VetSec Takes First in the Hacktober CTF: Summary & Steganography Write-up!

By **m4v3r1ck** in **CTF Write-ups**, **Steganography** on **October 18, 2018**

No comments

For the last week, VetSec competed in the Hacktober.org CTF event, which consisted of challenges in forensics, steganography, programming, offensive tactics, web application, reverse engineering, cryptography, and more. I am happy to announce that WE WON!

Our team consisted of the following members:

- Elliot Chernofsky (@emtuls)
- Reuben Booker (@reubadoob)
- Rob Fuller (@mubix)
- Myself (@hmaverickadams)

The competition itself was close up until the very end. We competed against well over 100 teams and the challenges ranged from pretty beginner stuff through some down right clever hidden flags. Shout out to Midwest Cyber Center for putting on the event and to the mods (@overallcoma1 & @attackd0gz) for staying on top of everything. They were the true MVPs.

In spirit of the win, I wanted to write up the entire Steganography section, which is my favorite CTF category. Other write-ups are in the works on some of the other challenges, so stay posted for those. Without further stalling, let's take a look at some stego!

Spooky Pumpkins – 100 Points

Challenge description:

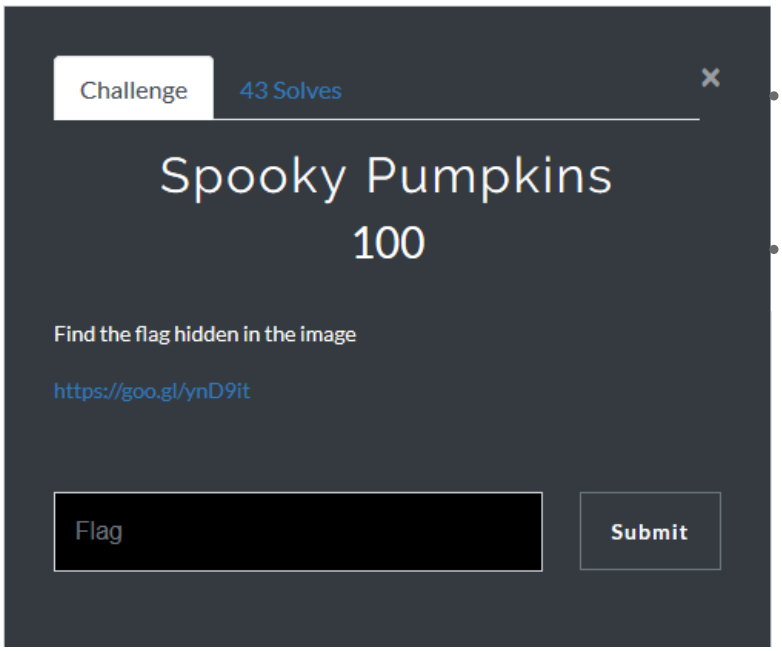


Image:

Solution:

So, one of the first things I like to try on stego challenges is using the *strings* command to view printable strings in a file. My go to is ***strings -a -n 7 <file>*** which shows all strings with a length of seven or more characters. Running this on our image produces the following flag and a quick win:

```
Mun`~2cc
flag-SpookyPumpkinIsSpooky=
RU%JJ/[
```

