560.5

Domain Domination, Azure Annihilation, and Reporting

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

SEC560.5 | Enterprise Penetration Testing

SANS

Domain Domination, Azure Annihilation, and Reporting

Welcome to SANS Security 560.5. In this session, we will have a look at authentication mechanisms used in Windows, such as NTLM and Kerberos, and how we can attack these mechanisms. Furthermore, we will see how we can escalate our privileges and bypass the UAC controls. Next to this, we will also focus on how to get control over Windows Domains using Domain Domination attacks.

The second half of this session will focus on Azure AD and its integration with on-premises Windows Domains. We will go over the basics of Azure AD and have a look at fundamental attacks, followed by some specific attacks targeting the cloud and on-prem integration. You'll notice how the Domain Domination attacks can also be applied here. Next, we will discuss applications and authentication, followed by specific attacks and we'll finish with some specific Azure defenses.

Without further ado, let's begin.

TABLE OF CONTENTS (1)	SLIDE

This slide is the table of contents (1). Note that all labs are in boldface.

TABLE OF CONTENTS (2)	SLIDE

This slide is the table of contents (2).

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

- Kerberos
- Kerberoast
  - LAB 5.1: Kerberos
- Domain Dominance
  - LAB 5.2: Domain Dominance
- More Kerberos Attacks
- Silver Ticket
  - LAB 5.3: Silver Ticket
- Golden Ticket
  - LAB 5.4: Golden Ticket
- Domain Privilege Escalation
- Azure Intro
- Azure AD
- Azure Recon
- Azure Password Attacks
  - LAB 5.5: Azure Recon and Password Spraying
- OpenID
- Azure Infrastructure
- Running Commands on Azure
- ngrok
  - LAB 5.6 Running Commands
- Permissions on Azure
- Reporting
  - LAB 5.7: Gaining Access and moving Laterally
- Conclusion

This page intentionally left blank.

Kerberos: Introduction

Kerberos is the primary authentication mechanism in the Microsoft Active Directory domain

- Kerberos is a network authentication protocol based on **tickets**
- Allows two parties (a client and a server) to authenticate to each other **over an insecure network channel**, provided that both parties **trust a third party**—the KDC
- Three pieces (heads) of Kerberos:
  - KDC (Key Distribution Center)
  - Client requesting access
  - Service the client is attempting to obtain access to

While Kerberos is the preferred mechanism, Windows will revert to NTLMv2 if Kerberos is not available (unless explicitly disabled)

With Active Directory, Microsoft introduced a new authentication scheme called Kerberos. Kerberos is a network authentication protocol based on tickets, developed by MIT. The protocol allows two parties (a client and a server, for example) to authenticate to each other over an insecure network channel, provided that both parties trust the third party: the Kerberos server. Given these three components, Kerberos is named after the three-headed mythological dog.

Each party in a Kerberos environment authenticates to the Kerberos server and receives a ticket. More precisely, this is a ticket-granting-ticket: a ticket that can be used to request tickets. When one party (client) wants to use a service from a second party (server), the first party uses its ticket-granting-ticket to obtain a ticket for the second party from the Kerberos server and then presents it to the second party, thereby authenticating to the third party. Since both parties trust the Kerberos server, a ticket provided by the Kerberos server is trusted by both parties, leading to authentication and authorization.

Before Kerberos was introduced, pre-Active Directory Windows domains (Windows NT) relied on NTLM challenge/response authentication. NTLM provides authentication between two parties without a third trusted party.

If Kerberos cannot be used (various reasons apply), Windows will fall back to NTLM authentication.

Simple Kerberos: How it works

Kerberos authentication comes in three parts, we can think of like an all-you-can-watch movie theatre

- Pre-authentication and getting a Ticket Granting Ticket
  - AS-REQ: You show up the theatre and get your movie pass by presenting your ID (and cash)
  - AS-REP: You now have pass (TGT) for all the movies, but aren't going to a movie yet
- Request a Service Ticket
  - TGS-REQ: You want to go to a movie, so you present your pass (TGT)
  - TGS-REP: The theatre checks the pass is legitimate (and not expired), then gives you a paper ticket (Service Ticket) for the movie
- Use Service Ticket
  - ST: Walk to the theatre, present paper ticket (Service ticket) to enter the movie

Kerberos can seem a bit complicated, but with this simply analogy you should see that it isn't that complex. Let's use the analogy of an all-you-can-watch movie pass.

You first need to obtain your special pass. You first present your ID (and cash). The theatre company then gives you a special card that allows you to watch all the movies you like. The ID you presented is like your username and password and proves who you are. The special card you get is your Kerberos Ticket Granting Ticket (TGT). At this point, all you have is the pass. You aren't in a movie yet.

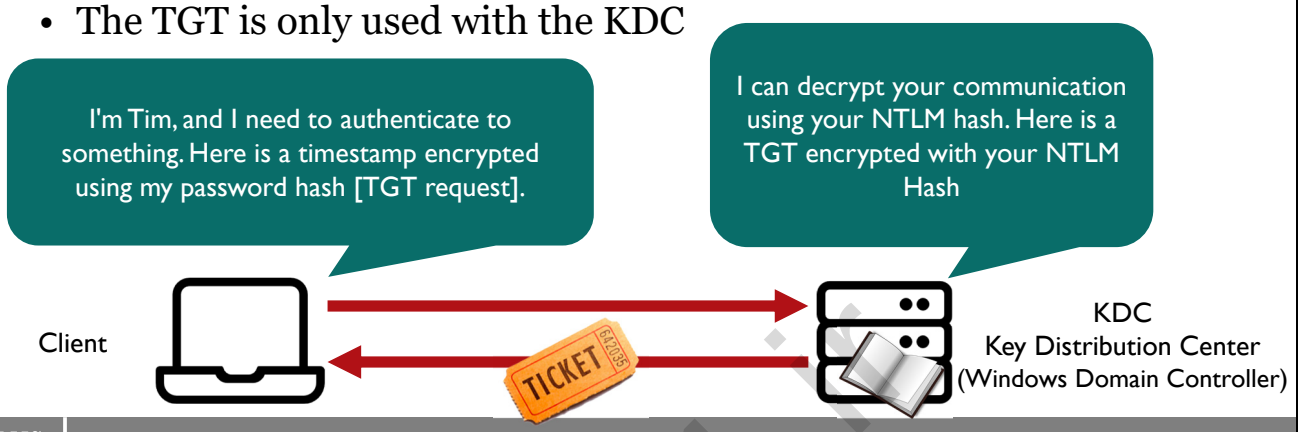
Now that we want to see a movie, we present our special pass (TGT). When you give them the pass, they check to make sure it isn't expired (much like a TGT is checked to see if it is still valid). If the pass (TGT) looks good, you get a paper ticket (Service Ticket) for your movie and walk towards the theatre.

At this point, you are met by a ticket agent close to the theatre. You present the paper ticket (Service Ticket). The agent looks at the ticket and checks if it is legitimate (correct date and right movie). You are then directed to your seat. At this stage, the ticket agent doesn't look at your ID (password) or the special pass (TGT), just the paper ticket (Service Ticket). You can only use the paper ticket at the movie you requested, it won't work for a different movie or a different theatre. If the time and name matches, it is time to enjoy the movie.

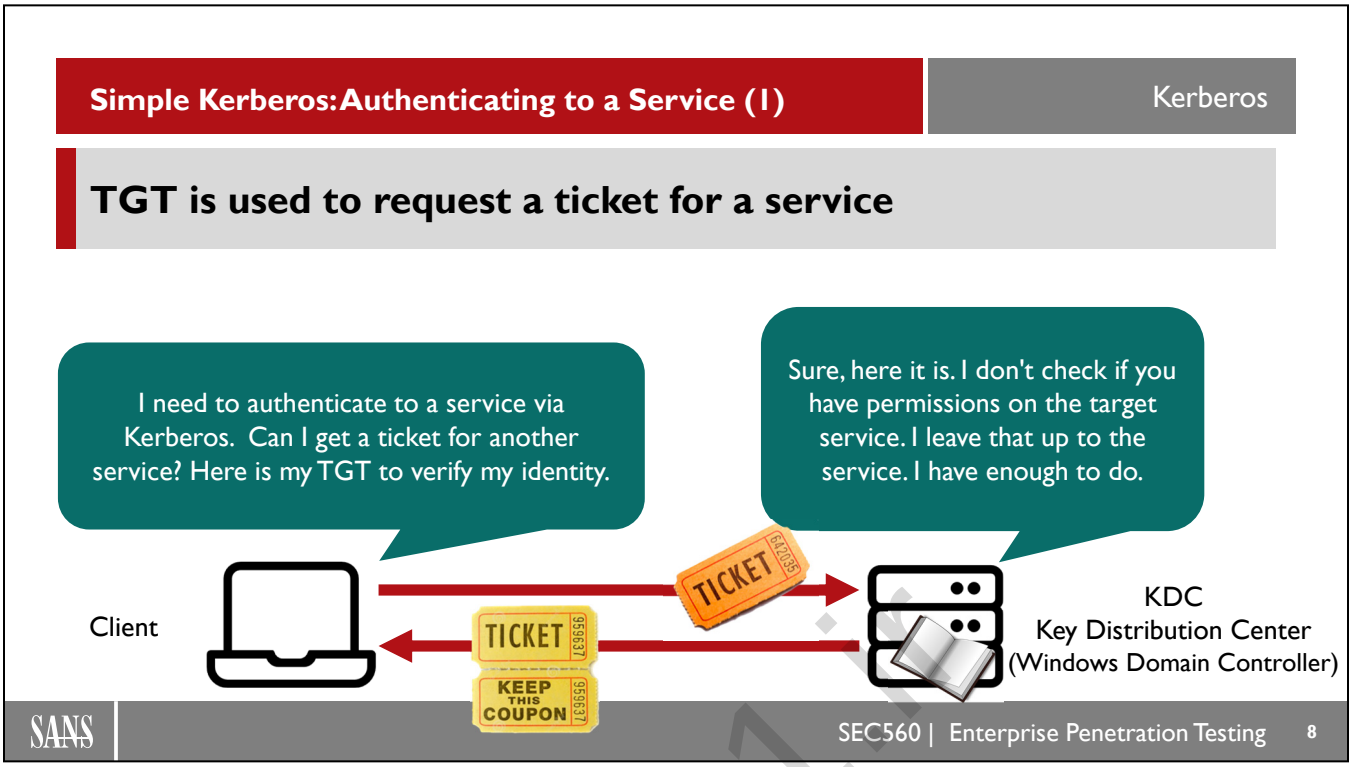
Simple Kerberos: Pre-Authentication and TGT

Before you can authenticate to a service you need a Ticket Granting Ticket (TGT)

- The TGT is only used with the KDC



Before we can authenticate to a service, we first need to obtain a Ticket Granting Ticket (TGT). To do this, we need to prove who we are. We can do that by using our username and password since the Domain Controller/KDC knows my password (hash). We encrypt a timestamp with our password hash. Since the KDC knows my password hash, it can attempt to decrypt the payload and confirm we used the same key. If successful, the KDC sends back a TGT.

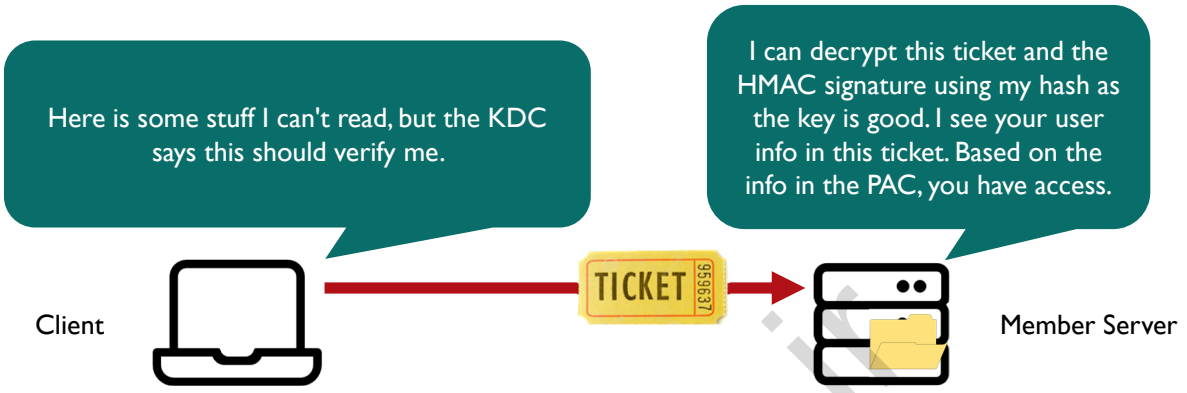


When we want to authenticate to a service, we need a ticket specifically for that service. To get one, we present the TGT to the KDC and ask for a Service Ticket (ST). The KDC doesn't ask for credentials again but uses the TGT to as proof of identity. The KDC then uses data in the TGT to build a Service Ticket specially for the target service.

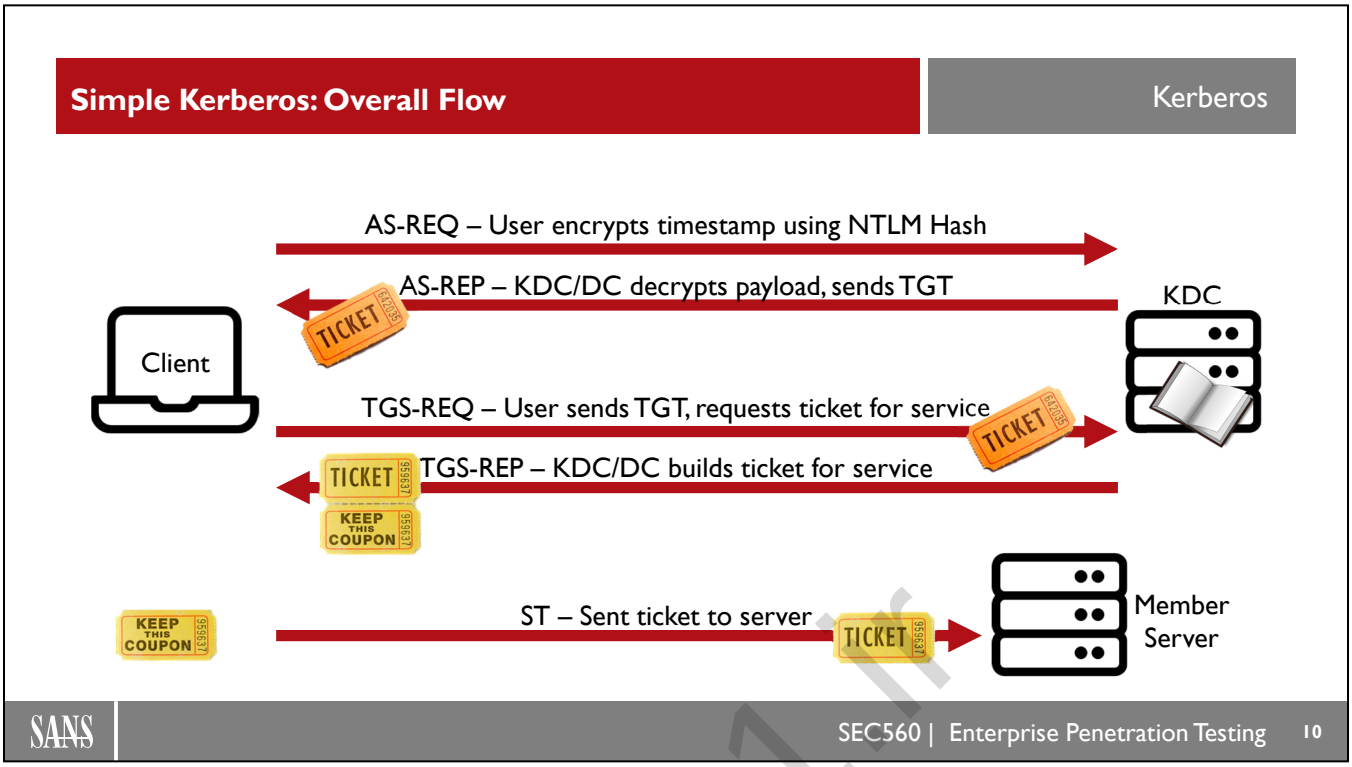
The KDC does not decide if we should have access to the target service. If it did, the KDC would have to know all the permission settings on every file, database record, and everything else in the domain. This would take tremendous computing power and would be quite slow. Instead, the KDC creates a ticket for the service that describes who we are, including username and group memberships (as part of the Privilege Attribute Certificate or PAC). The ST is encrypted using the password hash of the target service. We can't decrypt it, but the target service can.

Simple Kerberos: Authenticating to a Service (2)

The Server half of the ticket is sent to the target server, the server decrypts it and checks the PAC



Next, we present half of the ST to the server. We can't read the portion sent to the service (since we don't know the service's password or hash). However, the service can decrypt the ticket and examine the contents. The decrypted ticket also contains a PAC that describes who we are. The server then makes a decision if we should have access, if so, what level of access.



The diagram above shows the simple flow of information during Kerberos authentication. Let's discuss what's happening in the flow.

In Active Directory, when a client successfully authenticates to a Domain Controller, the client is given a ticket-granting-ticket.

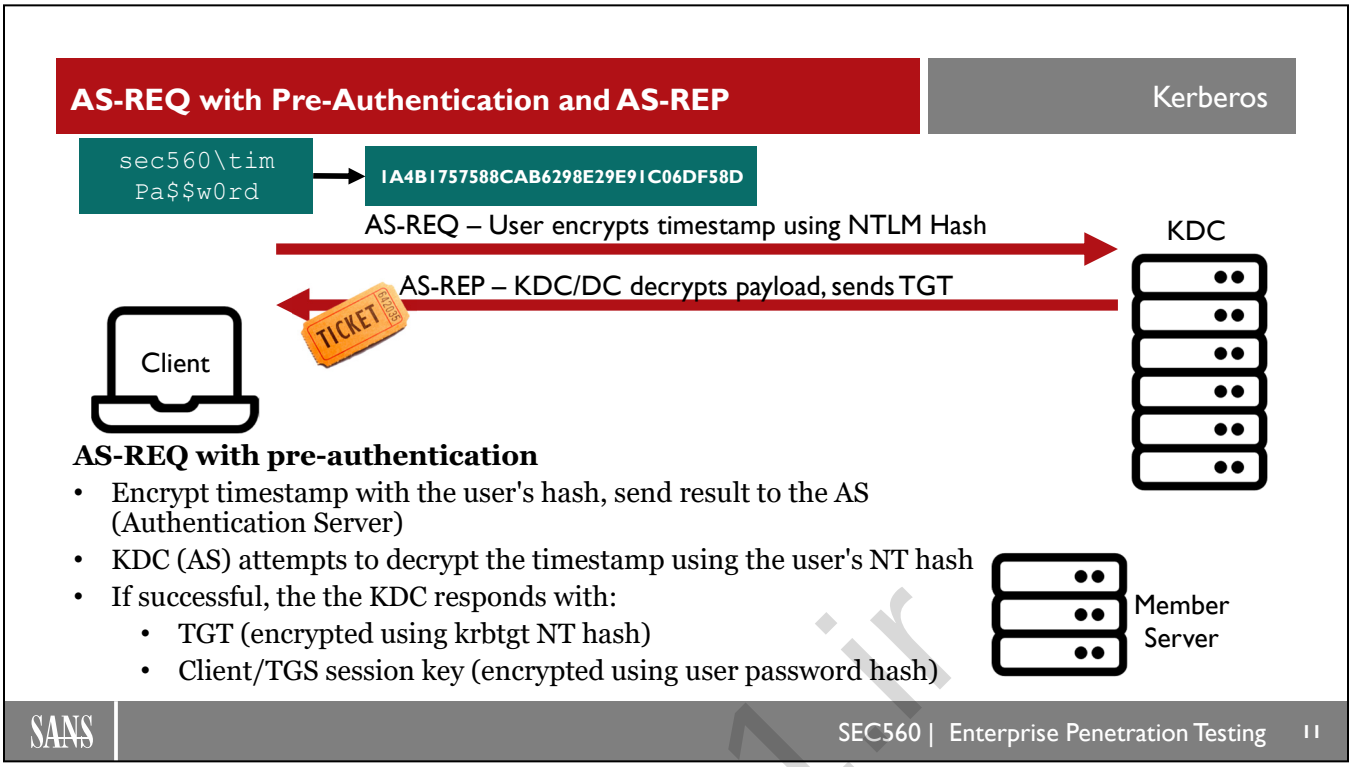
To authenticate, the user's workstation will send a request to the Authentication Server (AS) on the domain controller. This request includes timestamp encrypted with the user's NT password hash, which is also commonly referred to as the NTLM hash. The domain controller (AS) has the user's password hash in its database and will attempt to decrypt the message. If successful and the timestamp is still valid, then the client will be given a ticket-granting-ticket (TGT) by the Ticket Granting Service (TGS), a logical role of the KDC which is part of the domain controller.

Kerberos works with tickets: cryptographically secured pieces of data that grant access to a service for a particular user and during a well-defined period. For example, you can receive a ticket to access a share on a file server.

A ticket-granting-ticket is the ticket a client receives first, and it is a special ticket. It is a ticket for the krbtgt service and can thus be used to request other tickets. By default, a TGT is valid for 10 hours. When it expires, Windows will automatically and transparently request a new one.

Once a client has obtained the TGT, it can request access to other services, for example, a file share. To obtain this access, the client sends its TGT to the TGS along with the details of the service it wants to access.

To access the service (for example, a file share), the client will send the ticket to the service (the file server). Since the file server also trusts Kerberos (the TGS), it will accept tickets and grant access to the service once it has validated through cryptographic means that this is an authentic ticket issued by the TGS trusted by the service. The service will then decide if the user should have access and what level of access.

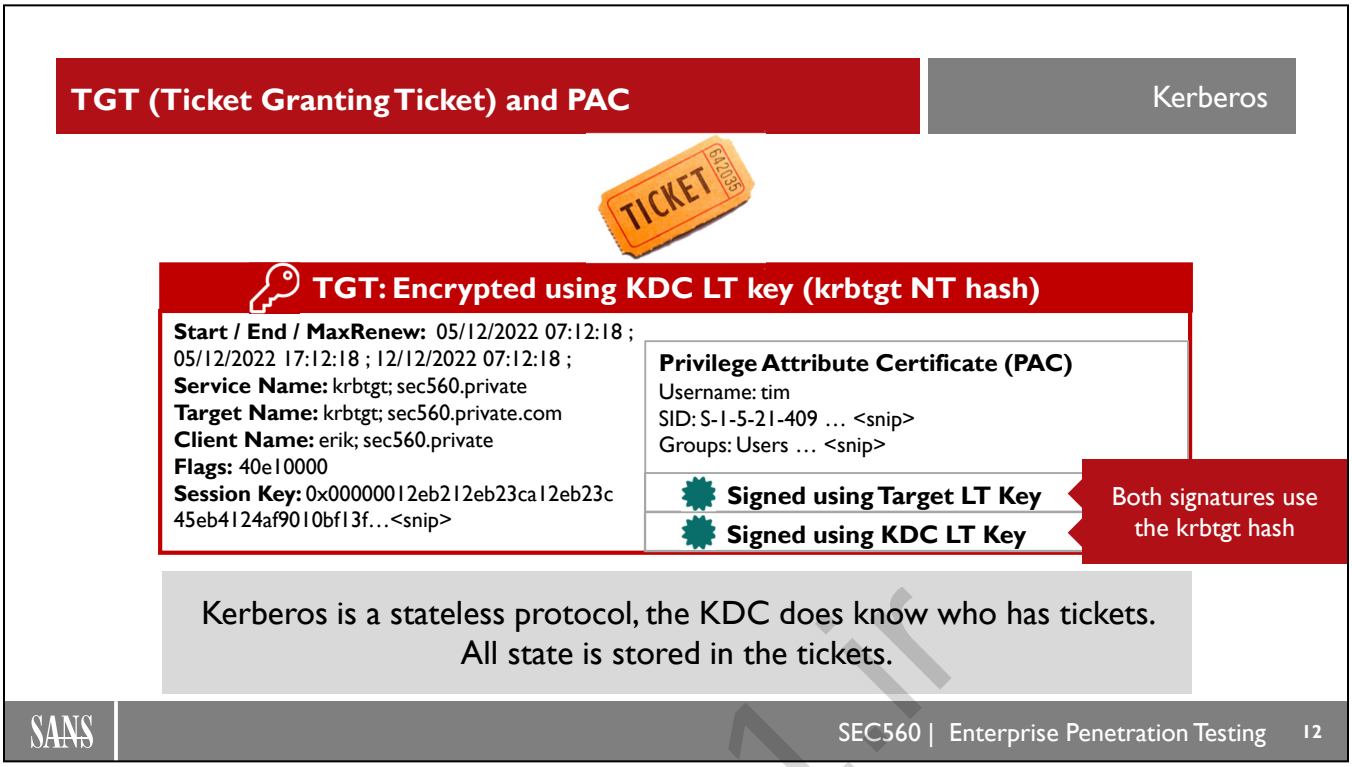


In Active Directory, when a client successfully authenticates to a Domain Controller, the client is given a ticket-granting-ticket.

To authenticate, the user's workstation will send an authentication request (AS-REQ) to the Authentication Server (AS), which is a logical role on the domain controller (DC). This request includes timestamp encrypted with the user's NT password hash. The domain controller has the user's password hash in its database and will attempt to decrypt the message. If successful and the timestamp is still valid, then the client will be given a ticket-granting-ticket (TGT) by the Ticket Granting Service (TGS), another logical role of the KDC which is part of the domain controller.

Kerberos works with tickets, cryptographically secured pieces of data that grant access to a service for a particular user and during a well-defined period. For example, you can receive a ticket to access a file server.

ATGT is the ticket a client receives first. It is a ticket for the krbtgt service and can thus be used to request other tickets. By default, a TGT is valid for 10 hours. When it expires, Windows will automatically and transparently request a new one.



The TGT is the first ticket received by the client and contains the following information:

Name: The user's name the ticket is associated with

Start time and End time: marks the validity period of the ticket. The default is 10 hours

Authorization data: Authorization data details the user's privileges and access rights. The authorization data takes the form of a Privilege Attribute Certificate (PAC) object

The Client/TGS session key that can be used for future communications between the client and the TGS

The PAC is extremely sensitive and should be protected due to its sensitivity. The PAC it is signed with two keys (in this case it is the same key both times) to protect the integrity:

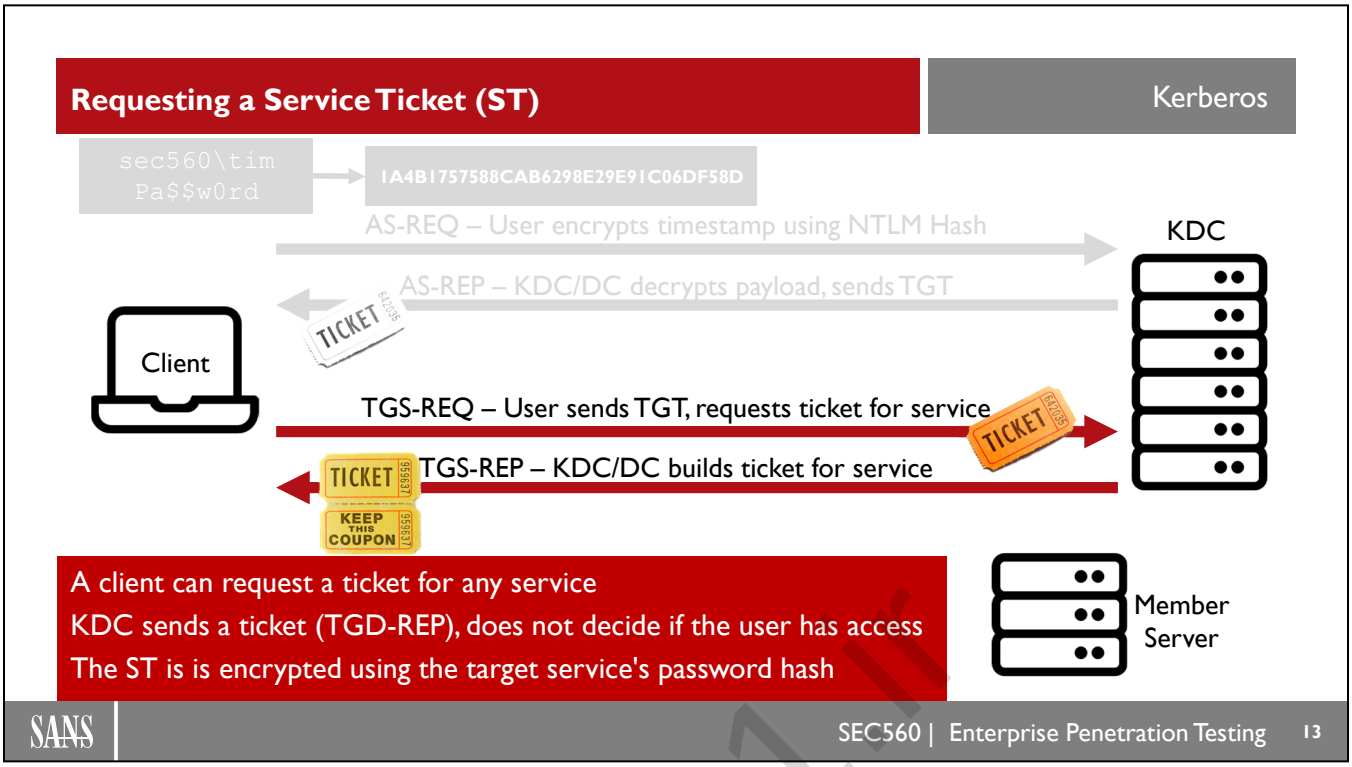
It is signed using the Target Long-Term Key. Because this is a TGT, the target is the krbtgt account (the key is the krbtgt NT hash)

It is also signed using the KDC LT Key, once again the krbtgt NT hash

To prevent the entire TGT from being tampered with, it is encrypted using the KDC long-term key (yet again is the krbtgt NT hash). The TGT cannot be read by the client, but that's fine, as it only needs to send it back to the KDC for future validation (e.g., when a Service Ticket is requested). In the TGS-REQ, the user wants to authenticate to a certain service and thus sends the following to the KDC (in this case, the TGS component of the KDC):

An authenticator message (encrypted using the Client/TGS session key)

The encrypted TGT and a ticket request (referencing a certain Service Principal Name, SPN)



If the KDC receives a TGS-REQ with a valid TGT, it will send back a response, TGS-REP. The KDC will validate whether the service to which the client requests access exists and subsequently creates a service ticket (ST) that is sent back. It's important to note that the KDC does not do any validation of privileges. The target service will do that by itself by reviewing the PAC that is included in the service ticket (see below).

The service ticket has two parts:

A client portion: encrypted using the Client/TGS session key (so this can be decrypted by the client).

A server portion: encrypted using the target long-term key (the password hash of the target service). This server portion also includes the PAC of the user. It is this service ticket that the user will subsequently submit to the service they are trying to access.

Illustration inspired by "Abusing Microsoft Kerberos - Sorry you guys don't get it", Benjamin Delpy (Blackhat USA 2014).

Service Principal Name

Kerberos

Service Principal Name (SPN) is how a Kerberos client identifies a service on a system and requests a ticket

- When we request a ticket, we specify the SPN
- Format: **`serviceclass/host:port`**
- KDC has a mapping of the SPN to the underlying account
  - Example: **`SMTP/cliff.sec560.local`** to the account **`mailsvc`**
- Mappings are created with setspn.exe
- We get a list of SPN with setspn.exe or other LDAP queries
  - Query available to any user

When the client requests a ticket it uses the name of the service, or the Service Principal Name (SPN). The SPN is in the format of serviceclass/host:port. Let's take a look at each piece:

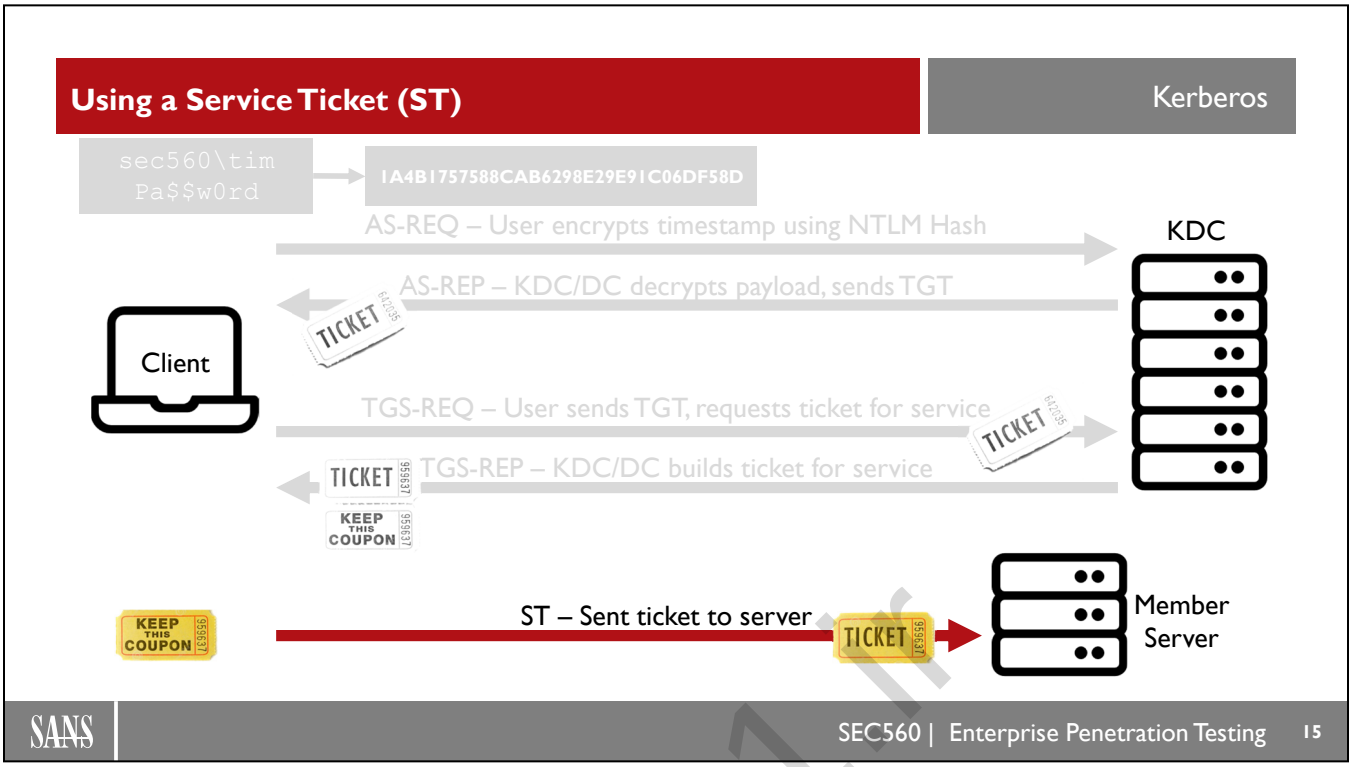
The "serviceclass" identifies the service. Examples include HTTP, HOST, TERMSVR, MSSQLSvc.

