Udajuicer - Threat Report

January 4, 2024

Introduction

This project serves to address issues with an insecure web application at Udajuicer - a very big juice shop in the world. The site would constantly go down and the people at Udajuicer weren't sure what the issue was.

Two things to note are these: web applications tend to be rated as the best attack vector for an attacker, primarily for two things:

- Web applications are difficult to build securely (without proper knowledge of all the moving parts that constitute a web application)
- Web applications found provide access to sensitive data found in an internal network segment.

As a security analyst from a world-renown cybersecurity consulting firm, I will get to the bottom of the issue and find out why the Juice Shop site keeps going down.

- I will build a threat model for Udajuicer's website.
- Using an internal threat model template provided, I will go through the process of helping Udajuicer mitigate their current issue and build a secure application.

1 Assessment

Despite being a big company, it is quite alarming that the system architecture at Udajuicer is poorly designed as shown in FIG 1.

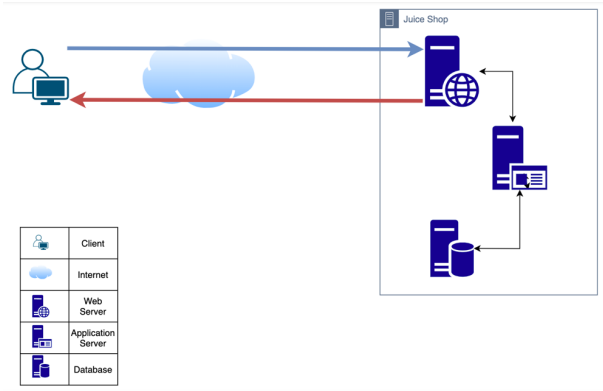

Figure 1: Udajuicer's Poorly Designed Architecture.

1.1 Asset Inventory

The first part of our threat model is being able to identify all the assets at Udajuicer. Looking at the architecture diagram, we can identify three components of the architecture: Web Server, Database Server and Application Server.

1.1.1 Web Server

A web server is a software application or hardware device that stores, processes, and serves website content to users over the World Wide Web, using the Hypertext Transfer Protocol (HTTP) to communicate. Key functions of a web server include:

- **Processing Requests**: Web servers handle incoming requests from clients (typically web browsers) and respond by serving the requested content.
- **Storing Content**: Web servers store website files, such as HTML documents, images, CSS stylesheets, and other multimedia files.
- **Communication**: Web servers communicate with clients using the HTTP (Hypertext Transfer Protocol) or its secure version, HTTPS. HTTP defines how messages are formatted and transmitted, and HTTPS adds a layer of security through encryption.
- **Hosting**: Web servers host websites and make them accessible to users on the Internet. Websites can be hosted on dedicated servers, virtual private servers (VPS), or through cloud-based services.

These dynamic functionality makes web servers an interesting target for attackers. Web servers are very common, and anyone can set one up, but doing it correctly without leaving doors open for attackers is a challenge.

1.1.2 Application Server

An application server is a software framework that provides an environment for running and managing applications. Key functions of an application server include:

- **Middleware Services**: Application servers often provide middleware services, such as messaging services, transaction processing, and connection pooling, to facilitate communication and coordination between different components of an application.
- **Application Execution**: They host and execute application code, allowing developers to deploy and run their applications in a controlled and managed environment.
- **Security**: They implement security features, including authentication and authorization mechanisms, to control access to applications and sensitive data. **Integration with Databases**: Application servers often integrate with database servers to retrieve and store data, providing a seamless connection between the application layer and the database layer.

Application server supports various communication protocols, such as HTTP, HTTPS, and others, making them suitable for web-based applications.

1.1.3 Database Server

A database server is a computer system that is dedicated to managing, storing and retrieving data from databases, and providing access to a database. Key characteristics and functions of a database server include:

- **Data Storage**: The database server stores data in a structured format, typically using a relational database management system (RDBMS) like MySQL, PostgreSQL, or Microsoft SQL Server.
- **Data Retrieval**: It allows users or applications to retrieve and manipulate data stored in the database through SQL (Structured Query Languages) queries and transactions.

In a typical architecture, applications or services interact with the database server to store and retrieve data. The database server manages the underlying data storage, indexing, and retrieval processes, providing a centralized and organized structure for efficient data management.

How A Request Goes from Client to Server

The general purpose of the Udajuicer's insecure web application is to allow users to make online orders for delivery or pickup. This process typically involves several components working together. Let us give a simplified overview of the how a a request gets from client to server and back. There are three components to a web application: the front-end client-side, back-end or server-side.

