Credentials in Shares

Description

Credentials exposed in network shares are (probably) the most encountered misconfiguration in Active Directory to date. Any medium/large enterprises will undoubtedly have exposed credentials, although it may also happen in small businesses. It almost feels like we are moving from "Don't leave your password on a post-it note on your screen" to "Don't leave unencrypted credentials and authorization tokens scattered everywhere".

We often find credentials in network shares within scripts and configuration files (batch, cmd, PowerShell, conf, ini, and config). In contrast, credentials on a user's local machine primarily reside in text files, Excel sheets, or Word documents. The main difference between the storage of credentials on shares and machines is that the former poses a significantly higher risk, as it may be accessible by every user. A network share may be accessible by every user for four main reasons:

- One admin user initially creates the shares with properly locked down access but ultimately opens it to everyone. Another admin of the server could also be the culprit. Nonetheless, the share eventually becomes open to `Everyone` or `Users`, and recall that a server's `Users` group contains `Domain users` as its member in Active Directory environments. Therefore every domain user will have at least read access (it is wrongly assumed that adding 'Users' will give access to only those local to the server or Administrators).
- The administrator adding scripts with credentials to a share is unaware it is a shared folder. Many admins test their scripts in a `scripts` folder in the `C:\` drive; however, if the folder is shared (for example, with `Users`), then the data within the scripts is also exposed on the network.
- Another example is purposely creating an open share to move data to a server (for example, an application or some other files) and forgetting to close it later.
- Finally, in the case of hidden shares (folders whose name ends with a dollar sign `$`), there is a misconception that users cannot find the folder unless they know where it exists; the misunderstanding comes from the fact that `Explorer` in Windows does not display files or folders whose name end with a `$`, however, any other tool will show it.

Attack

The first step is identifying what shares exist in a domain. There are plenty of tools available that can achieve this, such as PowerView's Invoke-ShareFinder. This function allows specifying that default shares should be filtered out (such as `c$` and `IPC$`) and also check if the invoking user has access to the rest of the shares it finds. The final output contains a list of non-default shares that the current user account has at least read access to:

Credentials in Shares

```
PS C:\Users\bob\Downloads> Invoke-ShareFinder -domain eagle.local -ExcludeStandard -CheckShareAcces

\\DC2.eagle.local\NETLOGON      - Logon server share
\\DC2.eagle.local\SYSVOL        - Logon server share
\\WS001.eagle.local\Share       -
\\WS001.eagle.local\Users       -
\\Server01.eagle.local\dev$     -
\\DC1.eagle.local\NETLOGON      - Logon server share
\\DC1.eagle.local\SYSVOL        - Logon server share
```

Cheat Sheet
Go to Questions

Table of Contents

Setting the stage
Introduction and Terminology
Overview and Lab Environment

Attacks & Defense
Kerberoasting
AS-REProasting
GPP Passwords
GPO Permissions/GPO Files
Credentials in Shares
Credentials in Object Properties
DCSync
Golden Ticket
Kerberos Constrained Delegation
Print Spooler & NTLM Relaying
Coercing Attacks & Unconstrained Delegation
Object ACLs
PKI - ESC1