The "host" is the name of the host which is typically, the fully qualified domain name (FQDN)

The "port" is included if the service runs on a non-standard port. The name would not include the port for an HTTP service running on the default port tcp/80.

You'll remember, the Service Ticket is encrypted using the password hash of the target service. The KDC needs to store a mapping of the SPN to the underlying account so it can select the correct hash to use as the encryption key.

The setspn.exe tool, built-in to Windows, is used to manually create this mapping of SPN to account. Registration requires significant privileges on the domain and can't be done by an ordinary users.

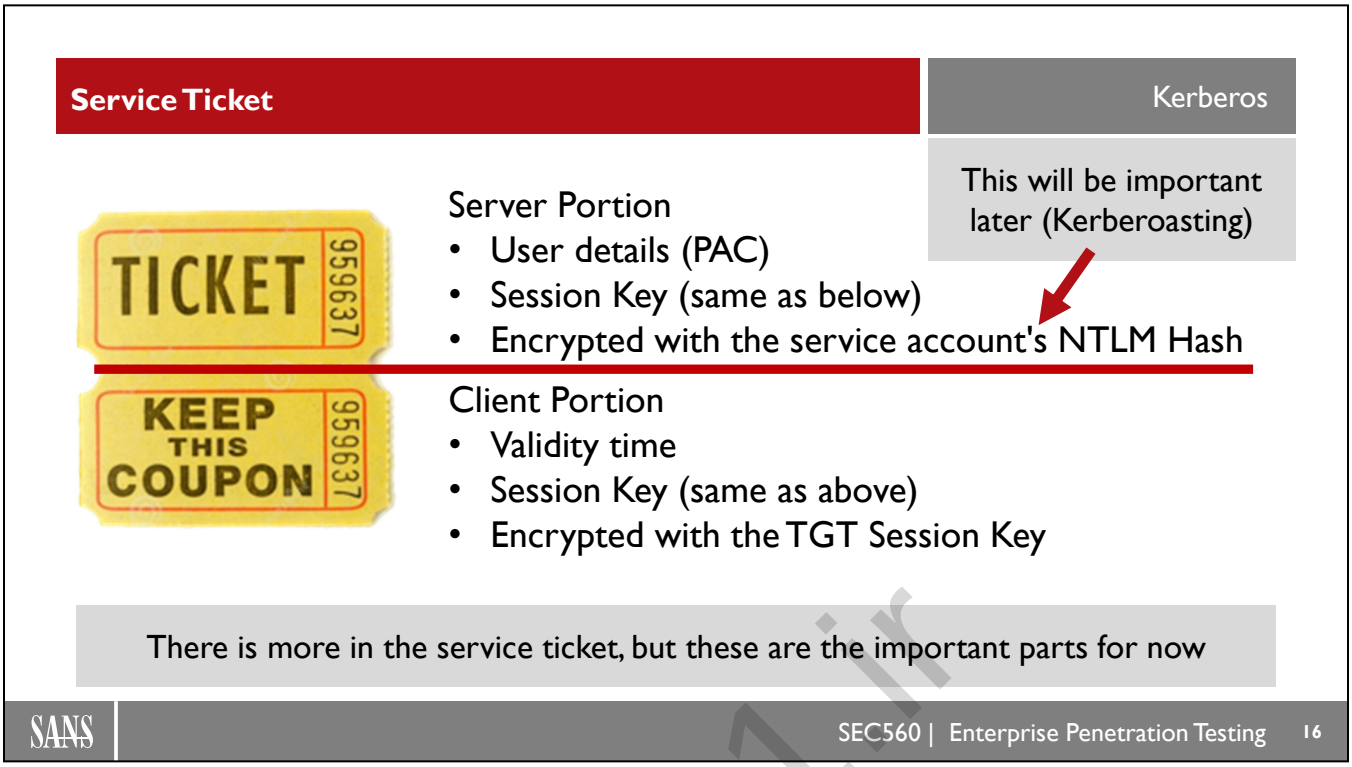


In Active Directory, when a client successfully authenticates to a Domain Controller, the client is given a ticket-granting-ticket.

To authenticate, the user's workstation will send an authentication request (AS-REQ) to the Authentication Server (AS), which is a logical role on the domain controller (DC). This request includes timestamp encrypted with the user's NT password hash. The domain controller has the user's password hash in its database and will attempt to decrypt the message. If successful and the timestamp is still valid, then the client will be given a ticket-granting-ticket (TGT) by the Ticket Granting Service (TGS), another logical role of the KDC which is part of the domain controller.

Kerberos works with tickets, cryptographically secured pieces of data that grant access to a service for a particular user and during a well-defined period. For example, you can receive a ticket to access a file server.

ATGT is the ticket a client receives first. It is a ticket for the krbtgt service and can thus be used to request other tickets. By default, a TGT is valid for 10 hours. When it expires, Windows will automatically and transparently request a new one.



The Service Ticket received from the KDC comes in two parts.

Server Portion

- Encrypted using the password hash of the target service (this will be important later when we talk about Kerberoasting)
- User details (PAC) containing details about the user, such as username and group memberships
- Session Key to be used to securely communicate with the client

Client Portion

- Encrypted using a key provided with the TGT
- Session Key to be used to securely communicate with the server (same key as the Server portion)
- Validity time

There is more information in the ticket, but these are the important pieces for now.

Kerberos: Three Long-Term Keys

Long-Term Secret Key	Key (password hash)	Use
KDC	Krbtgt	Encrypt TGT (AS-REP) Sign PAC (AS-REP & TGS-REP)
Client	Client account	Check encrypted timestamp (AS-REQ) Encrypt session key (AS-REP)
Target (Service)	Service account	Encrypt service portion of the ST (TGS-REP) Sign the PAC (TGS-REP)

Throughout a Kerberos transaction, three long-term security keys are used.

KDC long-term secret key is the password hash of the krbtgt service account. It is used in Kerberos to encrypt the TGT and sign the Privileged Attribute Certificate (PAC).

Client long-term secret key password hash of the client account. It is used in Kerberos to check the encrypted timestamp (AS-REQ) and encrypt the session key.

Target long-term secret key is password hash of the target service account. It is used to encrypt the TGS and sign the PAC.

Compromising any of these keys represents a security risk. There is, however, one "BIG" one: when we compromise the KDC long-term secret key, we can recreate TGTs and sign PACs, which allow us to obtain all privileges within the domain.

The Privileged Attribute Certificate (PAC) contains information about the user's authorization and privileges, e.g., group memberships. With the krbtgt hash, it is possible to forge the PAC to contain any desired privileges in the domain.

Course Roadmap

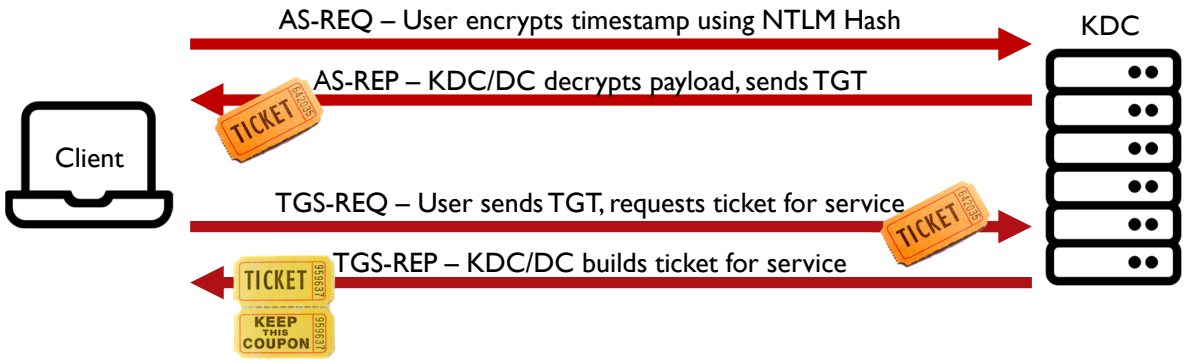
- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Requesting a Service Ticket (ST) Revisited



- A client can request a ticket for any service
- KDC sends a ticket (TGD-REP), does not decide if the user has access
- The ST is is encrypted using the target service's password hash

If the KDC receives a TGS-REQ with a valid TGT, it will send back a response, TGS-REP. The KDC will validate whether the service to which the client requests access exists and subsequently creates a service ticket (ST) that is sent back. It's important to note that the KDC does not do any validation of privileges. The target service will do that by itself by reviewing the PAC that is included in the service ticket (see below).

The service ticket has two parts:

A client portion: encrypted using the Client/TGS session key (so this can be decrypted by the client).

A server portion: encrypted using the target long-term key (the password hash of the target service). This server portion also includes the PAC of the user. It is this service ticket that the user will subsequently submit to the service they are trying to access.

Illustration inspired by "Abusing Microsoft Kerberos - Sorry you guys don't get it", Benjamin Delpy (Blackhat USA 2014).

A client can request a ticket for any service


The client does not need permissions to the service


The service could be inaccessible due to a firewall


The service could be offline


The server could be recycled, as long as the SPN exists

KDC/DC does not make decisions on whether a user should have access; that is solely up to the target service!

A client can request a ticket for any service. Remember, the KDC/DC does not make decisions on whether a user should have access; that is solely up to the target service.

A client can request a ticket for a service EVEN IF:

The client would not have permissions to access the service

The service is not accessible to the client, possibly due to firewall rules

The service could be offline, possibly due to a service outage or crash

The server running the service could be removed from the environment and sent to recycling, as long as the SPN still exists

Remember, the KDC does not make any decisions about whether the user should get the ticket. Access decisions are solely decided by the member server. The KDC doesn't know if the service is online and accessible either. Really, the KDC doesn't know much about the target service other than it exists because the SPN exists.

Kerberoast attack: Requests tickets and then crack the tickets

- STs are encrypted using the target service account's hash
- Which tickets should be requested?
- There are two types of accounts
  - Computer: Password hashes are random (not crackable)
  - User: Passwords are set by users/admins (may be crackable)
- We want SPNs/tickets tied to user (not computer) accounts

Service tickets are encrypted using the password hash of the target service. We need tickets for cracking, but which tickets?

First, we need to understand the types of accounts in the domain.

Computer accounts: these accounts identify the computers in the domain. These accounts still have a password hash, but the password hashes of these account is randomly generated. This makes cracking infeasible.

User accounts: these account passwords are set by a user or admin. This means that a human may have selected a bad (crackable) password. User SPNs are the ideal target for a Kerberoast attack!

Multiple tools exist to get tickets

- Impacket's GetUserSPNs.py – requests user SPNs tickets, saves in crackable format usable by JtR and Hashcat
- Invoke-Kerberoast – part of PowerSploit and Empire
- Other C2 features, manual with PowerShell, or extract from RAM
- Stealth tip: Go slow to bypass detections

We need to gather tickets that might be crackable. Multiple tools exists to do this:

Impacket GetUserSPNs.py will identify SPNs tied to user accounts and request the tickets. The tickets are then saved in a crackable format compatible with John the Ripper and Hashcat.

PowerShell can be used to request tickets, but it can be slow, and you have to have another tool, such as mimikatz, to extract the tickets in a crackable format

Tickets exist in memory, so any method of requesting a ticket will work (e.g., MSSQL Tools and access the server). Again, you will need another tool to extract the tickets from memory in a crackable format.

Some defensive tools will look for a user requestion a large number of tickets, so it may be stealthier to request one or two tickets at a time.

Kerberoast Attack Steps

1. Query the Active Directory for **accounts with an SPN**
2. **Request RC4 service tickets** from the DC using these SPN values
3. **Extract** received Services Tickets and dump them to a file
4. Launch **offline brute force** attacks against the tickets to recover the credential (NT hash/password) that was used to encrypt it

This attack is even more powerful since service account passwords are rarely changed!

So how can we launch such an attack? Several tools are available that implement this attack strategy. Most of them operate using the following steps:

Query the Active Directory for accounts with a Service Principal Name (SPN)

Request RC4 service tickets from the DC using these SPN values

Extract received Services Tickets and dump them to a file

Launch offline brute force attacks against the tickets to recover the credential that was used to encrypt the Service Ticket

This attack is even more powerful since service account passwords are rarely changed. A typical standard user account used by an employee is rotated every 90 days. The author of this course has seen service accounts with password over a decade old! Old passwords may not be compliant with modern password policies. Also, it means an attacker can spend a long time cracking the password with the knowledge that the account may be the same in the future, even a distant future.

What Service Accounts are Good Targets?

Service accounts often have elevated domain privileges or access to sensitive data

- Focus on service accounts that are interesting!
- AGPMServer: Microsoft Advanced Group Policy Management
- MSSQL/MSSQLSvc: SQL server (often Domain Admin too!)
- FIMService: Forefront Identity Manager Service
- STS: Security Token Service (VMware)

During a typical penetration test, you will see many others that are customary to the organization. Proper account and group membership enumeration will help you target interesting accounts!

Service accounts often have elevated domain privileges or provide access to sensitive data. If we can compromise the account, it could lead to compromise of the entire domain. At a minimum, it will provide access to the system running the service. You may be able to log in with the account directly to the computer or you may need to perform a Silver Ticket attack (discussed later).

What service accounts could be interesting? There can be quite a few of them! Here are some criteria to take into account:

- The account should obviously have some elevated privileges that could be interesting.
- The account should not be a computer account, as those are typically not possible to crack.
- Here are some examples of potentially interesting accounts (from adsecurity.org).
- AGPMServer: Microsoft Advanced Group Policy Management –Often has full control rights to all GPOs.
- MSSQL/MSSQLSvc: Admin rights to SQL server(s), which often has interesting data. This account is often a Domain Admin too!
- FIMService: Forefront Identity Manager Service – Often has admin rights to multiple AD forests.
- STS: Security Token Service (VMware) –VMware SSO service which could provide backdoor VMware access.

These are just some examples of interesting service accounts. During a typical penetration test, you will see many others that are customary to the organization. Proper account and group membership enumeration will help you target interesting accounts!

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Please work on below exercise.
Lab 5.1: Kerberos

Please go to Lab 5.4: Golden Ticket in the SEC560 Workbook.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

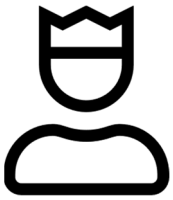
Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Domain Domination and AD Persistence

Domain Domination

- Throughout the course, we've discussed a variety of techniques that we can use to move laterally through a Windows environment and obtain administrative access. Of course, we also would like to persist the access we achieve!
- Common techniques that are used to consolidate/persist administrative access to the AD:
  - Dump the NTDS.dit file
  - Creation of a domain administrator account
  - Creation of a Skeleton Key
  - Use of DCSync or DCShadow
  - Creation of a golden ticket



Throughout the course, we've discussed a variety of techniques that we can use to move laterally through a Windows environment and obtain administrative access. Of course, we also would like to persist the access we achieve!

So, what is the next step in our toolbox? Typically, we can attempt one of the following tricks to consolidate/persist administrative access to the AD:

- Dumping the NTDS.dit file from which all credentials can be extracted
- Simply creating a domain administrator account (preferably with strong credentials)
- Creating a Kerberos golden ticket that will allow long-term access to the environment
- Creating a Skeleton key, which will allow us to authenticate as any user in the environment
- Using the DCSync attack
- Let's discuss these techniques in some more detail!

Domain password hashes are in the ntds.dit file

- It is only accessible with elevated privileges and locked by the OS while the domain controller is running
- It is encrypted with system key (stored in the registry)
- We have multiple ways to access this file:
  - Use the Volume Shadow Copy Service to create a read-only copy and download the file
  - Use ntdsutil.exe and the "Install from Media" (IFM) capability
  - Poorly secured backups of the domain controller drives (e.g., open network shares)
- We need to use special tools to decrypt and extract the hashes
  - secretsdump.py from Impacket is the most commonly used tool

Advanced attackers will often attack the Active Directory infrastructure of their targets because they contain the "keys to the kingdom". If we obtain domain admin rights, we will have unlimited access to the resources of that domain. We may also want to achieve persistence, unconditional access to the administrative functions of Active Directory, even when credentials are changed.

There are many attacks possible with Active Directory. Here we want to focus on attacks to obtain credentials and/or access. As we saw, the Active Directory database is stored inside a file called ntds.dit. And the hashes are protected by an encryption key stored in the system registry.

When we obtain a copy of these files, we can extract hashes. The hashes can be used for cracking or pass-the-hash attacks.

These files are present on a domain controller, but when the domain controller running, these files are locked and cannot be copied. One way to access the files is using the Volume Shadow Copy feature used for backups. This feature will create a shadow copy of the ntds.dit file and the SYSTEM registry hive.

Another command way to access these files is by using the ntdsutil.exe tool and the "Install from Media" (IFM) capability. This feature will also give us a readable copy of both files.

We don't always need access to a domain controller to obtain these files. Sometimes, organizations will leave backups on a poorly secured When centralized backups are made of the domain controllers, these files will be found on the backup servers too.

We need to use specialized tools to decrypt and extract hashes from ntds.dit. The most commonly used tool for this is secretsdump.py from Impacket.

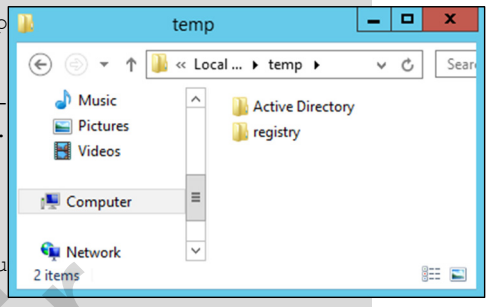
```
C:\> ntdsutil.exe
ntdsutil.exe: activate instance ntds
Active instance set to "ntds".
ntdsutil.exe: ifm
ifm: create full c:\temp
Creating snapshot...
...trimmed for brevity...
Initiating DEFRAGMENTATION mode...
     Source Database: C:\$SNAP_202112131421_VOLUMEC$\Windows\NTDS\ntds.dit
     Target Database: c:\temp\Active Directory\ntds.dit

                Defragmentation  Status (% comp

          0    10   20   30   40   50   60   70
          |----|----|----|----|----|----|----|---
          ....................................

Copying registry files...
Copying c:\temp\registry\SYSTEM
Copying c:\temp\registry\SECURITY
Snapshot {c6091d4b-1775-46ab-9158-36cfb2385133} u
IFM media created successfully in c:\temp
ifm: quit
ntdsutil.exe: quit

C:\> ntdsutil.exe 'ac i ntds' 'ifm' 'create full c:\temp' q q
```



Oneliner version

As mentioned, the domain password hashes are stored in the encrypted ntds.dit file. The encryption key is stored in the SYSTEM hive. We can use the ntdsutil.exe tool to extract the files. As shown in the output above, we run ntdsutil.exe and enter an interactive shell. Then we run a serious of commands to save the backup:

- activate instance ntds
- ifm
- create full c:\temp
- quit
- quit

The directory location above (c:\temp) can be change to another location.

Instead of running ntdsutil.exe in interactive mode, we can use quotes and run it in a single line.

- ntdsutil.exe 'ac i ntds' 'ifm' 'create full c:\temp' q q

The resulting file structure under c:\temp looks like this:

Active Directory

- ntds.dit

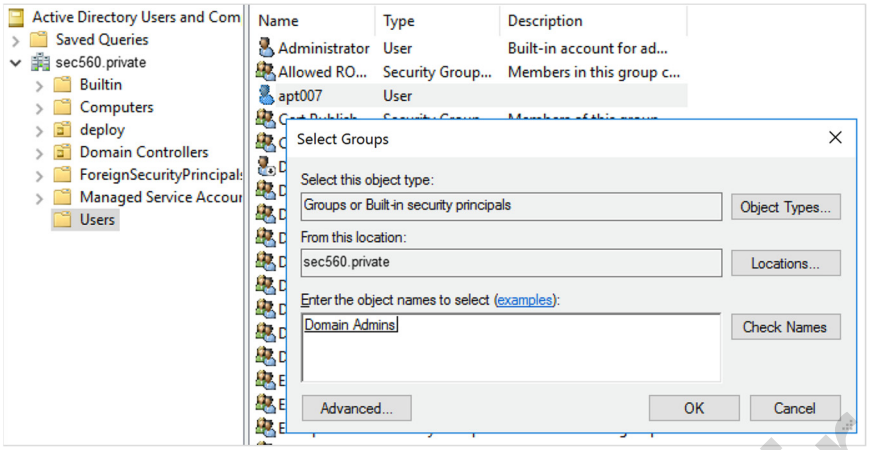
Registry

- SECURITY
- SYSTEM

The "Install from Media" feature is designed to help admins create a copy of their domain. Let's say an organization has one location in New York and they want to setup a new location in LA. They can make a backup of a New York DC, fly to LA, then import the data. The goal is to save the bandwidth and time syncing over the long link. Syncing over these long links was a bigger issue when we had slower internet speeds and internet was metered. With today's internet speeds this feature isn't often used by actual administrators anymore.

Creating a Domain Admin Account

Domain Domination



- Simple, yet highly effective
- Just create a new domain admin
- Set the password to not expire
- Note: Some orgs will alert on additions to administrative groups in AD

A simple attack to achieve persistence in Active Directory is to create a new domain admin user with a password that never expires.

Once we have administrative rights to the domain, we can create a new domain administrator. This does not necessarily imply that we must obtain domain administrator rights, but just obtain the right to create new users and assign them to groups. A domain administrator has these rights, but in Active Directory, these rights can also be delegated to administrative users that don't have domain admin rights. Because the domain administrator is such a powerful account, many organizations will only provide this account to a select group of security staff members.

Other users that need to perform common administrative tasks, like managing users, are only given the necessary rights to do this. But once a user has the right to create a new user and assign it to arbitrary groups, they can create a domain administrator.

This created account will give us domain admin access to the domain as long as the account is not discovered and removed. Companies often monitor their Active Directory infrastructure for the creation of new accounts with administrative rights.

Skeleton Key

Domain Domination

Skeleton key attack allows anyone to authenticate as any user with the password of "mimikatz"

- A skeleton key is a key that opens all the locks in a building. In a similar fashion,
- The Skeleton Key only works for Kerberos **RC4 encryption**
- The Skeleton Key is a backdoor that runs on the Domain Controller (**in memory**), allowing logon to any account using the skeleton password
- As it runs in memory, it **is not persistent** (by itself)

Another AD persistence attack we want to illustrate is an attack to achieve persistence inside a domain.

A skeleton key is a special key that opens all the locks inside a building. Each door lock inside the building requires its own key to be opened, but all the locks can also be opened with a special, single key: The skeleton key. Mimikatz provides a skeleton key attack for Active Directory.

When the Mimikatz skeleton key attack is executed on a domain controller, a bit of code is patched in memory so that all accounts can be logged in with a special password (mimikatz). This does not remove the existing passwords. It is now possible to log in to an account using two different passwords: the real password, or "mimikatz".

Technically, the Skeleton Key does this by manipulating the way the encrypted timestamp (AS-REQ) is validated. As a reminder, in RC4, the timestamp is encrypted using the NT hash of the user by the client, after which the domain controller attempts to decrypt the timestamp. When the Skeleton Key is installed, the domain controller will attempt to decrypt the timestamp using the user's NT hash AND the skeleton key NT hash (mimikatz default: 60BA4FCADC466C7A033C178194C03DF6, password "mimikatz").

This has to be done on a domain controller; it does not work on non-domain controllers. But this does not mean that we can only log in with the skeleton password on a domain controller. The password works on all domain members that use this compromised domain controller for authentication. Since the password is used to generate a normal valid TGT, the TGT can be used to access other domain resources.

If there are more than one domain controllers, the attack needs to be executed on all domain controllers to be sure that the skeleton password will work in all cases.

Skeleton Key in Action

Domain Domination

```
mimikatz 2.1.1 x64 (oe.eo)

  .#####.   mimikatz 2.1.1 (x64) built on Jun 18 2017 18:46:28
 .## ^ ##.  "A La Vie, A L'Amour"
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                     with 21 modules * * */

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # misc::skeleton
[KDC] data
[KDC] struct
[KDC] keys patch OK
[RC4] functions
[RC4] init patch OK
[RC4] decrypt patch OK

mimikatz # _
```

- Setup Skeleton Key with Mimikatz
- Allows anyone to authenticate as any user in the domain with the password "mimikatz"
- Note: there is no option to change the password (must recompile Mimikatz)

In the example above, we can see the skeleton key attack. Since this will patch the code of the LSA process in memory, administrative rights are required, and the debug privilege needs to be enabled. We just need to run the command "misc::skeleton" and the skeleton key is installed on the domain controller.

When there are several domain controllers inside a domain, domain members connect to a domain controller for authentication via a kind of load-balancing scheme.

This means that to be 100% reliable, the skeleton key attack needs to be performed on all domain controllers. Otherwise, a domain member might authenticate to a domain controller that does not have the skeleton key patch applied in memory.

Since this is a patch in memory, simply rebooting the domain controller removes the skeleton key. But, of course, we can install an autorun entry for Mimikatz to run automatically when the domain controller boots.

DCSync attack uses the domain replication protocol to mimic a DC

- For availability reasons, many ADs have multiple domain controllers and updates propagate via "AD Replication"
- Mimikatz "dcsync" will impersonate a DC and request replication to a target DC to obtain the credentials stored in its AD database
  - This attack requires Domain Admin or replication privileges
- Same impact as copying the ntds.dit file
  - Once we successfully launch an attack like this, all password hashes in the domain are compromised!

Benjamin Delpy, the author of Mimikatz, has pioneered many attacks on Windows security, and this has led to security improvements in Windows. In collaboration with Vincent Le Toux, Benjamin worked out another attack on Active Directory: Impersonating a domain controller.

For availability reasons, administrators deploy more than one domain controller in an Active Directory infrastructure. Each domain controller has a copy of the Active Directory database, and updates to this database on a domain controller (for example the creation of a new user) needs to be propagated to the other domain controllers in due time. This is called Active Directory replication: a set of methods and protocols to synchronize the database of Active Directory domain controllers.

Vincent and Benjamin have worked out these methods and protocols for use by Mimikatz: Mimikatz has a command (dcsync) that will make any computer that runs Mimikatz impersonate as a Domain Controller to a target Domain Controller to obtain the credentials stored in this Domain Controller.

Of course, normal users cannot access this information. One needs domain admin rights to be able to participate in data replication. A Golden Ticket can provide these admin rights.

dcsync can dump the hashes of all users or of a selected user.

Replicating the Domain: DCSync Example

Domain Domination

```
mimikatz # lsadump::dcsync /user:administrator
[DC] 'sec560.private' will be the domain
[DC] 'WIN-PGF4RHUE40J.sec560.private' will be the DC server
[DC] 'administrator' will be the user account

Object RDN              : Administrator

** SAM ACCOUNT **

SAM Username            : Administrator
Account Type            : 30000000 ( USER_OBJECT )
User Account Control    : 00010200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD )
Account expiration      : 1/01/1601 2:00:00
Password last change    : 10/07/2017 19:26:57
Object Security ID      : S-1-5-21-1737389956-3911202689-1728583289-500
Object Relative ID      : 500

Credentials:
 Hash NTLM: ae974876d974abd805a989ebead86846
   ntlm- 0: ae974876d974abd805a989ebead86846
   ntlm- 1: e19ccf75ee54e06b06a5907af13cef42
   lm  - 0: b84c6269fc243eefa5a4c667a7ec9656

Supplemental Credentials:
* Primary:NTLM-strong-NTOWF *
    Random Value : 6f537d1aecba5db626dbdfea767a84f3
```

- Request the hash with:

`lsadump::dcsync /user:`*`[username]`*

- Also retrieves previous password hashes
- Requesting the krbtgt hash would allow us to follow up with a golden ticket attack!

This example shows the dcsync command. By issuing the "kerberos::dcsync /user:administrator" command, we request the credentials for user administrator to a domain controller. Command "kerberos::dcsync" would list the credentials of all users.

When this command is issued without extra options, Mimikatz selects the domain and the domain controller automatically, and extract the credentials from this domain controller via replication; the Directory Replication Service Remote (DRSR) protocol.

This is a very powerful attack: Once we have obtained domain admin credentials, we can use dcsync to connect to a domain controller and extract the credentials of the krbtgt account. This data can then be used to create a Golden Ticket, and then it is game over: The only recourse a company has is to change the krbtgt account password. This password never expires and is never changed, unless it is done manually.

It is possible to detect a dcsync attack. DRSR network traffic should only occur between domain controllers. If a company detects DRSR network traffic between a domain controller and a workstation, it knows a dcsync attack took place.

Becoming a Domain Controller: DCShadow

Domain Domination

DCShadow is the reverse of DCSync, it pushes updates to the DC

- As a prerequisite, we need to obtain Domain Administrator rights;
- Using Mimikatz, temporarily register the system we are using as a DC
- We craft a change in the schema that benefits us (e.g., change the password hash of a sensitive account);
- Using Mimikatz, we can now trigger a replication, which forces a legitimate DC to commit the change!

Windows normally relies on event logs being generated on the source DC that creates the changes. As in this case, this DC does not exist; no Windows event logs are being generated!

"They told me I could be anything I wanted, so I became a Domain Controller". With this tag line, Benjamin Delpy and Vincent Le Toux introduced their new AD attack strategy in early 2018 called "DCShadow". While it appears very similar to DCSync, it's more intrusive and definitely different!

DCShadow rose to prominence after BlackHat USA 2018, where Vincent Le Toux and Benjamin Delpy presented their research. The slides can be found here: https://www.dcshadow.com/us-18-Delpy-LeToux-So-I-Became-A-Domain-Controller.pdf. It has a public website with information as well on https://www.dcshadow.com.

How does the "DCShadow" attack work? There are four main steps in a DCShadow attack:

1. As a prerequisite, we need to obtain Domain Administrator rights;
2. Using Mimikatz, we temporarily register the workstation we are using as a DC;
3. We craft a change in the schema that benefits us (e.g., change the password hash of a sensitive account);
4. Using Mimikatz, we can now trigger a replication, which forces a legitimate DC to commit the change!

The changes performed by DCShadow are done using replication, which renders them almost "invisible" for normal Windows event logs. Windows normally relies on event logs being generated on the source DC that creates the changes. As in this case, this DC does not exist; no Windows event logs are being generated!

The "promoted" workstation is afterwards unregistered again (it does not become a permanent DC).

It's important to note that DCShadow uses the built-in features of the Directory Replication Service. It does not perform any type of exploit.