The interaction between the client, web server, application server, and database server in a web application typically follows a multi-tier architecture. Here's a step-by-step overview of how these components interact:
The Client-Side: This is the device that is being used to connect to the web site. It includes the client's device and web browser.

- **Client Interaction (Client side)**:
  - The user with an active internet connection from an Internet Service Provider (ISP), Wi-Fi, or a mobile data, launches a web browser on their device (such as Chrome, Firefox, Safari, or Edge).
  - The user enters the Udajuicer's URL into the address bar of the web browser. A URL is a human-readable web address that identifies the location of a resource on the internet. It typically starts with "http://" or "https://," followed by the domain name.
  - The DNS (Domain Name System) translates the domain name into an IP address, which computer can understand. This is the network address of the web server that is hosting Udajuicer's online application.
  - The browser constructs an HTTP (or HTTPS) request, including the IP address and the path specified in the URL and sends it to the web server associated with the obtained IP address.
- **Web Server Interaction (Web Tier)**:
  - The web server receives the HTTP request from the client and processes it. It locates the requested resource based on the path in the URL and any additional parameters.
  - The web server may handle static content (like HTML, CSS, and images) directly and generates a HTTP response to the client's request for such resources.
  - The web browser receives the HTTP response and renders the web page or resource, displaying it on the user's device.
  - The user can now interact with the displayed web page, click on links, submit forms, and perform various actions.
- **Dynamic Content and Application Logic**:
  - If the request involves dynamic content or requires application logic (which is our case), the web server forwards the request to the application server.
  - The application server hosts the business logic of the web application and processes the request.
- **Application Server Interaction (Application Tier)**:
  - The application server interacts with the database server to retrieve or update data as needed for the request.
  - It may perform tasks such as user authentication, authorization, and execution of application-specific business logic.
- **Database Server Interaction (Data Tier)**:
  - The application server sends queries or commands to the database server to retrieve or update information stored in the database.
  - The database server processes these queries and returns the requested data to the application server.

- **Application Server Response (Application Tier)**:
  - The application server formulates a response based on the processed data and application logic.
  - The response may include dynamic content, such as user-specific information, order details, or any other relevant data.
- **Web Server Response (Web Tier)**:
  - The application server sends the response back to the web server.
  - The web server receives the response from the application server and sends it back to the client as an HTTP response.
- **Client Response**:
  - The client's web browser receives the response and updates the user interface accordingly.

Throughout this interaction, various protocols come into play:

- **HTTP/HTTPS**: Used for communication between the client and both the web server and application server.
- **Database Protocol**: Specific to the type of database system (e.g., MySQL, PostgreSQL, MongoDB) and used for communication between the application server and the database server.

2 Architecture Audit

Now that we've identified all the components that make up Udajuicer's web application, let?s conduct an architecture review. FIG 2 is an example of a secure architecture. We will use it as a baseline for addressing the flaws in Udajuicer's architecture.

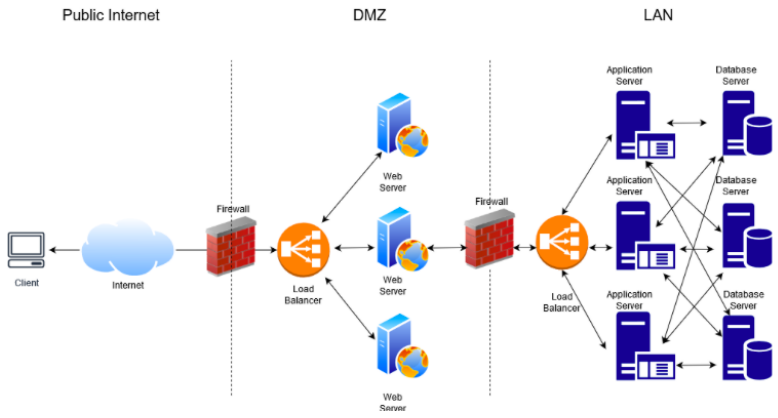

Figure 2: Example of a Secure Infrastructure

2.1 Architecture review

Adopting the concept of network segmentation, I have identified four tiers for Udajuicer's web application.

1. **Public Tier**: There's a public part of the architecture which includes the internet. Resources that reside on the internet cannot be trusted and, therefore, represent a tier of high-risk and trust.

2. **Semi-public (DMZ)**: This include the web servers housing Udajuicer's online application and DNS servers necessary for translating resources into corresponding IP addresses. DMZ systems are a great risk of compromise by the nature of being connected to a public tier.

3. **Middle Tier**: This includes the application server which separates the two ends of a communication, ensuring that he web server cannot communicate with database server and vice-versa.