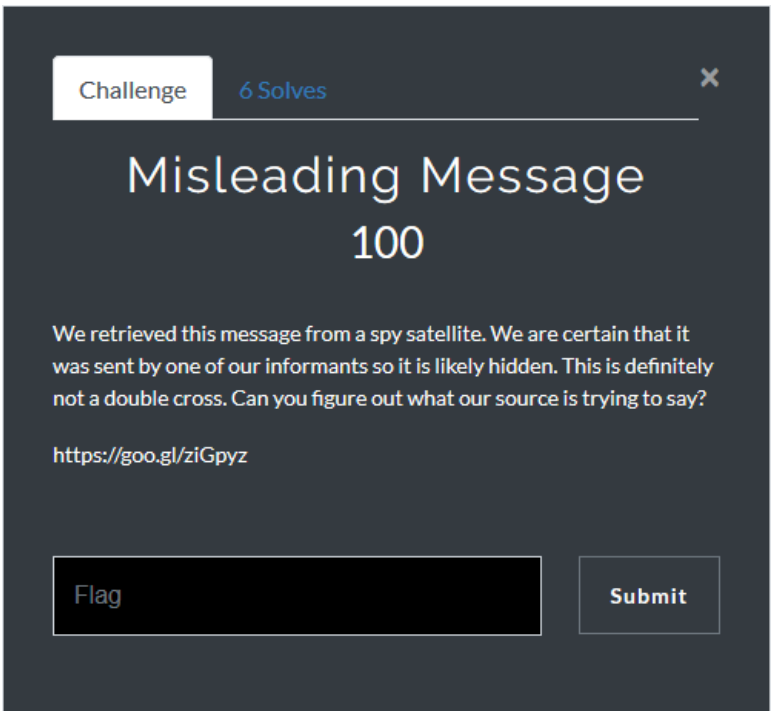

- **m4v3r1ck**
- VetSec Takes First in the Hacktober CTF: Summary & Steganography Write-up!
- Hack The Box – DevOops Walkthrough
- Hack The Box – Fighter Walkthrough
- Hack The Box – Sunday Walkthrough
- Not Your Ordinary OSCP Review

- **reubadoob**
- A Year Ago My Life Changed, From Soldier to Cyber

- RSS - Posts

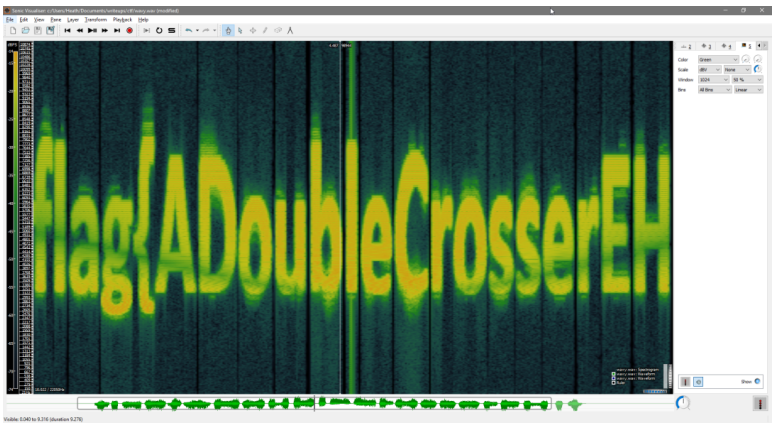
Misleading Message – 100 Points

Challenge description:



Solution:

In this challenge, we are presented with a wav file. Usually, on easy audio challenges, I like to go right to Sonic Visualizer to look for flags in the audio. The results don't disappoint as we immediately find a flag:



Except, when I went to submit the flag, it didn't work. Misleading message. Got it. So, in this instance, there are a few things we can do:

1. Look at the *strings* of the file. Nothing interesting.
2. Use *binwalk* to see if a file is hidden inside. No hidden files here.

3. Try *xxd* to look at the hex of the file for hidden messages. Nada.

So all of those tricks failed, but there's something else we can look for in wav stego: least significant bits (LSB). LSB is often used to hide messages in audio or pictures. I used WavSteg.py (https://github.com/ragibson/Steganography) to check for this:

```
root@kali:~/Downloads# python3 Steganography/WavSteg.py -r -s wavy.wav -o output
.txt -n 1 -b 1000
Reading files...                Done in 0.00 s
Recovering 1000 bytes...        Done in 0.00 s
Writing to output file...       Done in 0.00 s
```

With the output file producing our flag:

```
root@kali:~/Downloads# cat output.txt
flag-NoNoNoClearlyAWiseGuy?c???8???3??Is?R?
&mR??
?????3ehF)?
```

Information Leak – 100 Points

Challenge description:

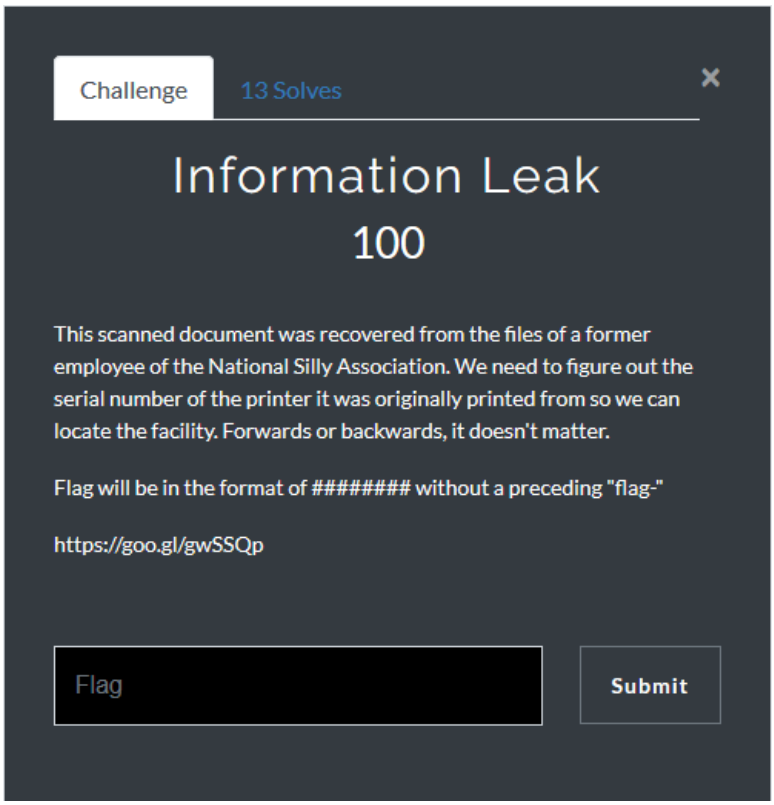
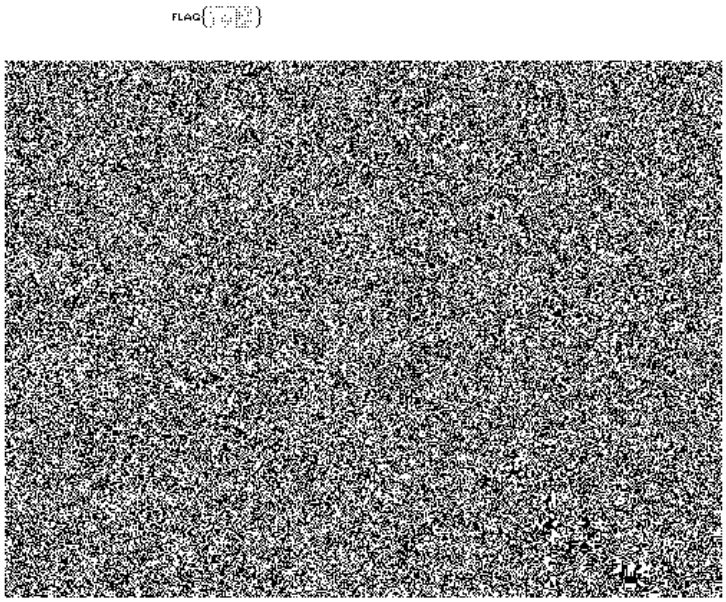


Image:

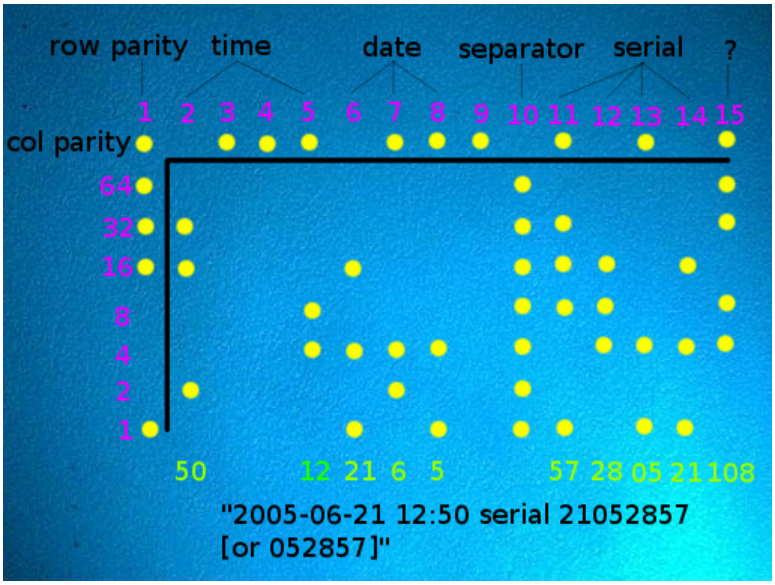
Solution:

The solution to this one was a bit tedious. First, I used one of my favorite steganography tools: StegSolve (https://github.com/eugenekolo/sec-tools/tree/master/stego/stegsolve/stegsolve), which allows me to toggle the color pane on an image and often find hidden flags very easily. By doing a little toggling, I find the flag:



It's hard to see, but if we blow it up, it looks something like this:

My initial thoughts were that this looks like braille. However, I've never seen braille that is eight digits tall. I reread the challenge and realized that it needed a printer serial number. This lead me down the path of printer steganography, which is where printers print secret dots on paper to identify information about it. Here is an example of what the code translates out to:



If you look closely, it's pretty similar. All we need to do to find the flag is add up the bits from columns 11-14. The answer is 57197153 or 53711957. Either work.

