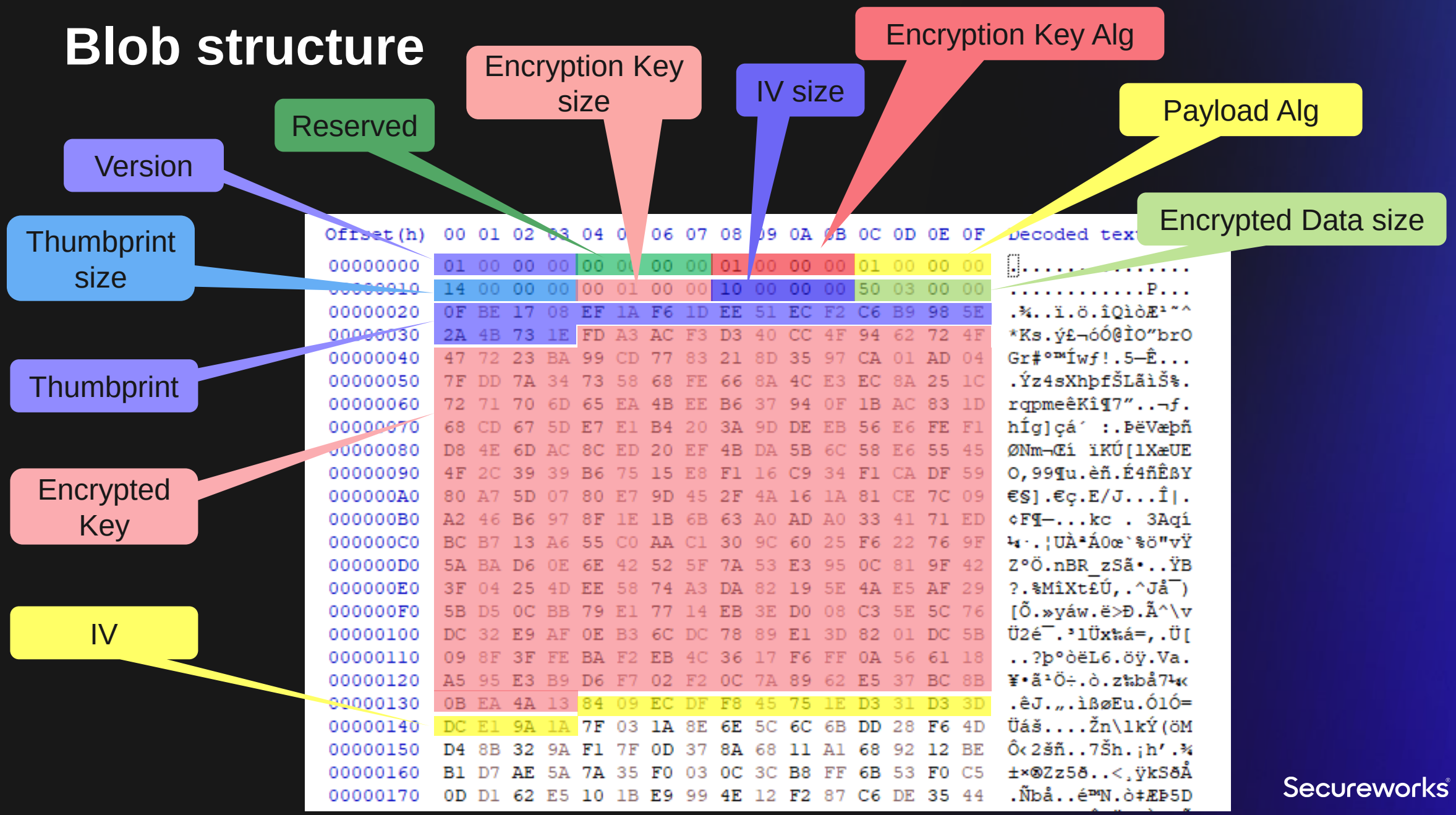
Encrypt & reverse the key

Blob structure

Decompile: ?WriteADAuthInfo@@$$FYAJKPEAU_BYTEARRAY@@00W4EncryptionKeyAlgorithm@@0W4PayloadEncryptionKeyAlgorithm@@PEAEPEAK@Z - (adauthinfogenerator.dll)

