Pentesting Azure Applications

Access Methods

Azure Deployment Models

Azure uses two models for authentication and permissions. The legacy *Azure Service Management (ASM)*, also known as *Azure Classic*, is used for older Azure services. *Azure Resource Manager (ARM)* is used strictly for new Azure services. Both models can coexist in a subscription and breaching one does not compromise the other.

Users can authenticate in ASM using their username and password or their X.509 certificate. X.509 certificate support was dropped in ARM due to certificate manageability. There are no certificate revocation lists (CRLs) implemented in ASM so certificate had to be manually removed from the Azure portal.

ARM uses role-based access control. The most common roles are Owner (full control), Contributor (all rights except the ability to change permissions), Reader (read only), and User Access Administrator (ability to change permissions only). When compromising a subscription, you'll likely want to target users with the Owner role.

Obtaining Creditionals

Our target account would be one that provides administrator access to a target's ASM resources, has Owner permissions for all ARM resources in the subscription, and has two-factor authentication (2FA) disabled. The first step in finding our target would be to locate a service account that uses a username and password to log in and that is a Co-Administrator of the target subscription in ASM. Service accounts are ideal because they rearely have 2FA enabled, infrequently change their password, and often have passwords left in source code. Failing that, the account of a human administrative user would do well, especially because they are likely to have full control over all resources. As a last report, consider ASM management certificates.

Mimikatz

Once you have administrative access to a system, it's time to download Mimikatz or Mimikatz convert to a Powershell script if Mimikatz is blocked by an antivirius. After Mimikatz is launched, run `privilege::debug` then `sekurlsa::logonpasswords`

Usernames and Passwords

If Mimikatz is not an option, other ways of grabbing usernames and passwords includes searching unencrypted documents, phishing, finding saved authentication thokes, or using educated guesses.

- **Searching Unencrypted Documents** - If your target is a service account, you will often find the account's password in source code and configuration files used by the service. When targeting a human, look for passwords in a text file or spreadsheet.
- **Phishing** - The downside of phishing is that it cannot target service accounts.
- **Looking for Saved ARM Profile Tokens** - Because developers often need to use different accounts when accessing ARM resources (perhaps for automation or testing purposes), Azure provides an ARM PowerShell cmdlet to save an Azure credentail as a *profile*: `Save-AzureRMProfile`. These profiles are just JSON files which can be used with the `Select-AzureRMProfile` cmdlet with the `-Path` parameter. Finding these stored profiles can be tricky because they don't have a unique extension (although some developers will choose *.json*). Try searching files for terms such as *TokenCache*, *Tenant*, *PublishSettingsFileUrl*, and *ManagementPortalUrl*.
- **Guessing Passwords** - Do some open source intelligence research to make an educated guess and use an online portal without rate limiting or account lockouts to test passwords.

Management Certificates

ASM uses asymmetric X.509 certificates. The most common file extensions for keys on Windows are *.pfx* for private keys and *.cer* for public keys. *Publish Settings* (*.publishsettings*) are XML documents that contain details about an Azure subscription, including the subscription's name, ID, and base64-encoded management certificate. Publish Settings files are used for developers to deploy projects to Azure. Once you have a Publish Settings file, copy the *ManagementCertificate* line and export the base64-encoded string as a *.pfx* file.

Mimikatz can retrieve certificates using the following commands

```
mimikatz # crypto::capi
mimikatz # privilege::debug
mimikatz # crypto::cng
mimikatz # crypto::certificates /systemstore:local_machine /store:my /export
```

The final command exports all certificates from the local machine store's personal certificate folder and saves them to the specified directory as both *.pfx* and *.cer* files. (For the names of other possible `store` and `systemstore` values, see the Mimikatz crypto wiki.)

Management certificates are typically used either to deploy a service or for an application to interact with a resource once it is running in Azure. Configuration files can contain base64-encoded management certificates. Look for files with the *.config* extension – most oftenly named *app.config* or *web.config* – that contain a base64-encoded string (usually a little over 3,000 character) and contain a subscription ID.

Cloud Service Packages (*.cspkg*) are ZIP files that contain specific parts of a developer application. Sometimes management certificates are embedded within the Cloud Service Package file.

Encountering Two-Factor Authentication

If 2FA is enabled, you have a significant hurdle to overcome (assuming you don't have the user's mobile device). If you want to avoid 2FA altogether, the best options are to use certificate authentication (for ASM) and use a service principal or service account. If you have access to the user's browser you can copy their access cookies into your own browser. An alternative to cookie stealing is proxying traffic through the user's browser using a command-and-control tool. You can install a Cobalt Strike Beacon on the user's system and use the Browser Pivot command to create a proxy for your Cobalt Strike server. Additional details on browser pivoting can be found at Cobalt Strike's blog and documentation.

Other options for bypassing 2FA is using Mimikatz to retrieve a smartcard's PIN, stealing a phone or phone number, and social engineering the user for their 2FA.