Becoming a Domain Controller: DCShadow in Action (1)

Domain Domination

- We need to run in the "NT AUTHORITY\SYSTEM" context

```
mimikatz # token::elevate
```

- Example: Change the description of a user

```
mimikatz # lsadump::dcshadow /object:"CN=Bob,OU=Users,DC=domain,DC=com"
/attribute: description /value: "DCShadow was here"
```

- Example: Add user to Domain Admins group

```
mimikatz # lsadump::dcshadow /object:"CN=Bob,OU=Users,DC=domain,DC=com"
/attribute: primaryGroupID /value: 512
```

- Push change with: `mimikatz # lsadump::dcshadow /push`

To begin this attack, we first need to run under the SYSTEM context. To get this token we use the elevate function in Mimikatz.

mimikatz # token::elevate

Next, we make changes to objects. In this example, we're change the Description for Bob to "DCShadow was here":

mimikatz # lsadump::dcshadow /object:"CN=Bob,OU=Users,DC=domain,DC=com" /attribute: description /value: "DCShadow was here"

The above change won't have much impact. Let's do something much more interesting and add a user to the Domain Admins group.

mimikatz # lsadump::dcshadow /object:"CN=Bob,OU=Users,DC=domain,DC=com" /attribute: primaryGroupID /value: 512

After we have all the changes we would like, we need to push the updated to the domain.

mimikatz # lsadump::dcshadow /push

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Please work on below exercise.
Lab 5.2: Domain Domination

Please go to Lab 5.2: Domain Domination in the SEC560 Workbook.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

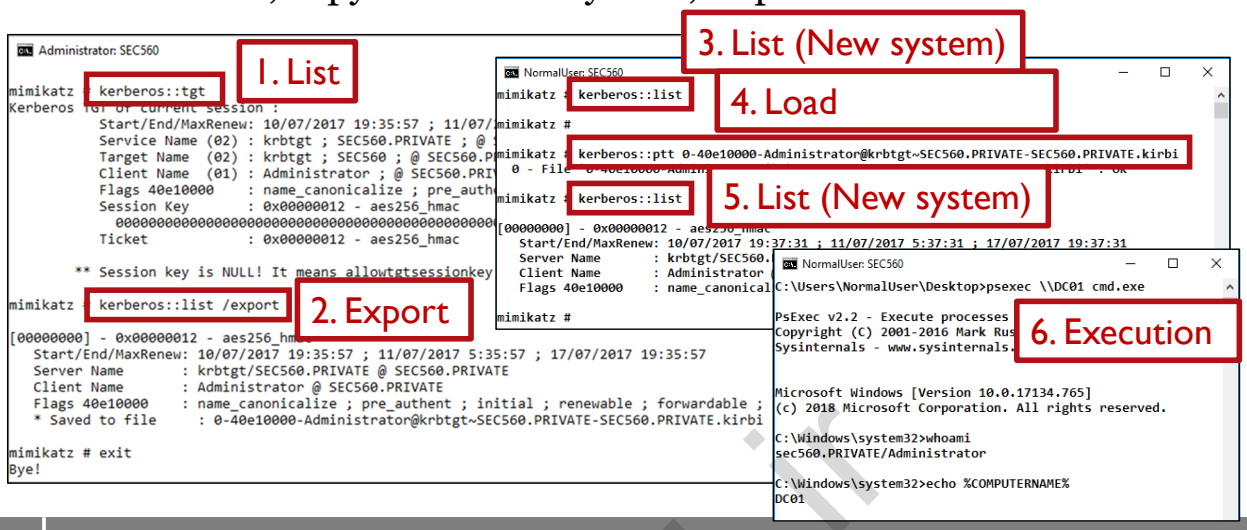
A stolen Kerberos ticket can be used on another system

- Credential theft technique that allows access as the user but without compromising the password
- A ticket can be taken from any system and be used anywhere else
  - Only valid until expiration time
  - ST: Only valid for the target service
  - TGT: Can be used for any service
- Tools: Mimikatz and Rubeus

This page intentionally left blank.

Pass-the-Ticket: Mimikatz Example

Save the ticket, copy to another system, import



To take advantage of this attack, we need to export a ticket from one system, copy it to another system, and then use a tool that will use the ticket. In the example above, we use Mimikatz to perform the attack. The steps above are listed here:

Victim system

1. We use the kerberos::tgt to list the ticket granting tickets
2. Export the ticket with kerberos::list /export

Attacker system

3. On the new system, we list the tickets and see that the system has no tickets in memory
4. We load the copied ticket using kerberos:ptt filename.kirbi to load the ticket into memory
5. When we list the tickets again, we can see the new ticket loaded into memory
6. We now use any tool that will use Kerberos authentication. In this example, the we are using Sysinternals PsExec.

Overpass-the-Hash

- The first step in Kerberos authentication is to encrypt a timestamp with the user's hash and receive a TGT
- If we have the user's hash, we can perform the AS-REQ/AS-REP and get a valid TGT for the user (not a privilege escalation)

When you are reading this, it's good to remember the initial Pass-the-Hash attack. Remember that Pass-the-Hash relies on NTLM authentication, so a possible defense might be to just disable NTLM. While this is not a straightforward control to implement, it also doesn't fully protect us against Pass-The-Hash:

We could use the stolen NT hash to perform pre-authentication and get a valid TGT for the user. The diagram on this slide provides an overview of how Pass-the-Hash (PtH) can still be an issue even if NTLM is fully disabled in an environment! This technique is commonly referred to as "Overpass-the-Hash".

We can forge a TGT if we have the hash of the krbtgt account

- The TGT is signed and encrypted using the password hash of the krbtgt account
- We can generate a forged PAC and give more permissions
  - Make ourselves a domain admin
- We'll cover this attack in more depth shortly

If we have the password hash of the krbtgt account, we can forge a TGT. As part of the attack, we generate a new PAC that gives us more permissions, usually, that of a domain administrator (RID 512). With this level of permissions, we can access nearly every system in the domain. This allows for very effective pivoting and persistence. We won't go into the full details of this attack right now, but we will cover it shortly.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

If we have the password (or hash) we can forge a Service Ticket

- ST is encrypted and signed using the service account's hash
- We can even modify the PAC to escalate privileges
  - Change the username, even to a non-existent name
  - Modify group membership
- This will work as long as the service doesn't ask the KDC to validate the second signature
  - Services set to "Run as part of the operating system" will not verify
  - CIFS (SMB), HOST, MSSQLSvc (SQL Server), TERMSRV won't verify
  - HTTP with IIS will always verify due to the nature of AppPools

Service Tickets are encrypted and signed using the service account's password hash. This means if we have the service account's password (or hash) we can forge the ST for the service since we can both sign the PAC and encrypt the ticket.

We can change anything we want to in a forged ticket. We can change the username, even to a username that doesn't exist on the computer or in the domain. We can also change the group memberships so that we can access the system that we normally wouldn't have privileges to access.

Remember, there are two signatures on the PAC. The first is signed using the password hash of the service, which we have. The second signature uses the password hash of the krbtgt account. If we don't have this password hash, then we can't forge this second signature. This isn't usually a problem since the krbtgt signature is only verified on some services. Services that "run as part of the operating system" will not perform this secondary check. Service classes that will not perform the check include CIFS (SMB), HOST, MSSQLSvs (MSSQL Server), and TERMSRV (Terminal Services). However, HTTP will check when running with IIS due to the way accounts are handled in AppPools.

Generating a Silver Ticket with Impacket


```
ticketer.py -nthash [ntlm_hash] -domain-sid [domain_sid]
-domain [domain_name] -spn [service_spn] [user_name]
```

- We can generate a silver ticket using ticketer.py from Impacket
  - **-nthash** Target service's password hash
  - **-domain-sid** Domain SID (e.g., S-1-5-21-1339291983-1349129144-367733775)
  - **-domain** Full domain name
  - **-spn** SPN of the target service
  - **user_name** Username in the forged ticket (does not have to exist)

We can use ticketer.py from Impacket to generate a silver ticket using the following syntax:

ticketer.py -nthash [ntlm_hash] -domain-sid [domain_sid] -domain [domain_name] -spn [service_spn] [user_name]

We can generate a silver ticket using ticketer.py from Impacket

-nthash Target service's password hash

-domain-sid Domain SID (e.g., S-1-5-21-1339291983-1349129144-367733775)

-domain Full domain name

-spn SPN of the target service

user_name Username in the forged ticket (does not have to exist)

We then need to load the ticket in Memory (Windows) or tell Linux to use the ticket. Let's take a look how to do that.

Silver Ticket Use

Silver Ticket

Linux:

- Set the KRB5CCNAME environment variable

```
export KRB5CCNAME=[TGS_ccache_file]
```

- Use any tool capable of using Kerberos authentication (Impacket!)

```
wmiexec.py [domain]/[user]@[hostname] -k -no-pass
```

Windows:

- Load the ticket with Mimikatz: `kerberos::ptt [ticket_file]`
  - Mimikatz can generate the silver ticket too!
- Load with Rubeus: `Rubeus.exe ptt /ticket:[ticket_file]`
- Almost every tool will use the ticket in memory

Once we create the ticket, we need to tell our operating system (and tools) to use it.

Linux

On Linux, we need to set the KRB5CCNAME environment variable to tell Linux to use the ticket when authenticating.

```
export KRB5CCNAME=[TGS_ccache_file]
```

Once we have this variable set, we can use any tool that will authenticate with Kerberos. The Impacket suite supports Kerberos, so we'll use that in our example.

```
wmiexec.py [domain]/[user]@[hostname] -k -no-pass
```

We need to specify the domain, username, and the target hostname. The -k tells the tool to use Kerberos to authenticate. Since we are using Kerberos, we have no need for a password and use the -no-pass option.

Windows

On Windows, we can use Mimikatz to load the ticket. We could use Mimikatz to generate the ticket, but we aren't going to demonstrate that here. In Mimikatz, we use the ptt feature (pass-the-ticket) to load the ticket into memory.

```
kerberos::ptt [ticket_file]
```

We can also use Rubeus, a Kerberos attack toolkit, to load the ticket.

```
Rubeus.exe ptt /ticket:[ticket_file]
```

We expect to use Kerberos authentication in the Windows domain, so most every tool will use this ticket in memory. We can use attack tools, or just normal admin tools, such as MSSQL Admin Tools.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Please work on below exercise.
Lab 5.3: Silver Ticket

Please go to Lab 5.3: Silver Ticket in the SEC560 Workbook.

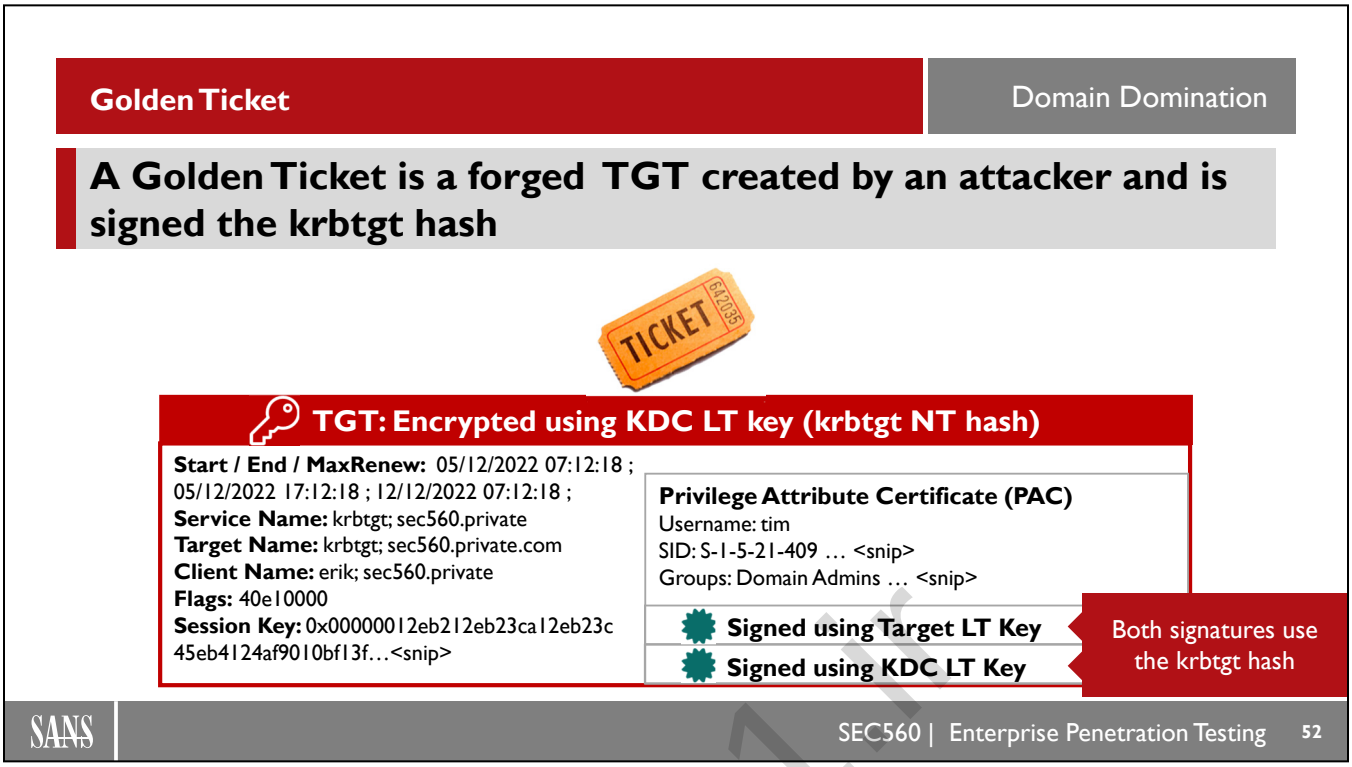
Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.



A Golden Ticket is a ticket-granting-ticket providing maximum access for a maximum period of time.

We discussed Kerberos attacks earlier before! One of the possible attack vectors we looked at was the creation of a "Golden Ticket"! A Golden Ticket is nothing more than a forged TGT created by an attacker.

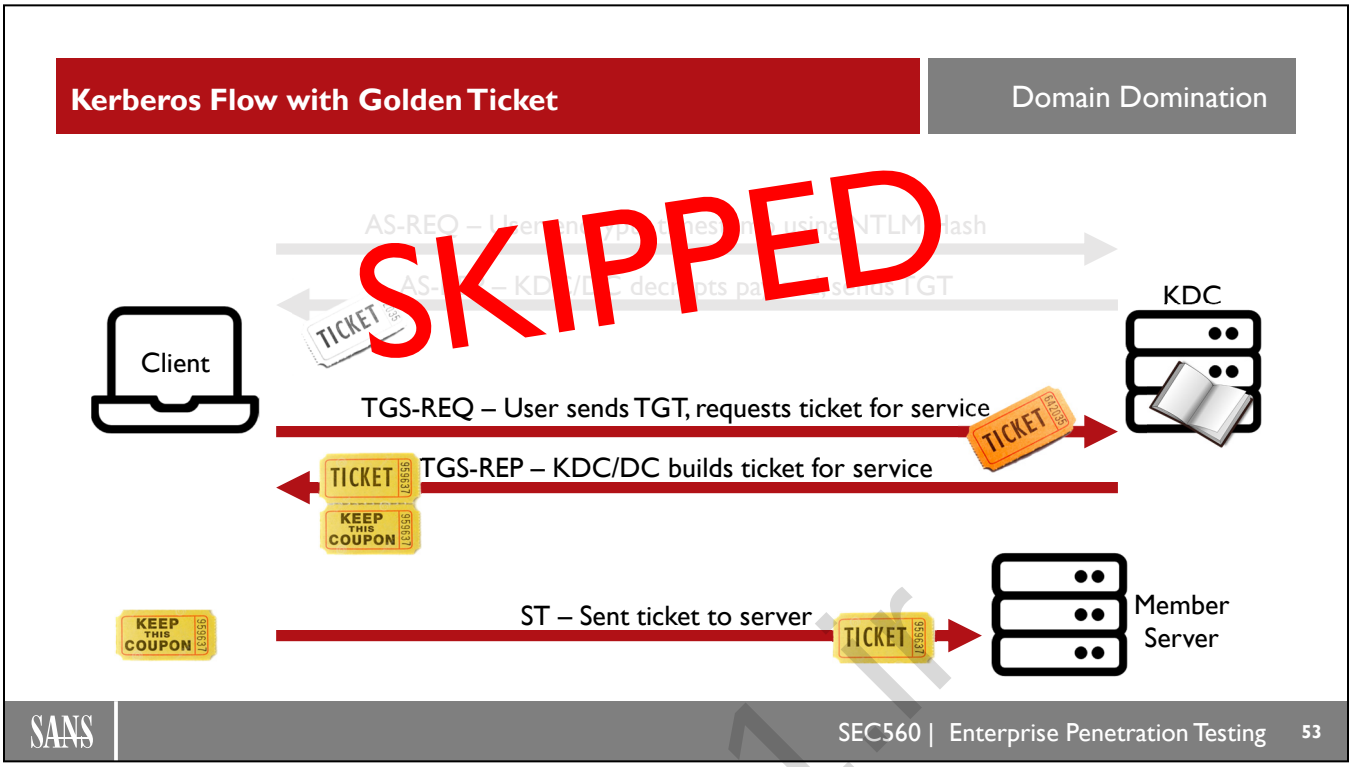
In order to create a valid TGT (with a valid PAC), we would require:

- The Target LT Key
- The KDC LT Key

In case of a TGT, the target and KDC keys are identical (krbtgt hash). We would thus have to obtain the NT hash of the krbtgt account (RC4) or the AES key (AES)!

When Active Directory is compromised, the NT hash or AES key of the krbtgt account can be extracted from memory. All the other information needed to create a Golden Ticket is not secret information but can be obtained readily. For example, the name of the domain will be needed. Although the domain name of an organization is not something that is publicized, it is not that difficult for us to obtain the name of a domain. Note that the NT hash or AES key of the krbtgt account can also be extracted from the Active Directory database and its backups.

The name Golden Ticket refers to the Willy Wonka movie. In this movie, children can win a Golden Ticket that provides them full access to a fabulous chocolate factory.



Let's have a quick look at the Kerberos flow when a Golden Ticket is in play. When we would use a Golden Ticket, the first interaction is a TGS-REQ (request for a Service Ticket) using the forged TGT (the Golden Ticket). There is no prior credential submission or AS-REQ/AS-REP!

One might assume that the lack of AS-REQ/AS-REP can be a trigger for detection. Fortunately, Kerberos is a stateless protocol, hence the AD does not keep track of prior AS-REQ/AS-REP before a TGS-REQ is performed. If one were to implement a rule like this, an avalanche of false positives would appear (e.g., TGT's generated by DC01 and subsequently used to request a Service Ticket on DC02).

Golden Ticket Properties

Domain Domination

- It's created without any interaction with the DC
  - This is possible because Kerberos is a "stateless" protocol
  - DC does not keep track of all previously created TGT's
  - We need the krbtgt hash to generate the ticket
- TGT is typically for an administrative account (e.g., RID 500 in the domain or a Domain Administrator)
- It's typically valid for a long time (10 years by default)

A Golden Ticket is nothing more than a forged TGT. However, it does have some very interesting features:

It's created without any interaction with the DC (it's "homemade"). This is possible because Kerberos is a "stateless" protocol. Forgery is possible once the attacker has the krbtgt hash

It's typically a TGT for an administrative account (e.g., RID 500 in the domain or a Domain Administrator) or a member of a powerful group (e.g., Domain Admins)

It's typically valid for a long time (10 years by default)

Golden tickets were initially described in Benjamin Delpy's BlackHat USA 2014 presentation "Abusing Microsoft Kerberos—Sorry you guys don't get it". This presentation is often referred to as "the Golden Ticket talk". You can find the slides here:

https://www.blackhat.com/docs/us-14/materials/us-14-Duckwall-Abusing-Microsoft-Kerberos-Sorry-You-Guys-Don't-Get-It.pdf.

Shortlink: redsiege.com/560/golden

Golden Ticket Tools

- ticketer.py from Impacket
- Mimikatz – the original tool for ticket generation
- Most C2 frameworks
  - C2 will commonly load the ticket into memory, stand alone tools require another mechanism to use or load the ticket
- To generate a Golden Ticket, you need:
  - krbtgt hash
  - Domain SID (S-1-5-21-XXXXXXXXX-YYYYYYYYY-ZZZZZZZZZ)
  - Full domain name
- ticketer.py example:

```
ticketer.py -domain-sid S-1-5-21-721047592-4068106649-2889670365
-domain sec560.local -nthash 5525e655c...e2cc5621fb3 Administrator
```


There are multiple tools we can use to generate a Golden Ticket.

- ticketer.py (part of Impacket)
- Mimikatz – the original tool for ticket generation
- Most C2 frameworks have a way to generate tickets

To generate a Golden Ticket, you need:

- NT password hash of the krbtgt account

Domain SID – This is the Security Identifier for the domain. It begins with S-1-5-21 and then three sets of numbers.

Full domain name

To generate a ticket with ticketer.py from Impacket we use a command like this:

ticketer.py -domain-sid S-1-5-21-721047592-4068106649-2889670365 -domain sec560.local -nthash 5525e655c06299c7e4179e2cc5621fb3 Administrator

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Please work on below exercise.
Lab 5.4: Golden Ticket

Please go to Lab 5.4: Golden Ticket in the SEC560 Workbook.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

In this section, we will be looking for ways to escalate our privileges in the domain. We will be looking for misconfigurations on the local host that will lead to elevated privileges, places in the network where our current user has administrator-level privileges and looking for file shares that may contain credentials or access to sensitive information.

PowerView Find-InterestingDomainShareFile

Domain PrivEsc

- PowerView Find-InterestingDomainShareFile
- Identifies file shares then…
  - Searches for files accessible by the current user on all available file shares
  - Searches for files containing
    - Credentials
    - Passwords
    - Configurations
    - VMDK
- Additional tile types can be added

Another awesome function in PowerView is the Find-InterestingDomainShareFile function. Find-InterestingDomainShareFile searches for all available file shares, creates a list where your user has access, and then searches recursively for files containing keywords in the file name or extension. The default search terms include *cred*, *password*, *config*, and *vmdk*. Additional search terms can be added during the search. We find credentials and configuration files laying around just about every time we use this tool.

PowerView Find-LocalAdminAccess

Domain PrivEsc

PowerView Find-LocalAdminAccess

- Searches for hosts where your current user has local administrator access
- Can be configured with other credentials during scan

Yet another great function in PowerView is the Find-LocalAdminAccess. This function checks every computer in the domain for administrator-level access as your current user. Once you find additional credentials, you can re-run this scan as that user to find additional access.

Privilege Escalation (1)

Domain PrivEsc

Process Memory Dumps

- Requires local admin or SYSTEM level access
- Plaintext credentials may not be available, but NTLM hashes are often present
- Methods
  - Sysinternals Procdump or Process Explorer
  - Task Manager
  - RunDLL32 with Comsvcs.dll
  - PowerSploit Out-MiniDump
  - Direct System Calls/Windows API Calls

Once some level of administrator access is achieved, we can create process memory dumps. The usual target for memory dumps is lsass.exe, which can contain plaintext or NTLM hashed credentials. Starting in Windows 8.1, Microsoft started fixing the plaintext credentials left in memory from Wdigest, CredSSP, TSPKG, etc. However, the NTLM hash of the password can be just as good if we have tooling to perform Pass The Hash attacks. Some methods for performing process memory dumps are:

Sysinternals Procdump or Process Explorer - Both are signed by Microsoft, so they are not usually caught by antivirus. Both will require the ability to execute binaries though so if the application whitelisting is extremely strict, you may have issues.

Task Manager - If you can execute task manager as administrator, you can right-click any process and create a memory dump, including lsass.exe. What's even better is task manager is built into every copy of Windows.

RunDLL32 with Comsvcs.dll - This method requires that you execute from PowerShell as it will have the required SeDebugPrivileges by default. Much like Task Manager, this is built into Windows, so no 3rd party binaries are needed.

PowerSploit Out-MiniDump - For a PowerShell version of memory dumping, you can use the PowerSploit Out-MiniDump function. Similar to the aforementioned methods, administrator access will be required.

Direct System Calls/Windows API Calls - As AV and EDR continue to monitor for the specific methods used by the previously mentioned tools, Red Teamers are having to find new ways to access processes like lsass.exe. Writing custom binaries to read the process memory are becoming more prevalent since they aren't signatured until after they've been executed and will often walk right around AV/EDR.

Privilege Escalation (2)

Domain PrivEsc

We've already discussed a few techniques that may work here!

- Kerberos Attacks
- Kerberoasting
- AS-REP Roasting (sniffing the initial auth request and cracking)
- Responder style attacks

Kerberos tickets are another great way to escalate privileges in the domain. The requested ticket will be encrypted with the target domain user's password. Any domain user can request Service Principle Name (SPN) Kerberos tickets for available services in the domain. The ticket will be encrypted with the service account's password. This is the basis for the Kerberoast attack. Another domain wide Kerberos attack is requesting tickets for users configured without Kerberos Pre-Authentication. This attack is known as AS-REP Roasting. Once you have administrator level access to a host, you can use the tool Rubeus to access Kerberos tickets outside of your current session. Once you have these tickets, you can import them into your current session and effectively become the user for whom the ticket was issued. Additionally, you can pull these tickets to a Linux host and use them with tools such as the Impacket Framework and SMBClient.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Azure

Introduction

- Microsoft has several online services
  - Microsoft Platforms: Azure, DevOps, Xbox cloud
  - Microsoft Business Platforms: Office 365, Exchange, Teams, Power BI
  - Microsoft Identity Services: Azure AD, Microsoft Account
- Microsoft Statement: 95% of Fortune 500s use Azure.
- Many companies "accidentally" using Azure through their Office apps.

Over its long history, Microsoft has created a multitude of online services. Microsoft has had a consumer-facing set of online services like Microsoft's Hotmail. Microsoft Accounts, and even today, Xbox cloud. What about business or enterprise services? Microsoft started its online history somewhat haphazardly. Consumer accounts and security issues show up in backward compatibility challenges that Microsoft has to support across its user base.

Microsoft's foray into Online Services includes its introduction with MSN, Hotmail, and some Consumer services. These services still live today using Microsoft Accounts, known as personal accounts and can sign into some enterprise services. These Microsoft Accounts are akin to Google Consumer accounts that can exist in a consumer environment or are mixed with GSuite accounts. There are several modes that we will discuss, such as Azure B2C and Azure AD External Support.

For Enterprise Users, which is the focus of our class, you have several Online Services that Microsoft offers to businesses. Microsoft claims that 95% of the fortune 500 uses Azure (https://azure.microsoft.com/en-us/overview/what-is-azure/). How is this so? Microsoft's product ranges from Infrastructure and Platform as a Service in Azure and integral business apps like Office, SharePoint, Exchange, PowerBI, and more. Those applications are tightly integrated into Azure AD whether you bring a third-party IdP like Okta, Google, Ping, SailPoint, or Native Active Directory. We will be exploring some of these technology systems throughout the next section.

Azure Services

Introduction

- **Microsoft Azure:** Microsoft's Infrastructure as a Service and Platform as a Service
- **Azure Active Directory (Azure AD):** OpenIDConnect Identity Provider (OIDC IdP)
- Azure allows for Infrastructure to be run in the Microsoft Cloud such as: Virtual Machines, Virtual Networks, File Shares, Shared Databases, etc.
- Azure AD provides the authentication for both Azure, Office365, and third-party services that support Oauth2 and SAML

Microsoft's Platforms include Microsoft Azure Platform and the Microsoft Azure Active Directory System. The Azure Platform is Microsoft's IaaS (Infrastructure as a Service) and Microsoft's PaaS (Platform as a Service). Azure Active Directory (Azure AD) is delivered as a component of the Microsoft Azure Platform but is also intimately tied into Azure.

When you first get access to the Azure Portal, you will be using Azure Active Directory in its freely available version. Azure Active Directory provides the Authentication and Authorization to the Azure Platform; it also can enable 3rd party federation as it is an OpenID Connect (OIDC) Identity Provider (IdP).

Azure IaaS features all of the Cloud-Based Components you would find in typical cloud providers. Virtual Machines, Block and Standard Storage, Networking Facilities, Shared Databases, and almost 100+ discreet services. Azure AD provides us the authentication to both Azure and Office365. Have you ever been in an environment with On-premise Active Directory and Cloud Hosted Exchange? They more than likely have Azure AD to facilitate that access.

- Each online service can have its own single or multiple management portals:
  - Catalogued at: https://msportals.io
  - About 150 individual portal links so far
- Microsoft Azure and Azure AD manageable through the their CLI (SDK) tools
  - Azure PowerShell Module(s) (.NET Based)
  - Azure CLI tool (python based, cross platform)
- Azure Cloud Shell available once your logged into Azure.

Each one of the Microsoft Services will have a separate management portal. For many non-Microsoft administrators, you may be familiar with the Azure Portal or the Office 365 portal. There are, however, over 150 management portals for the various Microsoft services. It is tough to keep track of them. Instead, a user has created a portal aggregation site at https://msportals.io. Some of the portals that you may find helpful in an engagement include:

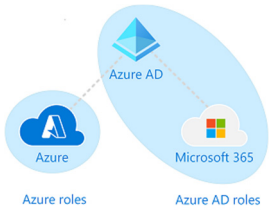
- https://portal.azure.com: The Main Azure Portal
- https://myaccount.microsoft.com: User account management page
- https://security.microsoft.com: Microsoft Security for Defender
- https://endpoint.microsoft.com: Microsoft Intune Page
- https://myapplications.microsoft.com: User enrolled Applications including 3rd party apps

To assess cloud assets' security, we will focus on using the CLI tools that provide equivalent access to the Microsoft systems. Azure PowerShell modules exist for PowerShell version 7, and for Cross-Platform usage, the azure CLI can facilitate access. These tools are typically loaded onto your system or a remote management system. Another option is a built-in shell in the UI called the Microsoft Cloud Shell. The cloud shell provides you with a Linux bash environment with your login credentials in memory. The shell also has a preloaded set of the management tools loaded. The concept of the Cloud Shell is also available in other cloud providers such as Google Cloud Platform.

Azure in relation to Azure

Introduction

- Azure AD's relationship to Azure *can* be confusing
  - Azure AD is a service in Azure
  - Azure AD users and permissions are required to ***use*** Azure
  - Azure AD is an Azure ***requirement***
- Azure's RBAC (Role Based Access Control) IAM permissions are set ***within*** Azure NOT Azure AD
  - This is unique to Azure as Azure AD can set Scopes for 3rd parties like Office'



When setting up Azure, there will be an Azure AD environment for the user and group management. Unlike the standard Azure Active Directory system, Azure AD and Azure are almost intertwined. It is accessible from within the Azure Portal. It is also required for Azure to communicate. Azure AD has many built-in roles that can be used when communicating with 3rd party systems like Microsoft Office or the case of the graphic shown, Microsoft 365.

For example, a User Administrator in Azure AD is also a User Administrator in Microsoft Office; the roles are transmitted between the systems. A User Administrator or Global Administrator Azure AD is not automatically a privileged user in an Azure Subscription. An Administrator must set up those permissions explicitly. This type of access can be confusing. What we have to remember as a tester is the following:

- Azure permissions are set within Azure IAM
- Azure AD roles can be used for specific Microsoft Services such as Office 365

Reference:
https://docs.microsoft.com/en-us/azure/role-based-access-control/rbac-and-directory-admin-roles

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Microsoft Azure AD

Azure AD

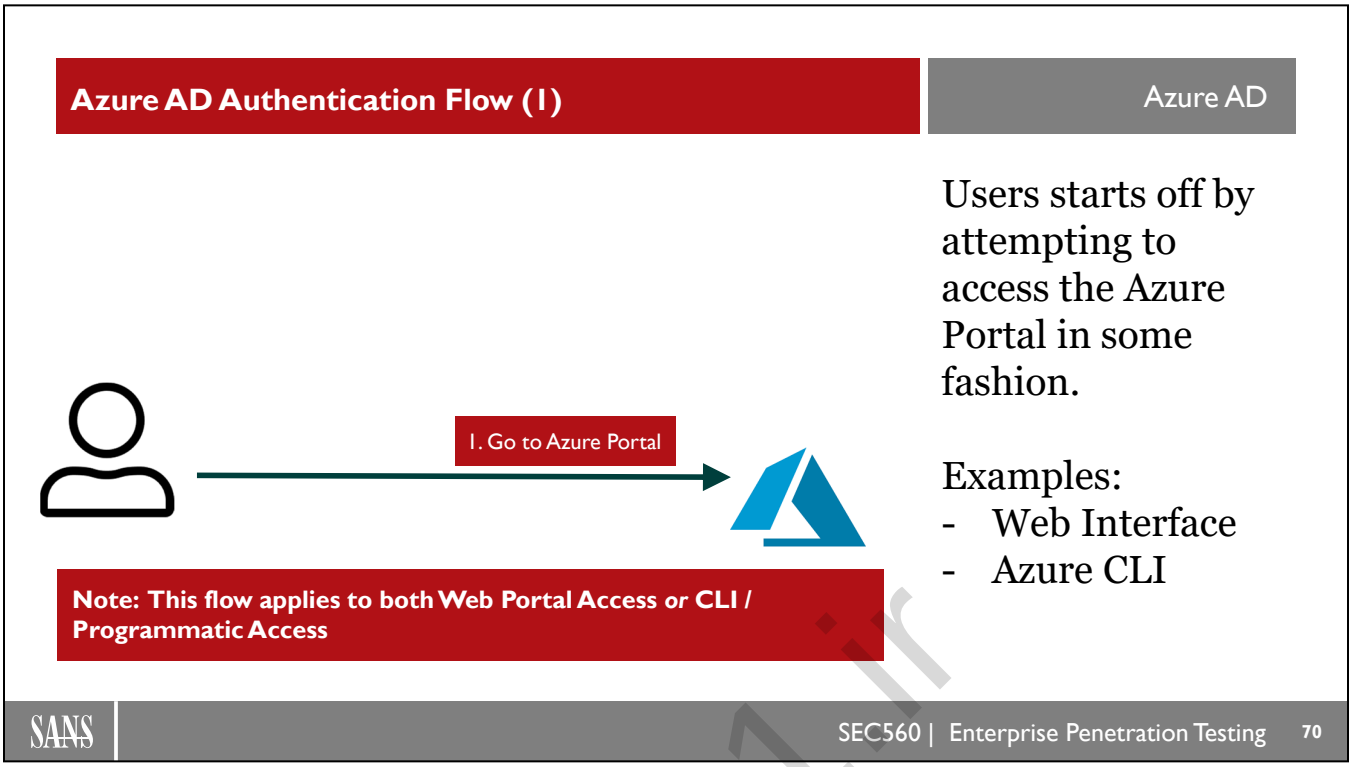
- Azure AD is an Identity Management System storing users and groups for authentication and authorization to third parties
- Azure AD supports several types of Hybrid Authentication modes and can also support "legacy" protocols
  - Current Supported Protocols for Azure AD include OpenID Connect, Oauth2, SAML, WS-Fed, and in preview Microsoft Kerberos
- Azure Seamless SSO can be used to sign-in users to Cloud Applications like Sharepoint automatically
- It does this by passing a domain hint such as:
    - http://myapplications.microsoft.com/hiboxy.com
  - User Accounts are seamlessly authenticated by Domain Joined machines or Azure AD Joined Machines

Microsoft Azure AD is the Authentication and Authorization System from Microsoft that is designed to replace the more classic Active Directory system. Azure AD stores our enterprise's user and group information for use in 3rd party systems and within the Microsoft EcoSystem. Azure AD does not support many of the legacy protocols of AD DS. Instead, it uses the following protocol types:

- Oauth2 (and OpenID Connect)
- SAML
- WS-Fed
- Kerberos (Preview Mode)

Some of the system features include a feature called Desktop SSO (or Seamless SSO), which provides the user with a seamless user experience once they are logged into either the Azure AD Domain by Joining with Azure AD or using AD DS join. With this

Seamless SSO command, a user can open a web browser and authenticate the user by leveraging the user's domain authentication or azure ad authentication within the system.