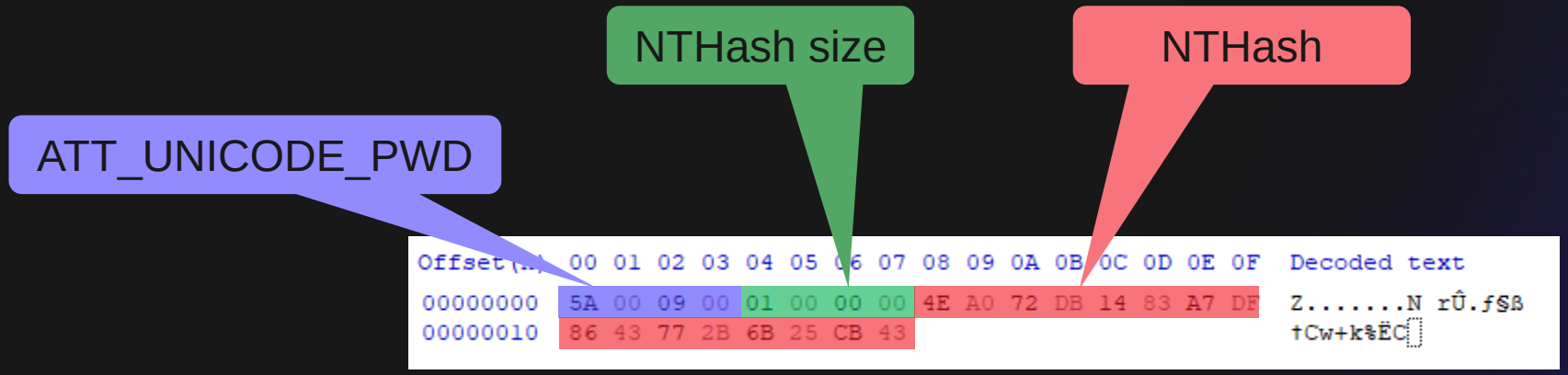
```
1
2  /* WARNING: Function: _guard_dispatch_icall replaced with injection: guard_dispatch_icall */
3  /* WARNING: Exceeded maximum restarts with more pending */
4
5  undefined4 __cdecl
6  ?WriteADAuthInfo@@$$FYAJKPEAU_BYTEARRAY@@00W4EncryptionKeyAlgorithm@@0W4PayloadEncryptionKeyAlgorith
   m@@PEAEPEAK@Z
7            (ulong flags,_BYTEARRAY *bThumbPrint,_BYTEARRAY *bKey,_BYTEARRAY *bIV,
8            EncryptionKeyAlgorithm encryptionKeyAlgorithm,_BYTEARRAY *bCipherData,
9            PayloadEncryptionKeyAlgorithm payloadEncryptionKeyAlgorithm,uchar *pbRetVal,
10           ulong *cbRetValSize)
11
12 {
13   int iVar1;
14   undefined8 uVar2;
15   ulong uKeySize;
16   ulong uThumbPrintSize;
17
18   if ((flags & 0xfffffffe) != 0) {
19     return 0x80070057;
20   }
21   if (pbRetVal != (uchar *)0x0) {
22     *(undefined4 *)(pbRetVal + 4) = 0;
23     *(EncryptionKeyAlgorithm *)(pbRetVal + 8) = encryptionKeyAlgorithm;
24     *(undefined4 *)pbRetVal = 1;
25     *(ulong *)(pbRetVal + 0x1c) = bCipherData->Size;
26     *(PayloadEncryptionKeyAlgorithm *)(pbRetVal + 0xc) = payloadEncryptionKeyAlgorithm;
27     *(ulong *)(pbRetVal + 0x14) = bKey->Size;
28     *(ulong *)(pbRetVal + 0x10) = bThumbPrint->Size;
29     *(ulong *)(pbRetVal + 0x18) = bIV->Size;
30     iVar1 = memcpy_s();
31     if (iVar1 != 0) {
32       if (g_ADAuthInfoTracer == (_func__cdecl_void_TraceType_char_ptr_int_T_HRESULT *)0x0) {
33         return 0x80004005;
34       }
35       uVar2 = 0x40;
36       goto LAB_1800036da;
37     }
```

Blob structure



NTHash blob structure

- `unicodePwd` attribute from the AD
- Example blob for “Summer2023!”:



