

CTF Forensics Write-Up

Fought the Law

20 Points

Problem:

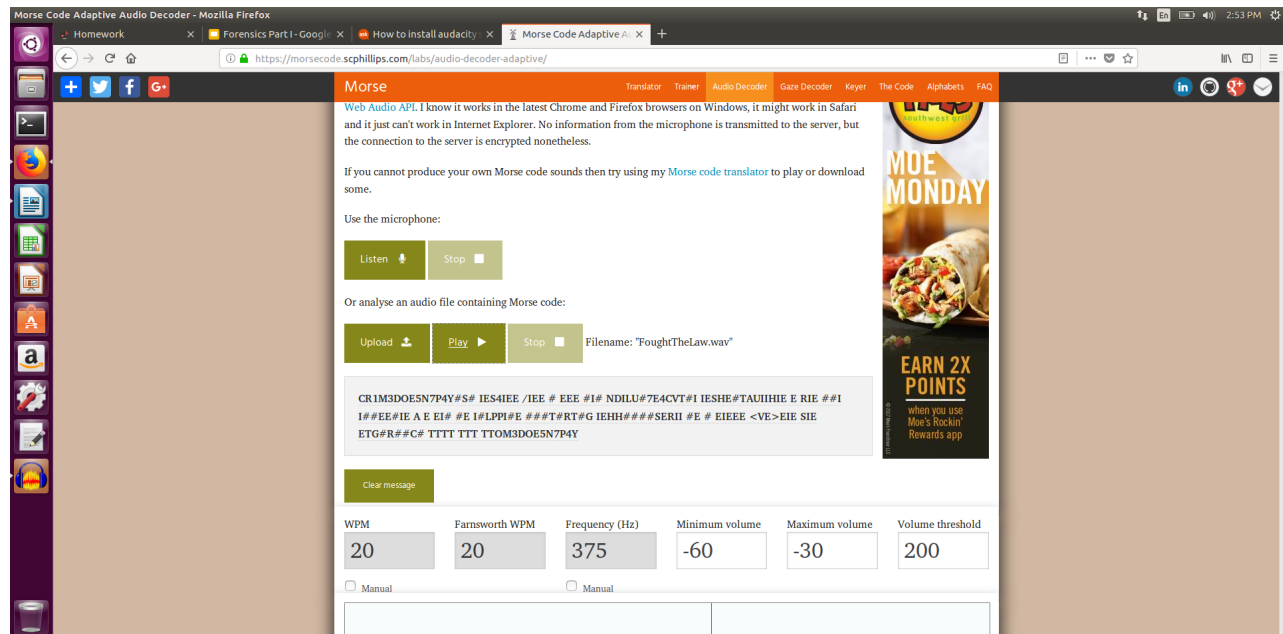
Some folk may tell you that crime pays. Listen to these guys and tell us what you think.

Note: when you find the flag it will not have flag{ } around it. Please add this when checking your work on the CTF site.

