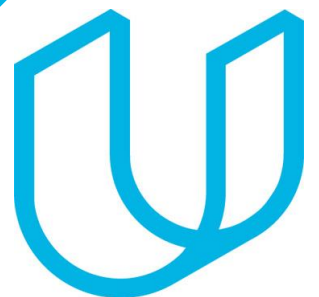


# Udajuicer: Threat Report



**Arwa AlQadheeb**

*November 11, 2022*



# Purpose of this Report:

This is a threat model report for **Udajuicer**. The report will describe the threats facing Udajuicer. The model will cover the following:

- Threat Assessment
  - Scoping out Asset Inventory
  - Architecture Audit
  - Threat Model Diagram
  - Threats to the Organization
  - Identifying Threat Actors
- Vulnerability Analysis
- Risk Analysis
- Mitigation Plan



# Section 1

## Threat Assessment

# 1.1: Asset Inventory

## Components and Functions

- **Web Server:** A computer hardware and software that uses HTTP/HTTPS protocols to deliver either static (i.e., HTML or CSS) or dynamic (i.e., JavaScript or PHP) web content of web pages on client-side browser.
- **Web Application Server:** An application program, typically resides behind a webserver, that offers dynamic web content of a web page upon request after retrieving scripting languages files from a database.
- **Database:** An electronic filing system that combines Database Management System (DBMS) (e.g., MySQL, PostgreSQL, or Microsoft SQL) which manages read/write access to data, and a storage where raw data itself relies (e.g., Scripts files, PII, login credentials, etc.).

# 1.1: Asset Inventory

## **Explanation of How A Request Goes from Client to Server**

In Three-Tire architecture the process goes as follows:

1. The End-user on the client-side uses a web browser to visit specific web page by entering its domain name (e.g., Udajuicer's URL).
2. The DNS resolver, provided by end-user's Internet Service Provider (ISP), translates the given domain name into its corresponding IP address.
3. The web browser establishes a TCP connection to make HTTP/HTTPS request addressed to the IP address of the web server that hosts the requested web page.
4. The web server hosts a set of related HTML pages and files. If the requested web content is static, then the web server delivers the requested content as it is to the web page and directly displayed on the web browser. However, if it is dynamic, then request is forwarded to the special software called application server for further processing.
5. The application server reads the script code and queries the associated database for the required information to be placed inside the HTML page.
6. The web application returns the generated HTML page to the web server.
7. The web server delivers the dynamic web content to the web page and then displayed on the requesting web browser.