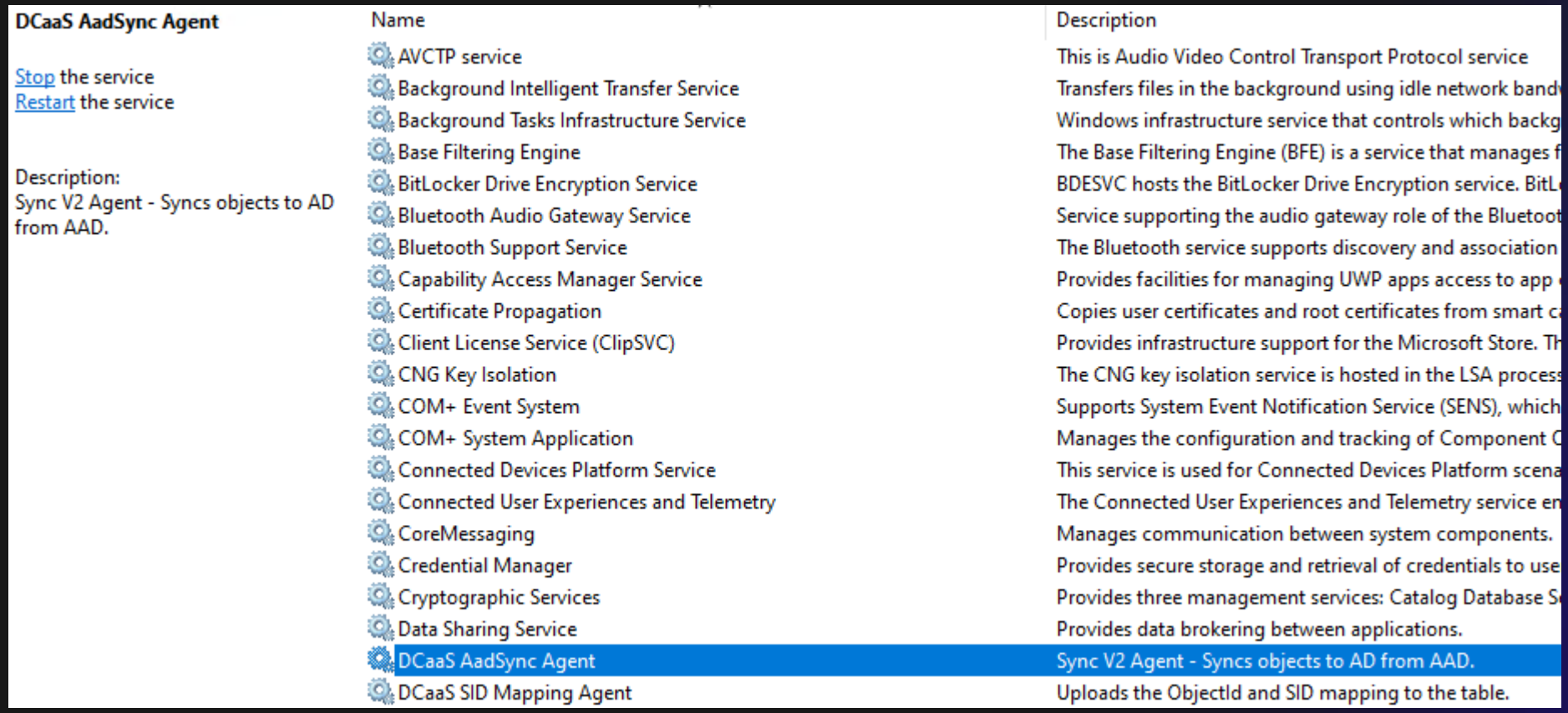
AADDS password synchronisation

- “Azure AD pushes the encrypted AES symmetric key, the encrypted data structure, and the initialization vector using an internal synchronization mechanism over an encrypted HTTP session to Azure AD Domain Services”[1]
- “Azure AD Domain Services retrieves the private key for the tenant's instance from Azure Key vault.” [1]
- These hashes are encrypted such that only Azure AD DS has access to the decryption keys. No other service or component in Azure AD has access to the decryption keys.” [2]

1. https://learn.microsoft.com/en-us/azure/active-directory/hybrid/connect/how-to-connect-password-hash-synchronization#password-hash-sync-process-for-azure-ad-domain-services
2. https://learn.microsoft.com/en-us/azure/active-directory-domain-services/synchronization#password-hash-synchronization-and-security-considerations

Azure AD → AADDS 1/5

- AADDS uses `DCaaS AadSync Agent` service to sync objects from Azure AD



Azure AD → AADDS 2/5

- Sync agent pulls information from Azure AD using MS Graph API
- Initial request:

```
https://graph.microsoft.com/v1.0/users?$select=id,city,country,mailNickname
givenName,displayName,department,faxNumber,mail,mobilePhone,officeLocation,
postalCode,preferredLanguage,proxyAddresses,state,streetAddress,surname,
businessPhones,jobTitle,userPrincipalName,accountEnabled,onPremisesSyn
employeeId,companyName,
windowsLegacyCredentials,windowsSupplementalCredentials,
onPremisesSecurityIdentifier,lastPasswordChangeDateTime,passwordProfile,
passwordPolicies
```