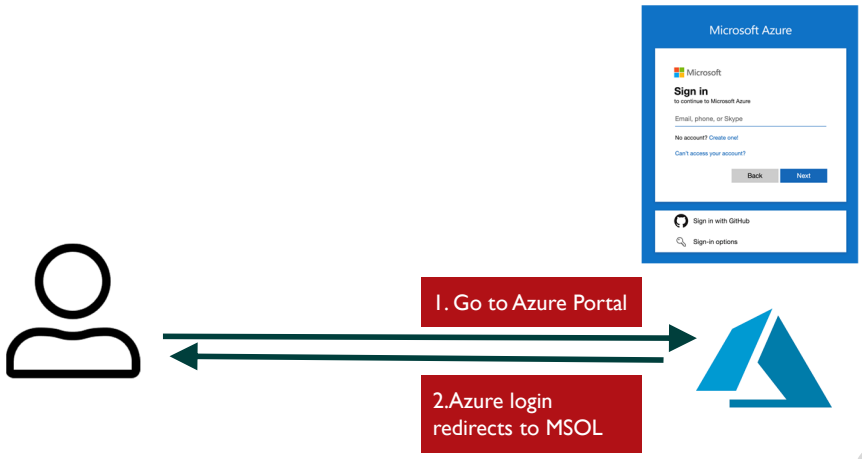
To better understand Azure Active Directory (Azure AD), it is probably best to understand how the user experience and the authentication system operate. It is also good to look at how the different parts of the system are supposed to operate and how they are not. Abuse will occur in the parts of the system that have to support protocols or mechanisms that may have not necessarily been designed with security in mind. When a user wants to log in to a Microsoft Property such as:

- Microsoft Azure
- Microsoft Office 365
- Microsoft's AZ CLI Tool
- Microsoft's PowerShell Azure Module
- Microsoft Online
- Microsoft PowerBI
- And others

The first thing that will happen is Microsoft's authentication system will detect that the email (or User Principal Name) is registered with an Azure AD tenant and will be forwarded to a Microsoft Sign In the page that may seem familiar to you. This flow (usually an OpenID Connect Code Flow) will occur with a command-line interface tool launching a browser. At this point it also good to know that many of the systems that are known as "fat clients" will generally have a registered application in the Azure Active Directory system, and these clients will send that application identifier as a client identifier in the system. This would include clients like Outlook, Word, Azure cli, PowerShell CLI and others.

Azure AD Authentication Flow (2)

Azure AD



Azure Redirects the user back to the MicrosoftOnline Login portal. This is the main front end for many Azure Services.

This is not the only Endpoint to handle Authentication

To make the authentication flow work, any property integrated with Azure AD, including Azure, will redirect you back to the Azure AD login endpoint, which points to login.microsoftonline.com. This is not exclusively the only authentication endpoint; other API endpoints handle authentication that we can also abuse and sometimes do not respect newer technologies like password-less authentications and MFA.

The URL below is an example of what the request to your browser will look like

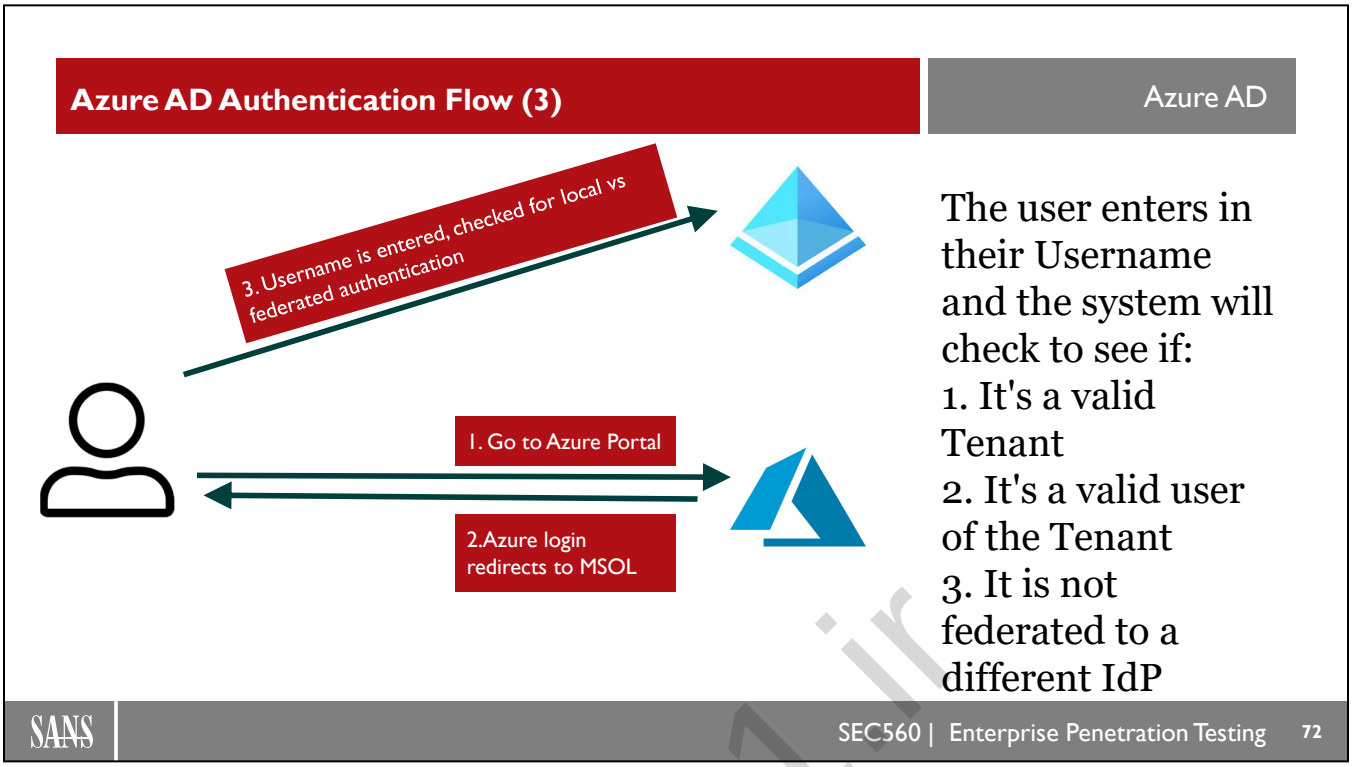
```
https://login.microsoftonline.com/organizations/oauth2/v2.0/authorize?redir
ect_uri=https%3A%2F%2Fportal.azure.com%2Fsignin%2Findex%2F&response_type=co
de%20id_token&scope=https%3A%2F%2Fmanagement.core.windows.net%2F%2Fuser_imp
ersonation%20openid%20email%20profile&state=OpenIdConnect.AuthenticationPro
perties%3DysnyqREDACTEDhOSq7&response_mode=form_post&nonce=6377REDACTEDY1OT
Rk&client_id=c44b4083-3bb0-49c1-b47d-
974e53cbdf3c&site_id=501430&prompt=select_account&client-request-
id=d28c579b-b8e7-

4949-a820-98f0e00c824e&x-client-SKU=ID_NET472&x-client-ver=6.12.2.0
```

The important components of this URL to key in on are the following:

- redirect_uri: Where the browser will be redirected to after authentication.
- scope: This is what you're attempting to request; for example, the scope, in this case, points to https://management.core.windows.net
- client_id: This is the identifier for the web browser we are using.

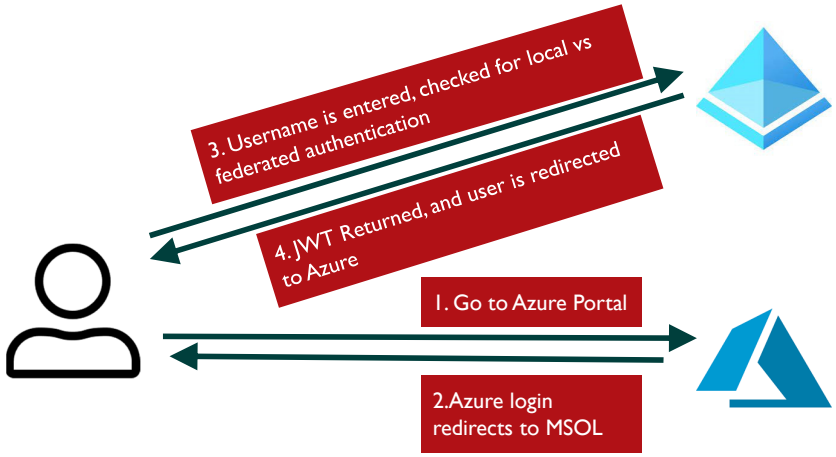
There are other components here that are useful for OAuth and OpenID Connect, but for now, these fields are relevant for our conversation.



The username is now entered into the page, and Azure AD will perform some checks. It will check to make sure that the domain in question is registered. Some domains will be federated; in other words, their IdP is not in Azure AD but instead hosted on another IdP. If you are an Okta or Ping Federate user, this will be where you are redirected to those systems. If not, the next check is that the UPN (User Principal Name), or you may recognize this as the "username," is valid. If the username and password are accurate, the next check is a conditional access policy check. Can the user log in? Does the user's signal match the policy? What are the signals? The browser user agent string, is it Windows Edge or Mobile? Does the user need to perform MFA? Is the user in a trusted location, or does the user require an MFA? These items are validated to ensure that the user can get Key Material.

Azure AD Authentication (4)

Azure AD



Once the user is logged in, they will receive a JWT which is passed to Azure to Validate the user.

Azure will have the user's UPN, Group, or Role check

Several things will happen now that we have a valid username and password. The first is that the Azure AD system will generate and provide a JWT. The JWT (JSON Web Token) serves us as the authorization token as well as the identifier of the user. The system will also redirect you to whatever the redirect_uri value stated in the second portion of this set of slides. Whatever the redirect_uri was set to will ask and receive the access_token. This value will be used to validate the user. We now see how a user can leverage Azure AD to perform authentication and authorization to third parties in this flow. We will be seeing more of this in our labs and over the following few sections.

Microsoft Authentication Systems

Azure AD

Description	Azure AD	On Prem AD DS	Azure AD DS
Communication Protocol	HTTPS	LDAP	LDAP
Authentication Protocol	OIDC, Oauth2, SAML*	Microsoft Kerberos or NTLMv2	Microsoft Kerberos or NTLMv2
Organizational Structure	Flat Structure	Domains and Forests	Single Domain
Access Control	Groups, Scopes for Permissions	Groups, ACLs for Permissions	Groups, ACLs for Permissions
Multi-Domain Support	Azure B2B or B2C	Multi-Forest Mode	Not available

Microsoft has several identity management systems. Its rich history means that Microsoft has legacy or classic identity management systems such as the On-Premise Active Directory Domain Services (AD DS) components and newer systems such as Azure Active Directory. Microsoft also has a hosted version of AD DS, known as Azure AD DS. Several unique differences exist between on-premise active directory domain services and the hosted version.

Microsoft AD DS On-Premise can have multiple forest modes, multiple domains, and many different types of administrators. The hosted version has limitations such as only supporting a single domain, no concept of forests, no extensions of schemas, and no domain administrators. It primarily supports Azure Active Directory users in a Windows Domain. Azure AD DS also doesn't support the AD Connect system similarly to on-premise. Instead, Azure AD pushes users and passwords into Azure AD DS.

Microsoft Azure AD supports multiple authentication protocols beyond OIDC (OpenIDConnect), Oauth v2, and SAML. It supports WS-Federation, Azure B2B Federation, Azure B2C Federation, and in preview as of the time of this writing Kerberos. It is possible to support more than just these protocols as one of its competitors, Okta, supports LDAP. Always check the compatibility of these legacy protocols to newer authentication mechanisms such as Multi-Factor Authentication, which it may not fully support. Multiple Domains and Forests are not possible in this system. Instead of multiple domains, you could have various tenants. There are many caveats to this; however, you can have external guest users in Azure B2B (Business to Business) instead of a cross-forest trust. There is a consumer component called Azure B2C (Business to Consumer). Azure B2C supports applications that wish to have user accounts stored in an OIDC system.

The most significant change you will see between Azure AD and Microsoft's Legacy Domain Services is access control. You may be used to seeing Active Directory ACLs with read and write permissions. These ACLs no longer exist in the same way in Azure AD. Azure AD uses Scopes, such as the scope of a virtual machine in Azure, the entire subscription, and the permission to read or write. Using scope, you can control the ability to "write" changes to a Virtual Machine but not a Network Group. Understanding scope is critical as it will impact what we can do in the system and perhaps, how to escalate privileges.

Identity Architectures in Microsoft

Azure AD

There are several types of Identity Architectures in Microsoft today:

- Microsoft Active Directory Domain Services (Microsoft AD DS)
  - Classic / Legacy On Premise non-hosted Domain Services from Microsoft.
  - Frequently found in Azure as a Domain Controller in a Virtual Machine in the Azure Cloud
- Azure Active Directory (Azure AD)
  - Hosted Azure Active Directory
  - Supports Federating Identities
- Azure Active Directory Domain Services (Azure AD DS)
  - Hosted version of Microsoft AD DS with many limitations

Microsoft has several Identity Management systems that it uses today. We are probably familiar with some of these and less familiar with others. Microsoft's Active Directory Domain Services (Microsoft AD DS) is the one that most users are familiar with. ADDS is an X.509 LDAP-based Object store that stores objects such as user accounts within the object tree. It is classically deployed "On-Premise" but can also be deployed within a virtual machine in the cloud. Frequently we find in Azure backup domain controllers are deployed in the Azure Cloud. The virtual machines deployed in the cloud may confuse some, as we may think this is cloud-hosted ADDS, but it is standard ADDS. It can have an Azure AD Connect Agent.

The second type of system discussed in this class is Azure Active Directory or Azure AD. Azure AD is an OpenID Connect Identity Provider or an OIDC IdP. It can federate 3rd party access and federate to other OpenID Connect IdPs. It is the primary mechanism to log into Azure and Office 365. If you see Office 365, you can rest assured they are more than likely running Azure AD.

The third type of server that we will only lightly discuss here is the Cloud Hosted Azure Active Directory Domain Services, or Azure ADDS. Its primary purpose is to help facilitate legacy SMB and Active Directory access to servers running in the Azure Cloud. The cloud-hosted version of Active Directory has much fewer options and is more secure than the on-premise Active Directory. It can not have users added to it, the users from Azure AD. It doesn't support some of ht more insecure features of Active Directory.

Synchronization and Federation

Azure AD

- Several mechanisms exist to Federate or Synchronize Directories
  - Password Hash Synchronization:
    - Can use the Azure AD Connect Application, of which only 1 agent can be active at a time, the rest are in "staging mode"
    - Azure AD Connect Cloud Sync, this is a hosted version of Azure AD Connect
  - Pass-through Authentication
    - Azure AD Passes the request back to the site
  - AD FS Federation
    - AD FS federates for On-Premise AD
  - Hosted Azure AD DS which has some special properties

Azure AD can either synchronize passwords from an On-premise Active Directory environment or federate to a third party. The federation to third parties can even include on-premise Active Directroy through ADFS. We will discuss these options here as you might encounter them in a penetration test.

Password Hash Synchronization, or PHS, relies on an Azure AD Connect Agent running on a server in the environment. Only one (1) agent can run in a single organization. The user object will then get a special LDAP GUID that will be used to specify that this user account is being synchronized. The object will have a different name, but Microsoft will refer to it as the sourceAnchor. Password Hash Synchronisation will perform several rounds of hashing to take an NT Hash and eventually store it as a PBKDF2 Hash[1].

Pass-Thru authentication is different than password hash synchronization. In pass-thru authentication, a pass-thru authentication agent obtains the request to check the hash. The system will then return a response to azure to sign the user on. This type of synchronization will rely on the usernames and passwords not being checked in Azure AD but being passed to the local on-premise exchange.

ADFS Federation works much like a federation to other third parties. Similar to Pass-Thru Authentication, once the authentication request comes into Azure AD, the request is redirected for federation. Instead of being in the background, the page will be presented to the user. This type of federation is not unique to ADFS but will also be used in other IdPs like Ping Federate.

[1] https://docs.microsoft.com/en-us/azure/active-directory/hybrid/how-to-connect-password-hash-synchronization

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

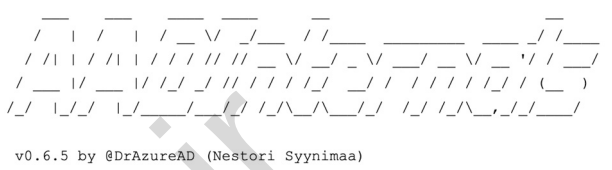
Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Introduction to AADInternals

Azure Recon

- Several tools exist to attack environments running both Azure and Azure AD
- AADInternals from Dr. Nestori Syynimaa (@DrAzureAD) is one of these tools
  - Found on his blog https://o365blog.com
- A collection of Powershell modules useful in both discovery, and recon, and attack of Azure AD



AAD-Internals is a collection of PowerShell scripts that have been created by Dr. Nestori Syynimaa (@DrAzureAD). He maintains a blog at https://o365blog.com. The toolchain comes from many hours of reverse engineering Azure AD calls, as much of the features within the toolchain come from undocumented API calls. The tool does not wrap itself around existing modules. Instead, it re-implements the functionality to display more data than the modules. The tool supports both authentication and unauthenticated sessions.

AAD-Internals has several tools that we can use to perform Reconnaissance Activities on Azure and attack Azure AD. There are over 50+ modules that exist in the system, and many of them are helpful for both external actions and internal actions. The one benefit of the AAD-Internals modules is that they can be used programmatically to make our attacks more realistic, instead of relying on manual scripts or manual interaction with APIs.

Several modules can help us attack an Azure AD environment by leveraging the local system integrations. One of those modules is AADIntPTASpy. This module installed a DLL inside a domain controller to hook and record usernames and passwords passed through the Pass-Thru-Authentication agent in the Azure AD Environment. This module is part of the hacking functionality of AAD-Internals, which is currently a small, but powerful collection of scripts.

Loading and Installation

To install AAD-Internals we can use the following PowerShell Command:

```
PS C:\> Install-Module AADInternals
```

Importing AAD-Internals collection to the current PowerShell Session for use:

```
PS C:\> Import-Module AADInternals
```

https://o365blog.com/aadinternals/#install-aadintptaspy

AAD-Internals Recon

Azure Recon

- AAD-Internals has several modules that are used both External Recon and Internal Recon
  - Part of the O365Blog AAD & M365 Kill Chain
- Several Modules exist to perform both domain reconnaissance
  - Does Azure AD have a tenant with this domain?
- As well as user reconnaissance
  - Do these particular usernames exist within this domain?

```
PS C:\> Invoke-AADIntReconAsOutsider -DomainName hiboxy.com | Format-Table
PS C:\> Get-Content .\Users.txt | Invoke-AADIntUserEnumerationAsOutsider -UserName
```


AAD-Internals does have modules that support both external and internal recon activities. The internal activities rely on having an authenticated session, while the external actions are unauthenticated. This module is part of the O365Blog AAD & M365 Kill Chain work that maps out how attackers abuse Azure AD and Microsoft Office applications.

External Reconnaissance

For the external reconnaissance parts, we want to start looking at what domains exist on tenants, Azure AD hosted, and federated. The module that we can use is:

```
Invoke-AADIntReconAsOutsider -Domain hiboxy.com | Format-Table
```

This module command performs a check with Azure for the domain that we pass; in the example, hiboxy.com. Once we know that the domain exists, we can then perform username enumeration using a different module.

```
Get-Content .\Users.txt | Invoke-AADIntUserEnumerationAsOutsider
-UserName
```

In this example, we read the contents of a file, users.txt. This file will then have its contents be passed into Invoke-AADIntUserEnumerationAsOutsider, passing the -UserName argument with the contents of users. There are three modes of operation that this module uses.

- Normal: This appears to pass information to one of the APIs discovered as a sign-in endpoint
- Login: This appears to be a different login API endpoint, and this one is tracked and shows up as a log in event
- Autologin: A different API that appears not to log the sign-in event failed.

The idea is that you can try different APIs to avoid triggering a username harvesting alert based on the API selections that you pick.

Username Enumeration Endpoints

Azure Recon

Some Azure endpoints provide unauthenticated access to information on users

Sending a request to them tells you if the user is **valid** or **not valid**

1. ***GetCredentialType*** endpoint for username enumeration
   - https://login.microsoftonline.com/common/GetCredentialType
2. ***OAuth Token*** endpoint for user enum & password guessing
   - https://login.microsoftonline.com/organizations/oauth2/v2.0/token

Take Caution with requests speed to prevent Azure throttling and Smart Lockout!

Microsoft Azure offers penetration testers useful application endpoints that facilitate key information on users. Both endpoints don't require authentication and provide excellent paths to finding whether a user is valid or not, with pros and cons to each approach. Generating a request is as easy as running a curl command line, with the parsed response telling us if the user is valid.

The two endpoints are:

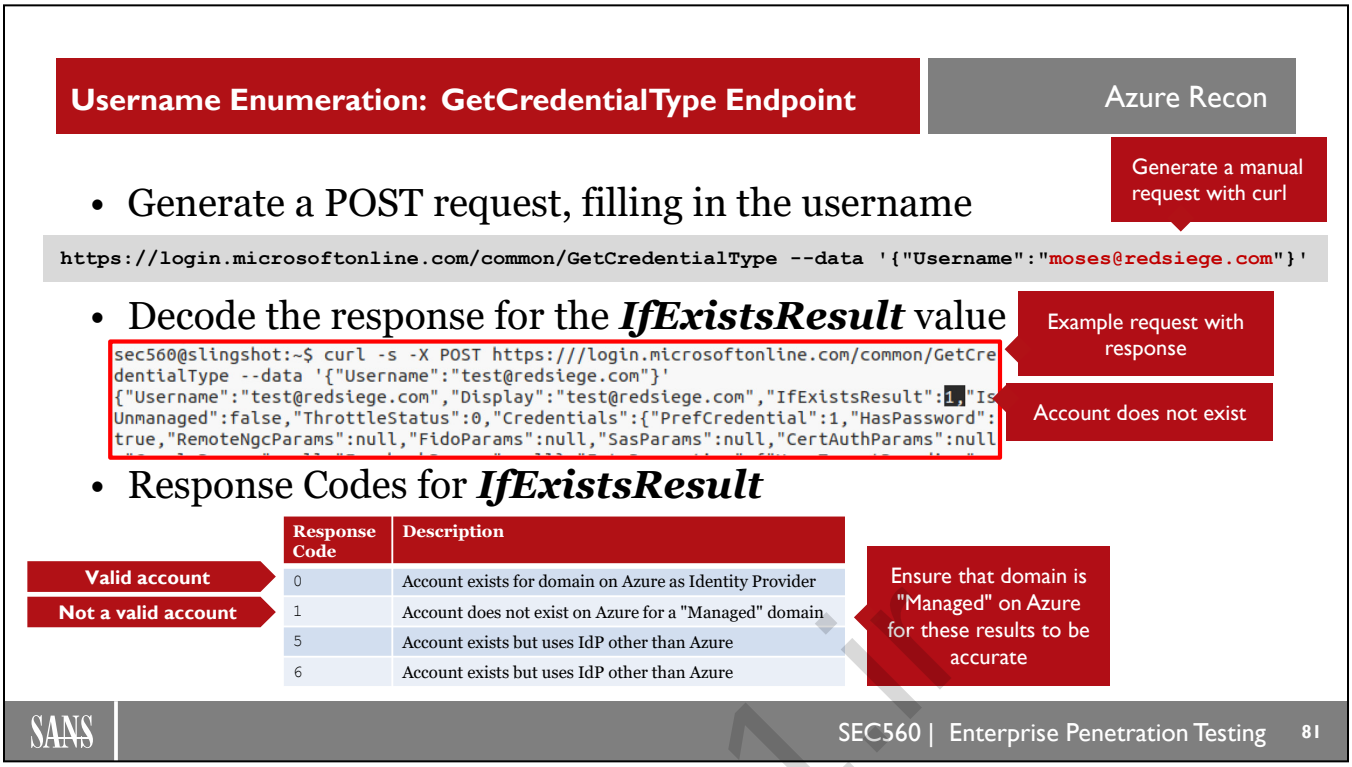
1. GetCredentialType: This provides username enumeration, telling us if the user is valid or not. Very easy to use with command line curl. Disadvantage is that excessive rate of traffic can be throttled by the Azure platform.

2. OAuth Token: This endpoint provides username enumeration as well. An advantage is that it also provides password guessing. A disadvantage is that it can also be detected by Azure with the Azure Smart Lock feature.

The two methods above are recommended for their ease of use, effectiveness, and automation capability; however, other methods exist for username enumeration. These other methods can involve more manual steps or Base64 encoding credentials. These approaches include using the Azure Portal web application (https://portal.azure.com), Outlook AutoDiscover, Microsoft-Server-ActiveSync, and https://login.microsoftonline.com/rst2.srf (which is used by OneDrive). [1]

We must be careful and aware of throttling, detection, and rate limiting by the Azure platform. Azure can detect an excessive rate of queries and throttle the response. This can lead to false positives in our penetration testing – we always want to avoid this!

[1] Red Siege Information Security: https://www.redsiege.com/bouncing-off-clouds



Using the GetCredentialType endpoint for correct username enumeration on Azure involves three steps.

Step 1: Ensure that the domain is "Managed" on Azure.

We must first ensure that the domain for this tenant is managed on Azure as the Identity Provider. This will ensure that the results returned by GetCredentialType are correct. To do this, query the getuserrealm.srf endpoint and look for the "NameSpaceType" returning a value of "Managed":

```
curl -s https://login.microsoftonline.com/getuserrealm.srf\?log
in\=<DOMAIN>\&\json\=1 | jq .NameSpaceType
```

An example of querying the domain "redsiege.com":

```
curl -s https://login.microsoftonline.com/getuserrealm.srf\?login\=red
siege.com\&\json\=1 | jq .NameSpaceType
```

Other possible values include "Unknown" and "Federated". To ensure proper results of querying the GetCredentialType endpoint we look to verify the response of "Managed" to indicate a domain that is managed by Azure as the Identity Provider (IdP).

Step 2: Generate a POST request to the endpoint.

Generate the request and rotate in the username desired for the query and query the "IfExistsResult" key's value. We can use Curl to generate this request. For example, in this query below we are testing for 'moses@redsiege.com' and asking to return the value of "IfExistsResult".

A user that does not exist:

```
curl -s -X POST https://login.microsoftonline.com/common/GetCredenti
alType --data '{"Username":"moses@redsiege.com"}' | jq .IfExistsResult
```

A user that exists:

```
curl -s -X POST https:///login.microsoftonline.com/common/GetCredenti
alType --data '{"Username":"tim@redsiege.com"}' | jq .IfExistsResult
```

Step 3: Evaluate the response

A response of 0 indicates that the user exists on Azure as a managed domain. This means that Azure is the Identity Provider (IdP) for the domain. A response of 1 means that the user does not exist.

Take caution to ensure that the domain is verified as "Managed" in the first step. In some cases, a result of 0 can be returned for a domain that is not managed by Azure as the IdP, leading to false positives.

Detecting Throttling: GetCredentialType Endpoint

Azure Recon

- Azure can detect a high rate and ***throttle*** the response to ***GetCredentialType*** endpoint, leading to false positives
- To detect when Throttling is taking place, decode the ***ThrottleStatus*** response and ensure that **2** is not returned
- A value of 0 or 1 indicates Azure is not Throttling the response

```
curl -s -X POST https:///login.microsoftonline.com/common/GetCredentialType --data
'{"Username":"tim@redsiege.com"}' | jq '.ThrottleStatus'
```

```
sec560@slingshot:~$ curl -s -X POST https:///login.microsoftonline.com/common/GetCredentialTyp
e --data '{"Username":"tim@redsiege.com"}' | jq '.ThrottleStatus'
0
```

Response is not being throttled

The Azure platform can detect a high rate of traffic sent to this endpoint and throttle the status. This will impact the efficacy of our results, leading to false positives. When building a script to perform username enumeration, we must ensure that throttling is not taking place. Evaluate the "ThrottleStatus" key's value and ensure that a value of 2 is not returned. A value of 0 or 1 is fine and does not indicate throttling or false positives.

Here is an example of using Curl and Jq to parse the response of "ThrottleStatus":

```
curl -s -X POST https:///login.microsoftonline.com/common/GetCredenti
alType --data '{"Username":"tim@redsiege.com"}' | jq '.ThrottleStatus'
```

Bringing all of this together, we can use curl and string interpolation to write out a single line showing the username queried, the response to IfExistsResult, and the response to ThrottleStatus. This can be used in a script to loop through a list of usernames and parse the response for a valid username and whether the response is being throttled by the Azure platform.

```
curl -s -X POST https:///login.microsoftonline.com/common/GetCredenti
alType --data '{"Username":"tim@redsiege.com"}' | jq '. | "Username:
\(.Username) IfExists Result: \(.IfExistsResult) Throttle Result:
\(.ThrottleStatus)"'
```

Username Enumeration: OAuth Token Endpoint (1)

Azure Recon

- A POST request to the ***OAuth Token*** endpoint
  - More input fields than the other method, including a password
  - Tells us valid users on the Azure tenant through an error code response
  - Additionally, tells us if MFA is enabled on the account (if correct password)
  - Several automated tools exist for this method
- Example of manual request using curl

```
curl -i -s -k -X $'POST' \
-H $'Content-Type: application/x-www-form-urlencoded' --data \
$'client_id=CLIENT_ID&grant_type=password&client_info=1&username=moses@redsiege.com&password=PASSWORD&scope=SC
OPE' \
$'https://login.microsoftonline.com/organizations/oauth2/v2.0/token'
```

Rotate Password

Rotate Username

Azure also provides an OAuth Token flow endpoint that is easy to use and provides a penetration tester username enumeration of Azure platform users. The OAuth Token flow endpoint has these properties:

- More input fields than the prior method, including a password field for password guessing
- An error response code that can be easily decoded
- Additionally tells us if MFA is enabled (if correct password)
- This endpoint is protected by "Azure Smart Lockout"
- Several automated tools support querying this endpoint; however, we can easily use curl and automation with our own script

We can use curl to easily query this endpoint, rotating the username highlighted in the request below:

```
curl -i -s -k -X $'POST' \

-H $'Content-Type: application/x-www-form-urlencoded' \

--data
$'client_id=CLIENT_ID&grant_type=password&client_info=1&username=moses
@redsiege.com&password=PASSWORD&scope=SCOPE'  \

$'https://login.microsoftonline.com/organizations/oauth2/v2.0/token'
```

Take caution when sending a high rate of requests to this endpoint because Azure has a feature called "Azure Smart Lockout". Azure Smart Lockout is enabled by default for all Azure tenants and all Azure users. We will discuss more on this and techniques for bypassing it in the password spraying section.

Username Enumeration: OAuth Token Endpoint (2)

Azure Recon

- Decode the error code response for enumeration of valid tenant users
- A valid username (with wrong password)
  - Response: 50126

Error code 50126 means valid user

```
"error":"invalid_grant",
"error_description":
"AADSTS50126: Error validating credentials due to invalid username
or password.\r\nTrace ID: ce378563-7b85-4154-860c-
bee8315bac02\r\nCorrelation ID: ce0115b3-035c-4676-ae06-
edb9d4e2627a\r\nTimestamp: 2022-01-06 06:51:07Z",
```

- User doesn't exist on the tenant
  - Response: 50034

Error code 50034 means invalid username

```
"error":"invalid_grant",
"error_description":
"AADSTS50034: The user account {EmailHidden} does not exist in the
rtcfingroup.com directory. To sign into this application, the
account must be added to the directory.\r\nTrace ID: b70fdd8d-7d92-
4186-a1c1-0d4d48433502\r\nCorrelation ID: ce4f3df5-1f1d-42b6-9b92-
```


Decoding the response to the OAuth Token endpoint for username enumeration is straight forward and easy. This endpoint uses error codes named AADSTS and they are documented at this Azure Active Directory documentation link [1].