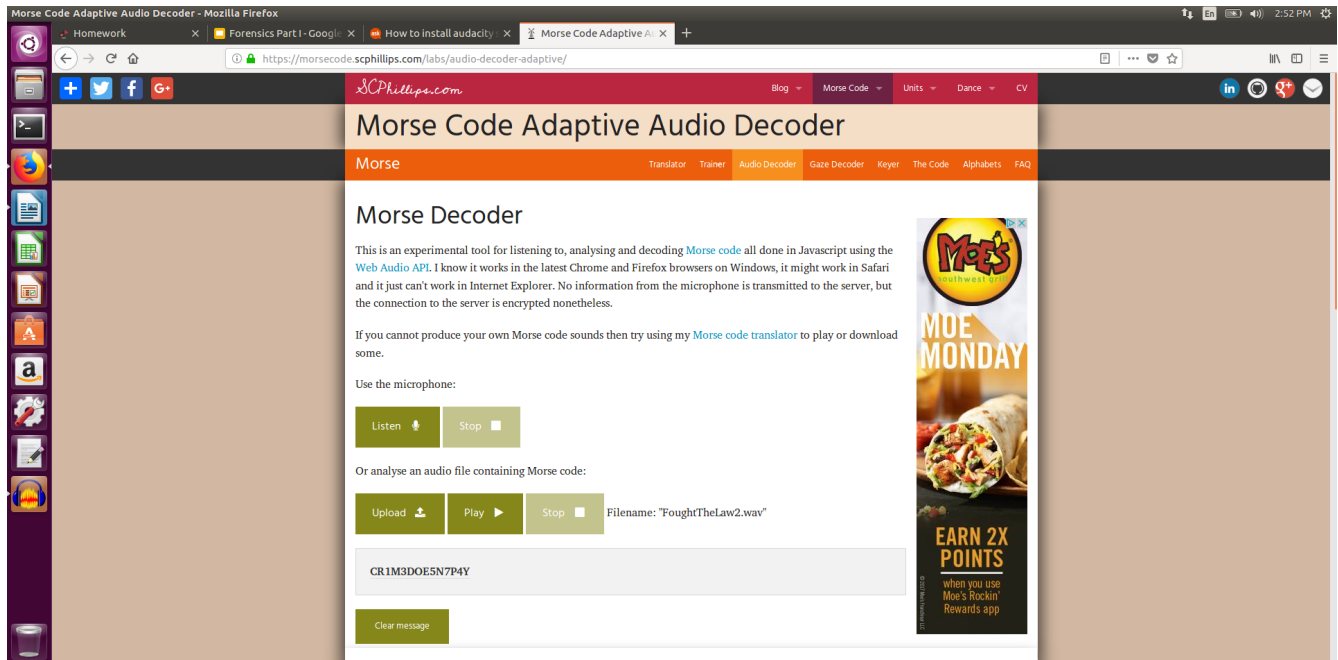
You can download the file here: https://drive.google.com/open?id=1HKGetXOZQ_OGOiWZEvP30lQYgQDE8ACC

Solution:

Upon inspection of the file, I see that it is a .wav audio file. This audio file opens properly. The first half of the file is the song “I Fought the Law”, and the second half of the audio file is a series of high and low pitches. Hearing the two different audio frequencies quickly alternating made me think that the flag was Morse code. I saved the audio file and threw it into an audio Morse code decoder found at <https://morsecode.scphillips.com/labs/audio-decoder-adaptive/>. This gave me the flag:



I tried entering “TTOM3DOE5N7P4Y” as the flag because it looks like a valid flag. However, it wasn’t the flag. Because the Morse audio decoder relies on two varying audio frequencies to decode the message, and the beginning half of the file contains multiple audio frequencies in the song, the decoder was being thrown off. To fix this I threw the audio file in Audacity, cut the audio file down to just the Morse code and ran it through the decoder again. This gave me the correct flag.



flag: flag{CR1M3DOE5N7P4Y}

RTFPPTX

20 Points

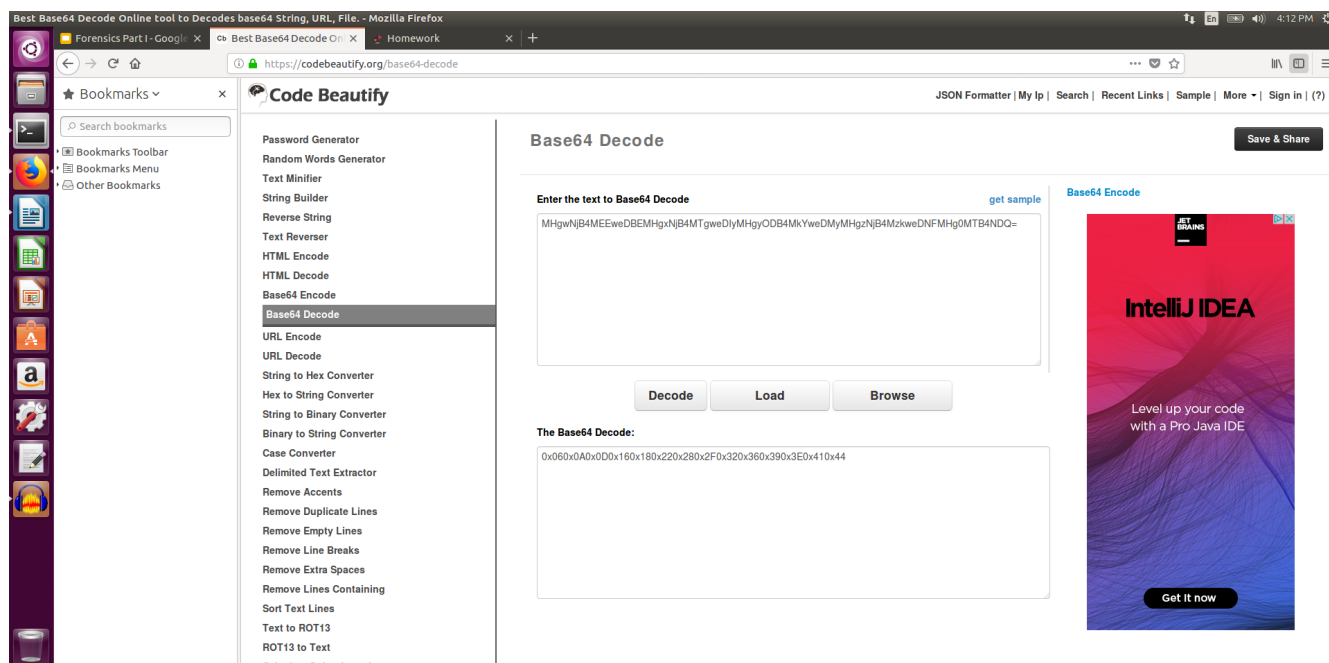
Problem:

How do you make sure students read the lecture materials? Put a flag in them. Take a look, it's in a powerpoint, follow the...

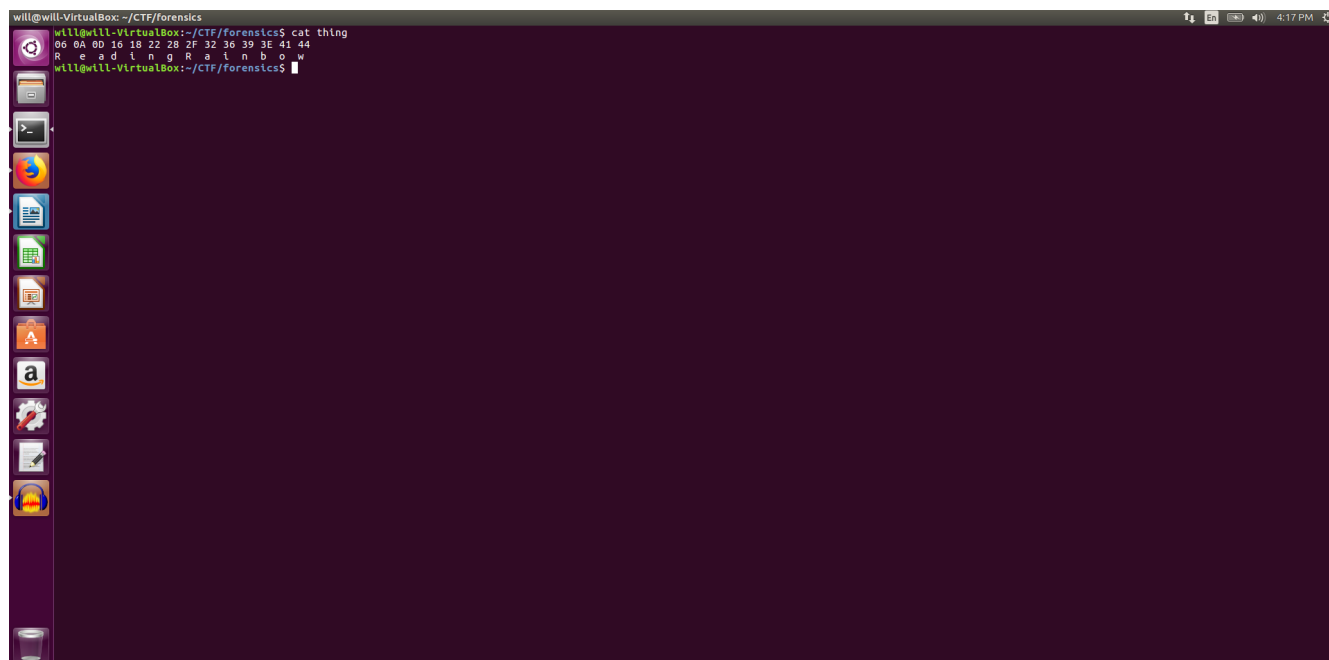
Note: when you find the flag it will not have flag{} around it. Please add this when checking your work on the CTF site.