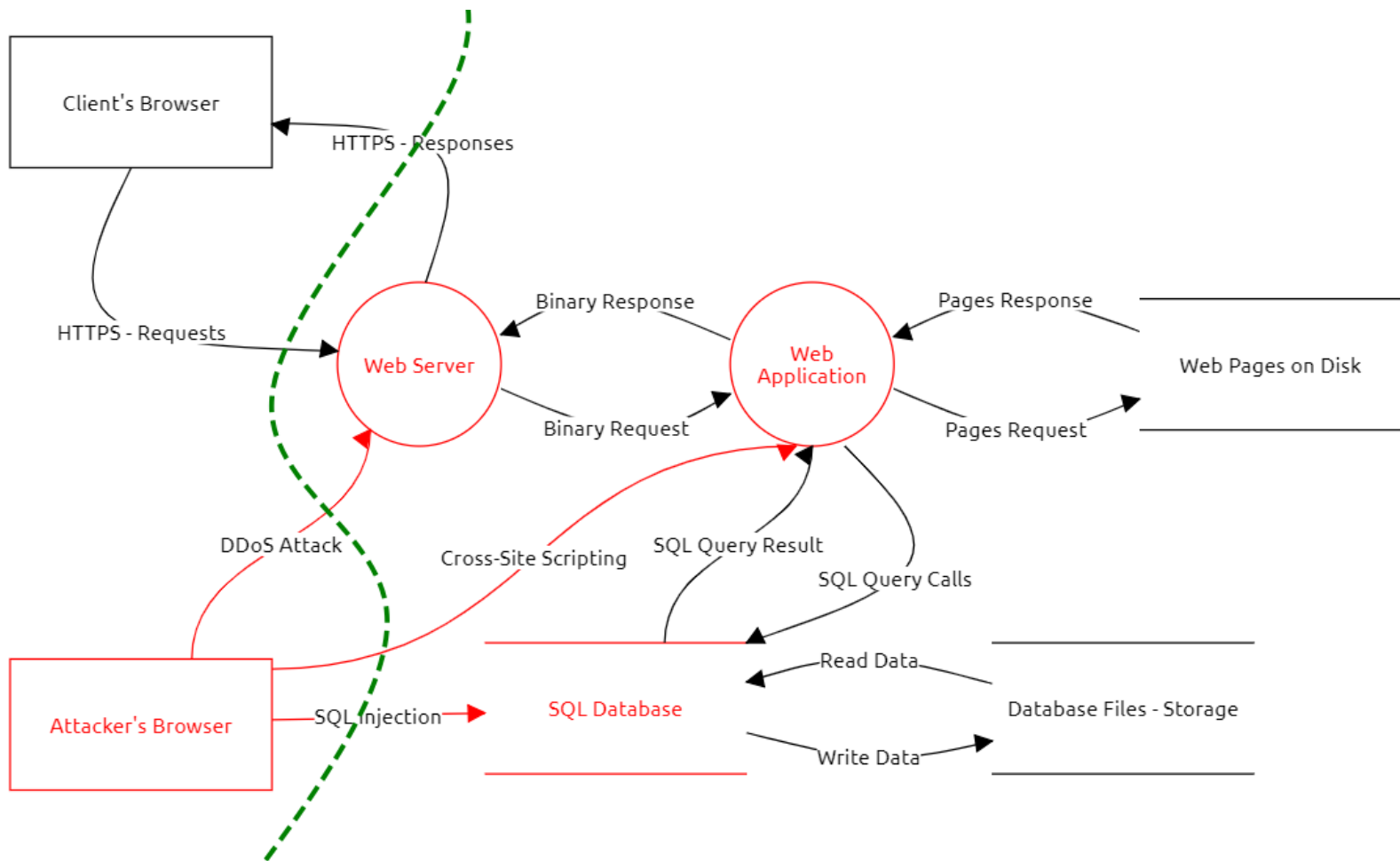
# 1.2 Architecture Audit

## Flaws

Just by looking to the given architecture diagram, we can see that essential security components are missing, resulting a poorly structured and vulnerable architecture. For example:

1. No redundant architecture was built to avoid having single point of failure. In our case, for example, if Udajuicer's web server failed for any reason and there were no secondary web server to take over, then the online web application will not be reachable and available to customers.
2. No network firewall was placed to protect the perimeter of the network by monitoring, allowing, and blocking incoming and outgoing Internet traffic.
3. No web application firewall was placed to protect the web application by analyzing HTTP/HTTPS requests, then detecting and blocking malicious requests before reaching the web application.
4. No Demilitarized Zone (DMZ) was implemented to separate the public-facing web server from other components within Udajuicer's network (i.e., web application and database).

# 1.3 Threat Model Diagram



# 1.4 Threat Analysis

## What Type of Attack Caused the Crash?

Udajuicer's online application crashed due to *HTTP Flood*, a type of *Distributed Denial of Service (DDoS)* attack, that was aimed to overwhelm the web server of Udajuicer with numerous HTTP requests which each one of them meant to be as processing-intensive as possible.

## What in the Logs Proves Your Theory?

After investigating the logs from right before the application went down, I found that multiple errors were logged, each of which was associated with HTTP GET request to [www.udajucier.com](http://www.udajucier.com). In addition, I have noticed that those requests were sent from different IP addresses (i.e., different machines controlled by the attacker) and at the same exact date and time (i.e., 2020/04/02 18:53:27) which proves that the attack that caused the crash is HTTP Flood.



# 1.5 Threat Actor Analysis

## Who is the Most Likely Threat Actor?

From my point of view, I can be almost certain that the threat actor is a *Script Kid* who obviously chose to exploit the insecure architecture to cause disruption rather than achieving specific destructive objective.

## What Proves Your Theory?

The nature of the attack and the resulted impact indicate that it was performed by unskilled cyber criminal who left a trail of breadcrumbs behind (i.e., log files) unlike an experienced cyber criminal who would leave no trace behind. In addition, any skilled attacker would not be satisfied with only taking down the online application, and would go far beyond, performing more sophisticated attack (e.g., SQL injection) and do more harm from the inside, or stealing sensitive information (e.g., customers' PII, credit cards information, delivery address, etc.).