- Subsequent requests:

```
https://graph.microsoft.com/v1.0/users/microsoft.graph.delta()?$deltatoken=ehCM
ic[redacted]sRTn4maJ.YTYhGIRkv794IAvoCKEWXaggFN7id6gP1bU37vJwZpE
```

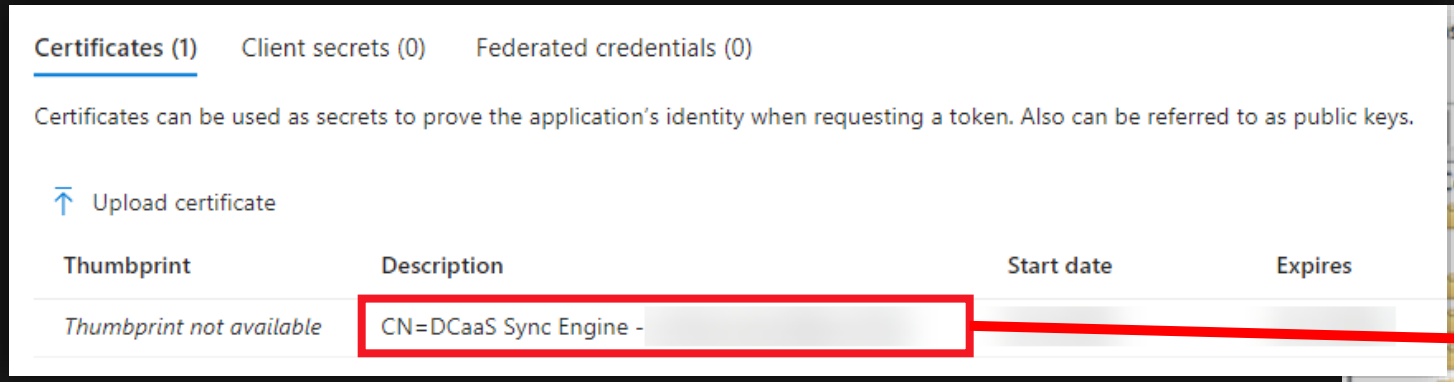
Azure AD → AADDS 3/5

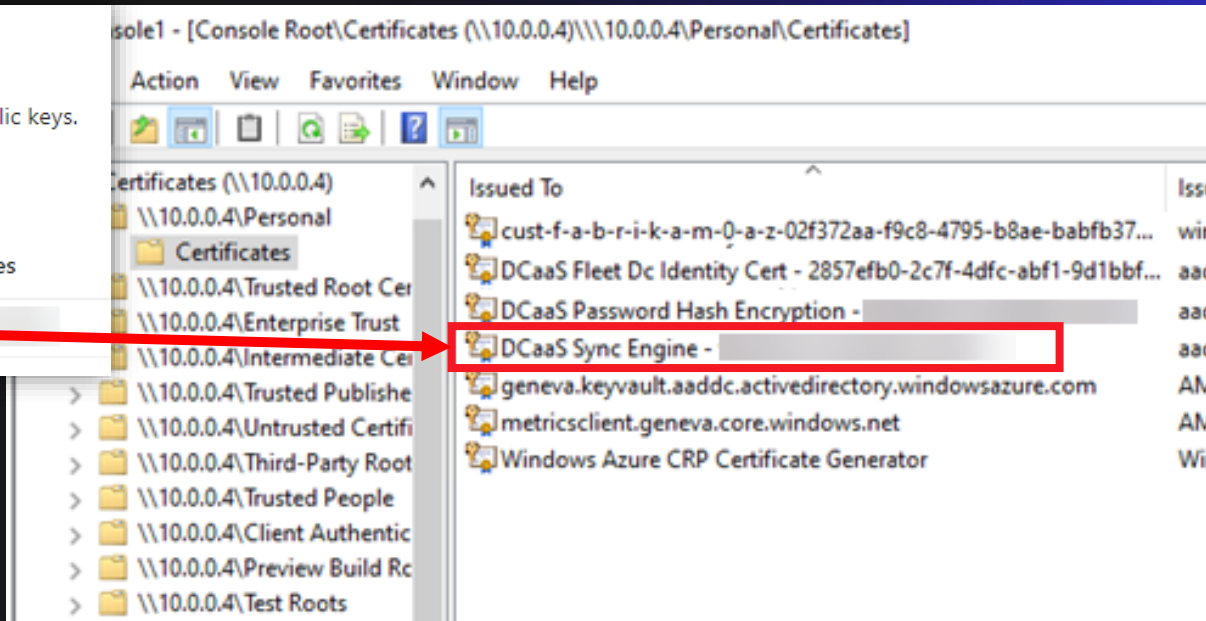
- `windowsLegacyCredentials` and `windowsSupplementalCredentials` properties are NOT part of MS Graph API user object schema *
- Sync agent uses `Azure AD Domain Services Sync` app for sync