Solution:

I started by inspecting the slides. First thing I notice is a string of ASCII characters on the first slide. Since there was an '=' at the end of the string, I inferred it was base64 encoded. After running the string through a base64 decoder I get a series of hexadecimal values.



After further inspecting the slides, I noticed that ‘R’ on page six was underlined. I tried XOR decoding the string with ‘R’ as they key but this didn’t work either. Upon further inspecting the slides I noticed it was more than just ‘R’ highlighted in the slides. There were multiple characters highlighted. This led me to believe the flag was being spelled out in the slides. Since it was hard to find all the underlined characters ,I began looking for a key to find all of the characters in the flag. Upon realizing the first underlined character was found on page six and the first base64 decoded hex value was also six, I realized the hex values corresponded to page numbers for the flag characters. After cross referencing the page numbers with the hex values I got the flag.



Flag = flag{ReadingRainbow}

BrokenFlag

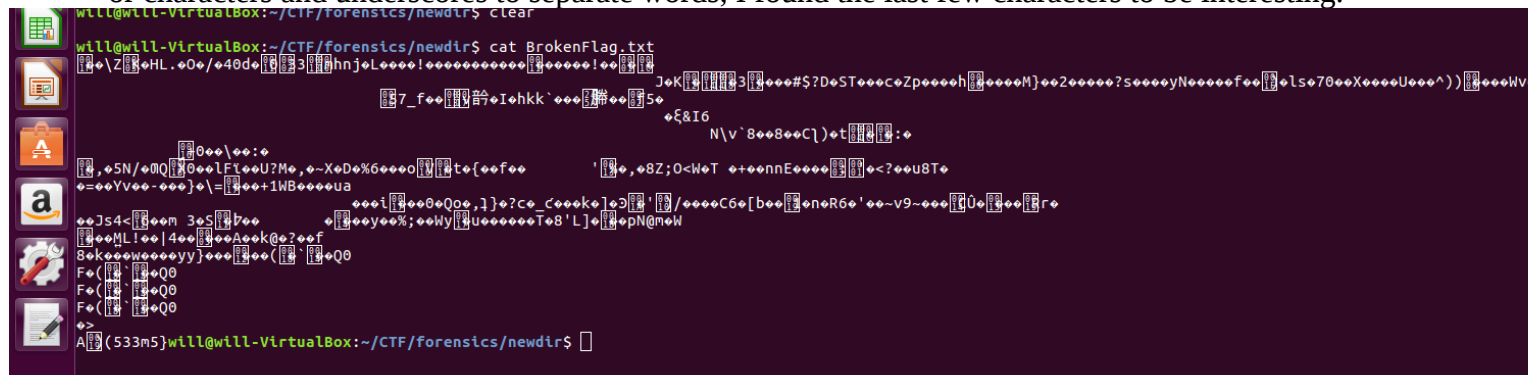
25 Points

Problem:

Someone broke our flag and spilled the pieces into a text file! Help us put it back together!

Solution:

First I opened the text file and didn't see anything interesting inside the source code or text document. This made me want to test the file type. I used 'wget' to download BrokenFlag.txt. After using 'file' on BrokenFlag.txt I noticed that the file was not a text file. It was in fact a gzip compressed data. Also, out of curiosity, I used 'cat' to display the gzip'd file's contents. Knowing that the flag contains letters or characters and underscores to separate words, I found the last few characters to be interesting.



```
will@will-VirtualBox:~/CTF/forensics/newdir$ cat BrokenFlag.txt
533m5}
```