## **Section 2**

# Vulnerability Analysis

## 2.1 SQL Injection

### Login


Email

' or 1=1 --

Password


' or 1=1 --

Forgot your password?

 Log in

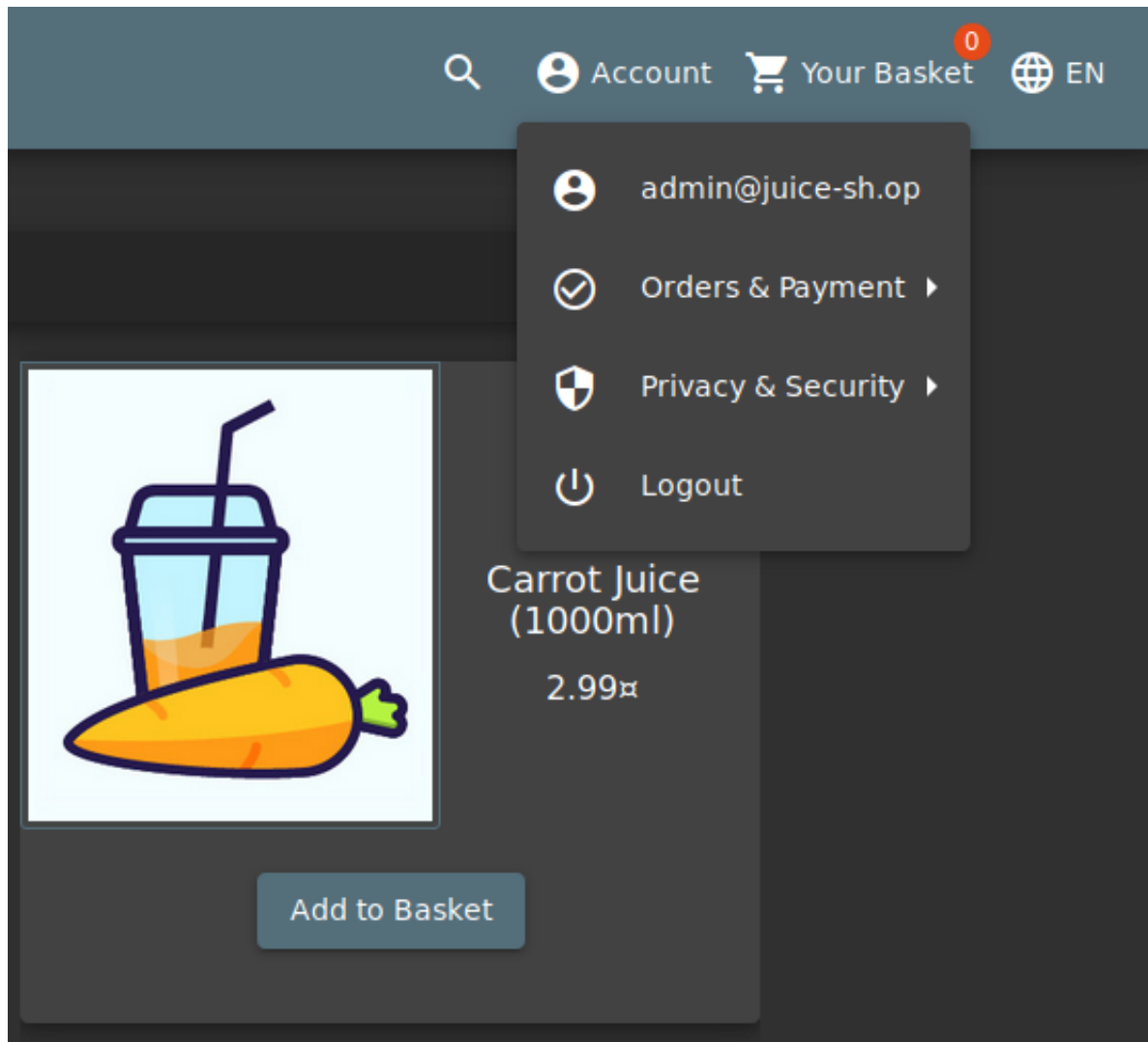
☐ Remember me

or

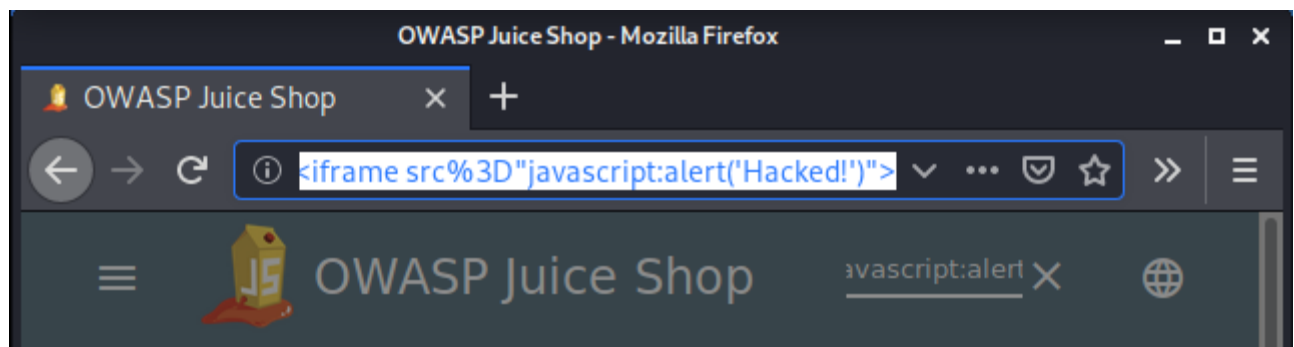
 Log in with Google

Not yet a customer?