Skills Assessment
Skills Assessment

My Workstation

OFFLINE

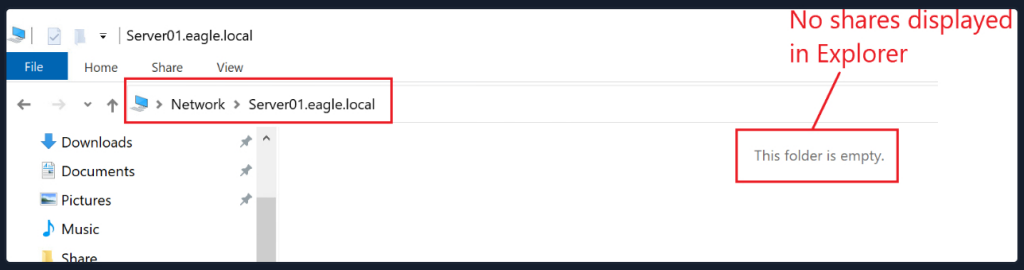
Start Instance

∞ / 1 spawns left

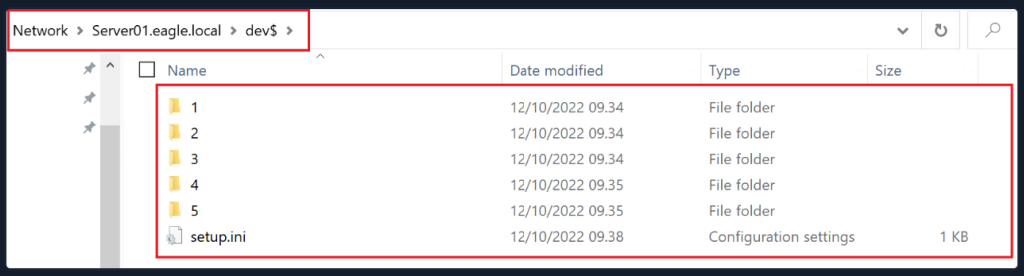
```
PS C:\Users\bob\Downloads> Invoke-ShareFinder -domain eagle.local -ExcludeStandard -CheckShareAccess
\\DC2.eagle.local\NETLOGON       - Logon server share
\\DC2.eagle.local\SYSVOL         - Logon server share
\\WS001.eagle.local\Share        -
\\WS001.eagle.local\Users        -
\\Server01.eagle.local\dev$      -
\\DC1.eagle.local\NETLOGON       - Logon server share
\\DC1.eagle.local\SYSVOL         - Logon server share
PS C:\Users\bob\Downloads> _
```

The playground environment has few shares in the domain, however, in production environments, the number can reach thousands (which requires long periods to parse).

The image above shows a share with the name `dev$`. Because of the dollar sign, if we were to browse the server which contains the share using Windows Explorer, we would be presented with an empty list (shares such as `C$` and `IPC$` even though available by default, Explorer does not display them because of the dollar sign):



However, since we have the `UNC` path from the output, if we browse to it, we will be able to see the contents inside the share:



A few automated tools exist, such as SauronEye, which can parse a collection of files and pick up matching words. However, because there are few shares in the playground, we will take a more manual approach (`Living Off the Land`) and use the built-in command `findstr` for this attack. When running `findstr`, we will specify the following arguments:

- `/s` forces to search the current directory and all subdirectories
- `/i` ignores case in the search term
- `/m` shows only the filename for a file that matches the term. We highly need this in real production environments because of the huge amounts of text that get returned. For example, this can be thousands of lines in PowerShell scripts that contain the `PassThru` parameter when matching for the string `pass`.
- The `term` that defines what we are looking for. Good candidates include `pass`, `pw`, and the `NETBIOS` name of the domain. In the playground environment, it is `eagle`. Attractive targets for this search would be file types such as `.bat`, `.cmd`, `.ps1`, `.conf`, `.config`, and `.ini`. Here's how `findstr` can be executed to display the path of the files with a match that contains `pass` relative to the current location:

Credentials in Shares

```
PS C:\Users\bob\Downloads> cd \\Server01.eagle.local\dev$
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> findstr /m /s /i "pass" *.bat
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> findstr /m /s /i "pass" *.cmd
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> findstr /m /s /i "pass" *.ini
setup.ini
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> findstr /m /s /i "pass" *.con
4\5\4\web.config
```

```
PS C:\Users\bob\Downloads> cd \\Server01.eagle.local\dev$
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> findstr /m /s /i "pass" *.bat
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> findstr /m /s /i "pass" *.cmd
```

```
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> findstr /m /s /i "pass" *.ini
setup.ini
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> findstr /m /s /i "pass" *.config
4\5\4\web.config
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$>
```

Matched an .ini file

Matched a config file

Similarly, we can execute `findstr` by looking for `pw` as the search term. Note that if we remove the `/m' argument, it will display the exact line in the file where the match occurred:

Credentials in Shares

```
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> findstr /m /s /i "pw" *.confi

5\2\3\microsoft.config
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> findstr /s /i "pw" *.config
5\2\3\microsoft.config:pw BANANANANANANANANANANANNAANANANANAS
```

```
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> findstr /m /s /i "pw" *.config
5\2\3\microsoft.config
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> findstr /s /i "pw" *.config
5\2\3\microsoft.config:pw BANANANANANANANANANANANNAANANANANAS
```

Matching a config file containing 'pw'

Line containing 'pw'

One obvious and yet uncommon search term is the `NetBIOS` name of the domain. Commands such as `runas` and `net` take passwords as a positional argument on the command line instead of passing them via `pass`, `pw`, or any other term. It is usually defined as `/user:DOMAIN\USERNAME PASSWORD`. Here's the search executed in the playground domain:

Credentials in Shares

```
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> findstr /m /s /i "eagle" *.ps

2\4\4\Software\connect.ps1
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> findstr /s /i "eagle" *.ps1
2\4\4\Software\connect.ps1:net use E: \\DC1\sharedScripts /user:eagle\Administrator Slavi123
```

```
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> findstr /m /s /i "eagle" *.ps1
2\4\4\Software\connect.ps1
PS Microsoft.PowerShell.Core\FileSystem::\\Server01.eagle.local\dev$> cat 2\4\4\Software\connect.ps1
net use Z: \\server1\administrators
net use E: \\DC1\sharedScripts /user:eagle\Administrator Slavi123
```

Matching a PowerShell script

The last command reads the text file and displays its content, including the credentials; this would be the domain's built-in account credentials (not uncommon to find domain admin rights in these script files).

Note that running `findstr` with the same arguments is recently picked up by `Windows Defender` as suspicious behavior.

Prevention

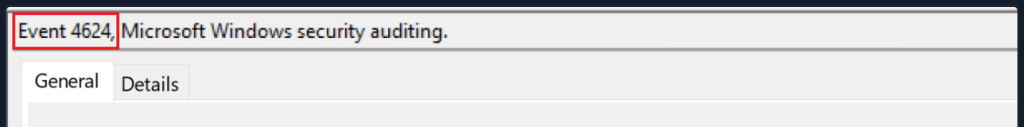
The best practice to prevent these attacks is to lock down every share in the domain so there are no loose permissions.

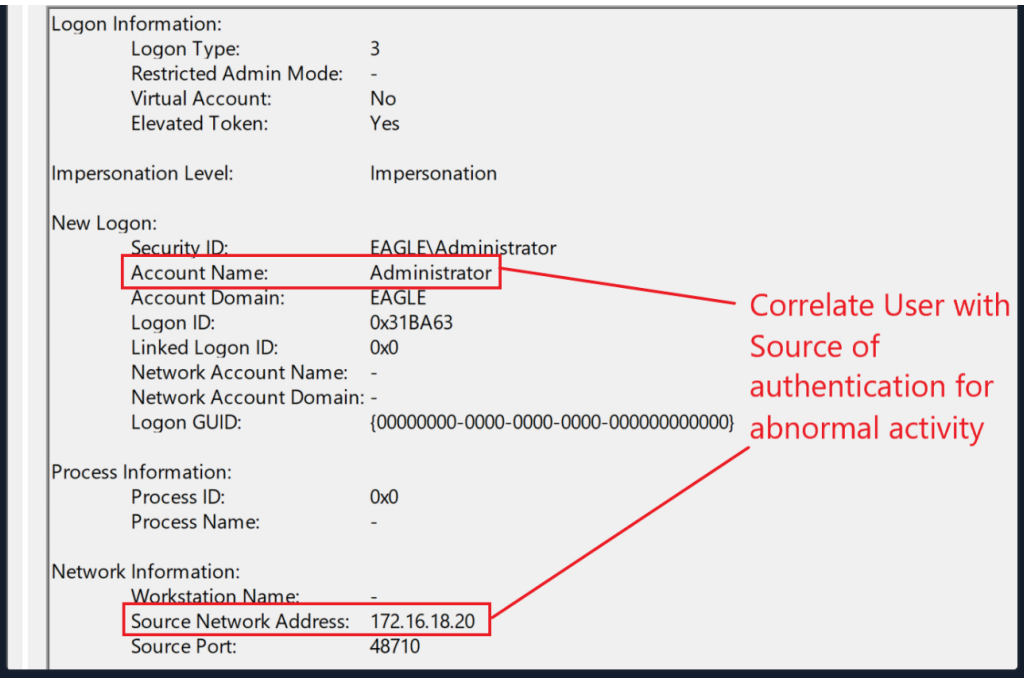
Technically, there is no way to prevent what users leave behind them in scripts or other exposed files, so performing regular scans (e.g., weekly) on AD environments to identify any new open shares or credentials exposed in older ones is necessary.