4. **Private Tier**: This refers to the internal network segment – the company's database. This system represents critical functionality and should have no direct connection between them and the internet. It is of higher trust and lower risk.

2.2 Udajuicer's Architecture Flaws

The overall goal of an architecture design should be to protect internal network from external attacks, provide defense-in-depth through a tiered architecture, and control flow of information between systems. The following flaws have been discovered:

- **Firewalls**: The architecture has no Firewalls. Network segmentation helps to separate critical infrastructures form the environment, resulting in a reduced capacity of communication and connection. This can be achieved using a firewall located:
  - Between the internet and other networks (the web server).
  - Between the semi-public (web server) and private network (database and application server).
  - Between tiers of varying trusting levels.

  Whether it is host-based or network-based, different types of firewalls exit. Web application firewalls and email filtering.

- **Content Delivery Network (CDN)**: The architecture has not adopted CDN system. A CDN is a distributed group of

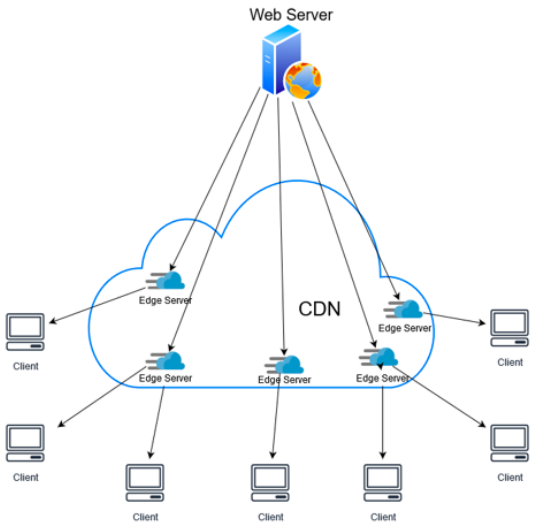

Figure 3: Secure Infrastructure : CDN

edge servers that serve cached content from the origin. Using this, traffic never hits your original server, instead are handled by edge servers. Thus in the event of DDoS attack, the edge servers handle the incoming request even if one gets down, there are multiple others still available. This makes the site or application highly available.

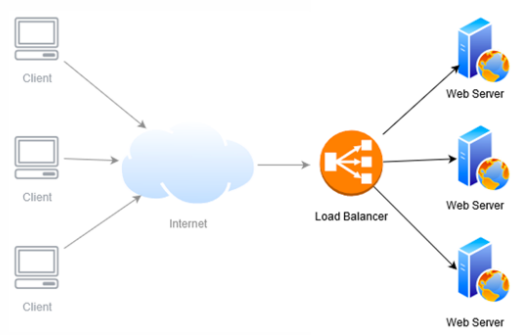

Figure 4: Secure Infrastructure : Load Balancer

- **Load Balancer**: Devices that act as intermediary; distributing network traffic evenly from client devices to servers.
  - It can handle incoming traffic and ensures no server gets over loaded.
  - This gives increased redundancy, less downtime and increased stability.
  - In the event of an attack that aims to overload the web application server, a load balancer can handle the traffic and ensures it is distributed or dropped altogether.

2.3 Threat Model

FIG 5 below is a threat model diagram for a secure web application at Udajuicer. This threat model was designed using OWASP Threat Dragon and the STRIDE model.

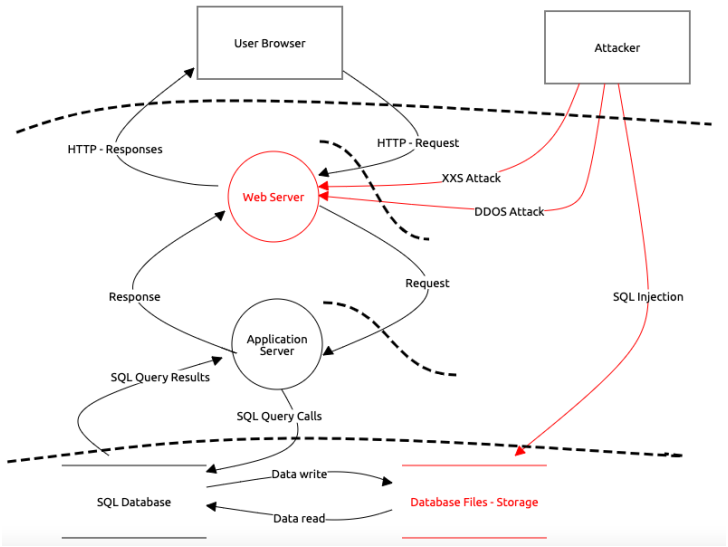

Figure 5: Data Flow Diagram

2.4 Threat Analysis - What Type of Attack Caused the Crash?

We are now ready to identify what was causing Udajuicer's architecture website to crash. The log file below will guide us through our analysis.
Looking at the log file, we see several GET request from diferent IP addresses, asking for Udajuicer's url, at the same time. The attack is HTTP flood attack, a type of Distributed Denial of Service (DDoS).