533m5} appears to be the end of the flag. This is because all flags are of the form 'flag{ some text }'. This text gives us the closing '}'. I will save this for later. Now, I must unzip the gzip'd file. In order to unzip a file you must add the .gz extension. I used the command 'mv BrokenFlag.txt BrokenFlag.gz' to rename the file and then 'gunzip BrokenFlag.gz' to unzip the file. This gave me a file called BrokenFlag. Upon using 'file' again on the new file, I see that it is a tar archive. In order to decompress the archive I used the command 'tar -xvf BrokenFlag'. This gave me two files called exact_copy01 and exact_copy02. The fact that they are called exact_copy made me think they are not exact copies of one another. To see if they were the same, I used 'diff exact_copy01 exact_copy02' and the terminal told me that "Binary files exact_copy01 and exact_copy02 differ". This led me to use the 'cmp' command to see the difference in the two files. I've always known the 'cmp' command allows for the comparison of files. 'cmp' told me the two files differ on line 2. Hoping for more information, I inspected them man page of 'cmp'. The -b flag allows me to print the differing bytes. This seems like what I want. The -l flag outputs byte numbers and the line of differing bytes. The -l flag adds more information to -b. After executing the command 'cmp -bl exact_copy01 exact_copy02' I noticed the following output.

```
will@will-VirtualBox: ~/CTF/forensics/newdir
will@will-VirtualBox:~/CTF/forensics/newdir$ cmp -bl exact_copy01 exact_copy02
390 137 _ 174 |
395 61 1 143 c
396 61 1 214 M-^L
398 65 5 144 d
404 137 _ 51 )
409 64 4 113 K
411 65 5 44 $
413 137 _ 255 M--
415 61 1 156 n
416 61 1 216 M-^N
417 164 t 366 M-v
423 137 _ 222 M-^R
will@will-VirtualBox:~/CTF/forensics/newdir$
```

Since a flag consists of letters and numbers separated by underscores. The left portion looks like it could be part of the flag. We will save ‘_115_45_11t_’ for later as it appears to be the middle portion of the flag. After using ‘file’ on exact_copy01 and exact_copy02 I noticed that the copies were bzip2 compressed data. In order to decompress bzip2 files they have to have the extension .bz2. So I renamed exact_copy01 and exact_copy02 to exact_copy01.bz2 and exact_copy02.bz2. There I used the ‘bzip2 -d’ command to decompress them. However exact_copy01.bz2 is corrupted, but exact_copy02.bz2 decompressed successfully. The new decompressed file is called exact_copy02. Upon using ‘cat’ on exact_copy02 I notice the last line contains ‘flag{n0th1n_’.

```
will@will-VirtualBox: ~/CTF/forensics/newdir
will@will-VirtualBox:~/CTF/forensics/newdir$ cat exact_copy02
junk2000066400017500001750000000034513227175464011073 0ustar shawshawnbzh91AY&SYuu ed! V&N`. @  ed
NNx~Q2; @#) @c Fw, ++Fu / @
+++++flag{n0th1n_will@will-VirtualBox:~/CTF/forensics/newdir$
```

Which must be the first part of the flag because all flags start with 'flag{'. Putting the three found flag pieces together I get flag{n0thin_115_45_11t_533m5} which is correct.

flag{n0thin_115_45_11t_533m5}

Cyberdog Ate The Homework

35 Points

Problem:

A student claims his dog ate his forensics homework and offered us an image of his file system to prove it. Personally, we're skeptical of the story. Investigate and see if you can prove whether or not we're dealing with a good student or a good doggie. Help us put it back together!

Solution:

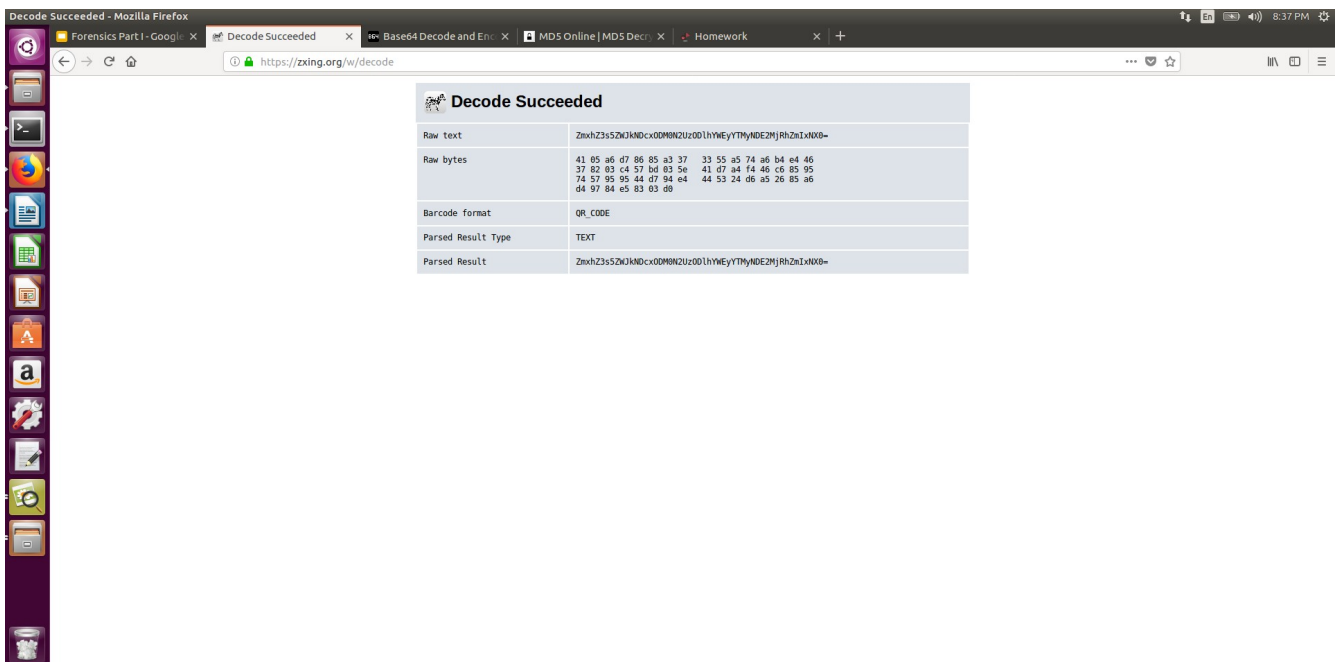
First things first I downloaded the file and installed the sleuthkit. Upon opening the terminal, I used the 'cat' command on the file.img file and that didn't show me anything too interesting. I tried using 'mmls' but no information on the partition was available. After inspecting the notes for more information on commands to get information from the file system, I tried the command 'fls -p -l -r -o 0 file.img' to recover data at offset 0 on the system. This gave me the following output

```
will@will-VirtualBox: ~/Downloads
will@will-VirtualBox:~/Downloads$ fls -p -l -r -o 0 file.img
r/r * 4:      Flag.docx      2018-02-05 02:45:04 (EST)      2018-02-05 00:00:00 (EST)      0000-00-00 00:00:00 (UTC)      2018-02-05 02:45:04 (EST)      0      0
d/d 6:      .Trash-1000    2018-02-05 02:45:18 (EST)      2018-02-05 00:00:00 (EST)      0000-00-00 00:00:00 (UTC)      2018-02-05 02:45:18 (EST)      4096      0
d/d 646:     .Trash-1000/info    2018-02-05 02:45:18 (EST)      2018-02-05 00:00:00 (EST)      0000-00-00 00:00:00 (UTC)      2018-02-05 02:45:18 (EST)      4096      0
r/r 775:     .Trash-1000/info/Flag.docx.trashinfo 2018-02-05 02:45:18 (EST)      2018-02-05 00:00:00 (EST)      0000-00-00 00:00:00 (UTC)      2018-02-05 02:45:18 (EST)
r/r * 778:     .Trash-1000/info/Flag.docx.trashinfo.NM78DZ 2018-02-05 02:45:18 (EST)      2018-02-05 00:00:00 (EST)      0000-00-00 00:00:00 (UTC)      2018-02-05 02:45:18 (EST)
d/d 648:     .Trash-1000/files    2018-02-05 02:45:18 (EST)      2018-02-05 00:00:00 (EST)      0000-00-00 00:00:00 (UTC)      2018-02-05 02:45:18 (EST)      4096      0
r/r 902:     .Trash-1000/files/Flag.docx 2018-02-05 02:42:32 (EST)      2018-02-05 00:00:00 (EST)      0000-00-00 00:00:00 (UTC)      2018-02-05 02:45:04 (EST)
v/v 15968291: $MBR      0000-00-00 00:00:00 (UTC)      0000-00-00 00:00:00 (UTC)      0000-00-00 00:00:00 (UTC)      0000-00-00 00:00:00 (UTC)      512      0
v/v 15968292: $FAT1      0000-00-00 00:00:00 (UTC)      0000-00-00 00:00:00 (UTC)      0000-00-00 00:00:00 (UTC)      0000-00-00 00:00:00 (UTC)      499200      0
v/v 15968293: $FAT2      0000-00-00 00:00:00 (UTC)      0000-00-00 00:00:00 (UTC)      0000-00-00 00:00:00 (UTC)      0000-00-00 00:00:00 (UTC)      499200      0
d/d 15968294: $OrphanFiles 0000-00-00 00:00:00 (UTC)      0000-00-00 00:00:00 (UTC)      0000-00-00 00:00:00 (UTC)      0000-00-00 00:00:00 (UTC)      0      0
will@will-VirtualBox:~/Downloads$
```

