

Deliverable 3 - Report

BTN710

Navid Esfahani

Group 1

Neil Ong, Julian Boyko, Andrew Tassone

July 10, 2020

Contents

| | |
|---|----|
| Introduction | 3 |
| Attacks | 3 |
| <i>Cross Site Request Forgery (CSRF/XSRF)</i> | 3 |
| Section 1: The Exploit | 3 |
| Section 2: The Attack | 4 |
| Section 3: Security Policy | 6 |
| <i>Cross Site Scripting (XSS)</i> | 6 |
| Section 1: The Exploit | 6 |
| Section 2: The Attack | 7 |
| Section 3: Security Policy | 9 |
| <i>NoSQL Injection</i> | 10 |
| Section 1: The Exploit | 10 |
| Section 2: The Attack | 10 |
| Section 3: Security Policy | 11 |
| Conclusion | 11 |
| Works Cited | 12 |

Introduction

Web security is a vital part of any web application. Various attacks can be used that can be extremely damaging and costly to any company. In order to prevent this, it is important that security experts ensure that all necessary precautions are taken to prevent the latest attacks. The team chose three attacks to use against the project found on GitHub. These attacks include CSRF, XSS and NoSQL injection was chosen due to the ability to attack the login page of the web application. NoSQL works with MongoDB which is the database used in the web application. XSS was chosen as it is an attack that is able to execute malicious JavaScript in another user's browser and exploit a vulnerability in the website. CSRF was chosen as it is another attack against a web browser. The web browser is tricked into submitting a false request by the attacker. These attacks were carried out locally on the researcher's computer, using the GitHub repository. The findings are documented in this paper.

Attacks

Cross Site Request Forgery (CSRF/XSRF)

Section 1: The Exploit

Brief Description

Cross Site Request Forgery is an attack that makes the user do or execute unwanted actions on a web application in where they are currently authenticated. (*What is CSRF: Cross Site Request*) They usually use some form of socially engineering to get the user to do the action. (*Cross Site Request Forgery (CSRF)*) If the attack is successful against a normal user the attacker can change user data and, where if it is successful against an admin account with more privilege the whole web application could be compromised. (*What is CSRF: Cross Site Request*)

Operation System (OS)

This attack is not operation system specific because it is a web application, it should work on all systems.

Protocols/Services/Applications

The applications this attack manly focus is web applications, especially ones that have back-end web services connected to it. (*What is CSRF: Cross Site Request*) These back-end web services could be anything from API's to databases that store user information and data. These attacks on these applications and services are mainly done through web browsers since they run the applications. The protocol this attack takes advantage of is the HTTP protocol that allows the fetching of resources and data from a server. (*An overview of HTTP*)

Variants

There are a few variants of the attack. They are all called Cross-Site Request Forgery but go about attacking the web applications a few different ways. Firstly, the social engineering aspect of the attack can be done in a few ways, but the two most common versions is either by going to a website that has a malicious script or having someone click on link in an email that either runs a script or tries to make the malicious http request. (*What is CSRF: Cross Site Request*) The malicious attack can use GET, POST, and even PUT and DELETE request to alter and compromise user data and operations. (*What is CSRF: Cross Site Request*)

Section 2: The Attack

Description and Diagram of Network:

The network where the incident can be visualized by the image above. The attacker sends an email to victim over the internet, where the user mistakenly runs the malicious script/action. (*What is CSRF: Cross Site Request*) This sends a request to the web service, which if the server cannot distinguish the attack then responds to the attack giving the attacker what they wanted. (*What is CSRF: Cross Site Request*)

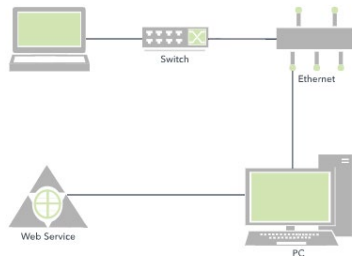


Figure 1

Protocol/Service Description

The protocol that is being attacked is the HTTP protocol and the service being the web service/server of the web application. (*What is CSRF: Cross Site Request*) Normally, this protocol is the basis of how the server and client side interact. The client uses this to fetch the HTML or data it needs. This is usually done by requesting a RESTful CRUD call to the server, where the server responds by giving the client side the resources that it is requesting. (*An overview of HTTP*) It's a weakness if this interaction is not protected then anyone can create, change, or receive important resources. (*An overview of HTTP*) An example of this would be a GET call that fetches important user data like an address or credit card information. If the attack gets access to this, they can fetch important data or even do malicious operations like depositing money into their accounts. (*What is CSRF: Cross Site Request*)

How the exploit works

The exploit is fairly straightforward to execute. The attack can either be very simple or more complex depending on the web application or how convincing the attack wants to be. The exploit can be broken down into two simple steps.