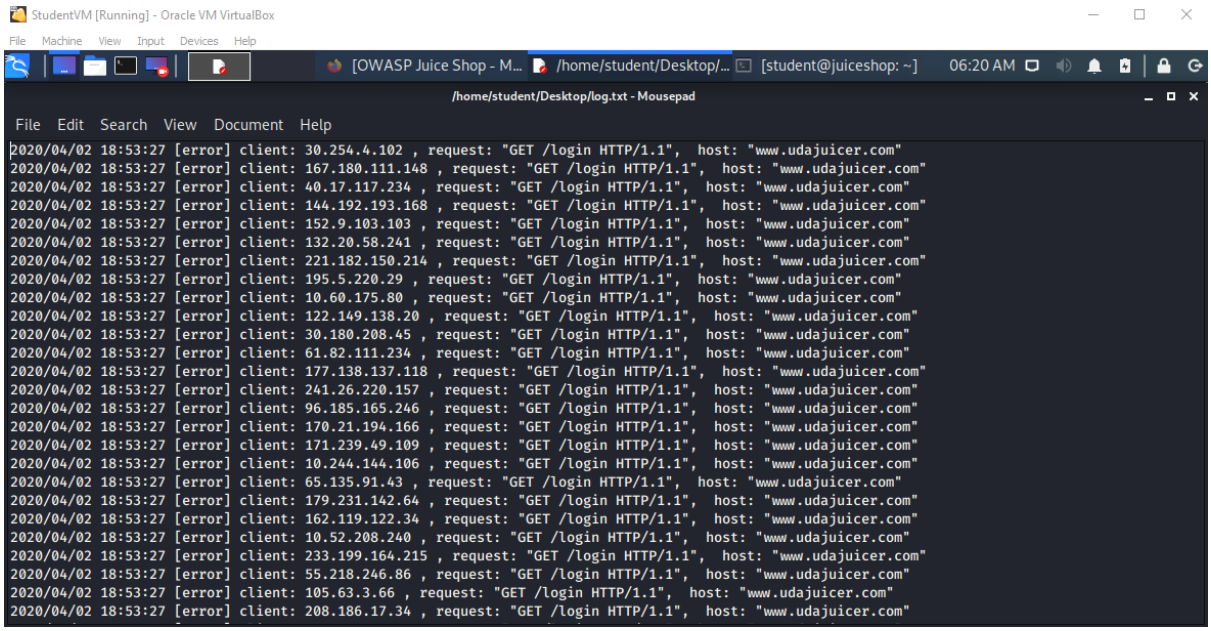

Figure 6: Captured Log Data

This is because a large volume of requests was launched simultaneously (at the same time) from different machines, evidence is in the different IP addresses.

HTTP Flood Attack

HTTP flood attack is a type of DDoS attack that targets web servers by overwhelming them with a massive volume of HTTP requests. The goal of an HTTP flood attack is to exhaust the server's resources, rendering it unable to respond to legitimate user requests.

2.5 Threat Actor Analysis: Who is the Most Likely Threat Actor?

I will say the threat actors here are Script Kiddies. The following points justify my theory

- Script Kiddies are amateur threat actors with no real purpose but to inflict as much damage as possible. Udajuicer's competitors are probably looking for ways to bring the business down (service disruption) since Udajuicer is considered the biggest juicing company. considered the biggest juicing company.
- There are more sophisticated attacks on Udajuicer's web application such as SQL Injection where the attacker may attempt to inject malicious SQL queries through user inputs to manipulate or extract data from the database.
- It is not a Hacktivist as there's no political concern, views or concerns about Udajuicer's products.
- The actor is neither an insider threat as there are easier ways for an insider to harm Udajuicer and go undetected.
- The APT groups, Organized Crime and State-Sponsored can be excluded as Udajuicer does not contain interesting information for these kind of threat actors.

3 Vulnerability Testing

Now that we have made it pass the initial assessment stage, identified the initial attack and shortcomings of the application setup, it is essential to continue with a deeper analysis of the application to see if there

are more vulnerabilities. We would exploit two vulnerabilities: SQL Injection and cross site scripting (XSS). The goal is to gain access into the website as an administrator, exploit the site and render a Hacked alert.