Valid Username:

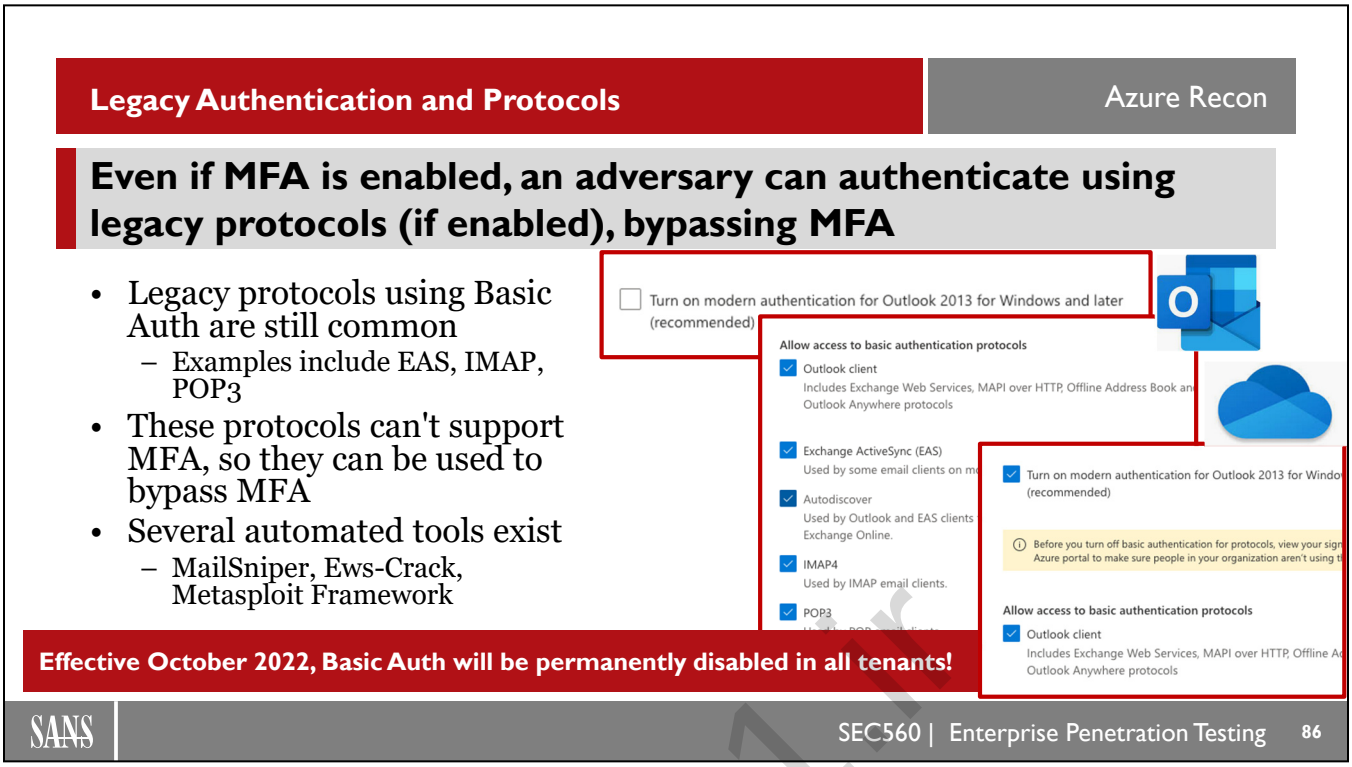
An error code of 50126 signifies a valid username has been queried but the password is incorrect. Since the focus of this section is on username enumeration, we can ignore the response for a valid password or that MFA is enabled.

Username Does Not Exist:

An error code of 50034 signifies that the user doesn't exist on the tenant.

[1] Azure AADSTS Error Codes

https://docs.microsoft.com/en-us/azure/active-directory/develop/reference-aadsts-error-codes#aadsts-error-codes



For many years, Microsoft Exchange Online has supported legacy authentication. Legacy authentication refers to clients using Basic Authentication in their authentication requests, such as Office 2010 client, or any client using older mail protocols such as IMAP or POP3. These older clients don't support Modern Authentication. Microsoft is pushing their customers to use Modern Authentication or OAuth 2.0 framework. These legacy authentication protocols include Exchange ActiveSync (EAS), Autodiscover, MAP4, POP3, Authenticated SMTP, Exchange Online PowerShell.

Azure is Moving Customers from Legacy to Modern Authentication

These legacy protocols do not support MFA. Microsoft's position statement is to encourage customers to "Block Legacy Authentication." Microsoft Azure is taking steps to better protect M365 and Azure users by enforcing secure defaults for new tenants by enabling MFA for all users and encouraging customers to use Modern Authentication (OAuth 2.0). On their own, customers are gaining awareness and enabling MFA. Since the legacy protocols do not support MFA, as a penetration tester we can authenticate as these users and bypass MFA! As of February 2022, The current posture of tenants is that Basic Authentication is enabled for these legacy protocols and can still be used by penetration testers. However, Microsoft has made new features available allowing customers to enable modern authentication across their tenant and disable all or specific basic authentication protocols.

As a penetration tester what we need to know is that multiple tools can be still be used to carry out testing for legacy authentication and on-premise Exchange, including newer tools that auto-detect MFA and automatically try legacy authentication protocols. Some Azure clients will still use legacy authentication protocols, Outlook Web Access, or Microsoft Exchange on-premise that is exposed over the public Internet. They include:

1. MailSniper: https://github.com/dafthack/MailSniper
2. Ews-crack: https://github.com/mikesiegel/ews-crack
3. Metasploit Framework Auxiliary module: auxiliary/scanner/http/owa_login
4. Metasploit Framework Auxiliary module: auxiliary/scanner/http/owa_ews_login
5. TrevorSpray (discussed in next section)

Modern Authentication: A Modern Approach

Microsoft has announced [1] that effective October 1, 2022, they will permanently disable Basic Auth in all tenants regardless of how Basic Auth is being used, apart from SMTP Auth. This means that this technique is still effective until October of 2022. The focus of this next section is on automation for password spraying against Azure users using modern authentication frameworks such as OAuth 2.0 endpoints.

[1] "Basic Authentication and Exchange Online – September 2021 Update"

https://techcommunity.microsoft.com/t5/exchange-team-blog/basic-authentication-and-exchange-online-september-2021-update/ba-p/2772210

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

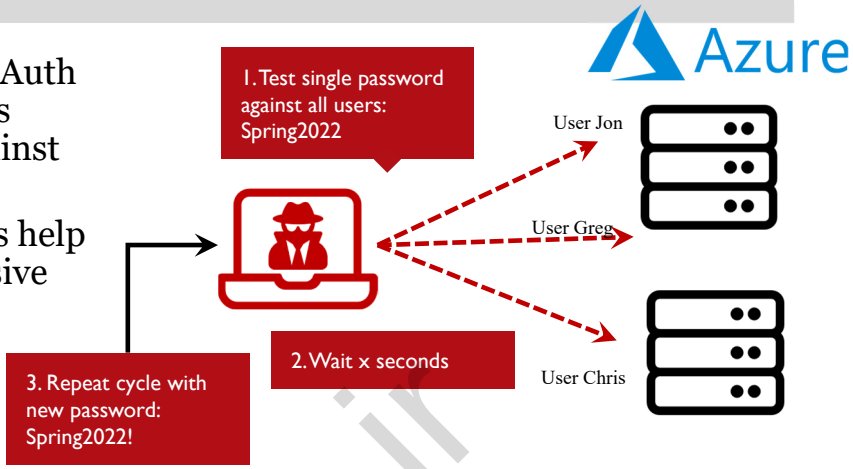
560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Password Spraying in Azure can be effective

- The Microsoft Azure OAuth token endpoint enables password spraying against Azure users
- Newer automated tools help to bypass Azure defensive countermeasures
  - TrevorSpray
  - Spray365



Password spraying can be an effective technique for password guessing that involves sending requests for a list of all known usernames (or guessing from a list of potential usernames) with a single password attempt. We must be careful of account lockout and ensure that enough time waits between each spray. This technique will help prevent account lockout since only one password is attempted for all users rather than sending many password attempts against a single user. This method is more effective than brute force or word list guessing and can yield excellent results. After guessing a single password, the penetration tester can start reconnaissance with an initial foothold, authenticated user account.

Password Spraying is effective in Microsoft Azure using the OAuth Token flow endpoint of https://login.microsoftonline.com/organizations/oauth2/v2.0/token. As a penetration tester, we can effectively send multiple username requests against this application endpoint, targeting a tenant with just a single password attempt. This can yield successful results. There are some caveats to this approach. Azure is evolving their defensive technology to make this technique less effective, so we need newer tools and approaches to bypass defensive technology. This endpoint is protected by "Azure Smart Lockout" by default for any Azure user [1]. Rotating IP addresses in requests can be effective technique to bypass Azure Smart Lockout. Gone are the days of classic password spraying where it was very easy to be effective against internal services such as an on-premise Domain Controller. A low and slow approach can work; however, it might make the password spraying infeasible given the time constraints of a client penetration test.

Automated Tools:

One of the tools to arrive at the scene and gain popularity for Azure password spraying was Beau Bullock's MSOLSpray. Since Azure is evolving their defensive countermeasures against this technique, new and modular approaches are required. Two excellent newer tools afford us advantages for advancing modular features and multi-platform Python support. More importantly, they both include support for detecting and handling Azure Smart Lockout.

1. TrevorSpray: https://github.com/blacklanternsecurity/TREVORspray

2. Spray365: https://github.com/MarkoH17/Spray365

Password Spraying Tool: TrevorSpray

Passwords Attacks

TrevorSpray: Python modular spraying tool for detection of smart lockout, MFA bypass, custom timers, excellent logging, and proxy SSH support (for rotating IP addresses)

– Website: Github.com/blacklanternsecurity/TREVORspray

`trevosrspray --recon redsiege.com`	Perform recon on domain for Azure tenant information
`trevosrspray -u emails.txt -p 'Spring2020'`	Perform spray against users in file with single password
`trevosrspray -u emails.txt -p 'Spring2020' -delay 60 -jitter 20 -lockout-delay 60`	Jitter: Random deviation of delay between requests up to 20 seconds

TrevorSpray [1] is a python-based, modular Azure password spraying tool that was designed with the intent of getting around the defensive countermeasure of Azure Smart Lockout and MFA. From the blog post from the author [2] when describing the increasing difficulty in effectively carrying out password spraying against Azure.

As pen testers, we've been forced to dial back the intensity of our password sprays so that they take hours or days to finish. And even when we find a valid credential, it sometimes doesn't lead anywhere thanks to security policies like MFA. But since we're hackers and it's our job to hack stuff, it's hard to sit idly by and let our favorite pastime of password spraying go the way of the dodo.

What I'm trying to say is that we're frustrated. And when hackers are frustrated, they write code. So, it is with great delight that we are open-sourcing some new tools which are the product of our frustration and will hopefully help to make password spraying fun again."

Tool Capabilities:

The nice thing about TrevorProxy is its dead simple to use, stable, and capable of many features and modules that help to bypass Azure Smart Lockout and MFA.

* Round-robin proxying of requests through SSH proxies to help avoid Azure Smart Lockout
* A separate proxy tool (TrevorProxy) to proxy traffic through Burp Suite
* Multi-threaded
* Multiple modules support (msol, adfs, okta, anyconnect) and custom modules
* Spoof User-Agent string
* Options for delay, jitter, and lockout-delay between requests (to help defeat Azure Account Lockout)
* Automatic MFA bypass support when it detects MFA support using 8 methods (IMAP, POP, Exchange Web Services, Exchange ActiveSync, and several more)
* Comprehensive logging and IPv6 support
* Enumeration feature for doing recon on tenant domain

Installation on SEC560 Slingshot Linux:

```
$ pip install --no-cache-dir
git+https://github.com/blacklanternsecurity/trevorproxy
$ pip install --no-cache-dir
git+https://github.com/blacklanternsecurity/trevorspray@trevorspray-
v2
```

Basic Usage and Examples:

```
$ trevorspray --recon redsiege.com
```

Perform recon against the domain, finding the token endpoint and DNS entries

```
$ trevorspray -u emails.txt -p 'Winter2022'
```

Perform password spraying against all users in emails.txt with password of Winter2022

```
$ trevorspray -u emails.txt -p 'Winter2022'  --delay 60 --jitter 20 --
lockout-delay 60
```

Perform password spray with delay of 60 seconds between request, with a jitter of 20 seconds, and a lockout delay of 60 seconds

[1] TrevorSpray Github: https://github.com/blacklanternsecurity/TREVORspray

[2] TrevorSpray Blog: https://github.com/blacklanternsecurity/TREVORspray/blob/trevorspray-v2/blogpost.md

-

Password Spraying Tool: Spray365

Spray365: Python password spraying tool for detection of smart lockout; supports timers, randomizing and spoofing user agent, execution plans, HTTPS proxying, and conditional access detect

- Website: Github.com/MarkoH17/Spray365

`python3 spray365.py generate --ep plan.365 -d redsiege.com -u users.txt -p Spring2022`	Generate an execution plan named *plan.365* for domain *redsiege.com*, testing single password of *Winter2022* for username list in *users.txt*
`python3 spray365.py generate --ep plan2.365 -d redsiege.com -u users.txt -p Spring2022 -rUA -delay 15`	Generate an execution plan named *plan2.365*; randomize user agent string and delay 15 seconds between each request
`python3 spray365.py spray --ep plan.365`	Execute the spray execution plan, *plan.365*

Spray365 [1] is a python-based M365 password spraying tool developed by Mark Hedrick. It utilizes a novel concept of first requiring creation of an "execution plan." After creation of an execution plan, the user runs the plan for execution of the spray. Spray365 has support for spoofing user agent strings, randomized and shuffling the order of spraying attempts, and native support for proxying HTTPS/HTTP traffic through Burp Suite. It supports execution plans in JSON and logging output in JSON, making the tool flexible and extensible with other tooling. Spray365 was also created for the purpose of novel approaches to identifying users with support for Azure AD conditional access policies. From the author [2]:

The current state of password spraying Office 365 accounts could benefit from new approaches to bypassing Azure AD conditional access policies and other techniques that make it difficult to detect password spraying techniques.

Tool Capabilities:

- Pre-generate an execution plan in JSON for later running; Includes built-in features to help bypass Azure Smart Lockout technology
- Logging results stored in JSON
- Randomizing user agents
- Spoofing a custom user agent
- Custom delay timers
- Shuffling authentication and order attempts
- Proxy requests through an HTTPS proxy such as Burp Suite

Basic Usage and Examples:

```
$ python3 spray365.py generate --execution_plan
<path_for_saved_execution_plan> -d <domain_name> -u
<file_containing_usernames> -p <single_attempted_password>
```

Creates execution plan

```
$ python3 spray365.py generate --execution_plan
<path_for_saved_execution_plan> -d <domain_name> -u
<file_containing_usernames> -p <single_attempted_password> --rUA --
delay 15
```

Creates an execution plan with randomizing the user agent string and a delay of 15 seconds between request

```
$ python3 spray365.py spray --execution_plan <path_to_execution_plan>
```

Runs the execution plan

[1] Spray365 Github: https://github.com/MarkoH17/Spray365

[2] Article on Spray365 ("Spray 365: A New Twist on Office 365 Password Spraying"): https://depthsecurity.com/blog/spray-365-a-new-twist-on-office-365-password-spraying

Azure Smart Lockout

Passwords Attacks

Azure Smart Lockout detects password spraying tools & invalidates their results; Enabled for all Azure tenants and Users

- Azure tracks' unfamiliar locations across data centers (based on IP address)
- When 10 logins with failed attempts are detected, returns an account locked error response
- Returns error response whether password is valid or invalid
- The user account in Azure ***is not actually locked out*** by spraying tool

AADSTS50053: *IdsLocked* - The account is locked because the user tried to sign in too many times with an incorrect user ID or password. The user is blocked due to repeated sign-in attempts.

TrevorSpray detects

```
[ERRR] twolf@rtcfingroup.com:Winter2022 - AADSTS50053: Account appears to be locked.
[WARN] swhite@rtcfingroup.com:Winter2022 - AADSTS50126: Invalid email or password. Account could ex
[WARN] shall@rtcfingroup.com:Winter2022 - AADSTS50126: Invalid email or password. Account could ex
[WARN] lhill@rtcfingroup.com:Winter2022 - AADSTS50126: Invalid email or password. Account could ex
[WARN] syoung@rtcfingroup.com:Winter2022 - AADSTS50126: Invalid email or password. Account could e
[WARN] jgolden@rtcfingroup.com:Winter2022 - AADSTS50126: Invalid email or password. Account could
                                  group.com:Winter2022 - AADSTS50126: Invalid email or password. Account could
                                  up.com:Winter2022 - AADSTS50126: Invalid email or password. Account could e
                                  group.com:Winter2022 - AADSTS50126: Invalid email or password. Account could
                                  group.com:Winter2022 - AADSTS50126: Invalid email or password. Account could
                                  group.com:Winter2022 - AADSTS50053: Account appears to be locked.
[ERRR] bfigueroa@rtcfingroup.com:Winter2022 - AADSTS50053: Account appears to be locked.
[ERRR] lweaver@rtcfingroup.com:Winter2022 - AADSTS50053: Account appears to be locked.
[ERRR] rchan@rtcfingroup.com:Winter2022 - AADSTS50053: Account appears to be locked.
[ERRR] asanchez@rtcfingroup.com:Winter2022 - AADSTS50053: Account appears to be locked.
[ERRR] sking@rtcfingroup.com:Winter2022 - AADSTS50053: Account appears to be locked.
[ERRR] aortiz@rtcfingroup.com:Winter2022 - AADSTS50053: Account appears to be locked.
[ERRR] Multiple Account Lockouts Detected!
[ERRR] 10 of the accounts you sprayed appear to be locked out. Do you want to continue this spray?
[USER] [Y/N] (default is N):
```

Both TrevorSpray & Spray365 detect Smart Lockout & support rotation of IP addresses!

Azure Smart Lockout is a defensive security control technology feature that is enabled by default for all Azure tenant and users. Since 2018, it has helped protect Azure users against automated brute force or guessing of user passwords. It is a technology feature that every Penetration Tester should be aware of when assessing the Azure platform. From the Microsoft documentation [1],

Smart lockout helps lock out bad actors that try to guess your users' passwords or use brute-force methods to get in. Smart lockout can recognize sign-ins that come from valid users and treat them differently than ones of attackers and other unknown sources. Attackers get locked out, while your users continue to access their accounts and be productive.

How it works:

Each Azure datacenter tracks login activity for the geographic region independent from the other regions. It uses familiar vs. unfamiliar locations to distinguish a malicious actor from a genuine user. The familiar vs unfamiliar locations have separate lockout counters. By default, Azure lockout will lock the account after 10 failed attempts for Azure public and 3 for Azure Government tenants. The account locks again after each failed login attempt, for one minute and then longer for subsequent attempts. Azure doesn't disclose the exact thresholds or rates for the lockout period growing.

The automated penetration testing tools should be built to automatically detect when Azure is returning a response indicating Smart Lockout and immediately stop sending requests. Azure AD will return a response of AADSTS 50053 (IdsLocked error code) regardless of the password validity:

> IdsLocked - The account is locked because the user tried to sign in too many times with an incorrect user ID or password. The user is blocked due to repeated sign-in attempts.

This might alarm the penetration tester, believing that the account is locked out. However, the actual user account is not locked in Azure. It simply indicates that malicious, anomalous activity is detected. Once this condition is reached, it is not possible for the automated tool to be effective in eliciting a valid password since the AADSTS 50053 response is returned regardless of a correct or incorrect password.

Other important points for Azure customizations used by clients. The Penetration Tester might need to know this:

Customization of Smart lockout features is possible but requires an Azure AD premium license P1 or greater.

Smart lockout can be integrated with Hybrid deployments that use password hash synchronization or pass-through-authentication in order to help protect on-premise AD DS.

- Federated deployments using ADFS can be protected with AD FS Extranet Lockout and Extranet Smart Lockout.

Why does this matter?

As an Azure penetration tester, this feature matters to us because it is vital that we avoid detection of Smart Lockout. It reduces efficacy of password enumeration. New tools, techniques, and methods are required to ensure accuracy for password spraying.

[1] "Protect user accounts from attacks with Azure Active Directory smart lockout"

https://docs.microsoft.com/en-us/azure/active-directory/authentication/howto-password-smart-lockout

There are several ways to avoid hitting lockout thresholds in Azure AD

- Use a load balancing approach, some tools support multiple egress connections
- Use BurpSuite IP Rotate, which leverages AWS API Gateway
- Use different APIs within the Microsoft Portals to attempt to circumvent detection
- Recall: Legacy Protocols sometimes provide an avenue to bypass more strict security items
  - Basic Authentication protocol has no notion of Lockout or Log Out

Several techniques can be used to avoid lockout via Smart Lockout. The primary approach will avoid using the same IP address or other identifying information in the Azure AD environment. To do this, we have several great options that we can deploy. The first one is to use a load balancing system to your advantage. One example is a tool like FireProx or using a standard Load Balancing system. The second option is to use a plugin like BurpSuite IP Rotate. The plugin will leverage API Gateway from Amazon to send traffic using the standard AWS API Gateway. The Gateway will naturally round-robin the connection across a very fast large number of IP addresses associated with the AWS API Gateway. The third option is to avoid using the same Azure AD APIs to prevent the lockout via a single API. The fourth and final option is that testers can also use legacy protocols and recall that specific systems still use protocols like Basic Authentication, a fragile system.

Bypassing Technique: Rotating IP Addresses (1)

- **TrevorSpray** supports load balancing requests through multiple SSH proxies with **`--ssh`**, bypassing Smart Lockout
- Excellent approach for using native Linux OS and SSH for rotating IP addresses
- Example to Proxy through 8 SSH servers:

This rotates requests between 8 SSH proxies, with a delay of 45 second per-proxy, and jitter setting of 23 seconds

```
$ trevorspray.py -e user_emails.txt -p Winter2022  --delay 45 --jitter 23
--ssh root@proxy1 root@proxy2 root@proxy3 root@proxy4 root@proxy5
root@proxy6 root@proxy7 root@proxy8 --key /home/sec560/.ssh/id_rsa
```

- **Tool benchmark:** During Azure penetration test, successful in spraying an Azure tenant containing 1,013 users without smart lockout detection in 6 hours. Timers can be more aggressive!

TrevorProxy not only has automated support for detecting Smart Lockout, but also has excellent support for using SSH proxies to round-robin or load balance across multiple spray requests. This serves a vital function of avoiding Azure smart lockout counters and timers across disparate, independent geographic regions. The more SSH proxies that are used, the more effective TrevorSpray is in avoidance of triggering smart lockout. This can be an important use case for environments that need to rely on native Linux OS and SSH capabilities to support automated security tooling where HTTP(S) proxies such as Burp Suite are not available or otherwise viable. Some organizations chose to use SSH proxies and SOCKS support to carry out security simulations. For these organizations, TrevorProxy is a great fit for bypassing Smart Lockout.

Advanced Usage Example with SSH Proxies against a Tenant with over 1,000 Azure AD Users:

As of January 2022, running a penetration test against an Azure tenant with over 1,000 users and TrevorSpray can be successful in bypassing Smart Lockout. This penetration test was run with the following settings and was successful in detecting valid user passwords:

- 8 SSH proxies load balanced to rotate IP addresses
- A delay of 45 seconds per-proxy between request (--delay 45). This is the delay between each SSH proxy sending a password spray request (not the delay between overall requests)
- A jitter setting of 23 seconds (--jitter 23)

Example command:

python3 trevorspray.py -n -e user_emails.txt -p Winter2022 --delay 45 --jitter 23 --ssh root@proxy1 root@proxy2 root@proxy3 root@proxy4 root@proxy5 root@proxy6 root@proxy7 root@proxy8 --key /home/sec560/.ssh/id_rsa

Result Notes: This spray took just under six hours to complete against 1,013 Azure AD users. This is certainly a conservative setting above. A more aggressive delay and jitter setting can speed up this with a smaller data set; however, it helps to get a benchmark for smart lockout detection behavior against a tenant of over 1,000 users. In some cases of more aggressive settings, Azure will start to detect malicious password spraying after 300 requests, invalidating the entire penetration test for that scan attempt.

Bypassing Technique: Rotating IP Addresses (2)

Passwords Attacks

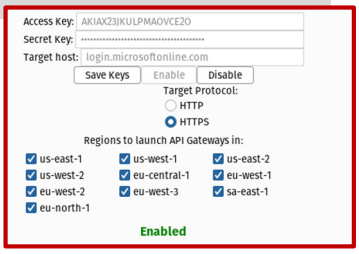
Using Amazon API Gateway to bypass Smart Lockout can be another effective technique

- Spray365 tool supports rotating IP addresses by proxying HTTP traffic
  - Amazon API Gateway
  - Burp Suite "IP Rotate" extension
  - Spray365
- **Usage Example:**

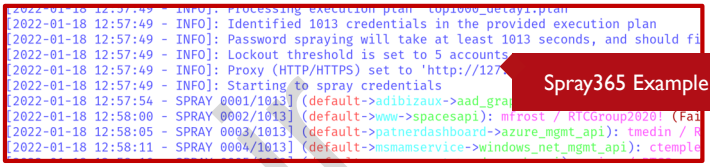
```
python3 spray365.py spray --execution_plan plan.365
--proxy http://127.0.0.1:8080 -k
```

Proxy request through Burp Suite

Burp Suite "IP Rotate" extension for proxying traffic through Amazon API Gateway



- **Tool benchmark:**
  - In an Azure penetration test against 1,013 users, used Spray365 + IP Rotate to successfully spray all users without detection in 16 minutes.
- Be careful to follow AWS Terms of Service for API Gateway



Spray365 Example

Using Amazon API Gateway to bypass Smart Lockout can be another effective technique. It works well for penetration testers that wish to use HTTP(s) proxies like Burp Suite to gain visibility into traffic by monitoring and analyzing the traffic. It can scale up with unique IP addresses per request offered by Amazon API Gateway across multiple region, making it very difficult for Azure to detect. There are cost savings to using Amazon API Gateway.

Bypassing Smart Lockout using Spray365:

One of the most effective techniques to bypass Smart Lockout is to use the infinite scalability and network addressing capabilities of a Cloud Service Provider (CSP)'s infrastructure. In summary: We use Amazon API Gateway to rotate IP addresses (https://rhinosecuritylabs.com/aws/bypassing-ip-based-blocking-aws/). Each request is proxied through Amazon API Gateway and regions, making it very difficult for Smart Lockout detections. How is this so? It is more difficult for Azure to detect malicious requests when nearly every source IP address is different. This technique has been available for several years with a maturity offered by several other tools such as Fireprox (https://github.com/ustayready/fireprox).

365Spray has a nice capability to proxy requests through an HTTP(S) proxy. A Burp Suite extension called "IP Rotate" is available in the Burp Extender store that enables rotating IP addresses through our Amazon API Gateway. The technique of choice is to use Spray365 together with Burp Suite + the IP Rotate Burp Suite extension + AWS API Gateway. This allows us to monitor and view all requests through Burp Suite, for later analysis. The steps are detailed here: https://depthsecurity.com/blog/spray-365-a-new-twist-on-office-365-password-spraying

AWS Terms of Service for using API Gateway:

Take caution when using Amazon API Gateway to carry out automated security simulations. Carefully review the Amazon Terms of Service (ToS) to ensure that you are not violating the Amazon terms of service. In their Penetration Testing Guidance (https://aws.amazon.com/security/penetration-testing/), Amazon lists API request flooding as "Prohibited Activities". Using API Gateway for security simulations should be permitted as long as the requests don't exceed 50,000 requests per second. A single password spray per second should stay under the threshold.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Please work on below exercise.
Lab 5.5: Azure Recon and Password Spraying

Please go to Lab 5.5: Azure Recon and Password Spraying in the SEC560 Workbook.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

OpenIDConnect Flows

OIDC Crash Course

- Azure ADs Main Flow type of Authentication is OpenIDConnect
- OpenIDConnect builds on top of Oauth2
  - Complex Protocol for using a Central Authentication Server over Web Flows
  - Not backwards compatible to older protocols
  - Azure AD Kerberos Support in Preview Mode
  - Competitors such as Okta, support Security LDAP
- Microsoft deems this "Modern Authentication" as it can support multiple password schemas such as MFA and FIDO2.

Azure Active Directory has several Oauth2 flow types that it supports to provide access. For the uninitiated, the OpenIDConnect framework can be very daunting as it is a framework that requires Oauth2 flows to be available. Let's start with OpenIDConnect itself and how the OAuth flows work.

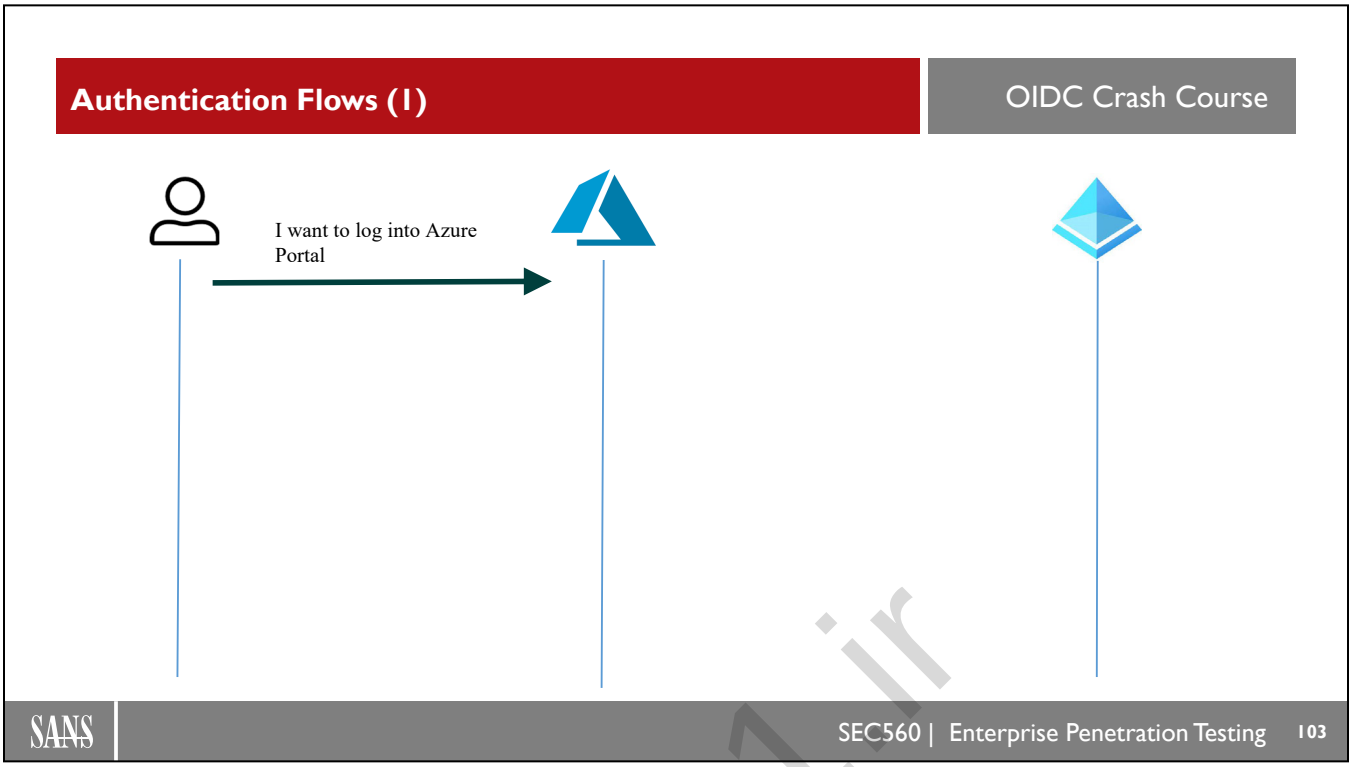
Oauth2 has several "flow" types. The most common flow type you may encounter is a flow type called Code Flow. The code flow type works like this:

A user goes to a website to authenticate, let's say its portal.azure.com

- The user will be redirected to log in to Microsoft Azure AD
- The successful authentication will provide the user with an access token
- That access token is then used to validate that the user has been scoped to access the resources in question

Oauth2 is not a legacy protocol. It was designed to federate and provide authorization and access control to different web services. It is the button that you see when you need to "login with Facebook" or "login with apple." Azure AD supports this protocol as a "First Class Citizen" but also supports SAML, WS-Federation, and in preview Kerberos support.