* https://learn.microsoft.com/en-us/graph/api/resources/user?view=graph-rest-1.0#properties

Azure AD → AADDS 4/5

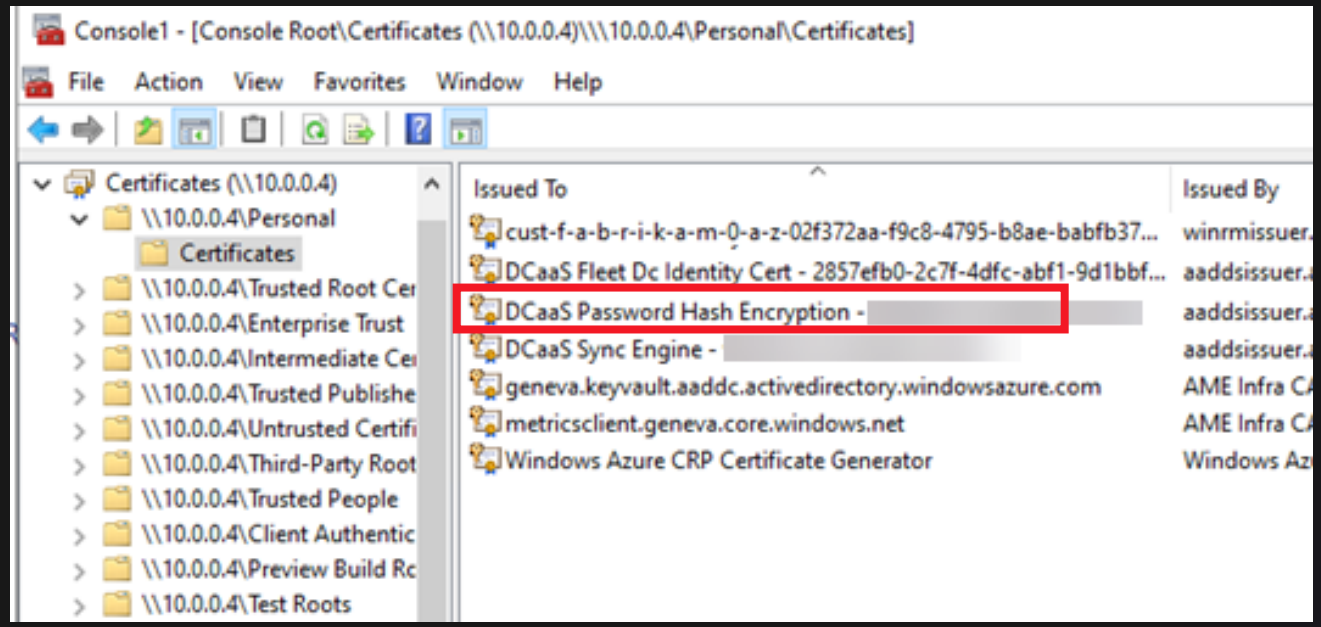
- Legacy credential properties available *only* for `Azure AD Domain Services Sync` app!
- The app is using certificate as client credentials
  - Stored in `Personal` store on AADDS DCs





Azure AD → AADDS 5/5

- Certificate used to decrypt legacy credentials stored in `Personal` store on AADDS DCs

