# SECURITY CONSULTANT REPORT

## INTRODUCTION:

As a freelance web security consultant, I have been hired to conduct a full-scale investigation of Brita Dawson's lyftknockoff web application. It is the case that the web application contains confidential information pertaining to customers who use it, thus it is of high importance that the web application does not leak information to the public and is secure against hackers. The eventual goal of the investigation is to expose at least three vulnerabilities in the security and privacy of Brita Dawsons web application.

## METHODOLOGY:

Initial stages, trying basic injection attacks:

http://warm-wildwood-31279.herokuapp.com/passenger.json?username=dwR3TbOH

http://warm-wildwood-31279.herokuapp.com/passenger.json?username[$ne]

http://warm-wildwood-31279.herokuapp.com/passenger.json?username[$ne]=b

Basic check for XSS vulnerability:

```
curl --data "username=JANET&lat=<scrJANET&lat=<script>alert("hello")</script>&lng=0"
http://warm-wildwood-31279.herokuapp.com/rides
```

Sending a post request that injects code to the server; did not work.

As also the formerly listed injection attack.

The next method to assess the security of this web application involved looking through reference material on cyber-security. It became obvious where this app blindly leaks non-trivial information and where the app fails to uphold regular security practices and standards.

## ABSTRACT OF FINDINGS:

We now present a high level overview of the privacy issues identified:

The findings from this study reveal that the web application is unsecure on moderate to high levels. The first main cause for concern is the issue that the website reveals the location of a user at a certain time, to any other user. This display of information is bad, and could cause a confidentiality issues.

The second main issue would pertain to the lack of protection of the web application when a service has been updated or deprecated. The web application is susceptible to crashing at any time due to it's inefficient plan to stay maintained through different programming ecosystems.

Although this application is bad in terms of security, it still has some benefits as it is invulnerable to basic hacking attacks, such as injections. Overall, I would give the application an 8/10, 8 being on the high side on a range of how urgent changes need to be made.

## ISSUES FOUND:

## Sensitive Data Exposure

**In an attack giving in detail below, by the OWASP:**

> *"A site doesn't use or enforce TLS for all pages or supports weak encryption. An attacker monitors network traffic (e.g. at an insecure wireless network), downgrades connections from HTTPS to HTTP, intercepts requests, and steals the user's session cookie. The attacker then replays this cookie and hijacks the user's (authenticated) session, accessing or modifying the user's private data. Instead of the above they could alter all transported data, e.g. the recipient of a money transfer."*

**This web application is susceptible to this form of attack. This web application serves through an insecure protocol: HTTP; thus making it easy and susceptible to these kinds of attacks. Compounding the issue is the nature of the information on the database (i.e, location of passengers using the lyftknockoff app). This form of threat would have to go under the category of exposure of sensitive data.**

# WHERE ARE THE VEHICLES?

JANET was looking for passengers at 10, 10 on Thu Jan 01 1970 00:00:00 GMT+0000 (Coordinated Universal Time)

JANET was looking for passengers at 90, 67 on Thu Jan 01 1970 00:00:00 GMT+0000 (Coordinated Universal Time)

JANET was looking for passengers at 90, 67 on Thu Jan 01 1970 00:00:00 GMT+0000 (Coordinated Universal Time)

suFKyeZg was looking for passengers at 98, 11 on Thu Jan 01 1970 00:00:00 GMT+0000 (Coordinated Universal Time)

suFKyeZg was looking for passengers at -111, 99 on Thu Jan 01 1970 00:00:00 GMT+0000 (Coordinated Universal Time)

suFKyeZg was looking for passengers at -111, 99 on [object Undefined][object Undefined]

nZXB8ZHz was looking for passengers at 56, 45 on 1554990466633

nZXB8ZHz was looking for passengers at 56, 45 on 2019-4-11T14:2:5.83Z

JANET was looking for passengers at 99, 77 on 2019-4-11T14:13:6.223Z

JANET was looking for passengers at 99, 77 on 2019-4-11T14:16:19.630Z

JANET was looking for passengers at 99, 77 on 2019-4-11T14:20:31.460Z

**Basic use of the app even reveals information that is confidential to other users, such as the location (lat and lng) of where they were at a certain time. This form of display is very bad, and so would have high cause for concern.**