1. The first is build an exploit URL or script. (*What is CSRF: Cross Site Request*) For the purposes my trail attack I used an exploit URL. An exploit URL is the malicious operation that the attacker wants the user to execute. (*What is CSRF: Cross Site Request*) An example of this is using an anchor tag or img tag that links to the exploit URL. For our attack, we used an anchor tag that had a link to get request for <http://localhost:8080/api/users>, which fetches all the users and their information in the database. CSRF can also be done using a POST request by delivering a "form" and tags to "submit" to the web service. (*What is CSRF: Cross Site Request*) The other method is by trying to use a script on an exploit page that you try to get the user to access. (*What is CSRF: Cross Site Request*) Once they enter the exploit page the script will try to execute by creating a new Http request to the browser to try and gain access to the resources. (*What is CSRF: Cross Site Request*)
2. The second step is to use social engineering to get the user to either click on the exploit URL or access an exploit page where a malicious script can be run. (*What is CSRF: Cross Site Request*) The key hear is hiding the attacker's true intentions and replacing it with something that entices the user. For the exploit URL it can be an image or a link that entices them to click on it like images or a link to cute dogs or even a sale for an item that they wanted to buy. The same goes with luring them to the exploit

page. (What is CSRF: Cross Site Request)

Once these steps are in place the attacker just needs to lie and wait for the user to fall into the trap. Once the exploit is run the damage is done, and the attack has access to the information they want like forcing the user to transfer money to the attacker's bank account. (What is CSRF: Cross Site Request) Similarly, if they want user information, they could use a script to fetch the user data and send them data.

Description and diagram of the attack:

For our attack we decided to socially engineer an email to entice the victim to click on link to cute images of dogs. This was actually a link to an exploit URL that tried to fetch user data. The two images below show how the hidden exploit URL was made and what it looks like from the victim's perspective. They see a link that tell them to click it for cute dogs. This could have been more convincing detailing a more believable story from a more believable email.

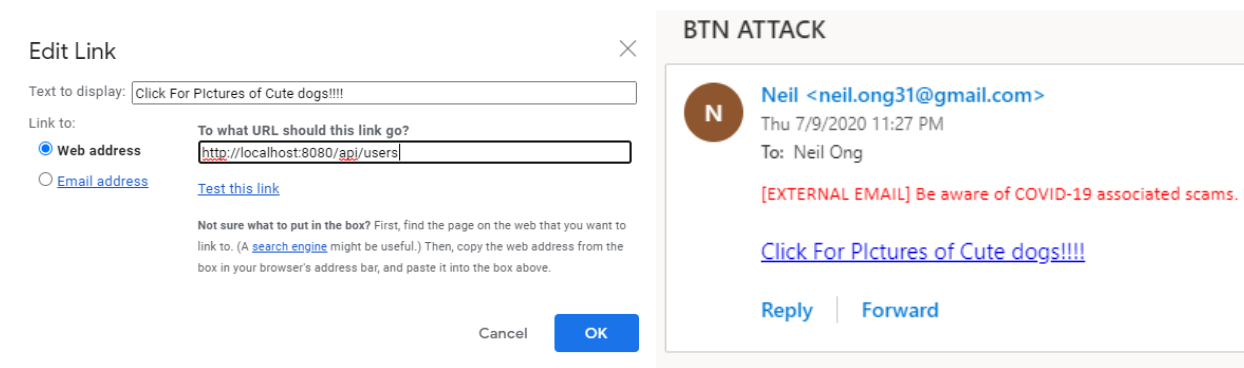


Figure 2

Figure 3

The two images below show that that victim was logged into the web application and clicked on the link for exploit URL, but it did not work. The server returned an Unauthorized Error telling the user that, "No authorization token was found".