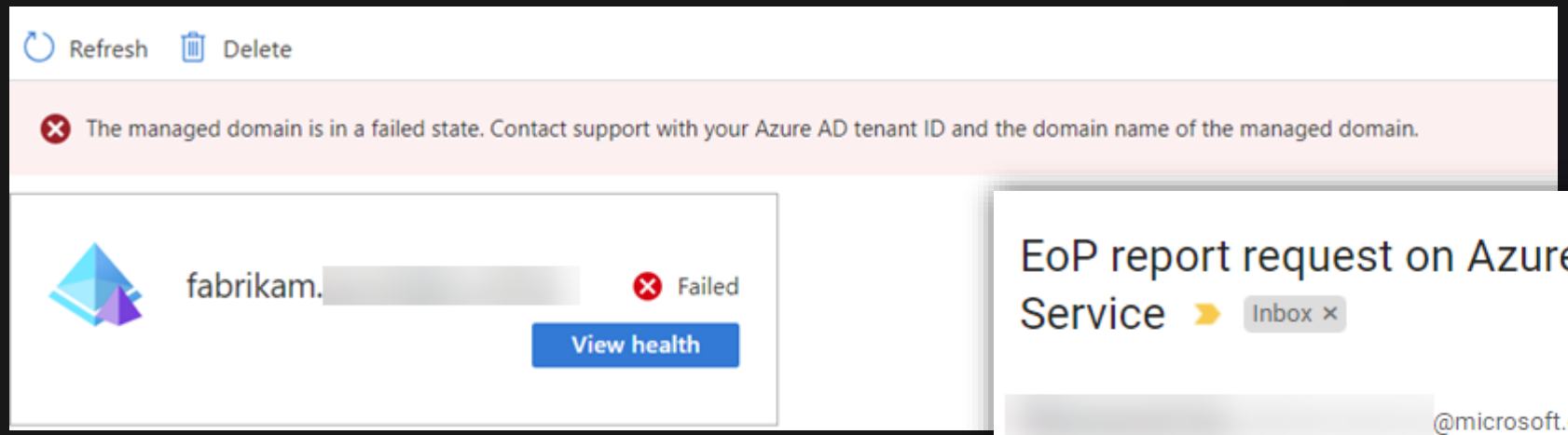


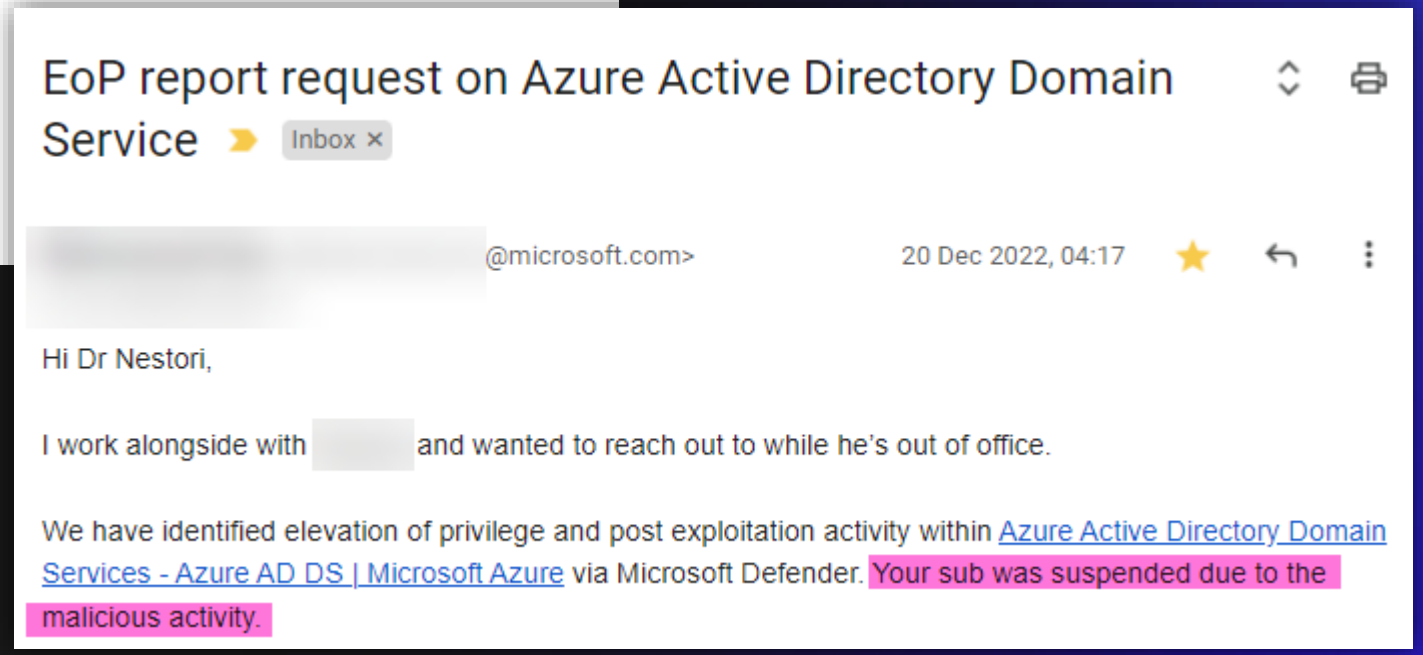
Research results 1/3

- AADDS pulls changed objects from Azure AD using `Azure AD Domain Services Sync` application
- Legacy credential properties only available for that app
- The app is using `DCaaS Sync Engine` certificate as client credentials
  - Extra credentials can be added – certificate not required