Spookier Kitty – 150 Points

Challenge description:

Image:

Solution:

The solution here is identical to the first challenge, Spooky Pumpkins. We can use strings to retrieve our flag:

```
root@kali:~/Downloads/writeup# strings -a -n 7 2.jpg
flag-TerrifyingKitty
```

Misleading Pumpkins – 200 Points

Challenge description:

Challenge 27 Solves

Misleading Pumpkins

200

Find the flag hidden in the image!

(No, it isn't that easy)

https://goo.gl/eJpzkL

Flag

Submit

Image:

Solution:

Using strings, I was able to see information for a file called “secrets.png”. I decided to take a closer look with *binwalk* to see if there was indeed a hidden file in our image:

```
root@kali:~/Downloads/writeup# binwalk 2.png

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 1920 x 1080, 8-bit/color RGB, non-inter
laced
41            0x29            Zlib compressed data, default compression
2075435       0x1FAB2B        Zip archive data, at least v2.0 to extract, compre
ssed size: 2076487, uncompressed size: 2076487, name: secrets.png
4152020       0x3F5AD4        End of Zip archive, footer length: 22
```

So, there is definitely data there. Let’s extract it and see what we have:

```
root@kali:~/Downloads/writeup# binwalk -e 2.png

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 1920 x 1080, 8-bit/color RGB, non-inter
laced
41            0x29            Zlib compressed data, default compression
2075435       0x1FAB2B        Zip archive data, at least v2.0 to extract, compre
ssed size: 2076487, uncompressed size: 2076487, name: secrets.png
4152020       0x3F5AD4        End of Zip archive, footer length: 22

root@kali:~/Downloads/writeup# cd _2.png.extracted/
root@kali:~/Downloads/writeup/_2.png.extracted# ls
1FAB2B.zip  29  29.zlib  secrets.png
```

And secrets.png with our flag:

Jack – 250 Points

Challenge description:

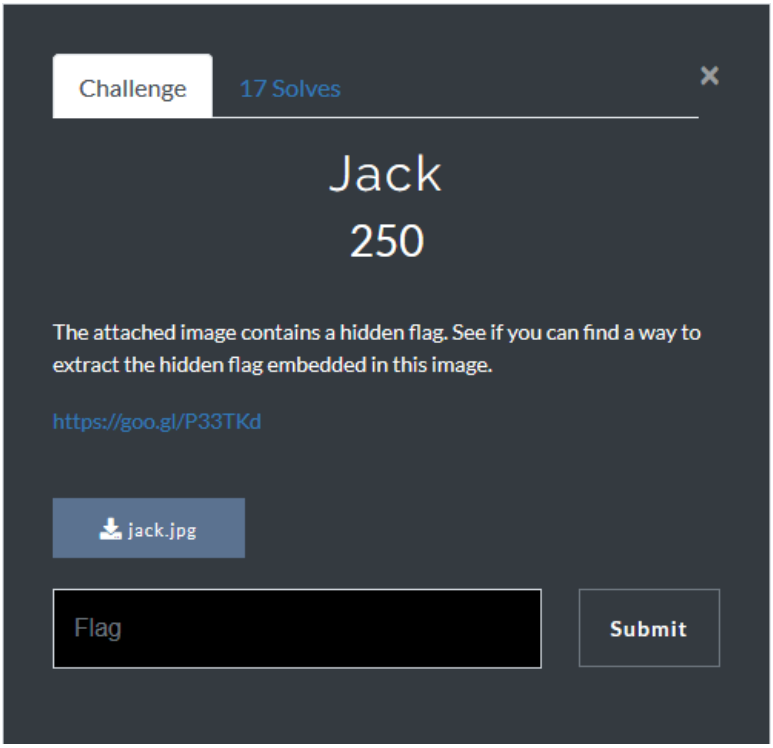


Image:

Solution:

Similar to the previous challenge (in fact identical), we can use *binwalk* to extract data from the picture. In this instance, it was a RAR file containing a file called “file.png”. Here’s a quick picture of the process:

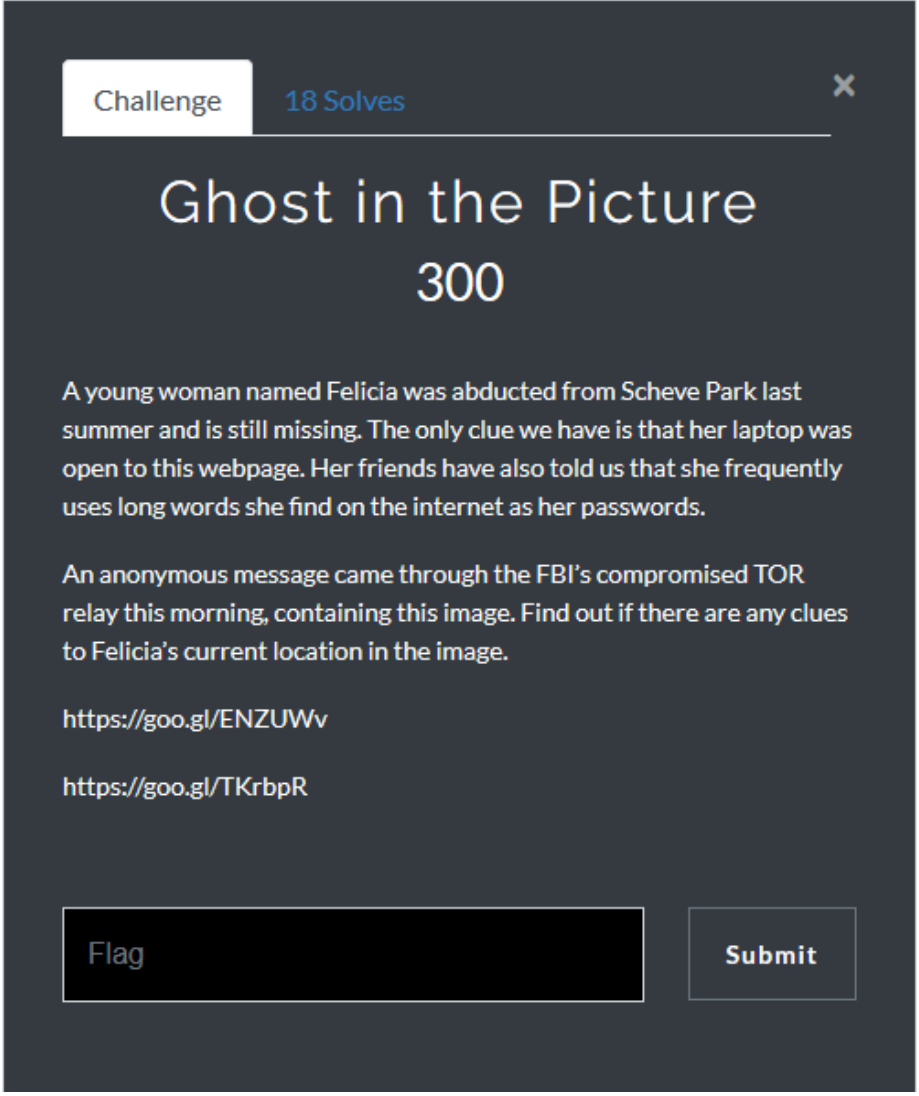
```
root@kali:~/Downloads/writeup# binwalk 3.jpg

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             JPEG image data, JFIF standard 1.01
104356        0x197A4         RAR archive data, version 5.x

root@kali:~/Downloads/writeup# binwalk -e 3.jpg

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             JPEG image data, JFIF standard 1.01
104356        0x197A4         RAR archive data, version 5.x

root@kali:~/Downloads/writeup# cd _3.jpg.extracted/
root@kali:~/Downloads/writeup/_3.jpg.extracted# ls
197A4.rar  file.png
```

Here’s the flag file:

flag-spookyglot

Ghost in the Picture – 300 Points

Challenge description:



Website & Image:

https://www.atlasobscura.com/lists/abandoned-psychiatric-hospitals

Solution:

Up until this point, the challenges have been pretty basic. The last few will start to get increasingly more complex. This challenge was not terribly difficult, but it did require making a wordlist and writing simple bash loops to crack a password. We're told in the challenge description that Felicia liked to use long passwords. To me, this reads that we need to generate a wordlist from the provided website and use that to unhide information in the picture that is likely hidden with a tool called StegHide (http://steghide.sourceforge.net/).