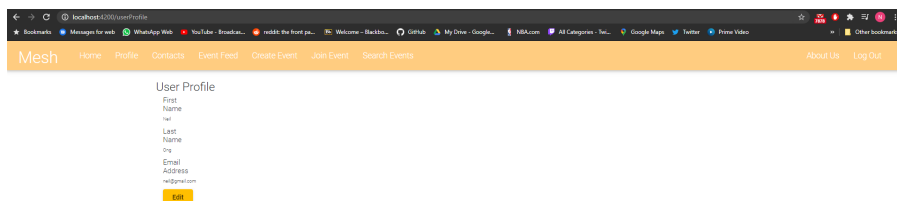


Figure 4



Figure 5

Signature of the attack:

For the attempted attack, there was no signature since it did not work, but even if it did there should be no signature since it was simple GET that only tried to fetch user data. Although, this type of attack can leave a

signature especially in the logs and database if the attack tries to manipulate user operations or data. This is because the data would be altered by adding events to the user's event list they have never been to or adding or even deleting contacts from their contact list.

How to protect against it:

There are few ways to protect from CSRF that the web application implemented. The first being the built-in support against CSRF attacks by using Angular's HttpClient to safely make requests by checking its XSRF-TOKEN, which was used in this project for every call to the web service. (*HTTP-level vulnerabilities*) Also, another added feature is since the Angular web application uses its router to only display pages or components from the Angular app any Http requests will not work since, all outgoing calls are handled by Services that use HttpClient to protect them. (*Security: XSRF protection*) Furthermore, if you use JSON Web Tokens to verify the user making the request, it stop attackers from trying to directly access the web-service because the server will throw an UnauthorizedError. (*JSON Web Tokens Introduction*)

Also, protection from an exploit page using a script to make a request on behalf of the user is implemented on most modern browsers already. (*What is CSRF: Cross Site Request*) They will not allow the request to go through unless the default restriction is life by not using CORS, which is implemented in the back-end (See Figure 6) (*What is CSRF: Cross Site Request*)

```
import { HttpClient } from '@angular/common/http';
import { Events } from '../classes/events';
import { Observable } from 'rxjs';
import { Attendee } from '../classes/attendees';
import { EventAttendee } from '../event-components/event-room/event-room.component';
import { environment } from 'src/environments/environment';

@Injectable({
  providedIn: 'root'
})
export class EventService {
  private currentEvent: Events;
  private url = environment.url;

  constructor(private http: HttpClient) {}

  // Get all
  eventsGetAll(): Observable<Events[]> {
    return this.http.get<Events[]>(`${this.url}events`);
  }
}
```

Figure 6

```
// Add support for CORS
app.use(cors());
```

Figure 7

There are a few other ways of protecting against CSRF, but these were the reasons why this web application could not be attacked using a CSRF attack

Section 3: Security Policy

Considering the attack failed, I don't feel it is necessary to update the security policy in regard to dealing with CSRF attacks.

Cross Site Scripting (XSS)

Section 1: The Exploit

Brief Description:

Cross-Site Scripting or XSS for short, is a type of attack that uses injection to input malicious scripts into websites that are considered safe and are generally trusted (Manico et al.). The way the Cross-Site Scripting attack works is the attacker will use a web application to send code that is malicious (Manico et al.). The attack is considered a client-side code injection attack, and when the victim of the attack visits the infected web page or web application, the attack occurs (What is Cross-site Scripting and How Can You Fix it?). This malicious code usually takes the form of a browser side script (Manico et al.).

Essentially, what happens is the web page or the web application that becomes infected acts a vehicle to deliver the malicious script to the victim's browser (What is Cross-site Scripting and How Can You Fix it?). When the attacker sends a malicious script to an unsuspecting user, the user has no idea that the script is not to be trusted, resulting in them executing the script (Manico et al.) The script can access any cookies, session tokens or other sensitive information that is retained by the browser and used on that site (Manico et al.)

Operating System (OS):

The attack can be performed on all operating systems as it is an attack that takes place through the attacker's web browser.

