

**Sri Lanka Institute of Information Technology**

**Secure Software Systems**

**Assignment 2020**

Group Details

|  |  |  |
| --- | --- | --- |
| Registration Number | Student Name | Contribution |
| IT18121148 | H.M.D.G.V. Perera | 2.3 | Cross-Site Scripting Vulnerability and Patch |
| IT18128550 | R.P.D.T.D. Pathirathna | 2.2 | Missing Rate Limiting Vulnerability and Patch |
| IT18157338 | U.P.B. Wijayawardane | 2.1 | Cross-Site Scripting Vulnerability and Patch |

Table of Contents

[Table of Content 3](#_Toc39349280)

[1 | Introduction 4](#_Toc39349281)

[1.1 | Injectify 4](#_Toc39349282)

[1.2 | Vulnerabilities found in the tool 5](#_Toc39349283)

[1. Cross-Site Scripting 5](#_Toc39349284)

[2. Missing Rate Limiting 6](#_Toc39349285)

[2 | Found Vulnerabilities and Patches in depth 7](#_Toc39349286)

[2.1 | Cross-site Scripting vulnerability 7](#_Toc39349287)

[How it will affect the security of the software 7](#_Toc39349288)

[Patch/Mitigation 8](#_Toc39349289)

[2.2 | Missing Rate Limiting 9](#_Toc39349290)

[How it will affect the security of the software 9](#_Toc39349291)

[Patch/Mitigation 10](#_Toc39349292)

[2.3 | Cross-Site Scripting Vulnerability 11](#_Toc39349293)

[How it will affect the security of the software 11](#_Toc39349294)

[Patch/Mitigation 12](#_Toc39349295)

[References 13](#_Toc39349296)

# 1 | Introduction

## 1.1 | Injectify

We Used Injectify, which is an open-source web-based advanced Man in the Middle (MiTM) attack tool for this assignment. Sam Denty created this tool, and this tool uses modern web technologies such as ReactJS, Redux, Typescript, and Webpack.

Injectify is mainly used to create reverse JavaScript shells between an attacker and the victim. This tool has a feature called "PageGhost," which can be used to monitor a victim's screen and also interact with it in real-time. Furthermore, this can be considered as a highly integrated framework with advanced API such as Data extraction API, which is used to extract sensitive data from victim browsers.

There are three contributors to this tool, and there are 593 commits in Github. The latest commit was done on 3rd May 2018. There are seven issues posted in Github for this tool. According to LGTM, the code quality for the JavaScript used in this tool is E, and there are 160 alerts. Out of these alerts, we have focused specifically on security errors in Injectify.

## 1.2 | Vulnerabilities found in the tool

### 1. Cross-Site Scripting

Cross-site scripting, widely known as XSS, happens when hackers execute malicious JavaScript inside the browser of a victim. Upon initial injection, the attacker usually doesn't wholly operate the website. The bad actor then adds their malicious code to a legal website, tricking browsers into running their malware once the website is launched. Because the JavaScript runs on the victim's browser, the session can steal confidential information regarding the authorized user, effectively enabling an attacker to target site privileges and deeply compromise a website. Another common usage of cross-site scripting attacks is when the bug is present on several of a website's publicly accessible sites. In this case, attackers can inject their code into elements in the website, such as ads and other content to target website visitors.

Mitigation:

* Sanitizing values

When adding user-generated material on a website, make sure that by replacing inappropriate characters with their respective individuals, this does not interfere in HTML material. They have the same appearance as standard characters, but they cannot be used to generate HTML.

* Using a Firewall

A firewall is beneficial in any scale of attack because it can practically repair threats on the website. This approach intercepts threats until the website receives unauthorized requests.

* Whitelisting

This method helps the server to obtain known and safe values. Restricting user inputs helps if you recognize what data you are likely to receive, so it's not realistic for custom user content.

* HTTPOnly Flag

HTTPOnly cookies can be used to prevent JavaScript from reading cookie material, making it harder for an intruder to steal a session. This means that attackers are not able to read the message, but the attackers can still submit requests using the active client session when operating as an admin user. This method is also only useful if cookies are used as a vital tool for recognition.

### 2. Missing Rate Limiting

The rate limit is used to control the amount of traffic to or from the network entering and leaving. When the threshold of the number of requests is exceeded, a fault is caused. The reasoning behind the usage of rate limits is to enhance the data flow and enhance protection by mitigating attacks such as DDoS. An HTTP request processor is vulnerable to denial-of-service attacks that carry out costly operations without limiting the rates of process. There are several ways of actually implementing rate limits. This is usually done at the server level, through a programming language or a caching framework.