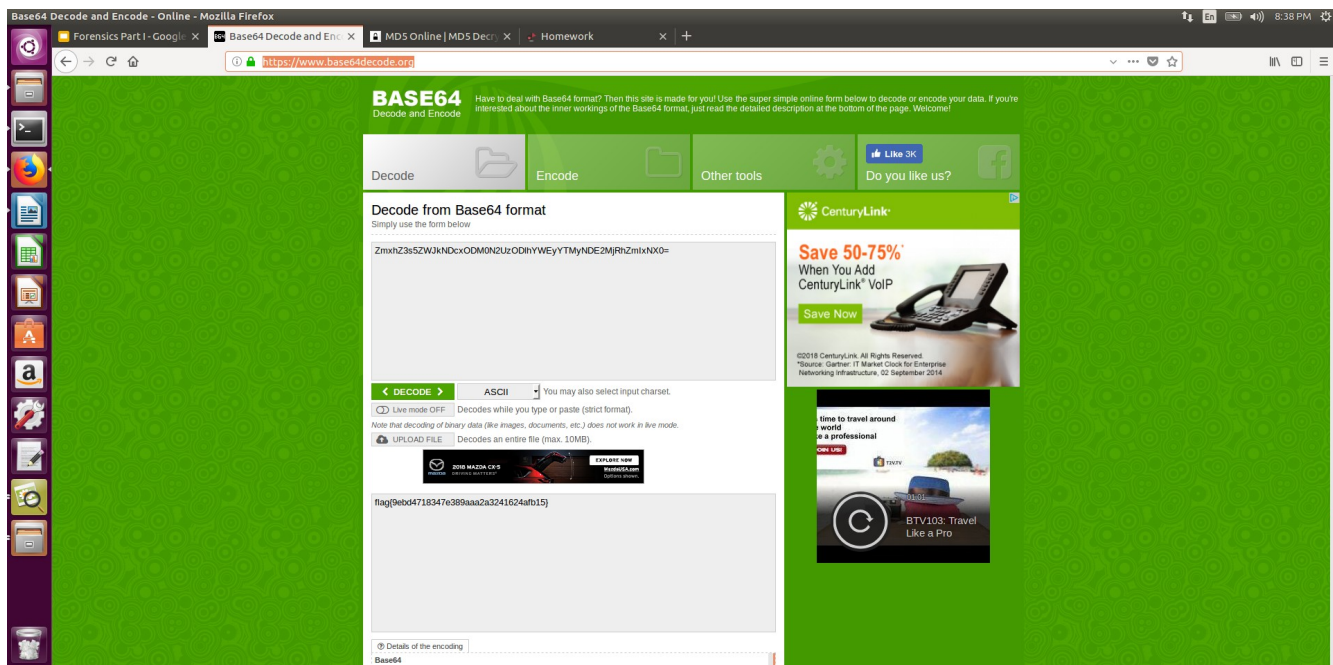
Which made realize there was a file called Flag.docx. I need to find this file. The notes made me think I had to use the tsk_recover command to recover files. However, this wasn't working. After inspecting the man page of tsk_recover, I narrowed my search range on the file recovery to just recover allocated files with the -a flag. After running 'tsk_command -a -o 0 file.img ./recovered' two files were recovered into the recovered folder. Inside the recovered folder nothing was showing. After executing 'ls -l -a' to show hidden files I noticed the folder .Trash-1000 which contained two folders, 'files' and 'info'. Inside files is Flag.docx. After opening the file and not seeing the flag, I knew there was more to Flag.docx then meets the eye. After using 'cat' to no avail, I remembered the example in class where files were hidden inside a Word document. I used 'unzip Flag.docx' to unzip the document. This gave me several folders. Inside the path '/word/media/' I noticed the file Image1.jpg. However, the image wouldn't open. This made me think that it wasn't a jpg after all. After running the 'file' command I noticed it was a 'png' file. I renamed Image1.jpg to Image1.png and copied it to my documents folder with the 'mv' command. I opened the picture through the file browser and saw the following.



