Javascript

1. **Low level**

- First seeing the src code, I find the javascript function `generate_token()` is called.

```
function rot13(inp) {
  return inp.replace(/[a-zA-Z]/g,function(c){return String.fromCha
rCode((c<="Z"?90:122)>=(c=c.charCodeAt(0)+13)?c:c-26);});
}

function generate_token() {
  var phrase = document.getElementById("phrase").value;
  document.getElementById("token").value = md5(rot13(phrase));
}

generate_token();
```

- This function gets the value of the element `phrase` and computes the md5 of the phrase coded through a Ceasar cipher (ROT13)
- Let's see if `PunatrZr` is a ciphertext encoded with `ROT13`

```
echo 'PunatrZr' | tr 'A-Za-z' 'N-ZA-Mn-za-m'
ChangeMe
```

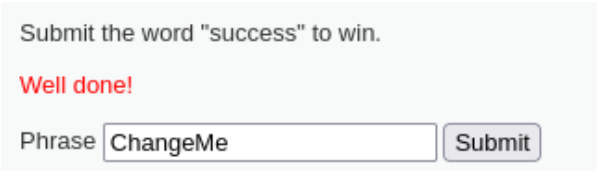
- The token is equal to `md5(rot13("ChangeMe"))`. I added an alias that computes the rot13 of the standard input

```
alias rot13="tr 'A-Za-z' 'N-ZA-Mn-za-m'"
```

- Compute a token for the string `success` and send it to the server

```
echo -n "success" | rot13 | md5sum
38581812b435834ebf84ebcc2c6424d6
```

- “Well Done” on the server



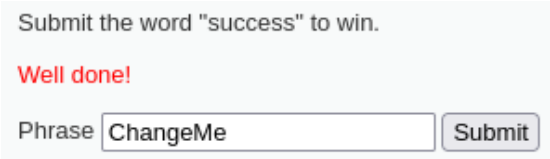
2. **Medium Level**

- The token always `XXeMegnahCXX`. If the phrase is not `success` we get the error **You got the phrase wrong.** When the phrase is `success` we get the error **Invalid token**
- Seeing the src code, we can see a script is called

  `/vulnerabilities/javascript/source/medium.js`

```
function do_something(e){for(var t="",n=e.length-1;n>=0;n--)t+=e[n];return t}setTimeout(function()
{do_elsesomething("XX")},300);function do_elsesomething(e)
{document.getElementById("token").value=do_something(e+document.getElementById("phrase").value+"XX")}
```

- From the script, we know
  - The function `do_elsesomething` is called after 300ms of waiting
  - The function `do_elsesomething` sets the token value with the help of the function `do_something`
- The default value of the phrase is `ChangeMe`, and the function `do_elsesomething` is set to execute 300ms after the page loads
- Open a console and execute a command `do_elsesomething('XX')` the token for the value `success` will be computed
- Submit and take the answer `Well done!`



3. **High Level**

- The token always is `28638d855bc00d62b33f9643eab3e43d8335ab2b308039abd8fb8bef86331b14`
- If the phrase is not `success`, we get the error message `" You got the phrase wrong "`. Otherwise we get the error message `" Invalid token "`
- Let's see the src code and find that script creates an array that is shifted by the following function

```
(function(array_to_shift, nb_shift) {
var shift_function = function(nb_shift_bis) {
// Shift the array "a" 0x1f4 + 1 times = 501 times
while (--nb_shift_bis) {
```

```
array_to_shift
 "push" ;
}
};
shift_function(++nb_shift);
})(special_array, 0x1f4);
```

- Right after the script defines a function to retrieve an element from the array

```
// Return a[index],
var get_element_from_a = function(index, unused_parameter) {
index = index - 0x0;
var element = special_array[index];
return element;
};
```

- The script creates a new function from the array and then uses `eval()` to launch the newly crafted function. To get the newly created function we put a breakpoint at the end of the function `function(d, e, f, g, h, i)` and we retrieve the code

```
function() {
// function that seems to define the sha256 function used later
})();
// Functions that help crafting the token
function do_something(e) {
for (var t = "", n = e.length - 1; n >= 0; n--) t += e[n];
return t;
}
function token_part_3(t, y = "ZZ") {
document.getElementById("token").value = sha256(
document.getElementById("token").value + y
);
}
function token_part_2(e = "YY") {
document.getElementById("token").value = sha256(
e + document.getElementById("token").value
);
}
function token_part_1(a, b) {
document.getElementById("token").value = do_something(
document.getElementById("phrase").value
);
}
```

```
// The rest of the code defines how the token are constructed
document.getElementById("phrase").value = "";
setTimeout(function() {
token_part_2("XX");
}, 300);
document.getElementById("send").addEventListener("click", token_part_3);
token_part_1("ABCD", 44);
```

- From the src code, we understand that the first part defines some functions that are used later on while the last part crafts the token from the different function

- The token is generated when the page is loaded with the predefined value "ChangeMe". When the input value is modified, the token is not updated
- We modify the input value to "success" and we will try to craft the token from the code logic

  ```
  token_part_1("ABCD", 44)
  "sseccus"
  ```

- The first function just scrambles the word "success", we just generated the first part of the token

  ```
  token_part_2("XX")
  7f1bfaaf829f785ba5801d5bf68c1ecaf95ce04545462c8b8f311dfc9014068a
  ```

- The second part generates a SHA-256 sum from "XX" concatenated with the first part
- From the code we see that the last part computes a SHA-256 sum of the previous step

  ```
  token_part_3(null, "ZZ")
  ec7ef8687050b6fe803867ea696734c67b541dfafb286a0b1239f42ac5b0aa84
  ```

- Config the token after using burp suite to intercept the connect

  ```
  echo -n "7f1bfaaf829f785ba5801d5bf68c1ecaf95ce04545462c8b8f311dfc9014068aZZ" | sha256sum

  ec7ef8687050b6fe803867ea696734c67b541dfafb286a0b1239f42ac5b0aa84
  ```

- The page pop up “Well done!”