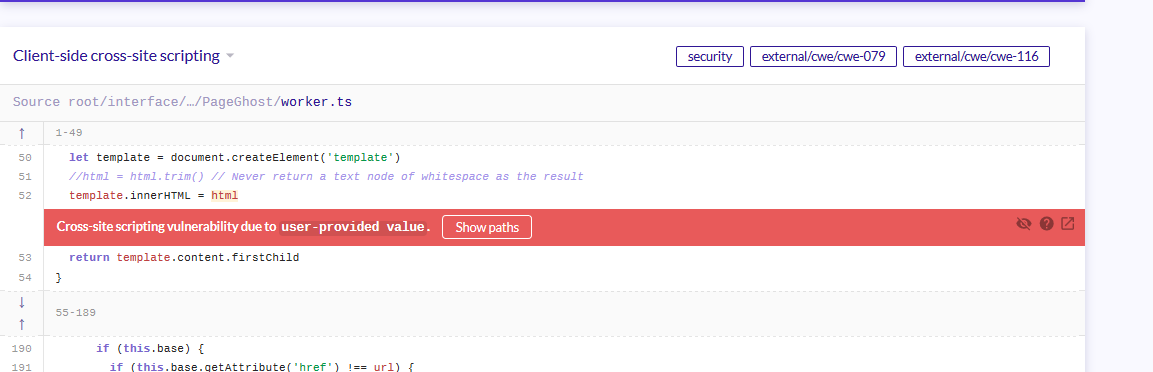
Mitigation:

Using a Middleware to prevent such attacks

# 2 | Found Vulnerabilities and Patches in depth

## 2.1 | Cross-site Scripting vulnerability

This was on DOM (Document Object Model)



*Figure 1*

This was DOM-based XSS vulnerability because that is a client (browser) side injection issue.

This software has a function called htmlToElement in the worker.tc page; line no 49. The parameter got HTML pass into the htmlToElement function, and it is vulnerable.

How is that parameter vulnerable?

1. The HTML parameter comes from a function called "change" (line no 286). This function gets user input values within the HTML parameter.
2. That HTML parameter passes into the htmlToElement function (line no 49), and inside this function, HTML parameter assign to an element called "template" (line no 52). That is vulnerable because that parameter assigning to element as prematurely (with without encoding/sanitizer mechanism)

### How it will affect the security of the software

The attacker executes some malicious script in a web browser and that malicious script assign to the HTML parameter. That malicious HTML parameter is passed to the "template" element (line 52), and the actual attack occurs when the victim visits the web page that executes the malicious code.

### Patch/Mitigation

My solution was to sanitize the HTML parameter and that output pass into the element called "template". Here I use the “node-esapi" encoder and follow these steps.

1. Open the terminal and install node-esapi npm package (without this installation "require" function not work in your program)

**Command:- npm i @types/node**

1. Create ESAPI variable and assign require('node-esapi') function. (line number 50)

**Code: Let ESAPI = require('node-esapi');**

1. After passing the HTML value user provides value within sanitization code and assigning the output to an element called template. (line number 56 )

**Code: template.innerHTML = ESAPI.encoder().encodeForHTML(HTML);**



*Figure 2*

## 2.2 | Missing Rate Limiting

An HTTP request handler that performs expensive operations without restricting the rate at which operations can be carried out is vulnerable to denial-of-service attacks.

In this Injectify application, two routes allow users to perform file system operations without any restrictions.



*Figure 2*

### How it will affect the security of the software

These mentioned two routes are responsible for performing file system operations, such as

•fs.existsSync(path) - returns true if path exists, false otherwise.

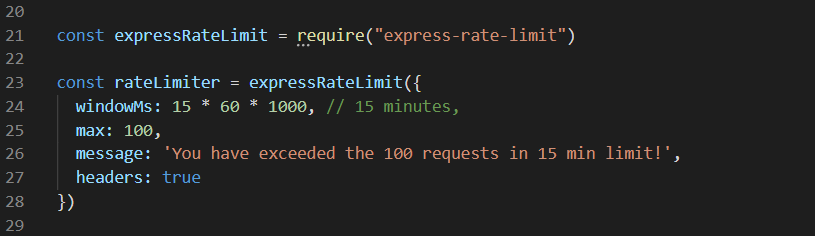
•Res.sendfile - to serve files with the response.

