# How to make CTF Challenges

Presentation by Thomas and Faisal

## Motivations

- We need people to develop challenges
  - Eventually the people currently making challenges will graduate, then what!?
- Developing challenges strengthens your understanding of
  - how to write secure code.
  - the topic your challenge is about.
  - Vulnerability analysis and other CTF challenges.
- You can solve your own challenge for points!
  - This is allowed on SIGPwny's internal server, this does not apply to public CTFs

## Challenge Development Steps

# Development Steps - Topic

Were going to focus on this side

### If you know the topic well

- Think about a challenge that people who are advanced in the topic can solve, but don't make it too niche.

- Try to avoid stereotypical challenges, more advanced challenges can be unique.

- Check recent CTFs (ctftime) for inspiration, but don't copy!

**Make the Challenges...**
Difficult
Solvable

Not too niche!

### If you are new to the topic

- Search for past CTF challenges, 50-100 points.

- Look at writeups for those challenges

- Try to understand the underlying fundamentals and assumptions that cause the vulnerabilities the other challenges are using.

- Solve some challenges from other CTFs, try to gain an understanding of the challenge fundamentals.

## Hacker Brain Level

This scales in either direction depending on the CTF

**100** — Difficult, but a script could probably solve it already

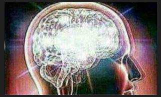
- Single step challenges
- Utilize security fundamentals
- Typically solvable in under an hour with appropriate knowledge (this applies to 200 +)

**200** — Good fundamentals, some googling, and grit will get you through this without issue

- More challenging single step challenges, multi-faceted (two category) challenges as well.
- Utilize general, well known security knowledge
- Solvable in 1-2 hours typically

**300** — You likely need to have some prior knowledge, otherwise be ready for a ton of learning

- Typically involve multiple vulnerabilities or one high difficulty vulnerability to solve.
- Utilizes deeper, more specific features of the category
- Solvable in 2-8 hours typically



**400** — Specific category knowledge required, challenge very difficult otherwise.

- Utilizes complex vulnerability chains
- May utilize newer vulnerabilities, or very deep extremely specific attack vectors
- Solvable in a day typically

**500** — Specialists may have a hard time solving this one

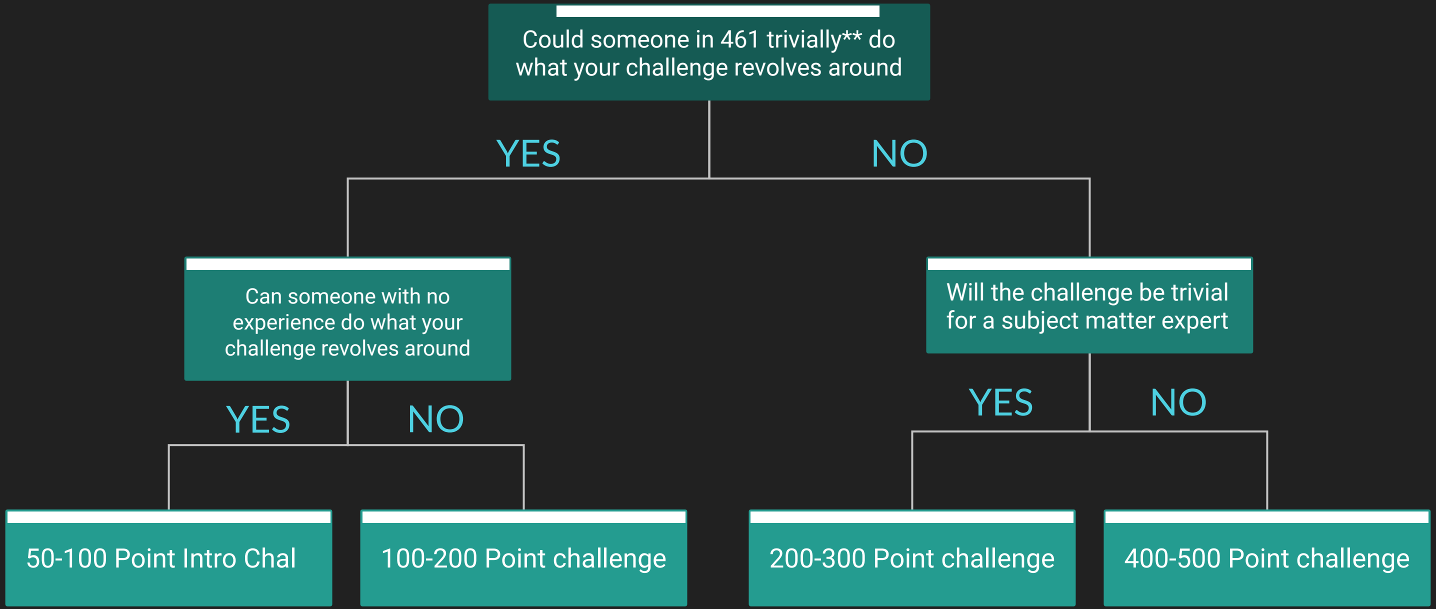
- Extremely challenging, typically multistep challenges (single step 500’s are horrifying)
- Might utilize a 0day
- Often take the whole CTF if alone