First things first, we need a word list. We can use a built-in Kali tool called CeWL to generate a wordlist from a website. The syntax looks as such: ***cewl -w list.txt -d 1 -m 9 <site>*** where d is for depth and m is for the minimum length of words to grab. In this case, I chose a depth of one and a length of nine as I would imagine a "long password" would not be shorter than this and the less words we use, the better. The list generates as such:

```
CeWL 5.4.3 (Arkanoid) Robin Wood (robin@digi.ninja) (https://digi.ninja/)
Washington
Destinations
rachelrummel
Philadelphia
Mysterious
Architecture
technology
Interactive
AtlasObscura
newsletters
subscribing
California
Newsletter
newsletter
Adventurous
Celebrations
Photography
Scientists
Prosfygika
Disappeared
Birthplace
```

There are ~3,500 words in the list. We can then write a simple for loop in bash to try and crack the password with StegHide. My loop looked like this: ***for i in $(cat list.txt); do steghide extract -sf scary.jpg -p $i; done***

It's an incredibly basic loop and thus, will not stop until it reaches the end. However, at some point, we can see that it wrote out a file:

```
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
wrote extracted data to "help".
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
steghide: could not extract any data with that passphrase!
```

We can *cat* the "help" file and find our flag:

```
root@kali:~/Downloads/writeup# ls
1.jpg  2.jpg  2.png  _2.png.extracted  3.jpg  _3.jpg.extracted  help  list.txt  scary.jpg  suspicion.png
root@kali:~/Downloads/writeup# cat help
please help me get out of this hospital! the secret word is flag-1nsaneasylum
```

Who's a Good Dog? – 300 Points

Challenge description:

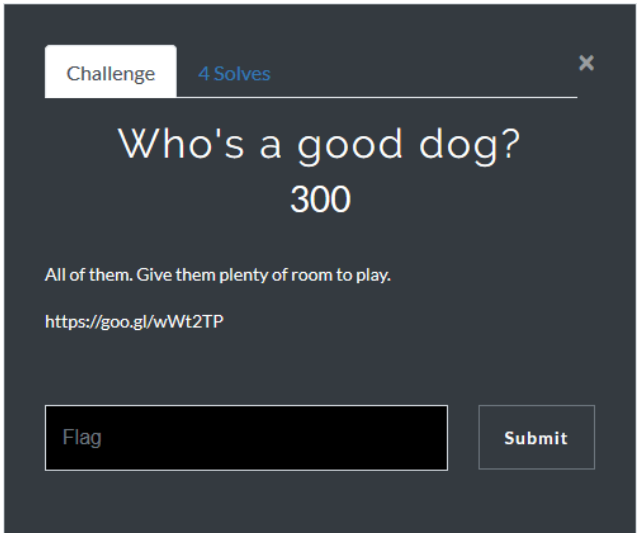


Image:

Solution:

The solution to this challenge was a bit on the trickier side and one you don't see much in stego challenges. All of the typical tools (e.g. strings, binwalk, stegsolve, etc.) did not produce any results. Reading the challenge again, I realized there's an obvious hint given: *give them plenty of room to play*. To me, this reads that the image has been shortened and is hiding the flag. In order to change the readable size of the image, I used a hex editor to modify the height and width information as it is read by the system. Here is a discussion thread talking about the location of these bytes: https://www.experts-exchange.com/questions/11416918/How-to-Read-JPG-Height-and-Width-from-Binary-Hex-data.html

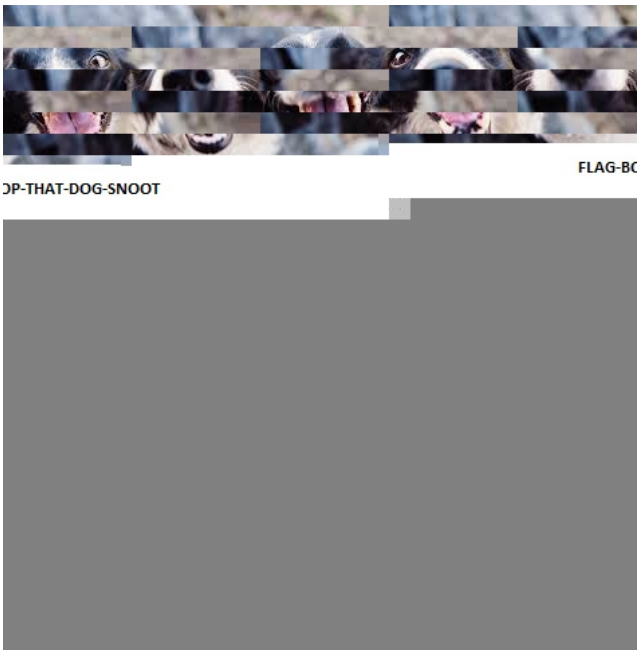
Essentially, we need to find the series of hex that reads FF C0 00 11 08. The next four bytes of hex after this information is the height and width. The original looked like this:

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	FF	D8	FF	E0	00	10	4A	46	49	46	00	01	01	01	00	60	ÿØÿà..JFIF.....`
00000010	00	60	00	00	FF	E1	00	22	45	78	69	66	00	00	4D	4D	.`..ÿá."Exif..MM
00000020	00	2A	00	00	00	08	00	01	01	12	00	03	00	00	00	01	.*..............
00000030	00	01	00	00	00	00	00	00	FF	DB	00	43	00	02	01	01	........ÿÛ.C....
00000040	02	01	01	02	02	02	02	02	02	02	02	03	05	03	03	03	....
00000050	03	03	06	04	04	03	05	07	06	07	07	07	06	07	07	08	....
00000060	09	0B	09	08	08	0A	08	07	07	0A	0D	0A	0A	0B	0C	0C	....
00000070	0C	0C	07	09	0E	0F	0D	0C	0E	0B	0C	0C	0C	FF	DB	00	............ÿÛ.
00000080	43	01	02	02	02	03	03	03	06	03	03	06	0C	08	07	08	C...............
00000090	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	....
000000A0	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	....
000000B0	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	....
000000C0	0C	0C	FF	C0	00	11	08	00	B4	01	13	03	01	22	00	02	..ÿÀ.....´....."..
000000D0	11	01	03	11	01	FF	C4	00	1F	00	00	01	05	01	01	01	.....ÿÄ.........
000000E0	01	01	01	00	00	00	00	00	00	00	00	01	02	03	04	05	....

Where 00 B4 01 13 is our height and width. I modified this and random as such:

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	FF	D8	FF	E0	00	10	4A	46	49	46	00	01	01	01	00	60	ÿØÿà..JFIF.....`
00000010	00	60	00	00	FF	E1	00	22	45	78	69	66	00	00	4D	4D	.`..ÿá."Exif..MM
00000020	00	2A	00	00	00	08	00	01	01	12	00	03	00	00	00	01	.*..............
00000030	00	01	00	00	00	00	00	00	FF	DB	00	43	00	02	01	01	........ÿÛ.C....
00000040	02	01	01	02	02	02	02	02	02	02	02	03	05	03	03	03	....
00000050	03	03	06	04	04	03	05	07	06	07	07	07	06	07	07	08	....
00000060	09	0B	09	08	08	0A	08	07	07	0A	0D	0A	0A	0B	0C	0C	....
00000070	0C	0C	07	09	0E	0F	0D	0C	0E	0B	0C	0C	0C	FF	DB	00	............ÿÛ.
00000080	43	01	02	02	02	03	03	03	06	03	03	06	0C	08	07	08	C...............
00000090	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	....
000000A0	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	....
000000B0	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	0C	....
000000C0	0C	0C	FF	C0	00	11	08	01	E1	01	D9	03	01	22	00	02	..ÿÀ....á.Ù.."..
000000D0	11	01	03	11	01	FF	C4	00	1F	00	00	01	05	01	01	01	.....ÿÄ.........
000000E0	01	01	01	00	00	00	00	00	00	00	00	01	02	03	04	05	....