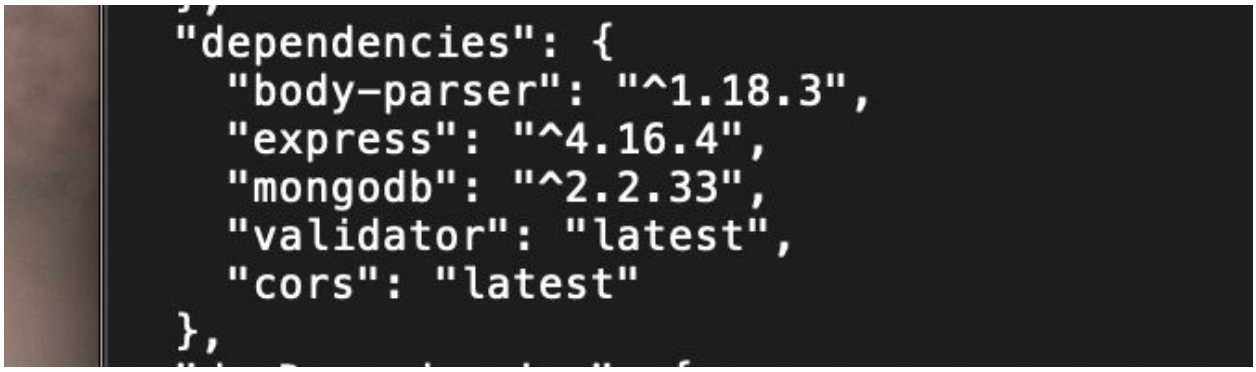
**Severity of issue: High**

**Viable resolutions for these issues are:**

- **Since the application uses a NoSql database, it is easy to alter what is being stored. I recommend another level of encryption before the data is at rest on the database.**
- **Thoroughly determine what information are sensitive to users and pick the relevant non-confidential information to share.**

## Using Components with Known Vulnerabilities

**No set up of a firewall to prevent intrusions in the case of an attack if the web application is not updated over a time. This application is susceptible to be falling out of date due to the many versions of components installed to run the web application(ie npm, nodejs). Failure to update these versions may lead to a vulnerability. This strategy is referred to as Virtual patching, and is recommended by OWASP as a way to protect applications that are not updated constantly from attacks. To redu**




**Since the packages used in this application have wide scale use among developers, the potential for subsequent versions of them being injected by attacks is quite low. So not updating the web application may not be very costly for this product, in terms of being hacked, this may lead us to conclude the severity of the issue is Low. However, the lack of a firewall is somewhat disconcerting, as the contents on this web application are confidential to users. The application failed to display any form of security in regards to this.**
**Severity of issue: Medium**
**Viable resolutions for these issues are:**

- **the recommended strategy by OWASP which is to virtually patch your web application with a firewall.**
- **keep track of all the dependencies on the client side and server side for updating.**

**A plus is the application keeps track of every user that logs in. It is good to audit the login of every user, according to OWASP an example event in security saw a software team had their source code scraped as an untracked attacked breached their system, due to their inability to log users login information.**

## Security Misconfiguration- CORS

**This app does not have any configured security setting for Cross-Origin Resource Sharing(CORS). It may not seem apparent as to why this configuration needs to be ensured, but with HTTP Requests, POST, PUT, DELETE, domains can access other domains resources. Since this app shares a resource with the google API server, it is therefore advisable to regulate the access of content across-domains that could interact with this application.**
**Severity of issue: Low**
**Recommended fix:**

- **Change Access-Control-Allow-Origin: // the specific domains allowed, and not a * //**

**Controlling access to the resources on this application, would minimize it's security risks and keep the website and users secure.**

## Miscellaneous

**On closer inspection of the code, there were no prevention measures put in place on the odd chance the database crashed. So no backup databases. Also, there was no means to prevent overflow of user input to the database.**
**However there was a means as to which injection was prevent by the validation of input into the database. These lines of javascript here ensure the users input is handled the way the server is to store it.**

```
var toInsert = {
    "username":
            info.username,

    "lat":
            latToInt,

    "lng":
            lngToInt,

    "created_at":
            timestamp
};
```

**This is good cause according to the 'IAS on CS', input validation is one of the first methods of defence to injection.**

## CONCLUSION:

As listed above, please pertain to these recommendations to ensure smooth secure running of lyftknockoff:

- **Since the application uses a NoSql database, it is easy to alter what is being stored. I recommend another level of encryption before the data is at rest on the database.**
- **Thoroughly determine what information are sensitive to users and pick the relevant non-confidential information to share.**
- **the recommended strategy by OWASP which is to virtually patch your web application with a firewall.**
- **keep track of all the dependencies on the client side and server side for updating.**

Besides these recommendations, it is commendable that the web application was not vulnerable to malicious injection attacks and was close to being impenetrable through cross-site scripting attacks. Another future consideration would be to keep up to date with the yearly OWASP yearly outtakes on top ten most critical web application security risks.

## REFERENCES:

The references used to successfully complete this study are listed below:


- **MongoDB: NoSQL Injection & Security: http://software-talk.org/blog/2015/02/mongodb-nosql-injection-security/#mongodb-nosql-injection**
- **OWASP Top Ten Projects 2018**
- **Complete Guide to Cross-Origin Resource Sharing; https://www.keycdn.com/support/cors. Date: October 4th, 2018**