- Credentials are encrypted/decrypted using `DCaaS Password Hash Encryption` certificate
  - Required to decrypt legacy credential properties
  - Updated on regular basis (supports n+1 older certificates)

Research results 2/3

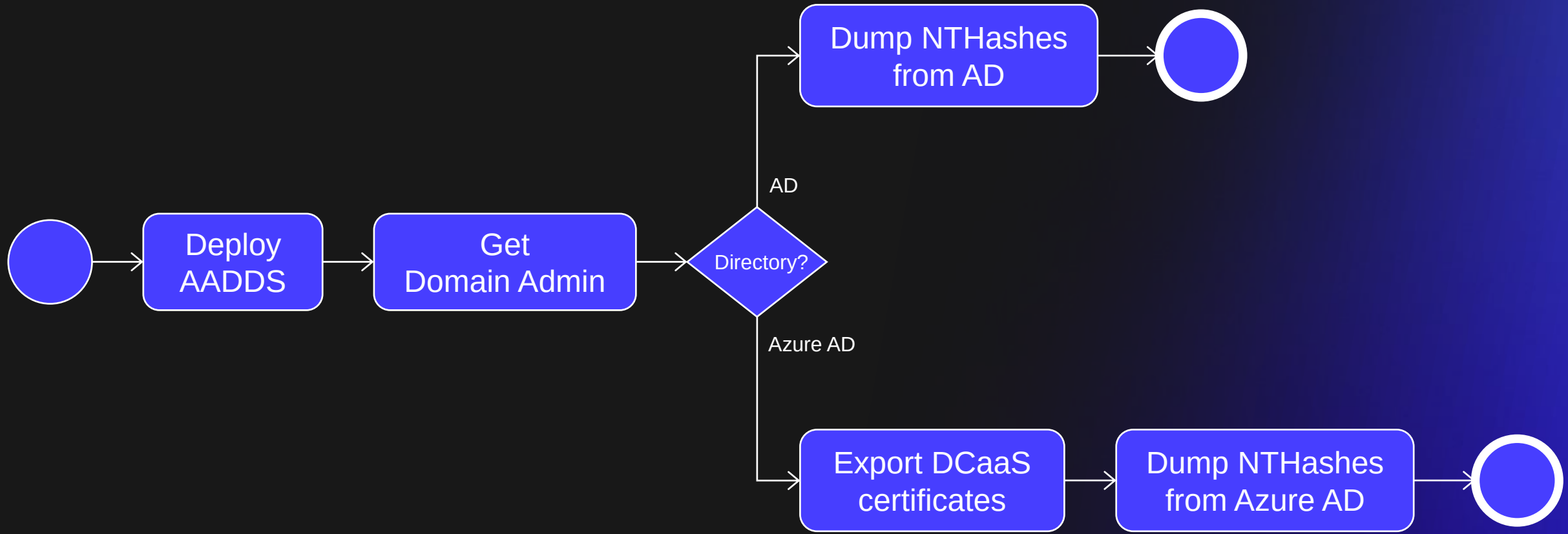
- Compromised instances will be suspended in 20min – 2d
- To continue, delete and deploy new one