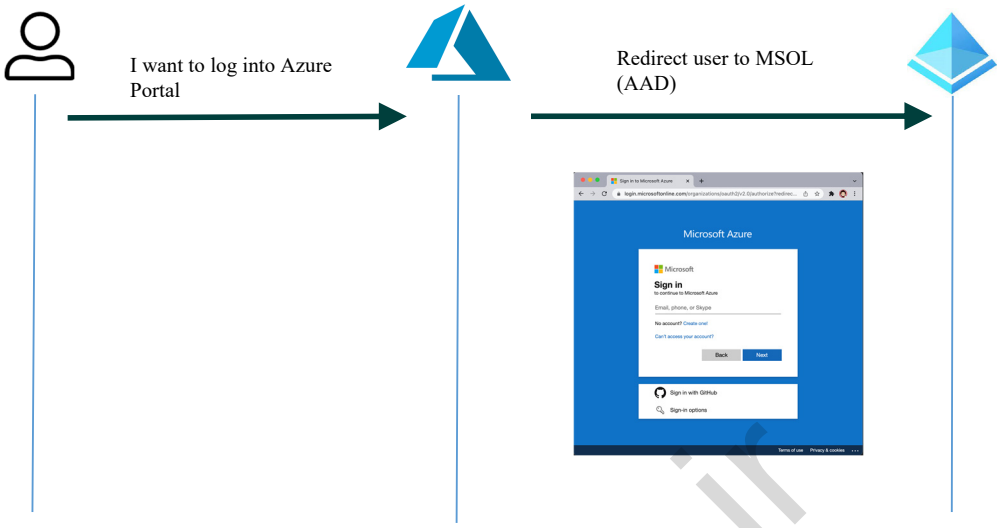
Azure AD is a modern authentication system as it can support multiple authentication types outside of the standard username and password, including Passwordless Authentication methods and Multi-Factor authentications. Modern authentication does not necessarily apply to legacy protocols.



Let's start with a flow in which a user wants to access a resource. In the example, the icon we are using is the Azure Infrastructure Icon, but this could be Office, Sharepoint, or any number of federated services. To simplify this transaction, consider that we will be going to the Azure Portal.

The first connection is from the user to the Azure Portal (The middle of the slide deck). The transaction happens within your browser; keep in mind where the transactions occur. This redirect is within your browser.

Authentication Flows (2)



The browser will be redirected to the Azure AD login page. This page is hosted in the domain: login.microsoftonline.com. The domain we are hitting may also be known as MSOL.

Recall from a previous slide the way the URL looks. It contains information that will be relevant to us:

```
https://login.microsoftonline.com/organizations/oauth2/v2.0/authorize?
redirect_uri=https%3A%2F%2Fportal.azure.com%2Fsignin%2Findex%2F&respon
se_type=code%20id_token&scope=https%3A%2F%2Fmanagement.core.windows.ne
t%2F%2Fuser_impersonation%20openid%20email%20profile&state=OpenIdConne
ct.AuthenticationProperties%3DysnyqREDACTEDhOSq7&response_mode=form_po
st&nonce=6377REDACTEDY1OTRk&client_id=c44b4083-3bb0-49c1-b47d-
974e53cbdf3c&site_id=501430&prompt=select_account&client-request-
id=d28c579b-b8e7-

4949-a820-98f0e00c824e&x-client-SKU=ID_NET472&x-client-ver=6.12.2.0
```

The redirect URI goes back to portal.azure.com. The URI is where the browser will return to once we are done.

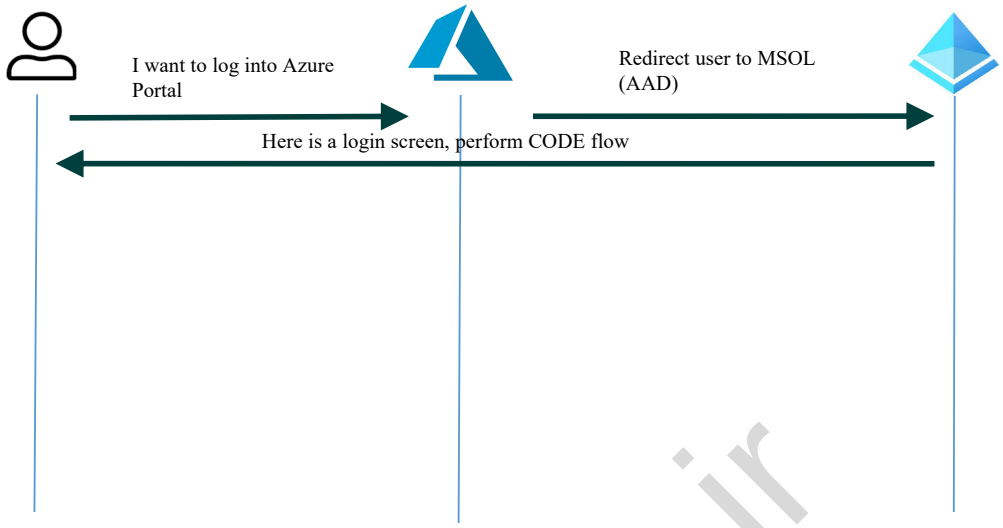
The response code is "code": This indicates that we are in an Oauth2 Code Flow.

The scope we are asking to be authorized includes specific OpenID connect scopes, such as the OpenID email scope.

The other items are specific to azure, such as the tenant and the SKU of the Client.

Authentication Flows (3)

OIDC Crash Course

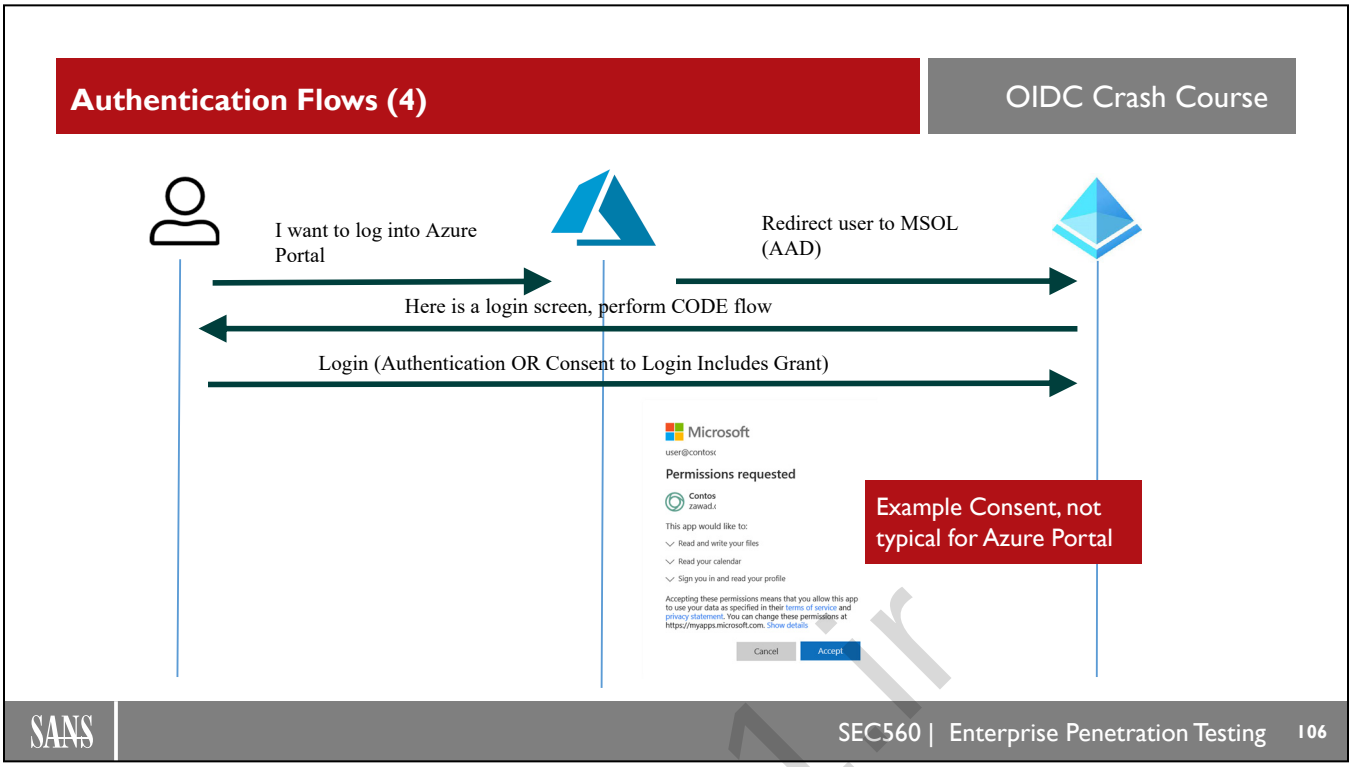


The user, at this point, is asked to log in. At this point is where a few things happen, including:

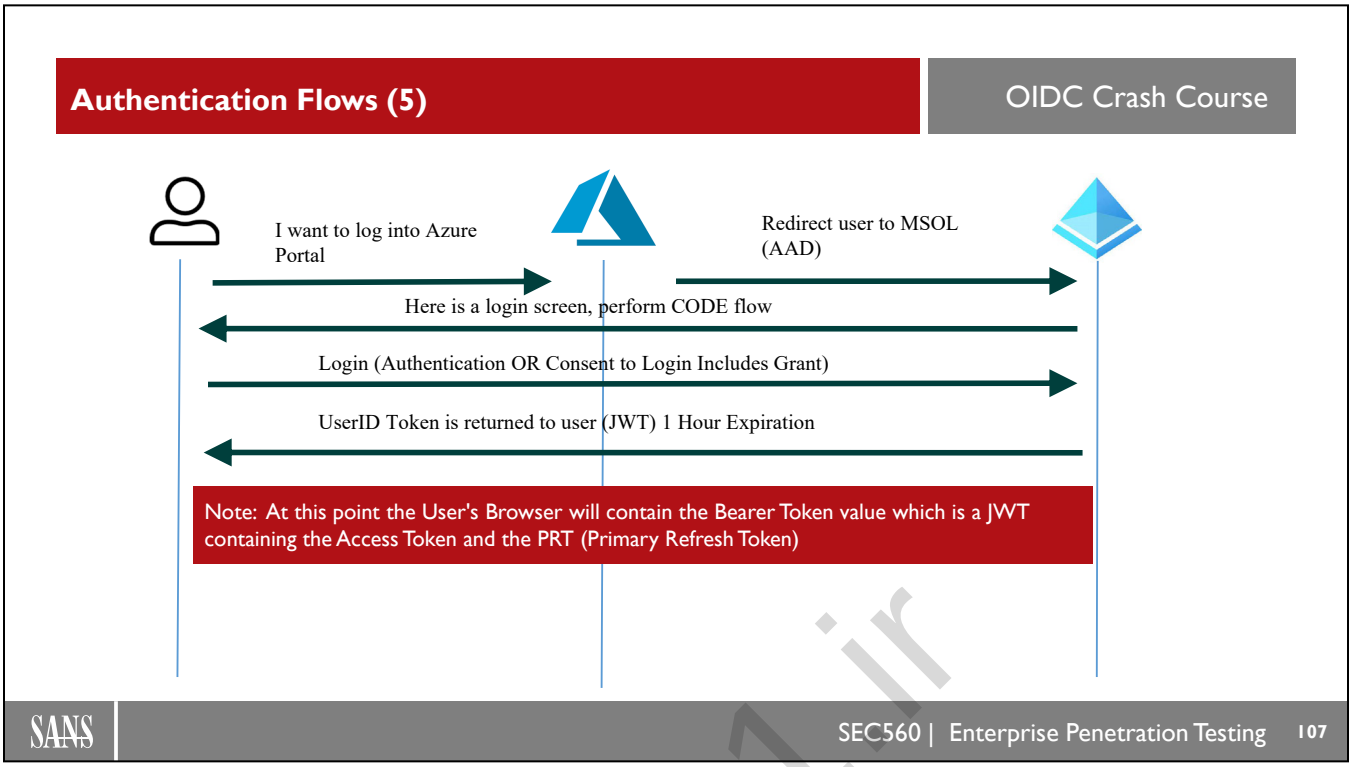
The user will be checked against Conditional Access Policies. Does the browser match? Is the user within the allowed listed IP Space? Does the user need to do MFA in this IP space?

The user will be prompted for MFA (MultiFactor Authentication) if it is enabled. It may also be that the user may have hardware FIDO2 tokens at this point.

Once the user passes this gate, Conditional Access Policies are no longer checked. Keep this in mind.



If the user flow is for a 3rd party application, such as an application working on your behalf, you may see a consent screen. Have you ever used a Salesforce Application? Perhaps you have seen this in an application that integrates with Facebook, Twitter, or Instagram? You may have seen this consent message. This message allows a 3rd party application to act on your behalf. This is the part of the flow in which you would see this message. In the case of Azure Portal, you will not see this. What happens if you write a malicious application that works on behalf of the privileged user?



We are now within an OpenID Connect flow that will return two key types. The first type of key material you will be getting is an access token. The access token contains your authentication material, the JWT (JSON Web Token). If someone were to steal this authentication material, they could log in to the Azure portal as if they were you. An example JWT will look like this:

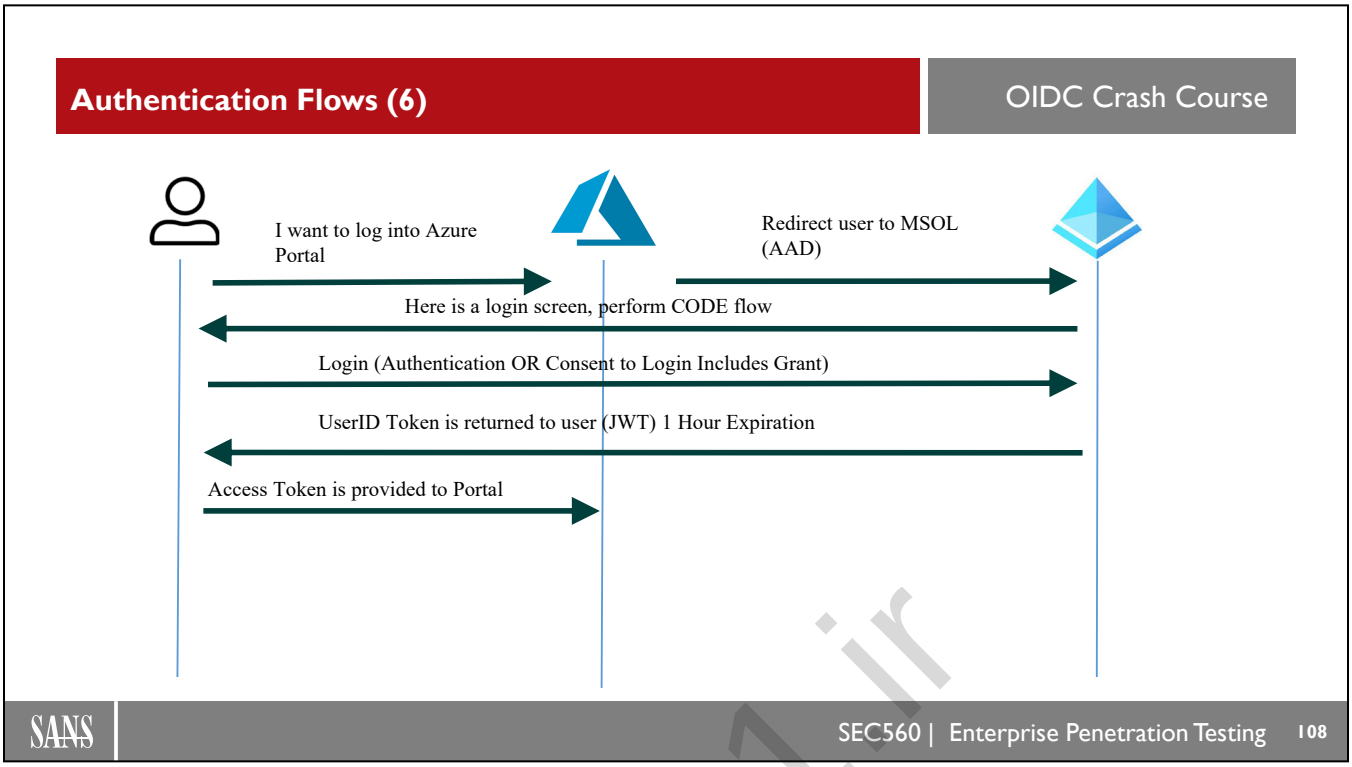
```
"access_token": "eyJ0eXREDACTEDCJ9.eyJhdWREDACTED.XREDACTED-IQ"
```

The value will be long; you need to key on a few things.

The JWT will typically start with "ey"

The JWT will have two (2) dots (.) that will separate each value, the encryption type, the body, and the signature.

There will also be a "refresh token", which we will describe later.

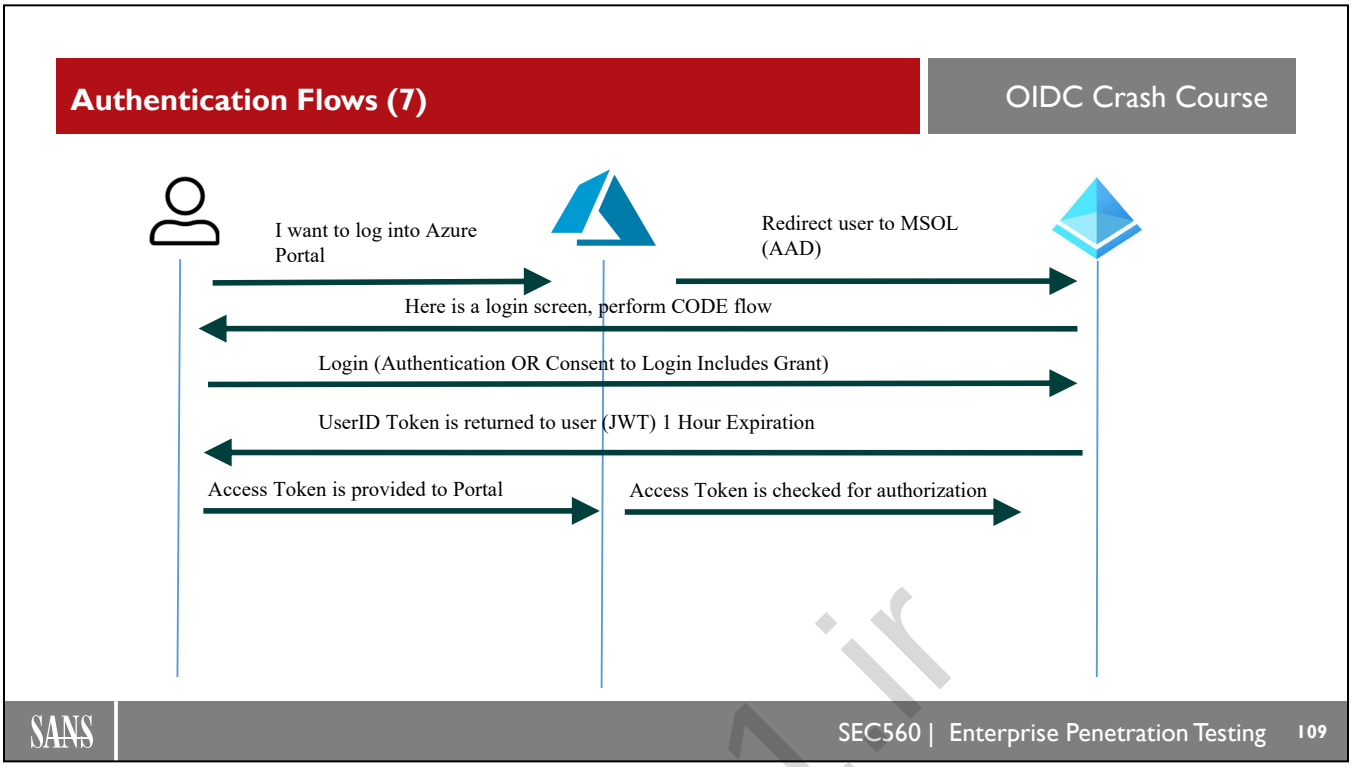


The access token is now provided to Azure via the Client's Browser. The Access Token contains various information about the user itself. This information is learned by sending the Access Token to a tool like http://jwt.ms, which can decode the token and provide all claims. The claims can help identify the user, the IP address of the request, and more pertinent information internal to Azure itself. The claims will include who provided the Token, the secure token server (STS), and what tenant this access token belongs to. Here are some of the more valuable claims found in a JWT.

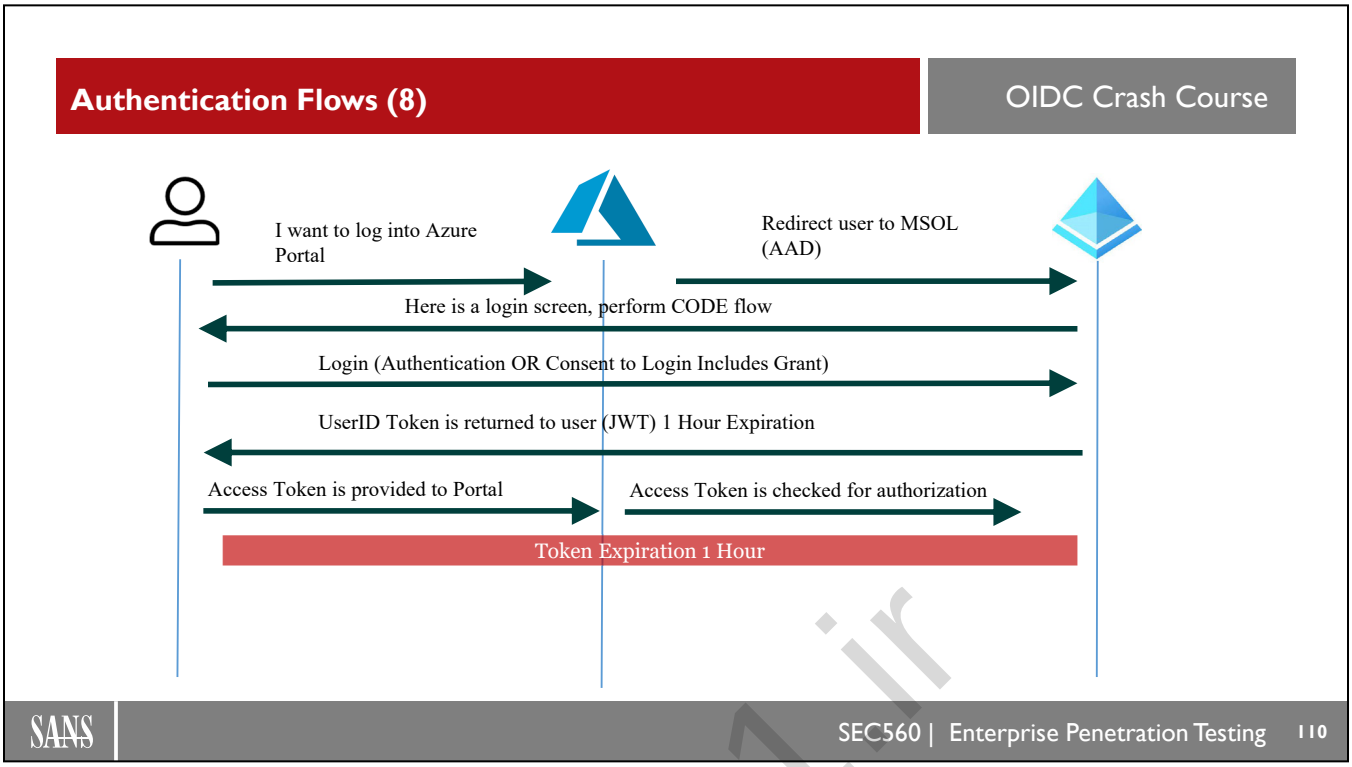
- AUD: The Audience of the ClientID of the application, this is used as a validation tactic for the Oauth Access Token
- IDP: This is the Secure Token Server address or restful URL that provided the token
- -EXP: In Unix epoch time, this is the Expiration Date of the Token
- AppID: This is the application ID of the client that requested the Token; every client is registered. Web Browsers, Azure CLI, Outlook Clients, each one of them has their respective identifier
- AMR: This claim is helpful to identify if the user came in over Multifactor (MFA) or Password Authentication (PWD)
- Ipaddress: This is a V1 claim that is useful to identify the IP address for the authentication

UPN: This is the username of the request.

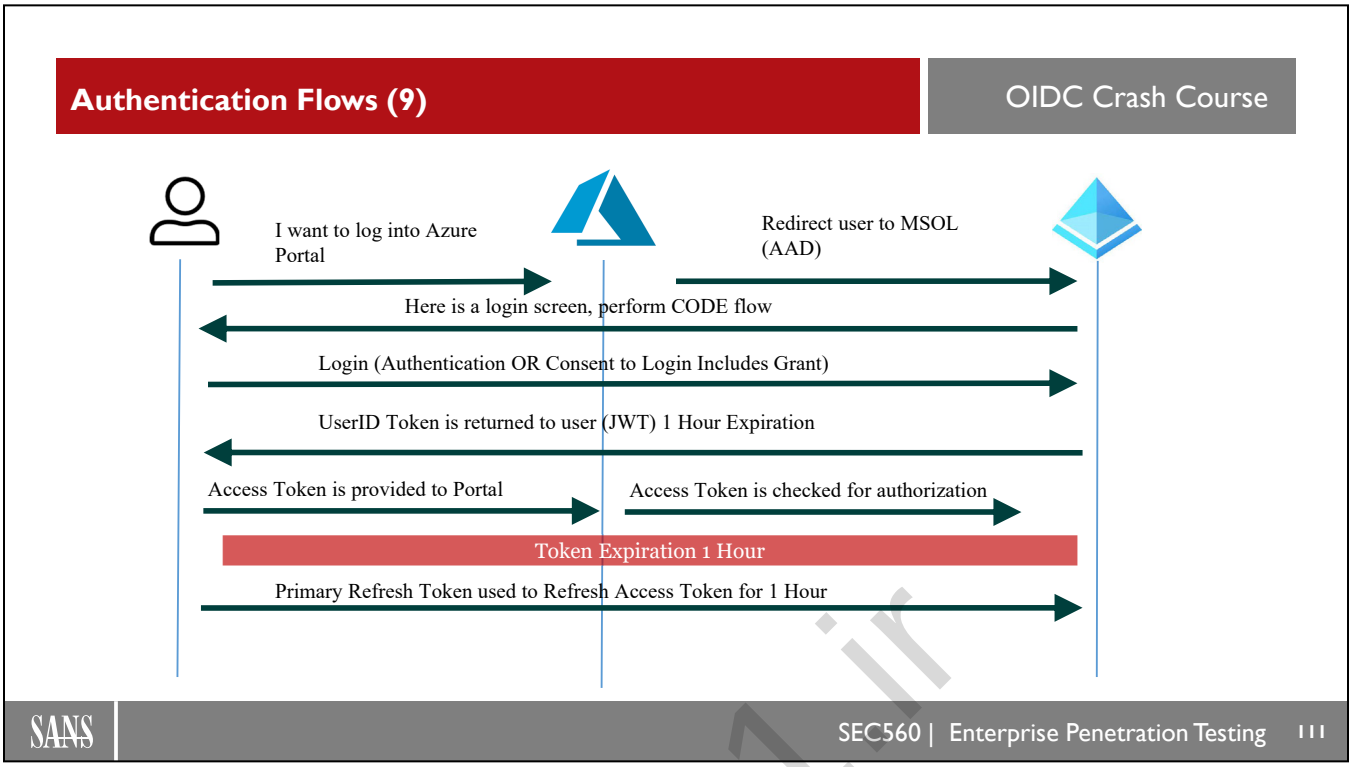
Reference:
https://docs.microsoft.com/en-us/azure/active-directory/develop/access-tokens



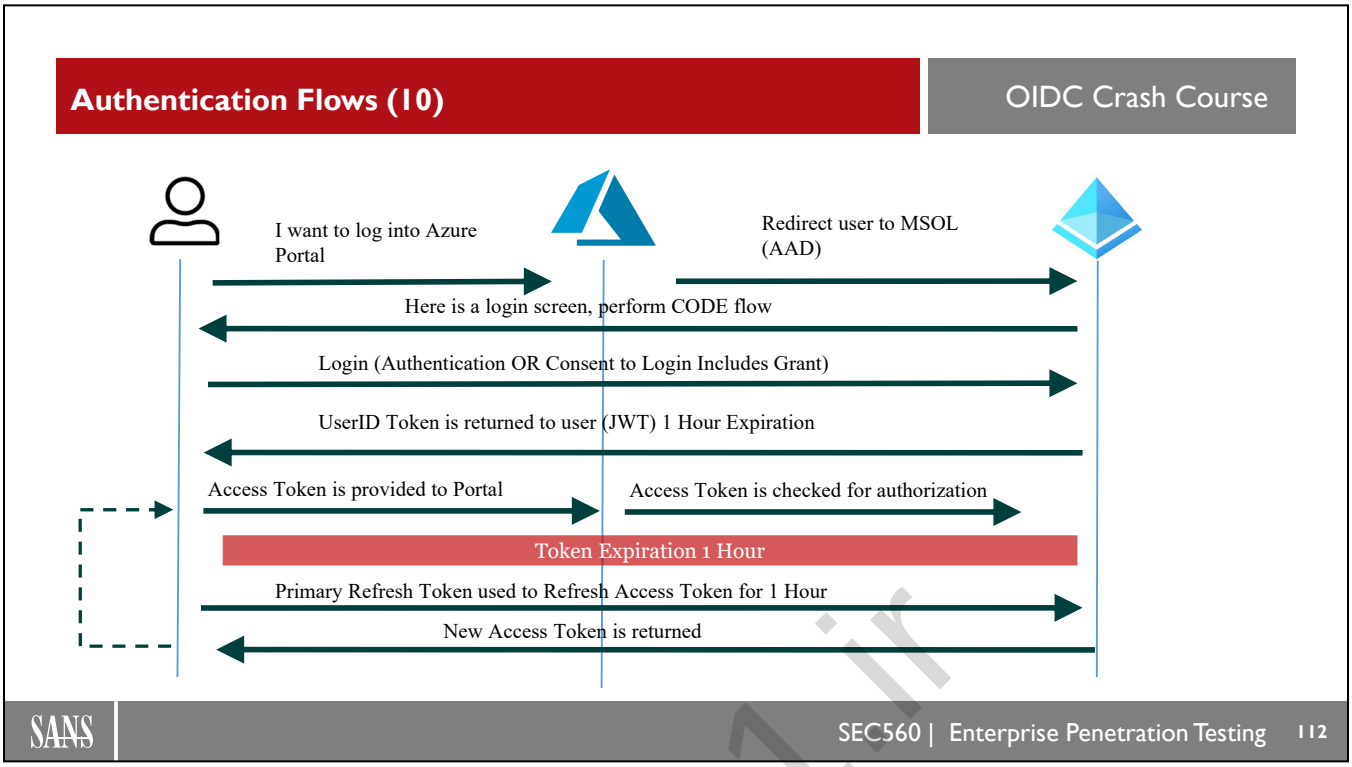
What occurs after the token is returned to the Azure Portal? The system must check that the Access Token is valid with the Azure Active Directory server. It must also check to ensure that the authentication is still valid and that it does not have strange expirations happening. This process is not happening with the user involved. Instead, it's between the Azure Active Directory Server and the Application, which is Azure. The values in here have to be validated for access. Conditional access policy does not apply here. The access was already checked. Not having continually checked access is a critical issue as stealing the credentials will bypass the security measures in that policy.



Let's talk about token expiration for a brief minute. The access token lifespan is 3600 seconds or 1 hour. After 1 hour, the tokens expire. The token expiration is designed so that an attacker cannot steal it and gain permanent access to a system. Having a short expiration window means that there must be a mechanism to refresh this token, or the system's usability will suffer. The user would constantly be asked to log in, which is sometimes not a great experience. What could we do? Well, what we have at our disposal is the Refresh Token. This is also referred to as the Primary Refresh Token (PRT). The Refresh Token is exchanged not between all the third-party systems but only to Azure AD. The Refresh Token itself can ask the Azure AD IdP to Refresh the Access Token, obtain a new one, and allow for authentication to continue. If someone were to get the Primary Refresh Token, they could gain permanent access to an environment.



At this point the user sends the refresh token over to the Azure AD System. Remember, that this is not heading through Azure and then Azure AD its going directly to Azure to reduce the places where this Refresh Token can be intercepted.



The final part of the flow will have the user get a new access token and return this value to the portal for validation. If an attacker can obtain the primary refresh token, they can perform the same flow bypassing conditional access policies, MFA, and other critical controls. Once the access token is returned refreshed, we can now see that the system will pass it back to the party requesting it for validation.

OAuth Flow Types

OIDC Crash Course

- OAuth2 supports several types of authentication flows
  - Code Flow: This is the most common OAuth flow, includes PKCE which prevents CSRF Code Flow
  - Device Code Flow: This is a newer flow type, designed for browser-less authentications
  - Refresh Token Flow: Refreshes the Access Token, longer lived, depending on the IdP. PRT for Azure is Fourteen days, constantly refreshed.
  - Client Credentials Flow: Designed to obtain an access token without a user being involved
- As a penetration tester we can abuse these flows such as Device Flow Authentication to perform attacks

Several flow types exist to help facilitate multiple clients and device types. Oauth2 supports up to eight (8) different flow types and what we will cover here are the most commonly seen ones.

- Code Flow: Code Flow, or three (3) legged flow, is the most common flow type. The idea behind code flow is that you, the user, can authorize and grant authorization to the client. The client is the application you wish to use. The flow will leverage your browser on the front end to get a grant. The grant is then exchanged for an access token. Every time you use a tool that "logs in with Google" or "logs in with Facebook," this is the flow that is used. The application asking you to login with google or face is the client.