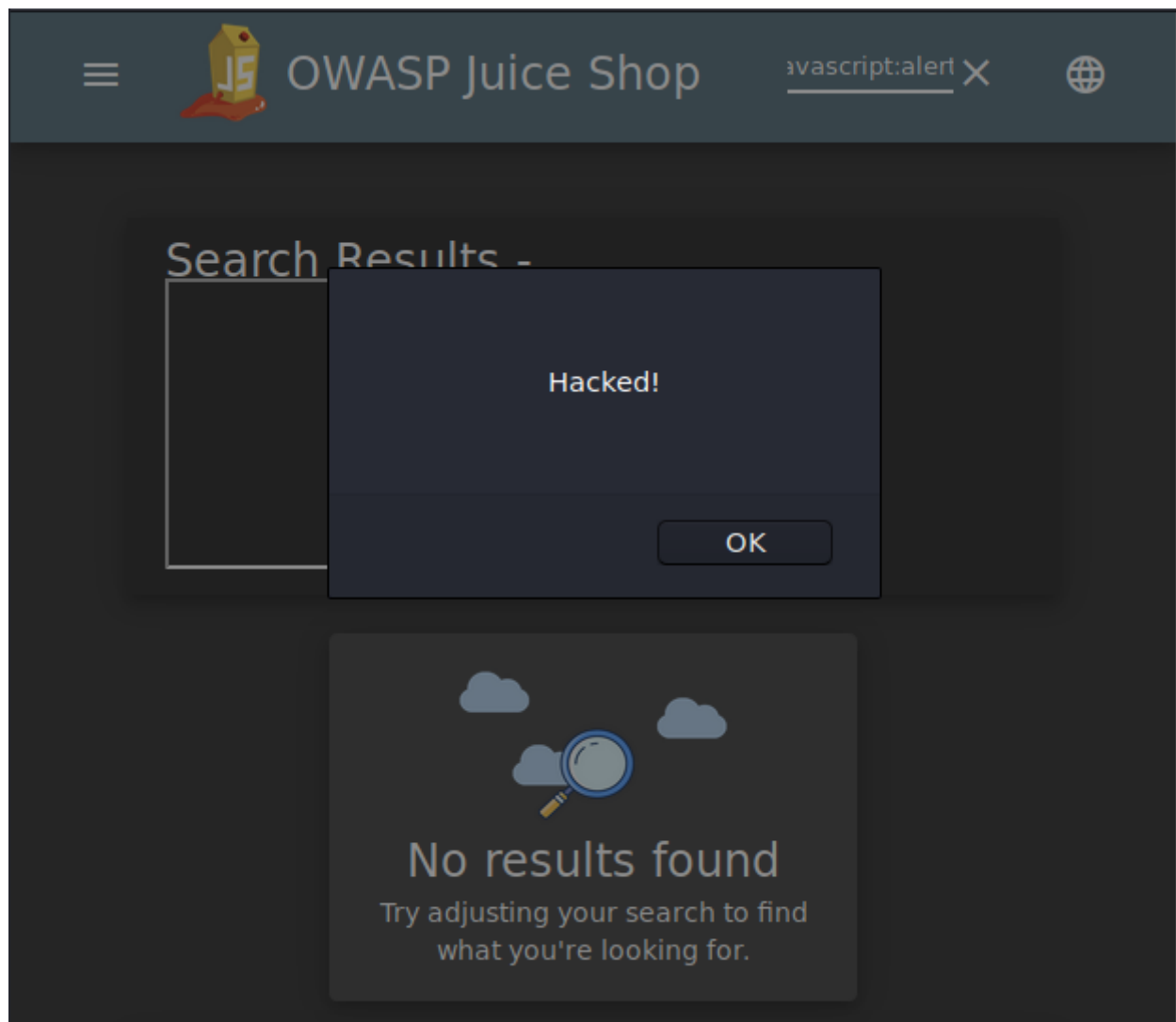
## 2.1 SQL Injection



## 2.2 XSS



## 2.2 XSS



# Optional Task:

## **Extra Vulnerabilities**

- Security Misconfiguration - Error Handling.



# **Section 3**

## Risk Analysis



# 3.1 Scoring Risks

Risk	Score <i>(1 is most dangerous, 4 is least dangerous)</i>
DDoS	1
Insecure Architecture	3
SQL Injection	2
XSS Vulnerability	4

## 3.2 Risk Rationale

### Why Did You Choose That Ranking?

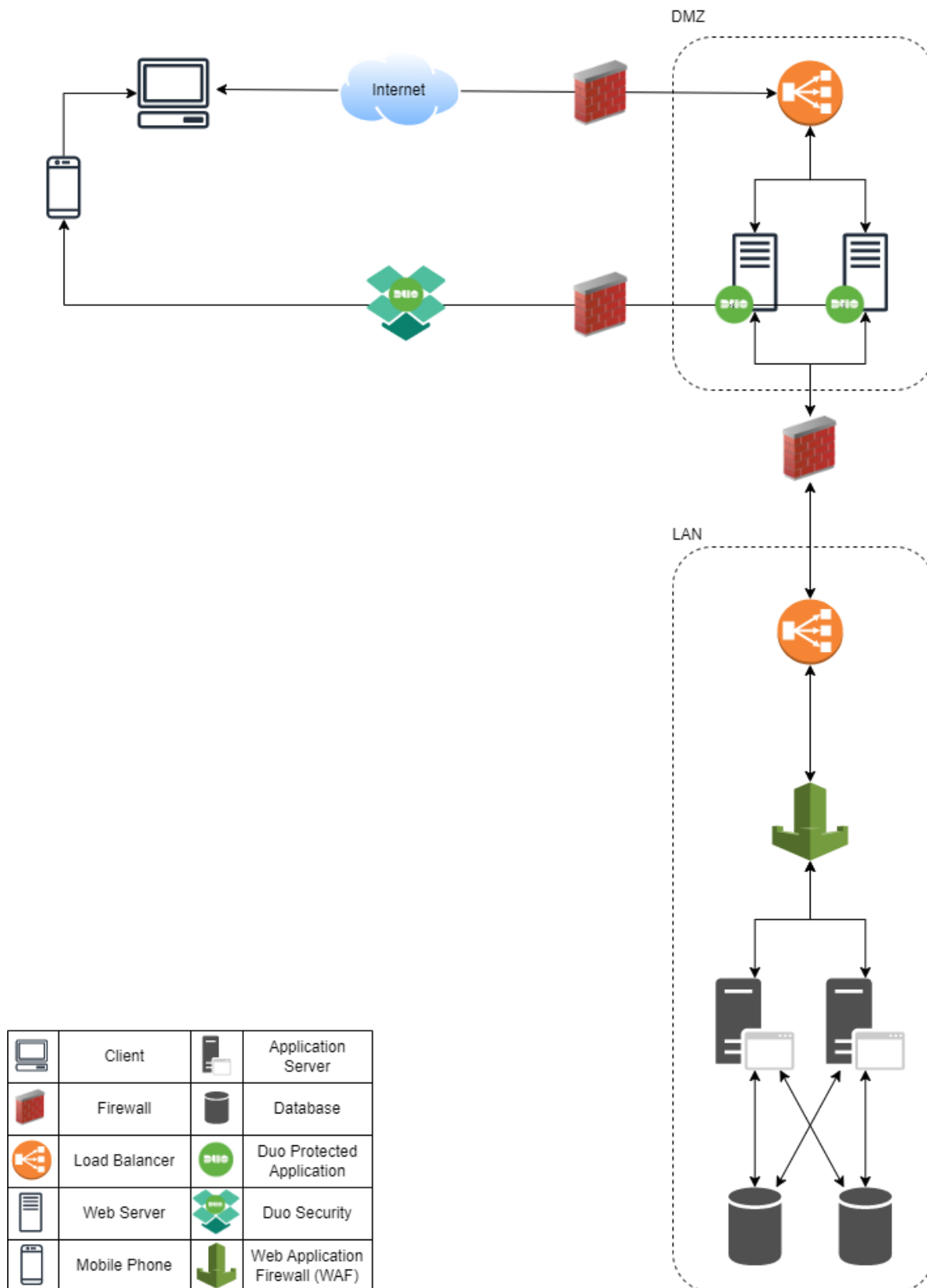
Taking into consideration that Udajuicer's online application was the last resort for the owner to save his franchise from going bankrupt due to the pandemic, I have considered HTTP Flood to be the most dangerous (i.e., score 1) attack because if the online application was continuously taken down and gone unreachable to customers, then as a result they will turn to Udajuicer's rival (i.e., Super Smoothies) causing the subject business a great financial loss. On the second place, I scored 2 for SQL injection because if an attacker managed to lay hands on the database that contains customers' PII, delivery addresses, credit cards numbers and so on, a series of consequences, including sensitive data exposure, would result from such an attack causing loss of customers' trust and reputational damage. Moreover, I have given insecure architecture a score of 3 because for example, in the event of SQL Injection attack and customers' PII was not protected due to developer negligence to apply proper encryption, then its damage would be far-reaching to the business. Lastly, a score of 4 was given to Cross-Site Scripting (XSS) attack because in case the compromised victim was a customer, then the attacker would be able to perform what an actual customer would do. However, it would more damaging if the compromised victim was a privileged user, then more harm would be done from the inside.