Since these two routes are not implemented with a rate-limiting mechanism, it allows an attacker to perform Denial of service attacks by sending a large number of crafted HTTP requests. So, this results in the app to crash by exhausting the available server resources.

### Patch/Mitigation

Using a rate-limiting library as a middleware can prevent Denial of service attacks. So here, I created a function to restrict max 100 requests per 15 minutes and return a response when the max limit is exceeded. And, I set the headers parameter to be accurate, to add an appropriate header to the reaction such as,

* X-RateLimit-Limit – shows the enforces limit
* X-RateLimit-Remaining – shows current usage
* Retry-After - shows waiting time before retrying



*Figure 3*



*Figure 4*

## 2.3 | Cross-Site Scripting Vulnerability

A Cross-Site Scripting vulnerability exists in this program because user-provided values are used in processes in the /PageGhost/worker.ts file. The vulnerability was detected with the assistance of lgtm.com



*Figure 4*

The source of the issue can be considered the same for both detected vulnerabilities. In this case, the input can be regarded as the message which will be passed on to a variable called data using addEventListener function at line 157 in the "worker.ts" file. In line 238, there are four characteristics (data, id, sender, timestamp) that are derived from the message. In the 239th line, the URL of the sender is obtained and passed to the linkify function.

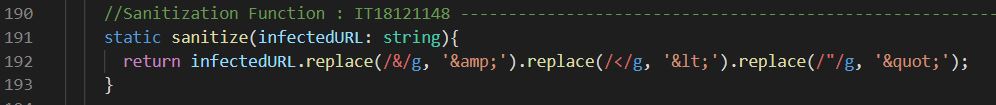
### How it will affect the security of the software

In this code, two Cross-site Scripting vulnerabilities are existing because the user-provided values are directly processed and written to the webpage without any sanitization procedures. This particular type of Cross-Site Scripting is known as DOM-based Cross-Site Scripting. To execute a Cross-Site Scripting attack based on DOM, you need to insert data in a source to propagate it to a sink, triggering arbitrary JavaScript execution. The most popular source for DOM XSS is the URL, usually accessed with the item "Window.location." An attacker will create a link in the query string and fragment portions of the URL to send a victim to a vulnerable page with a payload.

When inspecting the linkify function, we can find out the Cross-Site Scripting vulnerability exists within the function. In lines 192 and 197, we can find out that the value of the "URL" variable (the variable that should be initially passed into the function) is given to the "href" element using the setAttribute () function. This can be considered vulnerable because any value that is inserted into the linkify function is directly passed into the href element. The attackers can exploit this vulnerability by inserting malicious payloads in the message. Eventually, it will be directly processed when the value is passed to the "href" element in line 192 and 197.

### Patch/Mitigation

I found out that an easy and effective method is to mitigate this vulnerability by sanitizing the "url" variable in the linkify function. To do this, a sanitization function called "sanitize()" is implemented. The sanitization function is created in line 191 in the "worker.ts" file.



*Figure 5*

This function replaces characters that are considered to be used in a Cross-Site Scripting payload as well as characters that are inappropriate in a legitimate URL, to their respective HTML name codes.

The “sanitize ()” function is called in the linkify function to sanitize the initial “URL” variable in line 196. The sanitized value is saved in the “sanitizedURL” variable. Then it is being used in lines 199 and 204 to pass the values into the href element using setAttribute function without the Cross-Site Scripting vulnerability.

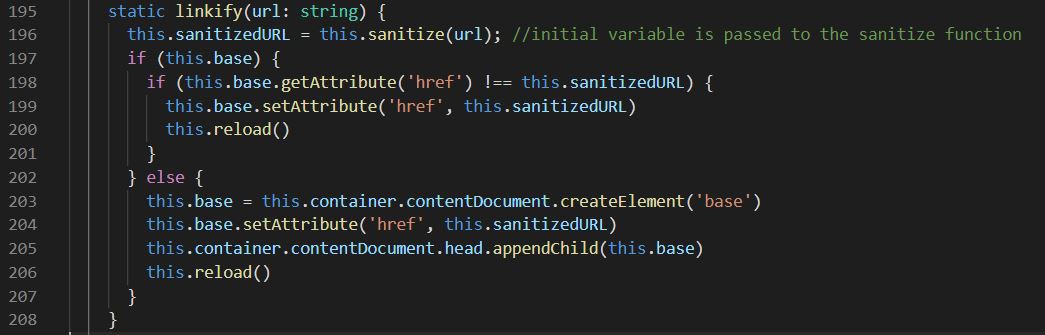


Figure 6

Furthermore, the same mechanism can be used in the source by sanitizing the sender URL before passing it to linkify in the original update () function. Still, it can be considered to be much more secure to patch the vulnerability in the linkify function because several other functions can use this function throughout this software.