Detection

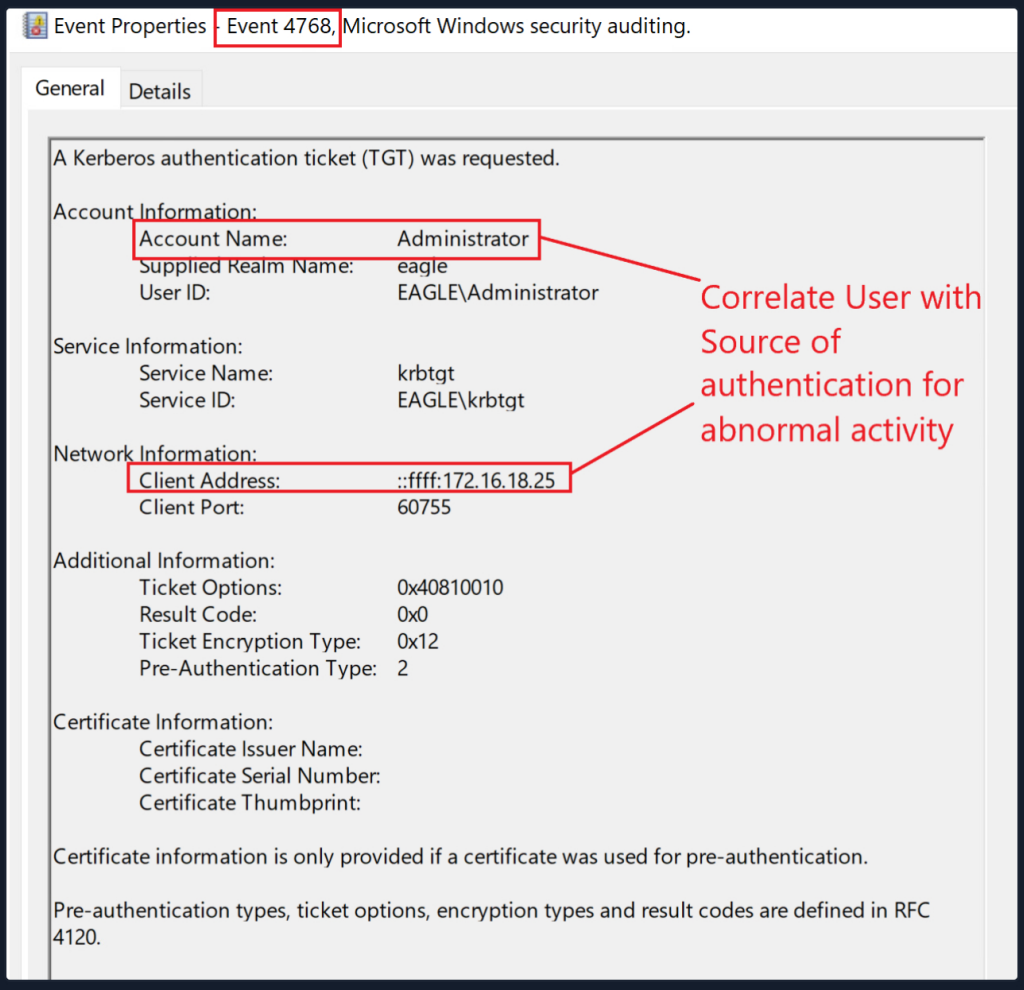
Understanding and analyzing users' behavior is the best detection technique for abusing discovered credentials in shares. Suppose we know the time and location of users' login via data analysis. In that case, it will be effortless to alert on seemingly suspicious behaviors—for example, the discovered account 'Administrator' in the attack described above. If we were a mature organization that used `Privileged Access Workstation`, we would be alert to privileged users not authenticating from those machines. These would be alerts on event IDs `4624`/`4625` (failed and successful logon) and `4768` (Kerberos TGT requested).