## **Section 4**

# Mitigation Plan

# 4.1 Secure Architecture



## 4.2 Mystery Attack Mitigation

### What is Your Mitigation Plan?

As I have identified in [1.4](#), the mystery attack was HTTP Flood that could be mitigated by the suggested **secure architecture** in [4.1](#) which included the below components that could assist with mitigating not only HTTP Flood, but also any type of DDoS attack:

1. Redundant Architecture: ensures uptime by preserving availability security property, such that if one component was flooded with too many requests, then there is another which can substitute and ensure that operations are up and running.
2. Load Balancers: ensures that inbound traffic is distributed evenly among resources to avoid having one resource being overloaded with processing too many requests.
3. Web Application Firewall (WAF): adds an extra layer of security to Udajuicer's online application by implementing rate limit rule that restrict the rate of incoming requests for each originating IP address, and apply the rule action (i.e., blocking/dropping traffic) on IP addresses that exceed the pre-defined limit.

## 4.3 SQL Injection Mitigation

### What is Your Mitigation Plan?

1. Web Application Firewall (WAF): used not only for mitigating DDoS attacks, but also to mitigate SQL injection attack. WAF acts as a barrier between Udajuicer's web application and the open Internet by filtering, monitoring, and blocking incoming malicious requests. The mechanism of WAF relies on inspecting and matching incoming requests against blacklist of known attacks and then deciding whether to block them or not.
2. Prepared Statement with Parameterized Queries: used to prevent malicious unexpected queries by taking user's input as variable and place it into a pre-built query and then execute it. In this way we mitigate the risk of malicious query being executed on the database which may result in sensitive data exposure, privilege escalation, or manipulation of the database.

## 4.4 XSS Mitigation

### What is Your Mitigation Plan?

Same as SQL Injection, we must never assume that user's input is secure and can be trusted. Therefore, mitigating Cross-Site Scripting (XSS) revolves around inspecting user input by applying the following techniques:

1. Input Sanitization: ensures that user input is clean by inspecting data supplied from the user and removing any characters or strings that might be malicious.
2. Input Validation: ensures that user input is genuine by taking data supplied from the user and matching it against pre-determined allow or deny lists.
3. Escaping: ensures that user input is safe by stripping out characters (e.g., tags or quotes) found in data supplied from the user to prevent them from being seen and executed.
4. Web Application Firewall (WAF): same as in [4.3](#).