# References

Injectify.js.org. 2020. *Injectify Documentation*. [online] Available at: <https://injectify.js.org/> [Accessed 30 April 2020].

Javascript, S. and ADVICE, I., 2020. s*anitizing user Input Before Adding It To The DOM In Javascript*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/2794137/sanitizing-user-input-before-adding-it-to-the-dom-in-javascript> [Accessed 30 April 2020].

Cheatsheetseries.owasp.org. 2020. *Cross-Site Scripting Prevention · OWASP Cheat Sheet Series*. [online] Available at: <https://cheatsheetseries.owasp.org/cheatsheets/Cross\_Site\_Scripting\_Prevention\_Cheat\_Sheet.html> [Accessed 1 May 2020].

Academy, W. and scripting, C., 2020. *What Is DOM-Based XSS (Cross-Site Scripting)? Tutorial & Examples | Web Security Academy*. [online] Portswigger.net. Available at: <https://portswigger.net/web-security/cross-site-scripting/dom-based> [Accessed 1 May 2020].

Microsoft.github.io. 2020. *Location | Typescript - V3.7.5*. [online] Available at: <https://microsoft.github.io/PowerBI-JavaScript/interfaces/\_node\_modules\_typedoc\_node\_modules\_typescript\_lib\_lib\_dom\_d\_.location.html> [Accessed 2 May 2020].

Blog.josequinto.com. 2020. *Typescript Functions To Get Current Domain, Site Collection, And Site Urls With Window. Location Fallback | José Quinto*. [online] Available at: <https://blog.josequinto.com/2017/03/09/typescript-functions-to-get-current-domain-site-collection-and-site-urls-with-window-location-fallback/> [Accessed 1 May 2020].

Rapidtables.com. 2020. *HTML Charset | HTML Character Codes & Symbols*. [online] Available at: <https://www.rapidtables.com/web/html/html-codes.html> [Accessed 2 May 2020].

Medium. 2020. *Securing Node Web Applications*. [online] Available at: <https://medium.com/@andy.neale/securing-node-web-applications-5d5f9bc21926> [Accessed 1 May 2020].

Cheatsheetseries.owasp.org. 2020. *DOM Based XSS Prevention · OWASP Cheat Sheet Series*. [online] Available at: <https://cheatsheetseries.owasp.org/cheatsheets/DOM\_based\_XSS\_Prevention\_Cheat\_Sheet.html> [Accessed 1 May 2020].

npm. 2020. *Node-Esapi*. [online] Available at: <https://www.npmjs.com/package/node-esapi> [Accessed 2 May 2020].

npm. 2020. *@Types/Node*. [online] Available at: <https://www.npmjs.com/package/@types/node> [Accessed 2 May 2020].

Google Books. 2020. *Essential Node.Js Security*. [online] Available at: <https://books.google.lk/books?id=vU\_6DQAAQBAJ&pg=PA53&lpg=PA53&dq=require(%27node-esapi%27)&source=bl&ots=SRkiAxU1q\_&sig=ACfU3U2QEPM05-YKsKxuxcJdGLOhBgcGHQ&hl=en&sa=X&ved=2ahUKEwibjMnAqZXpAhXRF3IKHR8IDTMQ6AEwA3oECAgQAQ#v=onepage&q=require('node-esapi')&f=false> [Accessed 2 May 2020].

GitHub. 2020. *BE-717 Code Quality Fix (#66) · Hyperledger/Blockchain-Explorer@Dac1dce*. [online] Available at: <https://github.com/hyperledger/blockchain-explorer/commit/dac1dce8e130fe108490155a67317dba50ca2ffe> [Accessed 1 May 2020].

Lgtm.com. 2020. *LGTM*. [online] Available at: <https://lgtm.com/projects/g/samdenty/injectify/snapshot/15a4b6fe4ee900654975742fa88e52d1a959d97f/files/src/main.ts?sort=name&dir=ASC&mode=heatmap#x7f761a20f9da2c00:1> [Accessed 2 May 2020].

npm. 2020. *Express-Rate-Limit*. [online] Available at: <https://www.npmjs.com/package/express-rate-limit> [Accessed 1 May 2020].