# Apache mod_python for red teams

LRQA's red team engagements are typically designed to be as highly targeted and as stealthy as possible. For the command and control (C2) infrastructure, this means layering several techniques.

- We hide all of our C2 infrastructure behind a number of Apache web servers
- Any traffic to the C2 is checked against an IP whitelist
- IP addresses that do not match the whitelist are directed to a legitimate site
- IP addresses matching the white list are filtered, any traffic that does not match our C2 traffic will be directed to the legitimate site
- Any traffic that does not match those rules is directed to the C2 infrastructure

We have previously used Apache rewrite rules (mod_rewrite) to accomplish all of this, and it works very well to accomplish this goal. However, if you need to do anything more advanced than regular expression matching, rewrite rules are not applicable.
I've recently been looking into adding malleable-comms to our private, in-house red team implant, that is; comms which can be changed to look like something else such as Zeus Bot traffic, in a similar way to Cobalt Strike. While doing this, I was playing around with mod_python and it occurred to me that we could replace our mod_rewrite rules with mod_python in order to make them more manageable. This post will give an overview of how that can be achieved.
Some knowledge of Apache and Python is assumed here; you must be able to set up a basic virtual host and understand how to write Python scripts.

## Enable Apache Python module

The Python module needs to be enabled on the server, on an Ubuntu installation. The following two commands, executed as root, will achieve this:

```
a2enmod python
systemctl restart apache2.service
```

## Example virtual host setup

For this example there are three virtual hosts; one internet facing which will run the Python code and redirect users to the correct location, a legitimate site and a C2 site. This example should be configured to match your own C2 setup.

### Legitimate site

A legitimate looking site that non-whitelisted users will be directed to, running on port 8000.

```
#
# Legitimate site
#
Listen 8000
<VirtualHost 127.0.0.1:8000>
    DocumentRoot /var/www/legitimate
</VirtualHost>
```

A test index.html page can be added to the legitimate site directory, for testing purposes

```
<html>
<body>
Legit Site
</body>
</html>
```

### Command and control

For illustration purposes only, here's a command and control site, running on port 9000. In real life, you would redirect traffic to your real C2 infrastructure.

```
#
# Command and control
# This is for illustration purposes
# In real life, you would direct to your external C2 server
#
Listen 9000
<VirtualHost 127.0.0.1:9000>
    DocumentRoot /var/www/commandcontrol
</VirtualHost>
```

A test index.html containing the following can be added:

```
<html>
<body>
Command and Control Site
</body>
</html>
```

### Internet facing virtual host

Finally, an internet facing host, which is bound to port 80. In real life you would want to also run this on port 443 with a valid SSL certificate appropriate to the engagement.

```
#
# Site that will screen visitors and redirect appropriately
#
Listen 80
<VirtualHost *:80>
    PythonPostReadRequestHandler /var/www/python/handler.py
    PythonDebug On
</VirtualHost>
```

In order to be able to redirect users, we have hooked the `PostReadRequestHandler` using the `PythonPostReadRequestHandler` directive. All requests will be directed through this script and can then be redirected appropriately.
Python debug is also turned on using the `PythonDebug` directive, to allow us to easily see any errors in our scripts.

### The handler script

For the handler, it is possible to write a Python class. The first class defined in the file will be instantiated. A class is overkill for this example, so instead a single function will suffice, which should be called `postreadrequesthandler`.
In this example, we will redirect curl clients to the C2 server and anyone else to the legitimate site. Log entries will be placed into the Apache error.log file.
The documentation for mod_python is quite sparse, so in order to work out how to perform a redirect correctly, I had to refer to both the mod_proxy and mod_rewrite source code. After some experimentation I found the correct method:

```
#
#/var/www/python/handler.py
#
import logging
from mod_python import apache
def postreadrequesthandler(context):
    #setup redirect
    context.handler = "proxy-server"
    context.proxyreq = apache.PROXYREQ_REVERSE
    useragent = context.headers_in.get("user-agent", None)
    if useragent is not None and useragent.startswith("curl"):
        context.connection.log_error("Redirecting client to c2 site" ,
apache.APLOG_ERR)
        context.filename = ("proxy:http://localhost:9000/%s" %
context.unparsed_uri)
    else:
        context.connection.log_error("Redirecting client to legitimate site"
, apache.APLOG_ERR)
        context.filename = ("proxy:http://localhost:8000/%s" %
context.unparsed_uri)
    return apache.OK
```

Once all of this is in place, after restarting the apache2 service you should get the following output from curl:

```
$ curl localhost:80

<html>
<body>
Command And control Site
</body>
</html>
```

Changing the curl user agent will send you to the legitimate site:

```
$ curl localhost:80 --user-agent "hello"

<html>
<body>
Legit Site
</body>
</html>
```

## Log Output

The Apache error log should show some useful output, thanks to mod_python.

```
[client 127.0.0.1:37420] Host : localhost
[client 127.0.0.1:37420] User-Agent : curl/7.47.0
[client 127.0.0.1:37420] Accept : */*
[client 127.0.0.1:37420] Redirecting client to c2 site
[client 127.0.0.1:37424] Host : localhost
[client 127.0.0.1:37424] User-Agent : hello
[client 127.0.0.1:37424] Accept : */*
[client 127.0.0.1:37424] Redirecting client to legitimate site
```

## Further Reading

Full documentation is available for mod_python here:
http://modpython.org/live/current/doc-html/contents.html