Protocols/Services/Applications:

The applications/services that are affected by the XSS are web applications and web sites (What is Cross-site Scripting and How Can You Fix it?). Previously, most web browsers such as Microsoft Edge, Firefox and Google Chrome offered some form of XSS protection, but it has since been removed (Cross-Site Scripting Protection Vanishing from Browsers). This means that before, Web Browsers did offer protection in some ways to prevent XSS attacks from taking place. Although, now the responsibility to protect users from XSS attacks is completely on the owner of the web-application and/or web-site (Cross-Site Scripting Protection Vanishing from Browsers). This may increase the risk for XSS attacks.

Variants:

There are various different types of XSS attacks. XSS attacks are divided into three different types: Persistent XSS, Reflect XSS and Dom-based XSS (Excess XSS).

Persistent XSS is what happens when the malicious string comes from the website's database (Excess XSS). Reflected XSS is what happens when the malicious string comes from the victim's request (Excess XSS). And lastly, DOM-based XSS is what happens when the vulnerability is on the client-side code instead of it being on the server-side code (Excess XSS).

Section 2: The Attack

Description and diagram of network:

I'll be attacking the front-end of the Mesh web application. One of the attacks I attempt will take place in the input field of the "Join Event" page (see Figure 8).

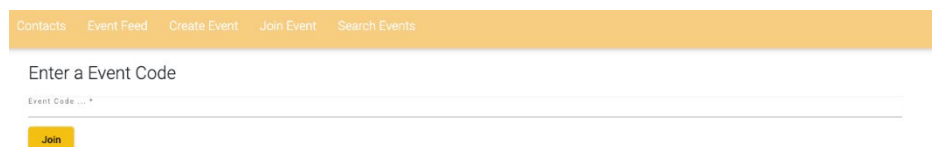


Figure 8

Protocol/Service Description:

When I attempted the XSS attack on the Web Application, I used input fields in the Web Application to try and inject JavaScript. I did this through the “Join Event” page on the Web Application and I also attempted it on the “Create Event” page on the application. It was unsuccessful both times.

How the exploit works:

Cross-Site Scripting is prominent mostly in software created in JavaScript, mainly because it is generally fundamental to the web-browsing experience – but it is also possible in VBScript, ActiveX, Flash and CSS (What is Cross-site Scripting and How Can You Fix it?). When people think of XSS, they generally think of the attack as one that affects users directly, but it’s important to keep in mind that XSS doesn’t just affect the user of a web application/web site (What is Cross-site Scripting and How Can You Fix it?). Additionally, XSS attackers can use JavaScript to change a website (What is Cross-site Scripting and How Can You Fix it?). Rather than an attacker attacking a user, they can instead inject scripts that change the content and layout of the webpage, but even worse – they can also even redirect the browser to another web page, again targeting the user (What is Cross-site Scripting and How Can You Fix it?).

Generally speaking, XSS attacks are seen as one of the more docile vulnerabilities, especially when compared to something like an SQL Injection (What is Cross-site Scripting and How Can You Fix it?). With that said, there are many consequences that can come from an attacker being able to successfully complete an XSS attack. You may think that because most web browsers run JavaScript in a controlled and closed environment that it is a non-issue, but JavaScript can be very dangerous when in the wrong hands (What is Cross-site Scripting and How Can You Fix it?). The following are some of the things that an XSS attack using JavaScript can do:

- Injected JavaScript can be used to gain access to all of the stored objects that the website or web-application has access to (What is Cross-site Scripting and How Can You Fix it?). This involves various forms of sensitive information such as data such as a user’s stored session cookie (What is Cross-site Scripting and How Can You Fix it?). This is an issue because if an attacker is able to gain access to a user’s stored cookie, they can then pretend to be that user, and not only gain access to that user’s personal data but can also impersonate that user on the website (What is Cross-site Scripting and How Can You Fix it?).
- JavaScript can be used to send HTTP requests with random content to random destinations with the use of XMLHttpRequest (What is Cross-site Scripting and How Can You Fix it?)
- With the use of social engineering, an attacker can use JavaScript with the use of HTML5 APIs to gain access to various forms of private information on users, that seriously infringe on their privacy. This can be done by gaining access to the user’s geolocation, webcam and microphone (What is Cross-site Scripting and How Can You Fix it?). Additionally, with HTML5 APIs the attacker can even access specific files from the user’s file system (What is Cross-site Scripting and How Can You Fix it?).