**EX** 500 > — S̰ṹm̃mọ́n a hacker god

- Often utilizes unique architectures / specialized environments. May also utilize zero days
- Typically covering something very niche / specific
- May take weeks or a team the whole CTF to do

SIGPWNY

## Development Steps - Specifics (This applies to all CTFs)*


Could someone in 461 trivially** do what your challenge revolves around
YES
NO
Can someone with no experience do what your challenge revolves around
YES
NO
Will the challenge be trivial for a subject matter expert
YES
NO
50-100 Point Intro Chal
100-200 Point challenge
200-300 Point challenge
400-500 Point challenge

|-------------------Tips will be for these two categories-------------------|

* Internal CTF is weighted differently
** TRIVIAL != FAST

## Development Steps - Development

- Make sure most teams would have someone who knows the environment your challenge is being developed in.
  - If you want to make a very niche challenge, just make one or two (Unless your CTF is centered around that niche)
  - Don’t make half your challenges one specific weird programming language unless people know that in advance & can prepare. (*cough* *cough* ocaml @ppp *cough* *cough*)

## Development Steps - Development

- Make sure your challenge is not too large (file size), and does not require excessive libraries to run (if giving local copy).

- Include a **solution.md**, and a **solution.py** if possible

- Make sure to include hosting materials if needed (docker etc).
  - See resources slide for pwn docker skeletons.

## Development Steps - Deployment

1. Compress and send your file to an exec, check to make sure you have
   a. Dockerfile
   b. solution.md (solve.py)
   c. info.md (points, title, description)
      i. Could also be challenge.yaml

2. Test your challenge live

3. Have SOMEONE ELSE test your challenge live
   a. If possible on a different architecture / OS

4. Make sure your challenge is up on CTFd

## Words of Warning

Unless explicitly stating you are doing so in challenge description and readme,
NEVER INCLUDE POTENTIALLY HARMFUL SOFTWARE IN YOUR CHALLENGE

That means
NO rm -rf (EVEN IF IT IS FAKE OR WON'T BE EXECUTED)
NO RATs or callback shells
Nothing legitimately malicious

Club admins should be able to see source / compile from source, and reserve the right to ask for the source to be changed should we see something malicious.

## Tips and Resources for Specific Topics

## Tips - RE

- Challenges that are just a ton of if statements can usually be solved with angr
- For beginners, writing a challenge that doesn't require lower-level knowledge is valuable.
  - That way experienced CTF players likely have scripts that can solve it.
- Don’t obfuscate source so much that it is annoying.

Resources / Inspiration for RE chal design

- https://challenges.re/
- https://infosecwriteups.com/tagged/reverse-engineering

## Tips - PWN

- For easier challenges, go by the rule “Stop at ROP”
  - Don't do challenges significantly harder than rop, newcomers won't know what to do and will give up before they can figure it out.

- The actual vulnerability does not have to be the hard part!!!
  - You can make the vulnerability simple, but triggering the vulnerability difficult
  - See Accounting Accidents (link)

- Grab skeleton code for c challenges and python based challenges on chal.dev

## Tips - Web

- Grab flask skeleton on chal.dev (resources)
- **SOMEONE HELP ME WITH THIS I DON'T DEVELOP WEB CHALLENGES**

## Tips - Crypto

- Hit the goldilocks zone of hint information
  - Doesn’t leave the challenge too guessey, doesn’t leave the challenge too obvious after reading the description.

- Brute-Force solvability
  - Optimized solution should generally be fast
  - Brute force can also work, but should take way more time.

## Tips - Forensics / Stego

- If you are going to make forensics, don't just make it stego
- If you are going to make stego, make sure it is good stego
    - Lots of people dunk on stego for some reason.

- Again, abide by the same rule as crypto

## Tips - OSINT

- Goldilocks zone of difficulty
  - You want OSINT challenges to be a *little* guessy.
  - Some amount of just looking around on the internet is good, but don't waste people's time
- No false rabbit holes
  - Even though this may happen in reality, it's a crappy thing to do with OSINT challenges.

- Developer tips
  - Websites that are OK with OSINT accounts
    - Twitter, Reddit, Youtube, Google accounts, Steam, Imgur, Instagram, Make your own website (google domains -> google sites, super ez and 1$ a month for a year, we will expense it).
  - Websites less OK with OSINT accounts
    - Facebook (quick to ban), Linkedin (professional website)

## Tips - Networking

- Often these challenges need to be in person

- If you can create a way to do networking challenges over a remote connection (before I do it), it would make a really good research project ;)

- Not many networking challenges exist, so typical attacks (replay, spoofing, wpa cracking) are cool to make.

## Extra Resources

## Extra Resources (https://chal.dev is something we own now)

C Challenge Skeleton

- cskel.chal.dev

Python Hosted Challenge Skeleton

- pyskel.chal.dev