Research results 3/3

- Dumping NTHashes

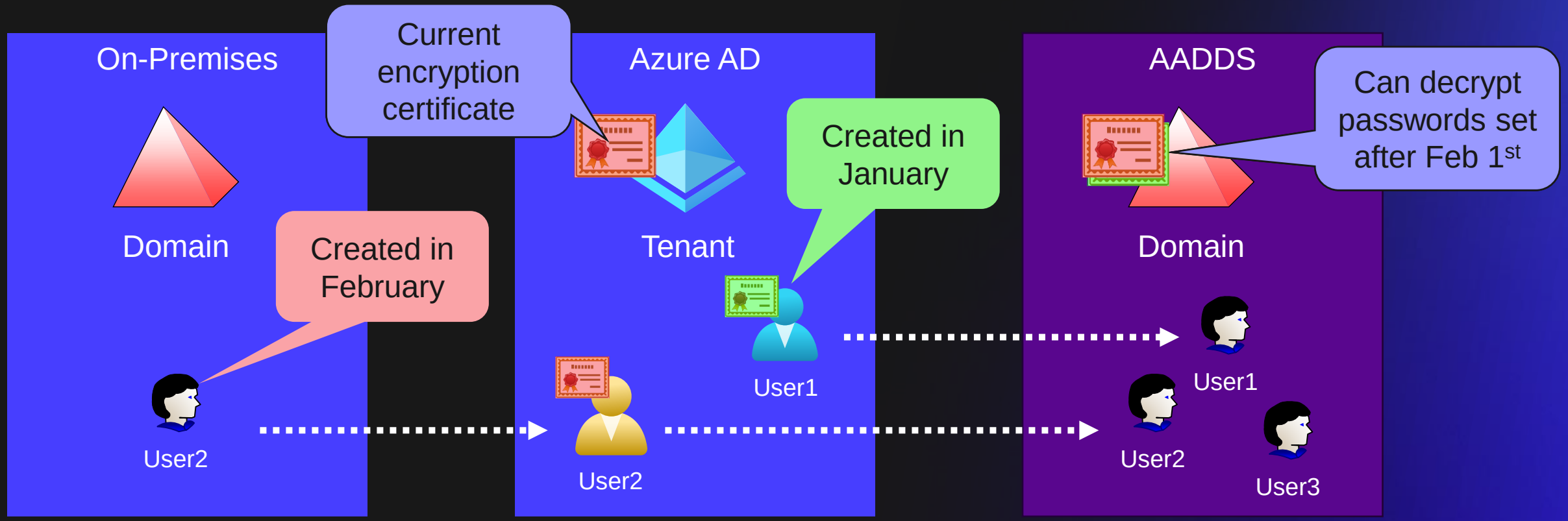


The story of a password – Mar 1st

Created on Jan 1st

Created on Mar 1st

Created on Feb 1st



Secureworks®

Demo

Open questions

- How the certificates are deployed to AADDS DCs?
- How `DCaaS Password Hash Encryption` certificate is updated on AADDS DCs?
- What other hidden properties the `Azure AD Domain Services Sync` app can access? (none)
- Are there other apps that can access undocumented/hidden properties?

@DrAzureAD

Secureworks®

From cloud admin to on-prem admin

From cloud admin to on-prem admin

- Deploy AADDS
  - Azure AD Connect starts syncing legacy credentials for the configured scope (may contain admins)
- Get Domain Admin
- Dump the hashes
- Use target user’s hash in the target domain or try to crack

Detecting AADDS deployment 1/2

```
// Check creation of AADDS service principal.
// Indication of future NTHash dump.
// Can be legit but very rare.

AuditLogs
| where Category == "ApplicationManagement"
| where OperationName == "Add service principal"
| where TargetResources[0].displayName == "Azure AD Domain Services Sync"
| where InitiatedBy.app.displayName == "Domain Controller Services"
```

Detecting AADDS deployment 2/2

```
// Check creation of AADDS sync app.
// Indication of future NTHash dump.
// Can be legit but very rare.

AuditLogs
| where Category == "ApplicationManagement"
| where OperationName == "Add application"
| where TargetResources[0].displayName == "Azure AD Domain Services Sync"
| where InitiatedBy.app.displayName == "Domain Controller Services"
```

Detecting adding credentials to AADDS sync app

```
// Check adding extra credentials to AADDS sync app by others than AADDS
service principal
// Indication of future NTHash dump.
// No legit use case to update credentials.

AuditLogs
| where Category == "ApplicationManagement"
| where OperationName contains "Certificates and secrets management"
| where TargetResources[0].displayName == "Azure AD Domain Services Sync"
| where InitiatedBy.app.displayName != "Domain Controller Services"
```

Detecting NTHash dumping from Azure AD

```
// Get the AADDS appid
let AADDS_appId = AADServicePrincipalSignInLogs
| where TimeGenerated > now() - 10m
| where ServicePrincipalName == "Azure AD Domain Services Sync"
| distinct AppId;

// Check AADDS sync app access to Graph API not using delta
// Can be legit but very rare.
// Indication of NTHash dump.
MicrosoftGraphActivityLogs
| where RequestMethod == 'GET'
| where AppId == AADDS_appId
| where RequestUri !contains '/users/microsoft.graph.delta()?$deltatoken'
```

@DrAzureAD

Secureworks®

Q(&A)

Secureworks®