Below is an example of a successful logon with event ID `4624` for the Administrator account:





Similarly, if Kerberos were used for authentication, event ID 4768 would be generated:



Another detection technique is discovering the `one-to-many` connections, for example, when `Invoke-ShareFinder` scans every domain device to obtain a list of its network shares. It would be abnormal for a workstation to connect to 100s or even 1000s of other devices simultaneously.

Honeypot

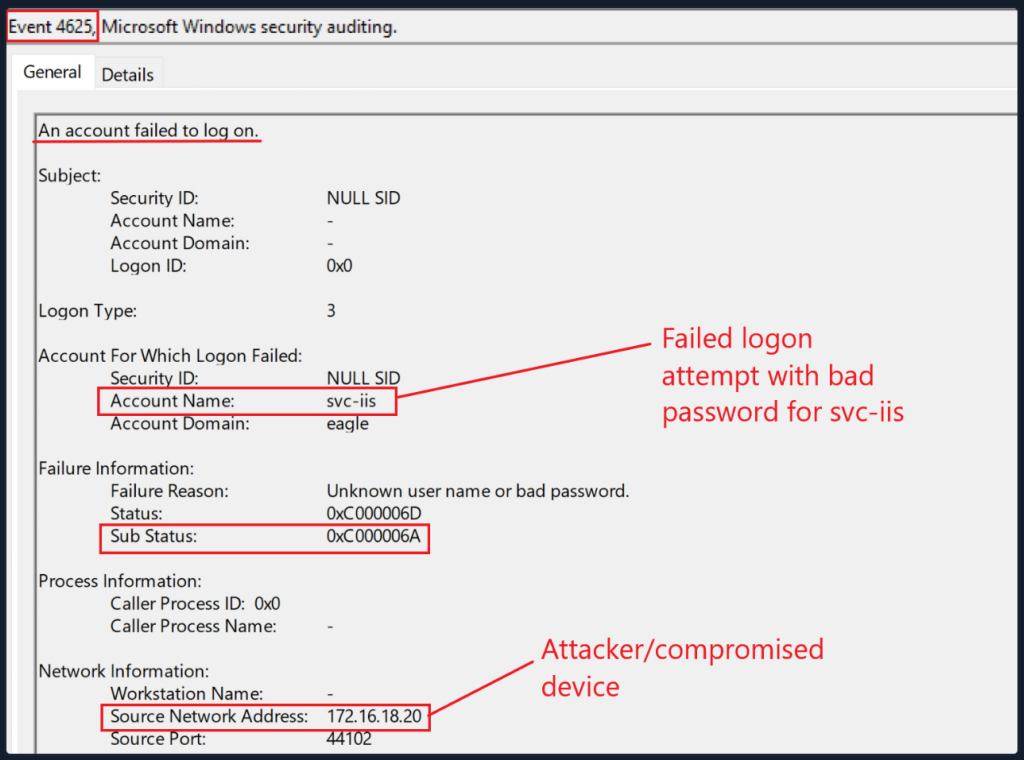
This attack provides another excellent reason for leaving a honeypot user in AD environments: a semi-privileged

username with a `wrong` password. An adversary can only discover this if the password was changed after the file's last modification containing this exposed fake password. Below is a good setup for the account:

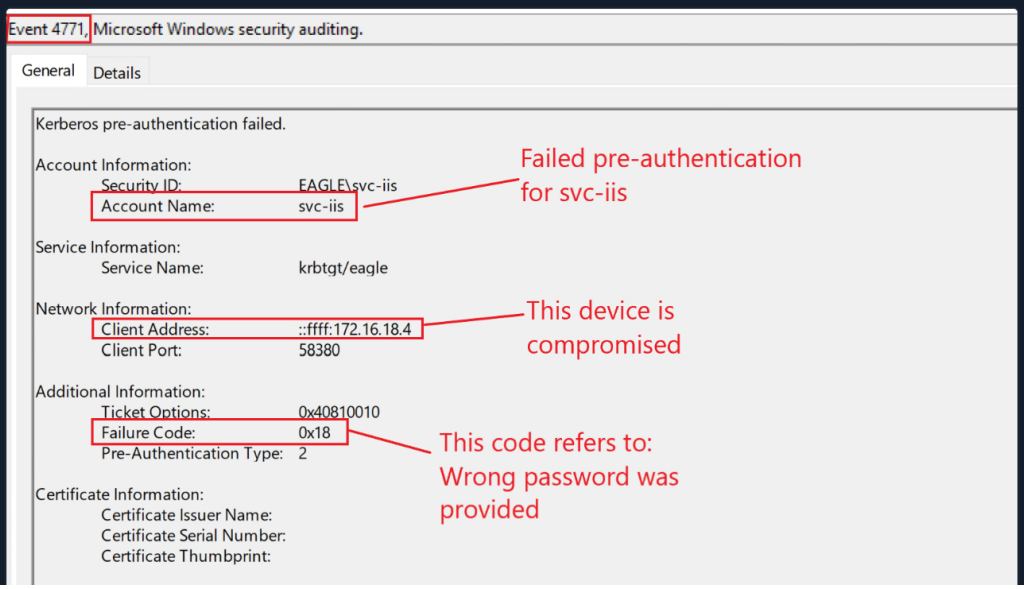
- A `service account` that was created 2+ years ago. The last password change should be at least one year ago.
- The last modification time of the file containing the `fake` password must be after the last password change of the account. Because it is a fake password, there is no risk of a threat agent compromising the account.
- The account is still active in the environment.
- The script containing the credentials should be realistic. (For example, if we choose an `MSSQL service account`, a `connection string` can expose the credentials.)

Because the provided password is wrong, we would primarily expect failed logon attempts. Three event IDs (`4625`, `4771`, and `4776`) can indicate this. Here is how they look from our playground environment if an attacker is attempting to authenticate with the account `svc-iis` and a wrong password:

- `4625`



- `4771`

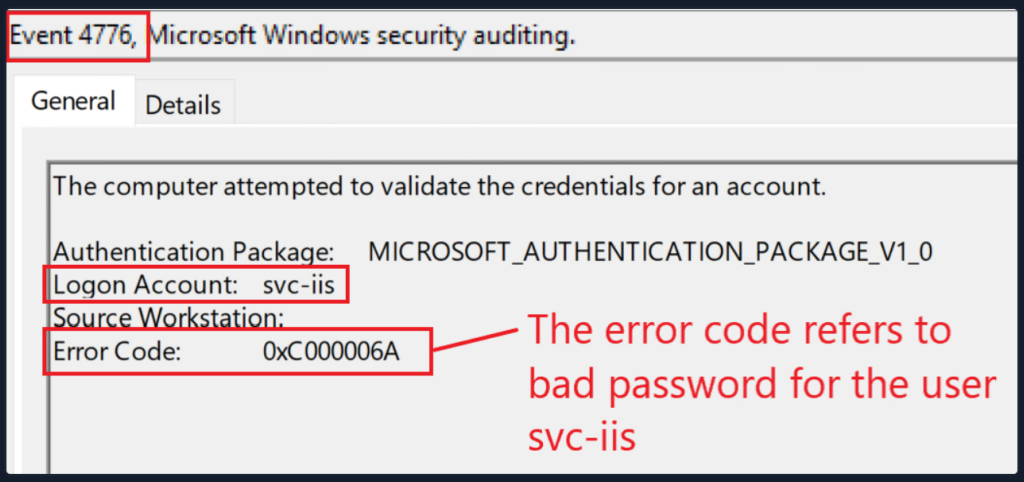


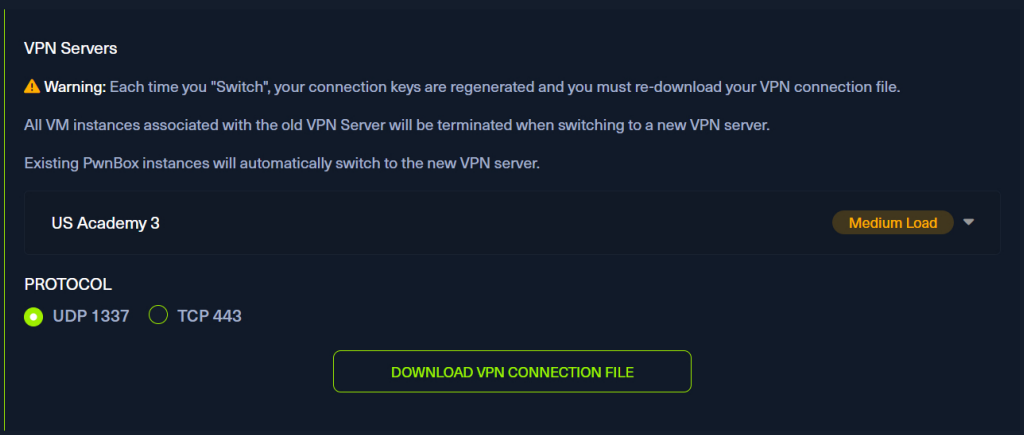
Certificate information is only provided if a certificate was used for pre-authentication.

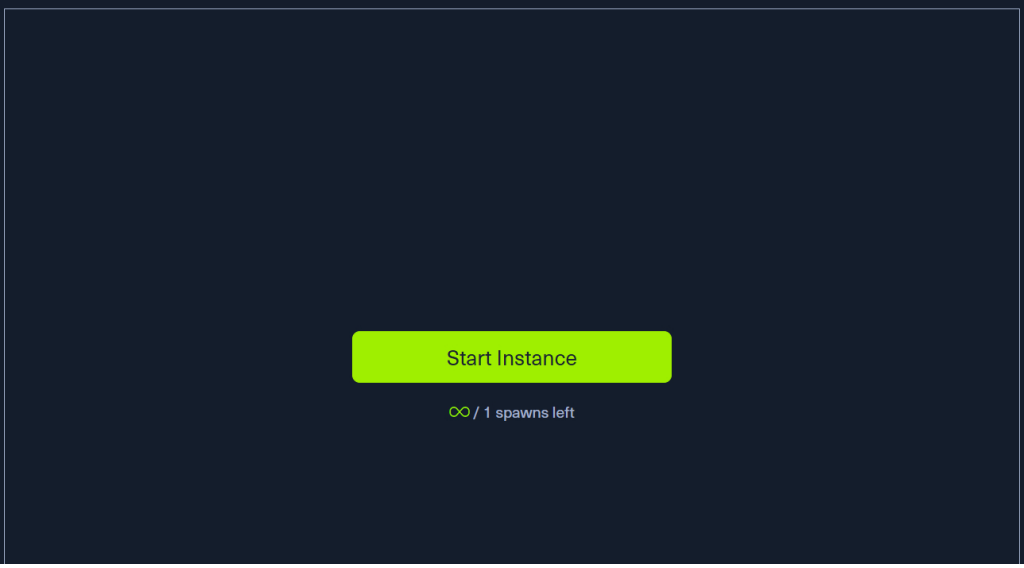
Pre-authentication types, ticket options and failure codes are defined in RFC 4120.

If the ticket was malformed or damaged during transit and could not be decrypted, then many fields in this event might not be present.

- 4776







Waiting to start...

Enable step-by-step solutions for all questions

Questions

Answer the question(s) below to complete this Section and earn cubes!

Cheat Sheet

Download VPN Connection File

Target(s): Click here to spawn the target system!

RDP to with user "bob" and password "Slavi123"

+ 1 Connect to the target and enumerate the available network shares. What is the password of the Administrator2 user?

Slavi920

Submit

Previous Next

Mark Complete & Next