Description and diagram of the attack:

When I attempted the XSS attack, I used the following small script
“<script>document.documentElement.innerHTML=""</script>”. The idea behind this was to inject JavaScript that accessed the innerHTML property of the document, setting it to a blank string. If the attack was successful, this would remove all HTML from the Web Application’s web page. Additionally, I also tried to insert some JavaScript alerts and HTML elements.

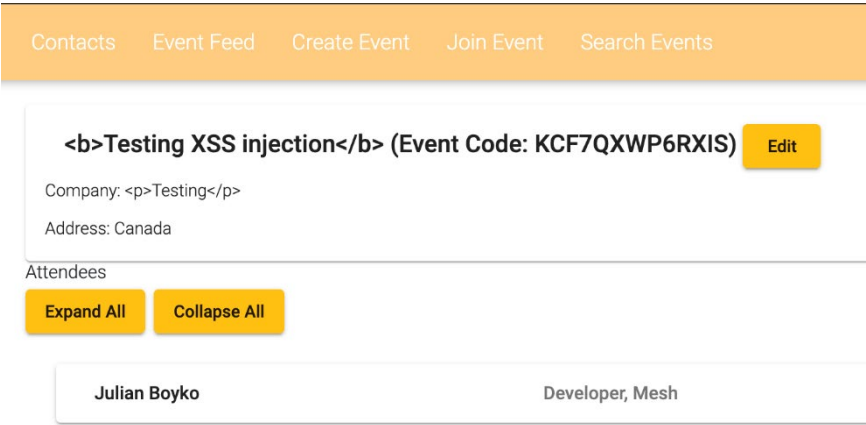


Figure 9

There are a few reasons that the XSS attack was unsuccessful. For one, the Web Application was built with the Angular framework, and Angular has some built in XSS protection (Angular – How to Prevent XSS Attacks – Code Examples). In particular, Angular will consider all data as untrusted data and by default will “sanitize” all data (Angular – How to Prevent XSS Attacks – Code Examples). What this means in particular is that any HTML tags which are found in the data will be escaped (Angular – How to Prevent XSS Attacks – Code Examples).

Secondly, when I tried to inject the JavaScript script to clear out the innerHTML, I believe it failed because the search value was being stored in a variable which was then used to match the value to other values in an array, rather than being ran as plain code.

Signature of the attack:

I would say that the main signature of attack is the changed state of the web application. If you inject a script that changes the appearance or functionality of a web application or web-site then you are clearly indicating that some sort of vulnerability has been exploited.

How to protect against it:

When it comes to the user of a web application or web-site, there is nothing you can really do to avoid the attack, except to be cognizant about social-engineering attacks trying to lure you away from the webpage, which can happen when an XSS attack has already taken place.

Regarding what the web site and web application owners can do to prevent the attack, they have the following techniques at their disposal, which are: escaping, validating input and sanitizing data (Vonnegut).

Section 3: Security Policy

Considering the attack failed, I don’t feel it is necessary to update the security policy in regard to dealing with XSS attacks.

Section 1: The Exploit

Brief Description:

NoSQL injection will be used in an attempt to access the personal information stored in the MongoDB database using the hosted Node JS Heroku web service. NoSQL Injection is similar to the classic SQL injection, but uses the NoSQL engine that is used in MongoDB databases. The username and password queries on the Sign-in page will be attacked. Using JSON objects, a statement can be inserted in both queries that always evaluates to true. The following is an example of MongoDB code vulnerable to NoSQL injection.

```
let query = {
  username: req.body.username,
  password: req.body.password
}

User.find(query, function (err, user) {
  if (err) {
    // handle error
  } else {
    if (user.length >= 1) {
      res.json({role: user[0].role, username: user[0].username, msg: "Correct!" });
    }
  }
});
```

Figure 10 ([NullSweep](#))

Operating System (OS):

The attack can be performed on all operating systems as it takes place through software that is able to intercept web traffic like Burp Suite.

Section 2: The Attack

Description and diagram of network:

The attack is done by modifying the JSON object that is in transit. The username or password is replaced with a valid MongoDB query, using the \$ne verb that always equals true so the password is always correct. Finally, using software like Burp Suite that is able to intercept traffic, catch the request and modify the body to something that looks similar to this:

```
{"username":"example","password":{"$ne": 1}}
```

Figure 11

Using this code, you can log into an account without knowing the password and could use other requests to extract the password with a few tries.