- Device Code Flow: Device code flow is useful when you cannot use a browser. You have probably used it. Have you ever seen on a TV App that you need to go to a webpage and enter a code like IXH34H? The code you see presented allows for device code flow OAuth. The Azure CLI will support this for flows that run on servers and do not have browsers.

- Refresh Token Flow: Refresh Token Flow will be the flow that is used to refresh your access token. Your Access Token lifetime will have a one-hour lifetime, while the refresh token will have somewhere between 7 to 14 days of life before expiration.

- Client Credentials Flow: Client Credentials flow is a flow type that is used for things like Service Principals where a static password or certificate is used to allow a client to authenticate. What types of clients are using this? Terraform automation is an example. The terraform module will use Client Credentials Flow to log in as a service principal and automate the creation of items.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Infrastructure Components

- Azure has several infrastructure components in the platform that you need to be familiar with
  - Azure Compute: This is the Virtual Machine Compute Section
  - Azure Blobs: This is the S3-equivalent Block Storage
  - Azure DevOps: A DevOps build pipeline to deploy services
  - Azure Backup: A backup service for azure components, such as VMs
  - Azure API Management: This allows for the ability to manage and use APIs and expose those to the internet
- Each one of the components can be enabled to be used within an Azure Subscription

Azure includes over 80+ services in which users can deploy individual services. However, many of them stem from essential Cloud Computing components that have been productized. Azure has its version of Compute, Storage, and Network, used in Services like Azure Compute. Azure Compute is the Virtual Machine compute environment in which standard operating systems can be launched. Azure Compute is backed by Hyper-V and uses the same Hyper-V constructs on premise hosts. The compute environment includes snapshots, disks, networking, and even backup and restore capability.

Azure also has a File and Block Storage environment called Azure Blobs. Storage Accounts create the Blob accounts, much like Servers hold the files in your file system. From there, individual folders, called containers, will contain your files. Azure Blobs and General Storage accounts can be used to provide both Block Storage and SMB Storage to Azure. It even has a Message Queue and Data like function to store objects.

Azure also has many developer-focused tools like Azure CosmosDB, Azure DevOps, Azure Apps, Azure Front Door, and many services that will allow companies to deploy and scale software. These services will all be contained in the Azure environment to enable users to roll out and deploy software in an Infrastructure Environment. Some of these services do not even apply to "Infrastructure as a Service" and are are "Platform as a Service" or Application Delivery Platforms.

How Azure Organizes Items

- Azure will organize items based on several organizational structures
- Management Groups will organize individual subscriptions
  - Subscriptions, which are tied to billing accounts is, where resources are held
  - Resources can be grouped into Resource Groups
  - Resource Groups can hold resources from many datacenters
- Permissions are applied from Management Groups down to the individual Resources

Azure is a bit different than other platforms like Amazon for organizing items. Azure offers one way to manage multiple subscriptions; they do this using Management Group. The Management group can have nested management groups. Each management group can have more than one subscription. Remember inheritance, a user can be a "contributor" or read/write in a management group, and permission can flow down to individual objects.

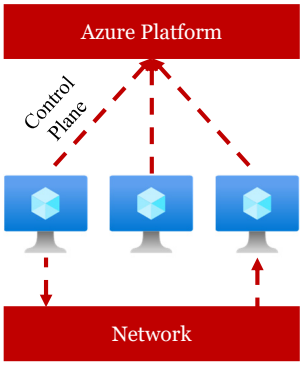
Subscriptions are used to provide for differentiation of billing and grouping of resources. An organization can have more than one subscription. Subscriptions were the classic or legacy model for Azure, so most organizations do not use the management group option; instead, you may only see subscriptions.

Within a subscription, an administrator can group permissions, resources, and networking by resource group. Each resource group can have its own set of resources, and it may not be unusual for you to see multiple resource groups. You will also find that vendor-supported virtual machines may require specific resource groups to be set up.

[1] https://docs.microsoft.com/en-us/azure/governance/management-groups/overview
[2] https://docs.microsoft.com/en-us/azure/cost-management-billing/manage/add-change-subscription-administrator
[3] https://docs.microsoft.com/en-us/azure/role-based-access-control/role-assignments-portal-subscription-admin

Control Plane and Data Plane

- To best understand how we will attack the Azure environment, it is best to understand what is control plane and what is data plane.
  - Control Plane is the components of the system in which were Azure Platform talks to the individual components.
  - Data Plane is what you typically interact with when you are moving packets on the network.
- When you login to Azure and tell the Azure platform to execute commands on a host, that is done through the Control Plane.



To understand how we can run commands in the Azure environment, we need to know how control and data planes work. When Azure machines boot up, they have built-in Azure agents pre-installed. These azure agents talk to the Azure platform to communicate different actions; some allow for machines to be booted and built on the first run; other options will let commands run on demand. The control plane components do not require VPN or any traditional network items to be set up, like in your traditional datacenter. They use an out-of-band communication channel to perform these actions. This agent is pre-installed and pre-configured within the Azure Platform.

The Data Plane is what we are typically using when we move laterally. The data plane generally is where the packets flow between network nodes. When you run WMIC or PowerShell remoting sessions, you or mount a share, this traffic is run over the Data Plane. The Data Plane is not the subject of our Cloud Penetration Testing logic. It is a component that we can use post exploitation to fall back to traditional penetration testing. The Data Plane uniquely exists within clouds enables cloud penetration testing to be different from standard penetration testing.

When we are within the Azure Platform, in the Azure CLI or the Web Interface, and use a specific item, like "azure run-command" or Azure Extensions, these actions are being taken through the Control Panel. The logging is done in the Azure Portal, not on the machine. The agents typically run in an elevated privilege such as "SYSTEM" or "root."

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Azure CLI Tools

- Azure has several tools that can interoperate with the platform
  - Azure CLI: The main python tool, the command to operate it is ***az***
    - Cross Platform, Windows, Linux, MAC, Azure Cloud Shell, Docker
  - Az PowerShell 7: Azure PowerShell 7 Modular.
    - Requires PowerShell to run
- The main tool that we will be using in this class will be the az cli tool
- We can also install these tools on conquered machines to facilitate leverage credentials from within the cloud as well

Microsoft, over the years, has published many different CLI tools to manage non-portal access to the Azure ecosystem. The traditional mechanism that was first published to end-suers was through PowerShell modules. These PowerShell modules required both Windows and specific PowerShell versions. Over the years, however, Microsoft started to work on multi-platform solutions for Azure Management. Having multi-platform support solved one key issue: Azure Cloud shell support for the Azure SDK.

The Azure CLI tool is a python-based application that is cross-platform. It has almost all the functionality of the Azure PowerShell equivalent. The tool itself keeps adding more and more features and should be considered the go-forward tool for azure administration. We can use this tool on all platforms. The CLI tools will also work on Azure Cloud Shell, a web-based bash shell delivered through the Azure Portal. The cloud shell environment ships with many tools and gives the administrators the ability to work with the Azure SDK without installing anything locally. Azure is not the only cloud provider; Google also features a cloud shell. After a specified time, the cloud shell will be stored and compressed for long-term storage and decompress and used when needed. You can determine that the Cloud Shell instance has been used by looking for Storage Accounts with the words "cloud shell" in them.

Azure CLI Basics

- Az login supports multiple options
  - Az login: Uses Code Flow to login
  - Az login -u <username> -p <password>: Submits the code flow credentials directly
  - Az login --i: Leverage a managed Identity
  - Az login --device-code: uses Device Code Flow
- Az tool has several subcommands, which call on the different services
  - az vm: This subcommand will control all aspects of the Azure Compute virtual machines

The Azure CLI supports multiple types of subcommands and authentication mechanisms that we can use to attack Azure. We need to know how the tool works to understand this better. The Azure authentication library, which is now called MSAL (Microsoft Authentication Library), supports multiple Oauth2 and OpenIDConnect flows for authentication. The standard flow is code flow, in which the Azure CLI tool is the "client" in the middle, and the user will see a browser window to support the code flow for authentication. It also supports several other flows.

The system can use a -u / -p on the command line to pass the username and password. Using the command line for this is convenient when MFA is not used, and the browser is unavailable. The other flow it can support can bypass code flow and instead use managed identities to log in to the CLI via the token values that the platform can send down to the system. Using this flow is convenient for when we want to attack from the inside of the platform, and we can leverage a managed assigned identity to do so.

The CLI tool can also support something called Device Code Flow. Device Code Flow will support browser less logins by supplying a code to the Microsoft online website (MSOL) and using a different identity to log in. We can even combine this flow with phishing to lure an unsuspecting user to perform the login on our behalf while allowing our remote CLI to log in as them. Device Code Flow Phishing is a nefarious attack, whereas the system administrator can be phished, and all the phishing lures are valid Microsoft accounts.

Finally, the tool supports many subcommands for standard access individual services; for example, we can specify running, stopping, and starting VM's from within the az vm. We can even do administration such as running commands or rebooting systems through the CLI tool. It is a very powerful way to walk through the permissions of the azure system.

Azure VM Operations

- Azure has several mechanism to facilitate VM Management
  - Run Command
  - Extensions
  - Configuration Management (Desired State Configuration)
  - Azure Automation
- Each one of these can run commands on the systems as an Elevated User (Root on Linux, NT Authority\System on Windows)
- Depending on your Azure Role Permissions you may be able to perform some of these actions

There are many ways that ew can execute cod eon the far end system through the portal. Azure supports within the system by default without any extra tooling, m; we can leverage multiple operational items for systems management. These items can be helpful for administrators but can be abused by attackers. We will be walking through one of these run commands in a subsequent slide. However, for completeness, we will also describe these other two mechanisms, as you may find that you have access to one but not the others.

Extensions are one of these options. These are scripts that can be applied to a machine to facilitate third-party software installation and allow us as the administrator of the system to customize the implementation of software. Extensions are run on application, run as SYSTEM or Root, and execute scripts such as bash or PowerShell scripts. The "custom" extensions can allow us to do almost anything. These extensions you may find can do things like building a domain controller, joining a system to a domain, updating windows, and so on. They must be hosted on a Blob Container, but they can do almost anything outside of that limitation.

Azure Automation is the Azure equivalent to SCCM. This toolchain has many items that help developers and system administrators keep their software running at its optimal state. Azure Automation can allow us to deploy runbooks that will automate the deployment of systems, the configuration of software, and so on. Azure Automation can be used to patch systems and update them. It also has a collection of workers that can watch for processes or network items to either exist or not exist. Using these workers, can we backdoor systems on events. Consider an event where your implant is detected by the blue team and is removed, and you can replace it automatically with a control plane action such as a worker script that installs a different implant.

Running Commands on Virtual Machines

- Run Command leverages the internal Azure Agent in the virtual machine to locally run commands
- The output is truncated to 4kb of output
- Scripts can run between 20 seconds and a maximum of 90 minutes
- The VM must connect outbound to return results
  - Required to be able to hit the Azure Public IP Address space on port 443

```
$ az vm run-command invoke -g ResourceGroup -n linuxvm --command-id
RunShellScript --scripts "sudo apt-get update && sudo apt-get install -y
nginx"
```


One of the mechanisms we will use to run commands remotely on a system is "Azure Run-Command." Azure Run-Command is a subcommand of the Azure VM compute engine; it is a single-use system that allows for the running of commands remotely. Azure Run-Command supports multiple pre-built scripts. There are some useful items in Run-Command:

- RunPowerShellScript/RunShellScript: This option (Powershell for Windows / Shell for Linux) allows you to execute scripts right from the command line or from the portal.
- EnabledAdminAccount: Enabled the local administrator account in Windows
- EnableWindowsUpdate: Enables the Windows Update in Windows
- IPConfig: Runs ipconfig
- RDPSetting: Can setup RDP on a Windows Machine

From within the Azure CLI, a user with the contributor permissions of Virtual Machine (Virtual Machine Contributor or Subscription Level Contributor) can execute run-commands on a machine through the portal.

The minimum permission set would be the following action:

Microsoft.Compute/virtualMachines/runCommand/action

An example of a command-line script to run would be:

```
az vm run-command invoke --command-id RunPowerShellScript --name
hiboxy-dc1 -g HIBOXY --script 'whoami'
```

In the above example, we have the run command invoking a command. The PowerShell Script option is set, and instead of running a local PowerShell script, we are passing the script from the command line. What is the PowerShell script we are passing? The command whoami. In the returned JSON, we will see a message value that will contain our output. Since our output is less than 4000 bytes, it should be fully displayed.

Reference:
https://docs.microsoft.com/en-us/azure/virtual-machines/windows/run-command

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

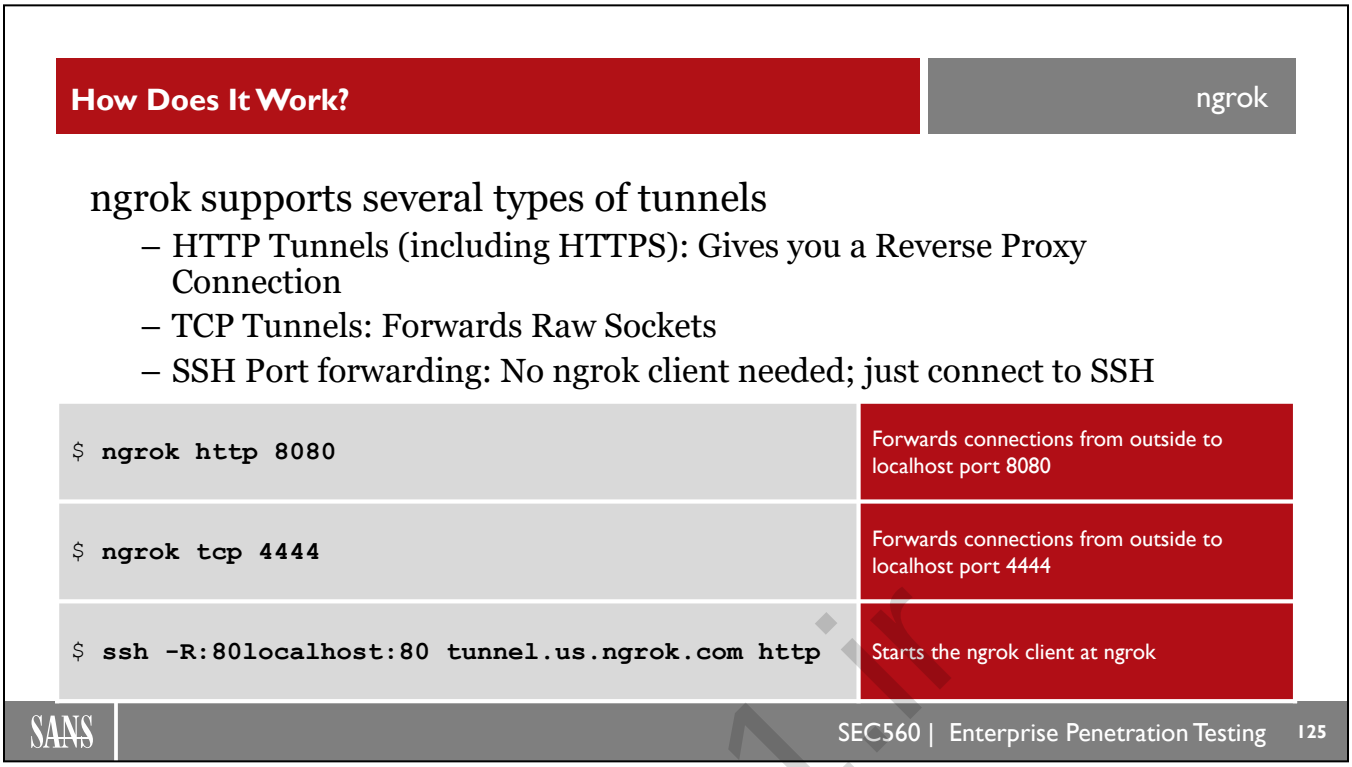
Introduction to ngrok

- ngrok allows us to expose local / internal ports to the internet
- Can be used for simple testing or in some cases Penetration Testing an Assessment
  - Always remember to encrypt payloads and communications
  - Never treat providers like this as you would a VPN

ngrok

The ngrok service can bypass specific network restrictions to facilitate service delivery. Ngrok will enable us to expose local or internal services to the internet. The heart of the ngrok system is the ngrok agent. The agent will connect to the ngrok cloud and provide the end-user with a connectivity method. We can leverage ngrok to facilitate connectivity for our implants or backdoors. A valid use case for ngrok? What about a service like Google Home or Amazon Alexa. If you wanted to build an application for those systems, you would need to host the code in a reachable way by those home assistants. Ngrok can facilitate that access to expose your code from your workstation to the internet in a reachable way. Can this be a security issue? Yes. Can we securely do this? Yes.

We cannot have a directly exposed server to the internet in the next set of labs. Consider what we have been doing in class so far this week. The servers and clients we have been hacking were laterally available to us; we could connect in both directions because we were effectively behind the firewall or on the same LAN. We do not have access laterally; our systems are behind firewalls and NAT interfaces. We can use ngrok to bypass this restriction.



Ngrok establishes a tunneled connection from within an environment out to the internet. The ngrok agent supports several tunnels to provide connectivity back to the client. Ngrok has a paid-for and free service; we would always recommend that you pay for a service like this for a commercial engagement. The pay-for features make the system much more secure and operationalized than you will see in class today.

Tunnel Types

While we will go over how this happens, we will start by describing the types of connections you can make from within an environment to the internet.

The first type is an HTTP Tunnel. The HTTP tunnel works much like an HTTP reverse proxy. Your agent establishes an outbound connection that will then be granted a set of addresses proxied to your local webserver.

You start the agent by typing: ngrok HTTP <port>, you will replace <port> with a number like 8000.

The second tunnel type is a normal TCP socket connection. This tunnel type requires that you have a valid authenticated session and are registered with ngrok. The TCP tunnel will establish an outbound connection on your host and then establish an open session in the ngrok cloud. This time instead of getting a reverse proxy connection, as you would with HTTP, you are getting a raw TCP socket. The socket connection will be displayed to you, and a port forward will exist.

You start the agent by typing: ngrok tcp <port>, you will replace <port> with a number like 4444.

The third type of tunnel is rather interesting. Instead of establishing a connection with the ngrok agent, you use SSH to reverse port forward the connection. The command to connect via SSH will run the ngrok agent on the ngrok cloud. This can then facilitate all standard tunnel types but will use SSH to connect from the host to the internet. This may even evade content filters that try and block ngrok connections.

You start the agent by typing: `ssh -R 80:localhost:80 tunnel.us.ngrok.io http`. Notice that the http part is running on the remote ngrok cloud.

Example ngrok Flow

Example ngrok Flow, how to create a connection

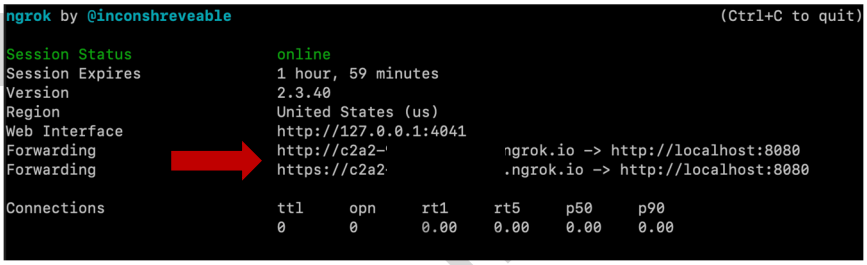
- Step 1: Start your local listener

```
$ python3 -m 'http.server' 8080
```

- Step 2: Start a ngrok forwarder forwarding connections to port 8080

```
$ ngrok http 8080
```

- Connect to the URL



To create a connection using ngrok, we follow a flow that will establish the listener connection. On the tester workstation, you will need to have the ngrok client, and in this example, the listening service. The first step for us will be to turn on a listening server, which is a python webserver in our example.

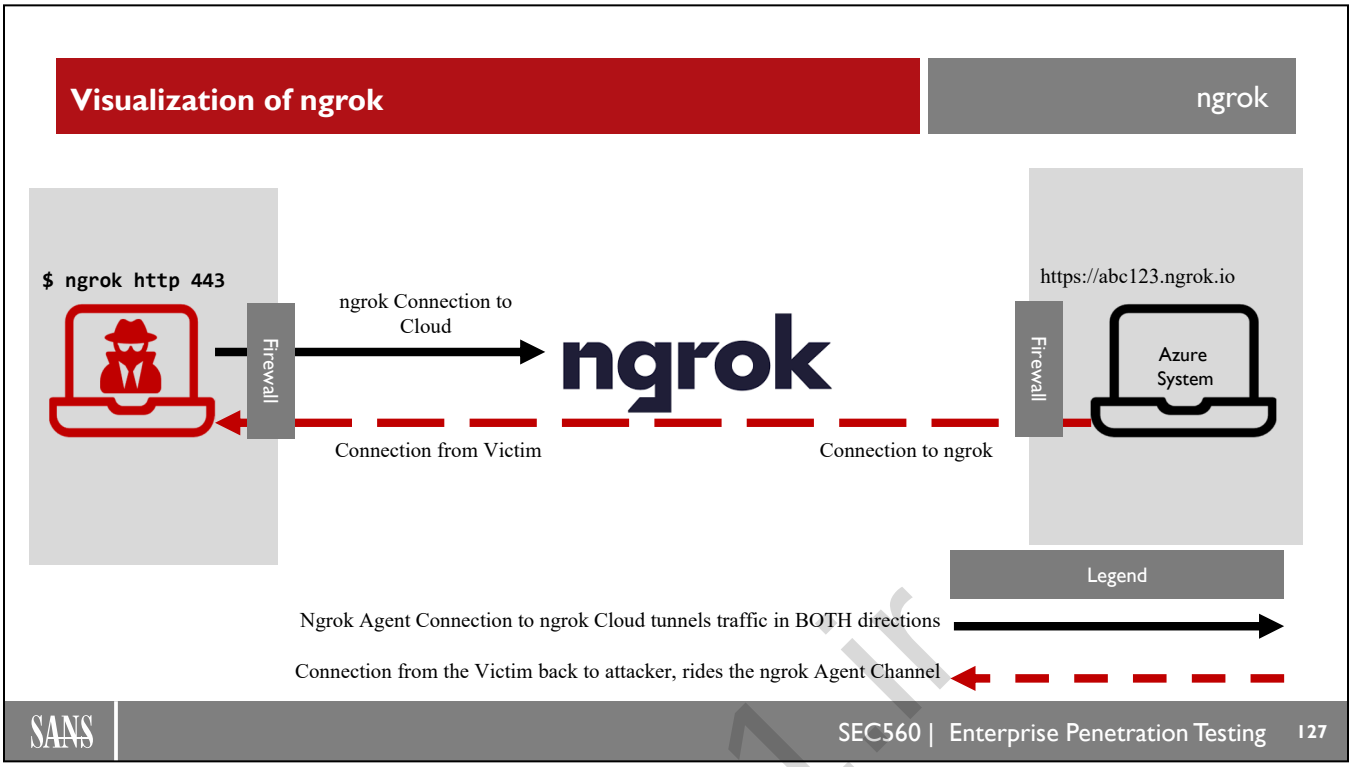
```
$ python3 -m 'http.server' 8080
```

The listener is now running on port 8080 on all interfaces, including localhost. We will start a new window with the ngrok client now that this is done.

```
$ ngrok HTTP 8080
```

The command we are running will connect to the ngrok cloud to establish a connection. Every packet that is sent to the ngrok address in the cloud will be forwarded locally to port 8080, which is our python listener. This will allow us to connect our internal pyhton3 webserver to the internet.

Pay attention to the window that ngrok keeps open for agent connection. Several items are items that we want to focus on. The first one is the redirection addresses we are given. We will be given both an HTTP and HTTPS address for an HTTP listener. The addresses you see ending in ngrok.io are available on the internet. You may also see a local ngrok listener on port 4040 that will serve as a logging system. At the bottom of the screen, you will see diagnostic information. For TCP-only connections, you see open and closed sessions. For HTTP connections, you will see the URLs that have been requested. This is helpful for troubleshooting connectivity.



This graphic depicts how ngrok works to provide tunnels bypassing restrictions. Ngrok itself is an agent. The agent offers a full TCP tunnel connection to the ngrok cloud. TCP Traffic is bidirectional, which means that having an open connection outbound from your machine to the cloud will establish a Firewall Connection and a NAT connection. This act will bypass restrictions that both NAT and Firewalling offer. Once the connection is established, the ngrok system will forward traffic that it receives to your connection.

Look at the graphic; the agent connection is started by typing the following command: ngrok HTTP 443. This will open a connection outbound to the ngrok system. The ngrok system will provide the HTTP URL to the agent that an external 3rd party can connect to send data back to the ngrok agent. The far-end system goes to the following URL: HTTP://abc123.ngrok.io. This will send traffic to the ngrok cloud, which gets redirected back to the agent listening for a connection. Whatever is listening on that device's interfaces on port 443 will now be reachable over the Internet. If this is an implant or a netcat shell, you will receive the connection from the Internet to your device. Since our labs are in Azure, this will help us perform our attacks.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Please work on below exercise.
LAB 5.6: Running Commands

Please go to Lab 5.6: Running Commands in the SEC560 Workbook.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

What Is Better than Domain Admin (DA)? GA

Permissions on Azure

- Getting Domain Admin on a Local On Domain typically means having the capability to perform 'almost' any function
- The equivalent role in Azure AD is Global Administrator
  - Granted to the "first" account or tenant account
  - Can be granted to any user
- Azure AD Connect Synchronization requires Global Administrator access

Domain Administrator is typically the most coveted group in a classic on-premise Active Directory environment. The Domain Administrator rights can be elevated to even more permissions through the use of Enterprise Administrators. A Domain Administrator is equivalent to the root user in Unix but can be root across all machines. Is there a comparable role in Azure AD? The answer is yes; it is called the Global Administrator.

Global Administrator is the first user in the Azure AD environment, and by default, it does not have access to Azure. Remember, Azure AD roles do not translate to Azure Roles. How does the Azure AD role of Global Administrator gain access to Azure items? The user can be elevated using a switch that adds to the User Access Administrator Built-in role in Azure. The User Access Administrator built-in role grants access to add users to subscriptions and resources for administration.

Global Administrators should be tightly controlled as they can do almost anything in the environment. More shocking is that because you are the "root" or "domain admin" of the Azure AD environment, you can also grant these permissions to third parties in the OpenID Connect Consent actions. Consenting to a third party means that a foreign application like Salesforce, or even malicious, can also be granted Global Administrator Access through the consent of the application. Users who constantly use Global Administrator permissions for their day-to-day work can inadvertently grant global permissions to third parties by just consenting as they browse the internet. Like On-premise Active Directory, we should consider these accounts only to be used when needed and not used for day-to-day work.

Azure Permissions

Permissions on Azure

- Azure has BuiltIn Roles and Custom Roles that can be used to perform actions on the platform
  - Microsoft may refer to this as Azure RBAC from time to time
- The BuiltIn roles have a Specific GUID that identifies each
- Roles in Azure have the following "nomenclature":
  - Owner: Owns a Service, can add users
  - Contributor: Consider this to be "Read and Write" access or modify access to the service
  - Reader: These types of roles only have Read access to the service

Azure, like other cloud providers, has many built-in Roles. We have already mentioned some of these roles, such as User Access Administrator. It is a role that allows someone like the Global Administrator to add new users to the Azure Platform and provide permissions to the resources in Azure. You may also find that Azure may call the IAM system Azure RBAC[1]. Azure has many overly permissive roles that you can find. One example is this one:

Log Analytics Contributor[2]

This role is alarming great permissions in an environment, and not uncommonly, it features */read, which means anyone with this role can read all aspects of the Azure resources in the scope of which someone with this role is granted access. It also gives the person the ability to run commands on virtual machines by creating and applying for extensions.

- Azure roles typically follow the following terminology:
- Owner: Owns a service, can add users to the service, and read, write, and execute on the service.
- Contributor: Consider this "Read and Write" access or modify access to the service but can never add a user to the service itself.
- Reader: These types of roles only have Read access to the service, cannot add any users to the service, and cannot modify items.

[1]https://docs.microsoft.com/en-us/azure/role-based-access-control/#:~:text=Azure%20role%2Dbased%20access%20control%20(Azure%20RBAC)%20is%20a,need%20to%20perform%20their%20jobs.

[2]https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#log-analytics-contributor

Permissions IAM Document

Permissions on Azure

```
    "properties": {
       "roleName": "machine-roles",
        "description": "",
        "assignableScopes": [
            "/subscriptions/27bce287-0193-4ec7-9c4f-
78f62be9eaac/resourceGroups/560ng"
        ],
        "permissions": [
            {
                "actions": [
                 "*/read",
                "Microsoft.ClassicCompute/virtualMachines/extensions/*"
                ],
                "notActions": [],
                "dataActions": [],
                "notDataActions": []
```

Role Name: Provides the name of the Role

Assignable Scopes: What this applies to

Permissions / Actions: */read <- All services Read access

extensions/* can write new extensions.

dataActions == Data Plane Permissions

Under the nuance of what is being described in this document, it is essential to understand how to discover overly-permissioned users, groups, and roles. All of the permissions in IAM are backed by these individual JSON documents that contain a list of permissions. Even Azure Built-In roles will have this type of permissions format. Let's look at this document which contains very loose permissions.

"role-name": This key describes the name of the role that you will see displayed. Role Names typically follow the terminology of "Contributor" for all read/write or modify roles and "Reader" for all Read-only views.

"assignableScopes": This key shows where this Permission can be applied. Many of the permissions in Azure are global; the assignableScope will be set to *. In this example, the IAM document will be assigned very specifically to one only specific subscription and one specific resource group. You will not see this role applicable anywhere else in the system.

"permissions"/"actions": The actions listed here are "control plane" only access. They specify what you can do in the Azure portal. In this case, the permissions are incredibly loose. Many of the built-in roles have this setting. The settings here are */read. In other words, every service every resource can be read in the console and the CLI. It also has a very specific read and write attribute. It has the classic computing scope and role for applying extensions. The permissions here grant the role the ability to execute code through extensions.

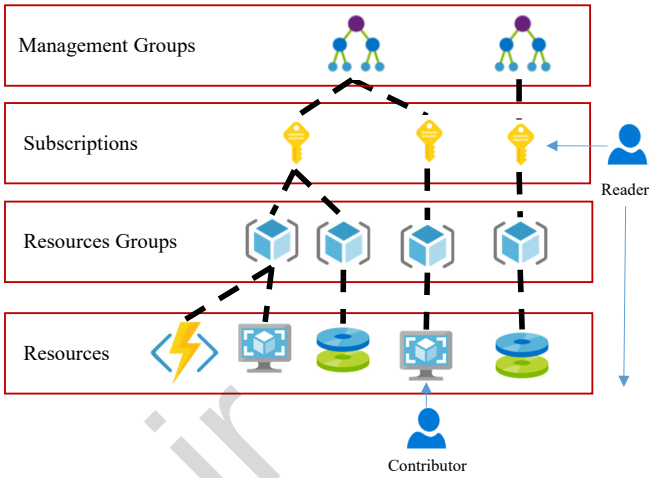
"permissions"/"not actions": Actions in here are also applicable to the control plane, but they are denied actions. Anything in here will be denied access explicitly.

"permisions"/"dataActions": These actions are interesting because they are data plane actions and are unique to Azure. They are actions that apply to items outside the console. What constitutes a data action? Reading and Writing Blobs would be a data action. This action is an application to the data channel of Azure.

"permissions"/"not dataActions": Actions in here are also applicable to the data plane, but they are deny actions. Anything in here will be denied access explicitly.

Where Do Azure Permissions Get Applied?

- Azure Permissions are inheritable
- The permissions can be applied at an upper level like management groups
- Permissions can then be inherited downwards
- Permissions can also be applied at the resource level itself, which is cumbersome to manage



Azure Permissions are inheritable; permissions can be applied at the top of the tree and applied down to the individual resource. If the Azure architecture is designed with this in mind, then permissions can flow downward more gracefully.

Subscriptions group resources that are billed under a single organization. Read and Write, or Contributor at this layer, will change all assets regardless of resource group status. Access at this layer and below can significantly impact assets beneath it.

Resource groups allow for like assets to be grouped. It allows for a collection of systems to be grouped for management. Resource groups are per data center, so you need to understand how to architect multiple resource groups across different data centers.

Permissions can also be applied at the individual resource level. Having permissions per resource would require a more rigorous set of controls that should be automated. With automation, this level of control can be achieved more quickly. Applying permissions here does not take into account inheritance. Inheritance still applies, but it is not necessary to use it when applying IAM controls at the resource level.

Instance Metadata Services in Azure

Permissions on Azure

- Every cloud provider has an Instance Metadata Service.
- The Instance Metadata Service allows for objects and resources in a cloud environment to get information about itself
- Instance Metadata Services are typically found on a locally significant IP.

```
$ curl -H Metadata:true "http://169.254.169.254/metadata/instance?api-version=2021-02-01"
```

- The output of this command from within a resource, like an Azure VM will output information about that VM.

The Instance Metadata Service (IMDS) in a cloud provider will provide resource information about itself. It can be used to manage and configure a machine. It can also be used to provide security information to the resource. The information that is found in the IMDS can sometimes be used to provide us with additional access. The URL for the Metadata service can be found in the curl command below. Notably, some headers need to be passed to have the API respond.