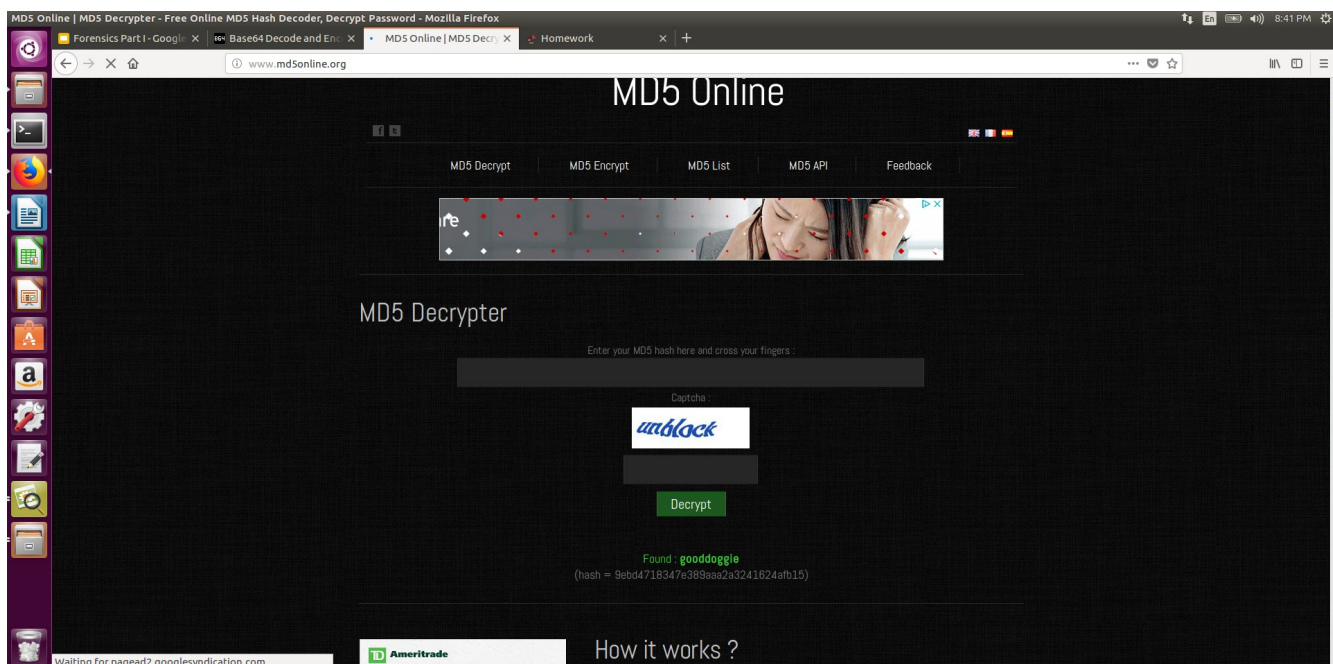
A QR code. I knew the flag had to be hidden inside the QR code because we discussed hiding flags in QR codes in class. I Google searched QR image decoder and used the tool found on <https://zxing.org> to decode the code.



The decode gave me the string 'ZmxhZ3s5ZWJkNDcxODM0N2UzODlhYWYyYTM5NDE2MjRhZmIxNX0='. I noticed the '=' on the end and knew it was base64. After running it through a base64 decoder found at <https://www.base64decode.org/> I got the following:



the string 'flag{9ebd4718347e389aaa2a3241624afb15}' However, this flag did not work. After inspecting the inside of flag{} I noticed there was 32 characters. MD5 Encoding uses 32 alphanumeric characters. Thinking that the code was MD5 encoded, I Google searched an MD5 decoder. I used the decoder found at <http://www.md5online.org/> and searched for the appropriate hash.



I got the string 'gooddoggie'. Upon entering flag{gooddoggie} I succeeded.

Flag = flag{gooddoggie}