Protocol/Service Description

When attempting the attack using the Burp Suite, I attempted to intercept the query that is in transit and attempt the injection with a username and password in a valid MongoDB query. Finally catching and modifying the body of the JSON request. "Unlike relational databases, NoSQL databases don't use a common query language". (Belmer).

How the exploit works:

This exploit is usually made possible due to a vulnerability in the lookup code. Using a modified JSON object, it is possible to modify the data and even escalate privileges of a user.

Signature of the attack:

The attacker could be detected once the user realizes their account has been hacked or the amount of password attempts is higher than normal.

How to protect against it:

This can be prevented by more strongly validating the users inputted data by using libraries such as mongo-sanitize. Also, setting javascriptEnabled to false in the MongoDB configuration file in order to disable JavaScript code from executing, will help to further protect from similar attacks. As Nullsweep states, "Don't use the Mongo where, mapReduce, or group with user supplied data." (Banach).

The attack failed to execute on the application selected by the team. Due to the secure third-party authentication system called auth0, access was denied to the MongoDB database. Therefore, it is important to have good validation in front-end applications.

Section 3: Security Policy

To prevent this attack from occurring, it is highly recommended to always use the latest version of MongoDB. It is also not recommended to use Javascript in your code unless necessary. If so, it is always recommended to validate and encode all user inputs.

Conclusion

In conclusion, Web Security is the center-stone of any Web Application or Web-Site. Various breaches have occurred over the history of the internet and with that privacy has been infiltrated and sensitive data has been exposed. The three attacks that we chose to do were CSRF, XSS and NoSQL Injection. The CSRF attack was used to try and trick the web browser into submitting a false request by the attacker. The XSS attack was chosen to execute malicious JavaScript through another user's browser, and the last attack, the NoSQL Injection attack tried to find vulnerabilities in MongoDB. We were pleased to find out that all our attacks failed, indicating a decent level of security on our web application.

Works Cited

Cross Site Request Forgery (CSRF). owasp.org/www-community/attacks/csrf.

Security: XSRF Protection. angular.io/guide/http.

"HTTP-Level Vulnerabilities." *Angular*, angular.io/guide/security.

"An Overview of HTTP." *MDN Web Docs*, developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

"What Is CSRF: Cross Site Request Forgery Example: Imperva." *Learning Center*, Imperva, 7 July 2020, www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/.

auth0.com. "JSON Web Tokens Introduction." *JSON Web Token Introduction*, jwt.io/introduction/.

Manico, Jim, et al. "Cross Site Scripting (XSS)." *Cross Site Scripting (XSS) Software Attack | OWASP Foundation*, owasp.org/www-community/attacks/xss/.

What Is Cross-Site Scripting and How Can You Fix It? 16 Mar. 2020, www.acunetix.com/websitesecurity/cross-site-scripting/.

"Cross-Site Scripting Protection Vanishing from Browsers." *Packetlabs*, 8 Aug. 2019, www.packetlabs.net/browsers-dropping-xss-protection/.

"Excess XSS." *Excess XSS: A Comprehensive Tutorial on Cross-Site Scripting*, excess-xss.com/.

"Angular – How to Prevent XSS Attacks – Code Examples." *Ripcord Systems*, 9 Mar. 2019, ripcordsystems.com/2019/03/09/angular-how-to-prevent-xss-attacks-code-examples/.

Vonnegut, Sarah. "3 Ways to Prevent XSS." *Checkmarx*, 3 Sept. 2018, www.checkmarx.com/2017/10/09/3-ways-prevent-xss/.

NoSQL Injection (Part 1 of the SQL Series). 8 July 2020, www.pentestpeople.com/nosql-injection.

Belmer, Charlie. *A NoSQL Injection Primer (With MongoDB)*. 6 Aug. 2019, nullsweep.com/a-nosql-injection-primer-with-mongo/.

Banach, Zbigniew. *What Is NoSQL Injection and How Can You Prevent It?* 17 June 2020, www.netsparker.com/blog/web-security/what-is-nosql-injection/.

What Is NoSQL Injection? 17 Feb. 2020, resources.infosecinstitute.com/what-is-nosql-injection/.