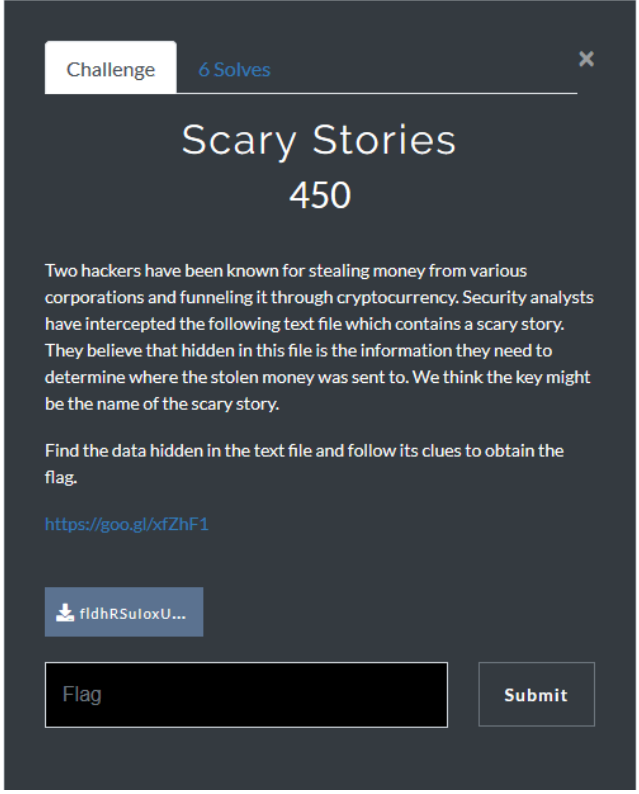
Which results in the following image (I could have tweaked it better, but you get the point):



As you can see, the flag was hidden underneath all along. This was a creative challenge. Hats off to the creator!

Scary Stories – 450 Points

Challenge description:



Text:

```
Last Halloween, I was waiting at my front do

I was just about to turn my porch light off

There were two children standing on my porch

Confused, I asked them, “What are you suppos

“No”, said the children as their mouths wide

Filled suddenly with a chilling sense of imp

When I looked out through the peephole, they

I don’t know about you, but this Halloween,
```

Solution:

This challenge was by far the most difficult for the team. The challenge actually functions in two halves. The first half involves extracting data from the text provided, which was not that difficult. The data extracted was a riddle and honestly, an incredibly clever one at that. Let’s walk through how we solved this one.

The first thing we noticed when we opened the text document was that there was a lot of white space. There is a form of steganography where you can use whitespace to hide information. A tool called Snow can be used to extract this information with the correct password. Now, the challenge hints that the

password is the name of the scary story provided. A quick Google search of the story produces its name: Halloween Candy

We can now extract information from the text file:

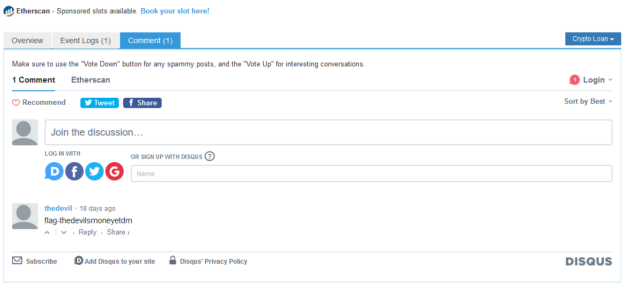
```
root@kali:~/Downloads/writeup# stegsnow -C -p 'Halloween Candy' 1.txt
I hid the flag in Devil Money.  I sent over about 10,000,000 tokens haha.
```

So, we receive the above riddle. This took us a very long time to figure out and many hours of Googling. Here's the thought process that lead to the solving of the riddle:

The initial challenge description talks about stolen money being funneled through cryptocurrency. The hidden statement then mentions that 10,000,000 tokens were sent over. However, what the hell is Devil Money? Is it a place? No, it's actually a cryptocurrency. Check it out: https://etherscan.io/token/0xba45eebaf4f5b4d70d1cbb8cba87f7de1271c897?a=0x463368d255a69a804a1ff9f666d1a165ac5911b3

Now all we needed to do was dig through 100+ pages of transactions and look closely at those involving a transaction of 10,000,000 or slightly more. That's when we found this page involving a recent transaction of exactly 10,000,000 Devil Money tokens: https://etherscan.io/tx/0xcd763f26de14151d053a044e60fa267cc125c7ee6685ef7f6

When you look at the comments on the page, you find the flag:



Pretty clever. We really enjoyed this challenge!

Conclusion:

The team at VetSec really enjoyed the challenges and look forward to sending a new squad forward next year to defend the crown. It is our first win as a group, but hopefully not our last. Thanks again to Midwest Cyber Center and the admins that helped make the event happen! We came away with increased technical knowledge, some cool prizes, and made some great connections in the cyber community.

Wanna chat? Add me on Twitter or LinkedIn!
Veteran? Join our Slack!