Instance Metadata's IP is typically sending the traffic to the physical host in the data center running an Agent that will proxy that request to the Azure Platform.

```
$ curl -H Metadata:true "http://169.254.169.254/metadata/instance?api-version=2021-02-01"
```

Azure CLI, for example, can use this Instance Metadata Service (IMDS) credential to log in to the Azure platform as that identity.

```
$ az login -i
```

An attacker can also move the instance metadata token onto their own systems to be able to persist outside of the host.

Managed Identities in Azure

Permissions on Azure

- Systems in Azure can have identities, like authentication profiles
- These identities can allow for systems, servers, services, to log into other Azure Services
  - Consider an Azure Application that wants to log into log to a File Blob.
  - The application service can have an Identity that is allowed to upload data to that bucket
- Two types of identities exist:
  - User Assigned: Created by you the end user
  - System assigned: Created by the Azure Platform
- If you can read the identity token, you can then leverage it for access
  - You can read the identity by querying the MetaData Service from the service or system

Objects such as Services, Systems, or Applications in Azure will need to authenticate, just like a user, to access other services within the platform. These managed identities will help facilitate access, but what types of access? Consider an application that needs to encrypt or decrypt data. Would it not be helpful for that application to talk to the Azure Key Vault service to perform those operations? A system administrator can grant a managed identity this level of access and then apply that managed identity to the application.

How would the application obtain the access token to log in to the platform? The system would obtain this token through the Instance Metadata Service. For example, if you wish to use curl to see the access token, it could be retrieved using the following command:

```
$ curl 'http://169.254.169.254/metadata/identity/oauth2/token?api-
version=2018-02-01&resource=https%3A%2F%2Fmanagement.azure.com%2F' -H
Metadata:true -s
```

Notice that the restful URL is sending the request to a specific oauth2 endpoint in the Instance Metadata Service that would be able to send the managed identity token to the local system.

Identity Types

Two types of identities exist in Azure. There are differences between each one. The first type is an end user-managed identity called a User Assigned Managed Identity. The User Assigned Managed identity is created as a standalone resource and is managed entirely by the end-user. It expires when the user expires it, it can be added to multiple systems and is somewhat not as secure as other options.

The second type of identity is an Azure Platform managed identity. The identity is called a System Assigned Managed Identity. The identity is fully managed by Azure and is created and managed per asset. The lifecycle of the identity is a platform managed, and it cannot be shared across multiple systems; each system must have its own identity. This would reduce the potential of multiple system compromise through a single identity.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Always Create a Report

Effective Reporting

- **Third-party testers:** The report is your leave-behind. Two or three years from now, it will be the only evidence of the work you did
- **In-house tests:** Report is important to communicate with management and to show that you've used due diligence in securing your network
- **Write your report as you go!**
  - Helps you to grab the right screenshots
  - It is important to document your methodology, and that is easier to do as you go
  - No forgetting what you've done
  - Prevents all the pain at the end
  - You can save final polish for the official post-test reporting time, but begin reporting during the test

Testers should always create a final report describing their work and findings. If the testers work for a third-party penetration testing or ethical hacking company, the report is the only evidence they leave behind of the project they performed. A high-quality report may be used for several years after an engagement as fixes are deployed, new architectures are considered, and new products and configurations are rolled out. Because of this potential long-term value of the report, make sure you focus on creating a high-quality deliverable. Reports that are technically incorrect, improperly formatted, or poorly written will linger and could cause significant problems in the future.

Sometimes testers who perform vulnerability scans of their employers, in-house, think that they don't need to write a detailed report. After all, you're just scanning your own enterprise network to see how things look. Why waste the extra time documenting your results and findings? Even in such circumstances, we still recommend that you create a report because it helps codify the work you've done and provides an organizational memory of the value you've provided. If management won't provide enough time or staff resources for creating a report, work hard to convince them of its importance. A report is concrete proof that your organization is exercising its due diligence in conducting vulnerability scans on a regular (or at least sporadic) basis. If there are major security problems in the future (such as a major breach), and your organization is investigated by the government or shareholders, the vulnerability scanning reports will be helpful in showing your attempts to measure your security stance in the past.

In addition, to increase the efficiency of your overall process, you should consider starting to write the report while the test is underway. That way, your reporting process will not take as long, and the overall quality of your results will better reflect the details of your work. As you conduct the test, take notes that follow along with the documented methodology that you use. Grab screenshots that illustrate your findings as you make those findings. When you have significant results, write them up immediately (or at least a bulleted list of their most salient points) while they are still fresh in your mind. You usually don't have to finish the report while the testing is underway. Most penetration tests include a reporting period after the hands-on testing is complete. But this post-testing report time will go much more smoothly and efficiently if you start writing during the test phase.

Don't Just Regurgitate Vuln Scan Results

Effective Reporting

- Copy and paste from vuln scanner is obvious—don't do it!
- Develop your own list of findings
  - Internal team: Findings already describe your business risk, and internal tools already in place to mitigate the risks
  - External team: Findings are worded in your style and include customizations that need minimal tweaking
- Review the results and help interpret the risk in context of the target
- Manually validate major findings to eliminate false positives
- You will often need to adjust the severity or risk rating

A common mistake of testers in writing reports is merely to cut and paste the output of their vulnerability scanning tools into a document and turn it in as the official report. Sometimes they don't even format their results, simply throwing vulnerability scanner output over the wall, hoping that someone will act on it.

Stock vulnerability scanner output is obvious. Its format is canned and often not easily interpreted by nontechnical personnel. Many of the issues that are identified are false positives, and the risks and recommendations do not tie in with the target organization's business objectives.

We strongly recommend that you review the results of your vulnerability scanning tools and other techniques and then help interpret them in light of the business of the target organization. What do these vulnerabilities mean? How could they impact the business? Given a realistic threat that exploits the vulnerability, what risk does it pose? And how should fixes be prioritized, whether they be patches, configuration changes, architecture alterations, and so on?

In addition, you should strive to verify all major findings manually, especially those that are high risk. If you do not validate findings, your report may be full of false positives, which significantly lower the value of your work and may call into question your skills and approach.

Also, note that you may need to adjust the High/Medium/Low risk indications of your vulnerability scanning tools. What Nessus or another scanner calls a High risk might be a Low risk for the target organization. Of course, the opposite is also possible. All these issues have to be addressed in the business context of the target organization.

Recommended Report Format

- Executive Summary
- Introduction
- Findings sorted by risk
- Methodology
- Appendices (Optional)

This slide contains a recommended report format. Of course, you may iterate on this structure, adding or removing components. Your organization may already have a report format for penetration tests and related work. If so, get it, and use it. If not, feel free to adopt this structure, reviewing it and tweaking it to meet your needs.

We recommend that the penetration testing, vulnerability assessment, or ethical hacking report include these elements:

Executive Summary: This brief up-front matter is meant for executives who may not read the full report, providing them the most important conclusions from the work.

Introduction: This component describes the project at a high level, answering the who, where, when, and why aspects of the project.

Findings: This section presents the actual findings, listed one by one, in the target environment with detailed technical descriptions. The findings are sorted so that the most significant risk issues are discussed first.

Methodology: This part of the report describes the "what" of the project: What did the team do? It covers the process of the penetration test or ethical hacking engagement.

Conclusions and Future Considerations: This last section summarizes the project results and is reminiscent of the Executive Summary. It also may optionally include future considerations for items to test more comprehensively or defend better.

The report also may include optional appendices of reinforcing data and results.

Let's look at each section in more detail.

1. Executive Summary (1)

- Arguably the most important part of the report
- Writing effective Exec Summaries is a vital art
- Write this last, so you can best summarize the test
- Should be 1 to 1.5 pages
- Very briefly (2 to 3 sentences), summarize project
  - Date, goal, who, overview
- Summarize overall risk identified during test
- Add high-level recommendations
- Positive findings are a nice touch

REDSIEGE

EXECUTIVE SUMMARY

SYNOPSIS

Red Siege experts evaluated the security of Nakatomi Trading Corp's (NTC) perimeter network during the course of a three-week period in July 1988. The goal of the assessment was to identify security vulnerabilities in NTC's internet facing systems and services. All issues identified by Red Siege have been manually verified and exploited (where applicable) to demonstrate the underlying risk to NTC, its employees and clients.

FINDINGS OVERVIEW

Findings grouped by risk severity:

Critical Risk issues 2
High Risk issues 4
Medium Risk issues 6
Low Risk issues 8
Informational issues 2

KEY FINDINGS

Red Siege found one critical vulnerability related to the authentication to NTC's VPN which allows an attacker to access internal systems. Additionally, Red Siege found two high severity vulnerabilities that have the potential to impact visitors to NTC's website which could impact Nakaomi's brand and reputation.

- NTC's VPN does not use two-factor authentication and multiple users were found using weak passwords. An attacker could easily guess these passwords and access the internal network, compromising the confidentiality of NTC's data and possibly leading to data loss.
- Red Siege found significant shortcomings in defenses and secure coding related to a common web related attack known as cross-site scripting (XSS). This type of attack allows a malicious actor to use the website to attack visitors, which could expose personally identifying information, authentication credentials, or even compromise the victim's computer.

STRATEGIC RECOMMENDATIONS

To increase the security posture of NTC, Red Siege recommends the follow strategic actions be taken:

- IMPLEMENT TWO-FACTOR AUTHENTICATION ON PUBLIC FACING SYSTEMS. Internet facing systems are regularly being probed and attacked. Extra care needs to be taken on these systems to prevent unauthorized access.
- STRENGTHEN PASSWORD REQUIREMENTS. NTC should use technical means to ban known bad/weak passwords and train users on safe password practices.
- REQUIRE DEFENSIVE CODING TRAINING FOR DEVELOPERS. Developers are the first line of defense when it comes to custom web applications. Developers should be made aware of the common mistakes that lead to vulnerabilities and learn ways to prevent these issues before the code is run on production systems.
- IMPLEMENT DATA WHITELISTING. Data sent from a user to the webserver should always be treated as potentially malicious. Developers should identify the data expected by the application and disallow characters that are invalid.

Red Siege would like to thank the Nakatomi Trading Corp for the opportunity to work on this project. Should you have any questions regarding these findings or the contents of this report, please feel free to contact us.

Page 3

The Executive Summary is probably the most important section of the entire report. Composing an effective Executive Summary is quite an art form and an important one for testers to master. The purpose of this section of the report is to describe to decision makers what the results of the report mean, and more importantly, recommendations for the actions they need to take based on its results.

Although it appears at the start of the report, we strongly recommend that you write the Executive Summary last, after you have completed the rest of the report. That way, you can properly summarize the findings and recommendations from elsewhere in the already-written body of the report.

The Executive Summary should be quite short. Ideally, it will be under one page in length and should be no more than one and a half pages. It should not get highly technical, nor should it go over every finding. It should be composed so that it can be read by itself, offering firm conclusions and advice for someone who doesn't necessarily read the rest of the document.

The Executive Summary starts with a brief description of the project in three or fewer sentences, describing the project's date and goals, as well as who participated in the project. It also provides a brief overview of what was done.

Next, the summary covers the overall risk posture of the target environment. Were major problems discovered? Is the environment worse than expected by testers and/or the people reading the Executive Summary? Or did the test results show that the security of the target environment appeared to be sound?

I. Executive Summary (2)

Effective Reporting

Keep the Exec Summary high-level and business focused
Do not use technical terms!

- Include a list of 3-6 significant findings and business impact
- Discuss root causes and management level recommendations
  - Changes to organizational structure
  - Altered policies or procedures
  - Changes to technology
- Include positive findings
  - Don't just beat them up; they are likely to have done something right
- A mediocre test with a good Executive Summary may be more valuable than a good test with a mediocre Executive Summary

And now comes the vital part of the Executive Summary: You need to explain the business impact of your findings. You can use a bulleted list of the biggest three to six findings, followed by several sentences that explain the business impact in terms of the risk for each finding. Focus on the institutional levers management can pull to change things to eliminate not just a given discovered vulnerability but the underlying reason that the vulnerability exists. Such levers could include changing organizational structure, altering policies and procedures, adding personnel, deploying new technology, or many other possibilities.

Don't roll your eyes about this component of the report and this mindset for composing it thinking that it is merely management fluff. Effectively communicating your results to management is critical. A mediocre test with a good Executive Summary is likely much more valuable in improving an organization's security stance than a good test with a mediocre Executive Summary. The former provides impetus for change and improvement, whereas the latter just reinforces security vulnerabilities in the minds of technical people who, in all likelihood, know what those flaws are already.

Remember that the Executive Summary should stand alone as a document without the rest of the final report. Often, these one or two pages are split from the rest of the report and sent up a management chain.

Introduction should provide an overview of the test

- Date range and time range
- Scope
- People associated with the test
  - Testers
  - People associated with the target who supported the testers
  - Include name, role, and contact info: email address and phone number
- Sometimes the scope and personnel is moved to the Appendix and the introduction becomes part of the Executive Summary

Next, we move to the Introduction component of the report. This one- to three-page section provides an overview of the project so that the reader understands when the project occurred, what was included in the scope (and possibly items purposely left out of the scope, if applicable), and who participated in the test.

Include a list of people, organized in a table, that identifies the testers and the individuals associated with the target environment who supported the testers (usually those who participated in the daily/weekly debriefings). The table should include each individual's name, role, and contact information, such as a phone number and email address. Then, provide a brief overview of the most salient findings, using language similar to that included in the Executive Summary but possibly with more technical detail.

These Introduction components of reports are immensely helpful six months after the project or later in analyzing what was tested. Many organizations frequently refer to these Introductions to determine what else in their environment needs to be tested and what needs to be retested.

Describe ease of exploitation and impact, and provide recommendations to mitigate the finding

- Vulnerable system(s) identified by IP address (and name, if applicable)
- Risk level: Usually High, Medium, Low (possibly adding "Critical" and "Informational")
- Explain the business risk and a detailed technical description
- Customize text in light of target's business and technical environment
- Recommendations
  - Ideally with multiple methods of dealing with the issue
  - Long-term and short-term fixes

And now we get to the Findings section, where the technical results are described. We recommend starting with high-risk findings, prioritized so that the highest of the high-risk issues comes first. Then move down to medium and low risk, again prioritizing each within its subsection.

For each finding, list the system(s) that exhibit the given flaw, identifying the machine by IP address and name (if you know it). Include the risk level and estimate of how easily the given issue could be exploited. The risk level is typically sorted into High, Medium, and Low, but your given organization may want to characterize risks in another fashion. Some pen testers also add "Critical" as a category above "High" risk, and "Informational" as a category below "Low" risk. Then provide a two-sentence summary of the finding, again focused on the business risk to the extent that you can. For some of the lower-risk findings, the business issue may be fairly small.

We then get into the technical details. Here we provide a description of what the flaw is and how an attacker could exploit it and cause an impact to the target organization's business. The discussion is described in technical terms, but it, again, ties back into the business risks.

And, finally, we include a Recommendation section, which should provide one or more means of remediating the discovered flaw. If there are multiple ways of addressing the issue, include them all, but point out the one most likely to match the business needs and technical environment of the target organization.

Be careful with including the successfully guessed or cracked passwords in your report because these reports are often passed around to numerous people in a target organization. Including passwords in a report that might be reused elsewhere could be a security risk, so verify with the target organization whether such information should be included in a report. Many penetration testers have a policy of describing the characteristics of cracked passwords (such as length or character set) but not including the actual passwords in the report.

In the Technical Findings section, screenshots that illustrate the issue can be helpful in conveying a lot of information and making the results feel "real". Network diagrams can also go a long way in explaining some technical ideas.

Screenshots help make findings seem "real"

- Include useful screenshots that demonstrate a vulnerability
- Suggested: Copy text from CLI tools for screen readers and accessibility
- Focus screenshots on the most important part of the screen
  - Don't include a giant screen dump with a bunch of useless info
  - Zoom in ... where is the "action"?
  - If you gain command shell access, run "hostname"

Illustrate, don't decorate!
Each image should support the finding, not just show you did something

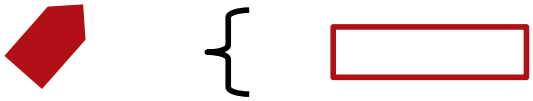
Screenshots aren't just eye candy. They can help a report have its intended effect (motivating the organization to improve its security stance) because they show the success of a penetration tester's or ethical hacker's work.

When including screenshots in your reports, make sure to focus the screenshot on the issue you are illustrating in the prose of your report. Some penetration testers include giant screenshots with a lot of detail distributed around the picture, but only a small portion of the figure is actually meaningful. The crux of your screenshot could be lost amid a sea of other unimportant information in the same figure. Focus your screenshots on the action you want to show, which might be a vulnerability discovered by a scanning tool, a command shell returned by a successful exploit (often running "hostname" to show the machine's name), or other useful items.

Screenshot Elements

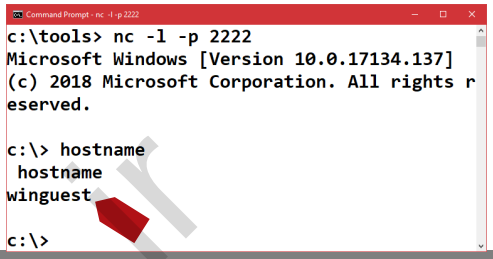
Effective Reporting

- Augment screenshots with various graphical elements to help focus attention on the most important part



Notation: Use meaningful text to illustrate a point

- Color can add a nice touch, but don't rely on it exclusively
  - Your report will likely be printed in black and white and may even be photocopied or faxed
  - Be aware of color blindness
- Black text on white background is ideal (especially in terminals)



Screenshots should almost always be augmented with graphical clues as to where the most important information is located in the figure. Use arrows, brackets, or squares overlaid on the screenshot to focus the reader's attention on the most important information. Small textual notations can also help hammer home the point of a given element on the screen. Try not to make your screenshots too busy, however. Remember, you need to demonstrate that a given vulnerability is real, not bewilder the reader with so much obscure detail in your pictures that they will not understand.

Color can help to bring out salient points. For example, you could use red graphical elements to zoom attention on a big problem. However, keep in mind that your beautiful color report will likely be passed on to other people in the target organization via photocopying and faxes and will likely not retain the color elements you create. Thus, while color can help, don't rely on it exclusively to make your point. Make sure that there is contrast between your screenshots and your graphical elements.

Terminals shown in screenshots often work best with black characters on white background. They are easier to see that way (especially when faxed) and require less toner from printers.

Screenshot Tools

Effective Reporting

- Use a tool that allows you to quickly and efficiently take high-quality screenshots
- Important features: Crop, Highlight, Annotate, Redact
- Photoshop and Gimp are powerful, but have too many features that get in the way

Snagit	US$50: It is widely regarded as the best screen-capturing tool Crosshairs mean no extra or missing pixels Designed for screenshot workflow
Flameshot	Linux, Mac, Windows Free and open-source
Snip & Sketch	Free in the Windows App store or pre-installed with recent updates Windows only
Greenshot	Windows: Free Mac: US$1.99 but missing features included in the Windows version

There are a number of screenshot tools that are useful when writing a report. The tools should make taking screenshots quick and easy. The tools should also include some basic functionality, such as:

- Crop
- Notation
- Highlight
- Redaction

Let's take a look at a few tools that can help us with screenshots.

Snagit is designed for screenshots and the workflow of taking, manipulating, and using the screenshots. The software operates very similarly on both Windows and macOS. Most online polls handily rank this as the best screenshot tool. It includes a number of nice features, such as the crosshairs when taking a screenshot to ensure you don't include extra pixels or miss pixels, thereby creating a slightly sloppy screenshot. The features incorporated into the application are designed for quick editing and to speed up the workflow. This is the tool used to generate the screenshots in the wiki. SnagIt is available at https://www.techsmith.com/screen-capture.html.

Flameshot is a free and open-source tool that runs on Windows, MacOS, and Linux. Its feature set is similar to Snagit. Flameshot is available at flameshot.org.

Snip & Sketch is a free in the Windows App store and is installed with more recent updates to Windows. This is a great tool that maybe already be installed on your system!

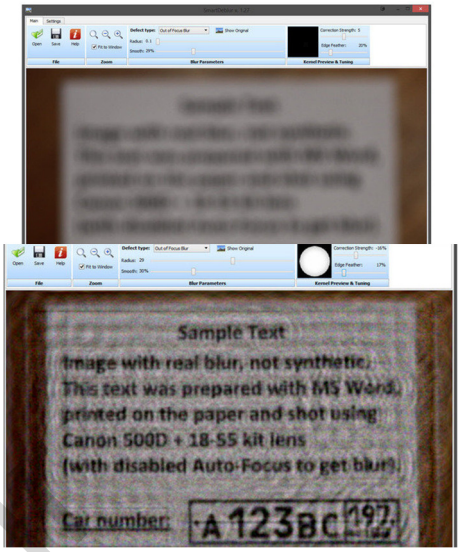
Greenshot is a free tool for Windows. It is designed for quick screenshots and annotation. You can quickly take a screenshot, annotate, and paste into a report. The tool is very good considering it's free. The Mac version of the tool costs US$1.99 and is missing significant features that are included for free in the Windows version. Greenshot is available at getgreenshot.org.

Redaction and Transparency

Effective Reporting

Be very careful of transparency and blur!

- If not used properly, you could leak sensitive information
  - Passwords and hashes
  - Keys
  - User information, PII, PHI
- Blackout is best, but can be visually harsh
  - Make sure the overlay is part of the image, and not just in the document
- Pixilation is good (with large pixels)
  - Use pixel size of 1/2 font height or larger
  - There are tools to depixilate if your pixels are too small



It is very important to properly redact information in reports. Taking safer screenshots is an import first step. Always disable transparency in your terminal windows (and others) when taking a screenshot to avoid information leakage. The author of this course once received a penetration test report from a prominent penetration testing company and learned about the company's other clients because of information leakage due to transparency.

We do not want to include sensitive information in the report, such as passwords, password hashes, encryption keys, API keys, private user information, or other sensitive data. We can't control the audience of the report after it is delivered, so as much sensitive information as possible should be redacted. Obviously, technical details on the findings can't be removed, but users' passwords should not be included in the report. When redacting, a color overlay in the image is the best, but a large black box can be visually jarring. Be very careful with the blur feature, as it can be deblurred if not done properly, as shown in the screenshots above. Pixilation using pixels of at least half the font height will sufficiently redact the data and result in a more visually appealing image. The large pixels prevents data exposure. One such tool is: https://bishopfox.com/blog/unredacter-tool-never-pixelation

Additional reading:

https://www.pcworld.com/article/2032916/review-smartdeblur-restores-blurred-images-indulges-csi-fantasies.html

Shortlink: redsiege.com/560/blur

Recommendations (1)

Effective Reporting

- Consider recommendations that fall into one or more categories:
  - Applying patches
  - Changing configuration
  - Hardening systems
  - Applying filters
  - Altering architecture
  - Changing processes: Always consider this kind of recommendation
- Make sure you consider the root cause of your findings
  - Why is this so, and how can it best be fixed?
  - How can we prevent it, or something similar, from happening again?
  - Even for deeply technical issues, what process allowed the issue to arise, and how can we stop it from happening again?

Because recommendations are so valuable, let's look at the recommendations element of the Findings section of the report in more detail. Most of your penetration testing and ethical hacking project recommendations will fall into one of the following categories:

Applying patches: Many times, a test discovers vulnerable software that can be fixed by upgrading to a more recent version or applying a patch. Such changes tend to be easy, provided that they are conducted in coordination with the change management.

Hardening systems: Tests often have findings associated with shutting off an unneeded service, removing software that doesn't have a defined business need, or otherwise changing the configuration of a target machine to harden it from a security perspective.

Applying a filter: Some issues identified during a test can be mitigated by applying a filter in front of the target service, through a network firewall, network-based Intrusion Prevention System, or even a host-based filtering technology, such as a personal firewall or host-based IPS.

Altering architecture: The most effective way of dealing with some flaws is to alter the overall architecture of the target environment, moving systems around, deploying new kinds of technology, applying filters in a different order, or tweaking the flow of data in the environment. Such solutions tend to be among the most expensive when compared to other issues on this list.

Changing process: For each one of your recommendations, always consider making a recommendation to improve processes. Even for deeply technical issues, some process associated with the given technical issue allowed the vulnerability to arise. Make recommendations for improving the process to avoid such an issue happening again in the future.

Most recommendations in a penetration testing or ethical hacking report fall into the first three categories here: patches, hardening, and filtering.

Whenever you make a recommendation, make sure you consider the root cause of the problem. Why is the issue present in the first place? How can target system personnel prevent it from happening again in the same instance or in other related areas? Answer those questions and you'll be providing additional business value for your work.

Recommendations (2)

Effective Reporting

- Make multiple recommendations, if possible, with discussion of trade-offs among them, considering:
  - Needed functionality and security
  - Operational complexity
  - Costs
  - Ancillary benefits: new features
  - Ancillary risks: new problems, such as denial-of-service exposure
- Value Add: Tell them how they can verify if a fix is in place and working
  - This can be difficult to do succinctly, but this gives you a chance to shine

Where possible, provide multiple recommendations for addressing a given discovered vulnerability. Discuss the trade-offs between the recommended solutions in light of the target organization's environment, which could be associated with different impacts on operational complexity, varying costs, and the ancillary benefits of some solutions over others (including the target organization's familiarity with a given type of operating system or software package, the enabling of future services or features from one solution over another, and more). Some solutions could have ancillary risks, as they open new vectors of attack, possibly including denial-of-service exposure.

Make sure that you base your recommendations on what you feel is truly the best way for the given organization to deal with a discovered issue, independent of pressures to raise or lower costs. If cost is a concern, you should list both higher- and lower-cost alternatives and discuss which ones make the most sense in light of your understanding of the target environment.

Don't be pressured to recommend only high-cost solutions, which some third-party penetration testing companies might do to raise the price of follow-on mitigation projects.

From the opposite perspective, recommend budget-conscious approaches to make sure you have a possibility of improving the security stance of the organization. But don't be pressured to use only low-cost solutions, even if budgets are slim.

If you want to go above and beyond in providing value in your penetration tests, provide in your recommendation some steps an organization can take to verify that your recommended fix is in place and working effectively. For example, if you recommend applying a patch or a filter, provide target system personnel with some command line activities they can run to verify that the patch is in place or that the filter is properly filtering. Such fix verification advice can be difficult to formulate succinctly, but it provides some extra verification for target system personnel that their defenses have improved because of your work. Might this prevent you from getting some extra follow-on testing work? Yes, that is possible, but it makes your initial test results and report much more valuable to the organization.

4. Methodology

- Describe the process used, listing results at each stage:
  - Recon
  - Scanning
  - Gaining Access—Exploitation
  - Post-Exploitation
- Include screenshots and of representative actions
- Especially important if there aren't many major findings

The next part of the final report includes a description of the methodology used in the testing. Provide a description of the step-by-step process used, and briefly list the findings discovered at each phase. Depending on how you conduct the test, this section may include details of the recon, scanning, and gaining access phases. You may want to include a list of tools you used in the project. Some target organizations want to see the tools to learn from your report, whereas other organizations are not interested in the list of tools. When in doubt, include the tools you used for the project in the narrative description of your methodology, and consider adding a table to briefly summarize the tools used.

This methodology section may run from 1 to 10 pages, depending on how the test was conducted. If there are few high-risk findings associated with the project, this section actually grows in importance to demonstrate the value provided by the testing. Sometimes a testing project finds little wrong with the target environment. This is, of course, is good news. But if the Findings section is skimpy, the target organization may question the quality, veracity, and completeness of the test. Thus, in situations in which there aren't many findings, we need to provide even more detail in this Methodology section to show at a fine-grained level what we did during the test. That demonstrates not only that we conducted a thorough test (again illustrating and recording our due diligence), but it also makes the test more reproducible in the future. A year later, the target organization may want to redo the test, and it can rely on this report to replicate it exactly, performing a gap analysis on changes in the interim, or purposely tweak the process next time.

The scanning component of the Methodology section should include an inventory of all the machines that were included in the test. We recommend a table with one row per target machine, listing the IP address, machine name(s), associated business unit (if it can be determined), and its method of discovery (DNS, ping sweep, Google searches, and so on). During the recon discussion (560.1) we discussed maintaining this inventory while the test occurs. We can directly use this inventory, in a condensed form, in our final report. For some tests, this inventory can get long and should be moved into optional appendices at the end of the report.

Move lengthy or cumbersome items and lists to an Appendix

- If a list, such as of vulnerable targets, is longer than a page, move it to an Appendix
  - Detailed vulnerability scan output, if required
  - Backup documentation associated with the project
  - Summaries of memos communicating with third parties
  - Very long items can be delivered in supplemental documents
- Add a link from the finding to the Appendix, and visa versa

The report appendices should include lengthy outputs that would be cumbersome to put inline with the rest of the report. You could include detailed vulnerability scan output in an appendix. However, make sure that the report recipients want such information. It is often counterproductive and may make it look like you are merely adding weight to the report without any useful business or technical reason.

You could include backup documentation associated with the project, like the Rules of Engagement or scope description, in an appendix.

You also may want to include any memos that were sent to third parties getting their permission to scan their systems, such as a DNS server or web-hosted environment. Also, if any requests were sent to these organizations to change their configurations, you may want to include a copy of the request and response for the change in an appendix.

Include other items as they are required or helpful.

Recommended Reading

- Brian "BB" King – Hack for Show, Report for Dough
  https://www.youtube.com/watch?v=c_LBWqNDY0M
  shortlink: redsiege.com/560/dough
- Example Reports:
  https://github.com/juliocesarfort/public-pentesting-reports
  shortlink: redsiege.com/560/reports
- SEC402: Cybersecurity Writing: Hack the Reader

Brian "BB" King – Hack for Show, Report for Dough
https://www.youtube.com/watch?v=c_LBWqNDY0M
shortlink: redsiege.com/560/dough

Example Reports:
https://github.com/juliocesarfort/public-pentesting-reports
shortlink: redsiege.com/560/reports

SEC402: Cybersecurity Writing: Hack the Reader

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Please work on below exercise.
Lab 5.7: Gaining Access and moving Laterally

Please go to Lab 5.7: Gaining Access and Moving Laterally in the SEC560 Workbook.

Course Roadmap

- Comprehensive Pen Test Planning, Scoping, and Recon
- In-Depth Scanning and Initial Access
- Assumed Breach, Post-Exploitation, and Passwords
- Lateral Movement and Command and Control (C2)
- ***Domain Domination, Azure Annihilation, and Reporting***

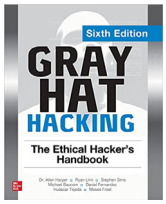
560.5 Domain Domination, Azure Annihilation, and Reporting

Kerberos
Kerberoast
LAB 5.1: Kerberos
Domain Dominance
LAB 5.2: Domain Dominance
More Kerberos Attacks
Silver Ticket
LAB 5.3: Silver Ticket
Golden Ticket
LAB 5.4: Golden Ticket
Domain Privilege Escalation
Azure Intro
Azure AD
Azure Recon
Azure Password Attacks
LAB 5.5: Azure Recon and Password Spraying
OpenID
Azure Infrastructure
Running Commands on Azure
ngrok
LAB 5.6 Running Commands
Permissions on Azure
Reporting
LAB 5.7: Gaining Access and moving Laterally
Conclusion

This page intentionally left blank.

Conclusion

- Where to go from here to learn how to Cloud Penetration Testing?
- Books:
  - Penetration Testing Azure (NoStarch)
  - Penetration Testing Azure for Ethical Hackers (Wiley)
  - Gray Hat Hacking 6th Edition
- Training:
  - SANS SEC588 Cloud Penetration Testing
- Training Lab:
  - Purple Cloud by Jason Ostrom



What are your next steps in exploring cloud penetration testing? This course section is just a fraction of what Cloud Penetration Testing is. For those interested in it, there are books on the subject that one can purchase. Penetration Testing Azure from No Starch Press is one such book. There is a newer book titled Penetration Testing Azure for Ethical Hackers. There are also a few pages in the Gray Hat Hacking 6th Edition book that contains Cloud Penetration Testing labs. SANS offers training in the SEC588 Course focused on Cloud Penetration Testing. Even a lab, a downloadable Azure, and Azure AD installation called Purple Cloud[1].We hope you enjoyed this section of the class.

[1] https://github.com/iknowjason/PurpleCloud

Conclusion for 560.5

Conclusion

- That concludes the 560.5 session
  - Microsoft Active Directory attacks allow penetration testers to perform lateral movement and gain deeper access into a target organization
  - Fully on-premises environments are getting scarce. Most organizations are incorporating Azure AD in one way or another
  - This integration opens up different attack venues, however, often similar or related to the Domain Domination attacks that are well-known already
- In 560.6, we'll have our Penetration Testing Workshop and Capture the Flag Event
  - A day-long lab that incorporates lessons from the entire class

That lab brings our 560.5 session to a conclusion. As we've seen, Microsoft Active Directory attacks allow a penetration tester to perform lateral movement and gain deeper access into a target organization and better understand its business risks. Fully on-premises environments are getting scarce. Most organizations are incorporating Azure AD in one way or another. This integration opens up different attack venues, however, often similar or related to the Domain Domination attacks that are well-known already.

Our next section, 560.6, will include a day-long hands-on lab, in which you'll be part of a penetration testing team attacking a target organization. As part of this Penetration Testing Workshop, you'll participate in a Capture the Flag event to demonstrate the skills you've built in the labs throughout the rest of the course.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z