3.1 Testing for SQL Injection Vulnerabilities

Structured Query Language (SQL) injection is a common web attack technique that exploits input validation flaws on applications that accept user input to interact with databases. The image (FIG 7) below shows a screen shot of the commands I used to gain access to Udajuicer's site as an admin

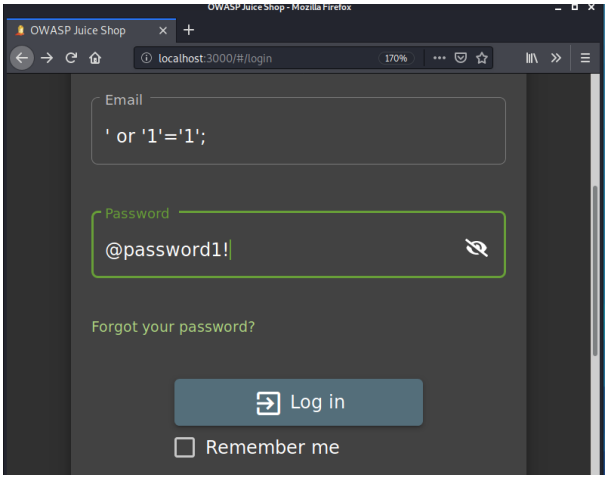

Figure 7: A screenshot of the login credentials I used to gain access as an administrator

One can confirm from the account setting image below that I gained access as an admin into the website

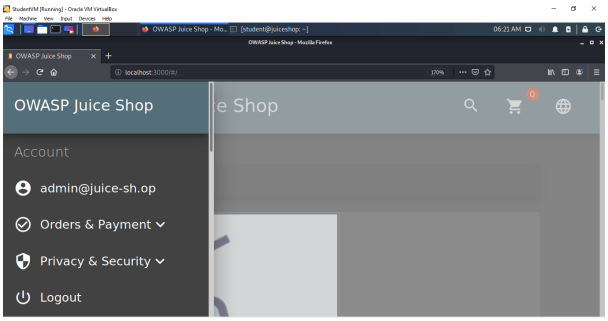

Figure 8: A screenshot showing me as an admin with email address: admin@juice-sh.op

Using the SQL Injection vulnerability, an attacker can change the account settings of the admin thereby gaining a privilege escalation. It is also possible to gain access to user names of all employees at Udajuicer.

3.2 Testing for XSS Vulnerabilities

Cross-site Scripting (XSS) is an attack that against an end-user, leveraging a vulnerability on a server that is not performing input or output validation. Here, the user runs a malicious code supplied by the attacker from the vulnerable scanner.

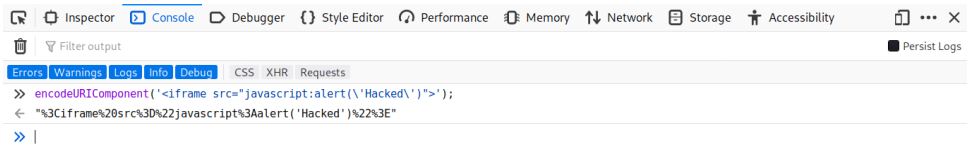

Figure 9: A screenshot of the code command

Using the code command, I am attempting to render an alert with the value Hacked. Usually, an attacker would exploit one form of an XSS attack by uploading malicious content to an XSS vulnerable server. The server record and stores the malicious content returning it to any victim that access the vulnerable page. You can see from the image below how the code command has rendered a a message box displaying "Hacked" as shown in the FIG 10 below.

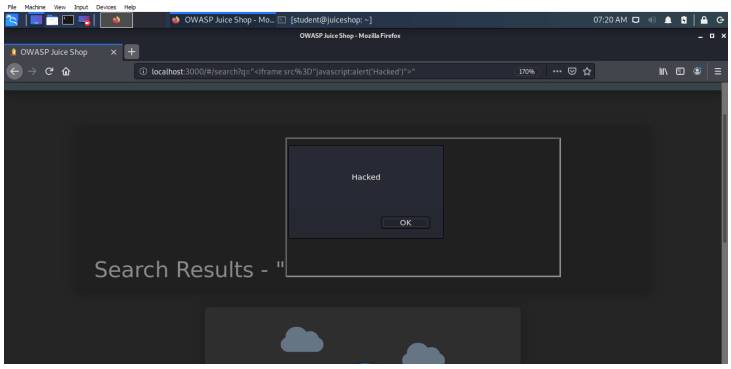

Figure 10: A screenshot of the message box displaying "Hacked"

4 Risk Analysis

To administer mitigations for the issues we have identified in the web application, we have ranked the risk to Udajuicer in order of what should be dealt with first. This ranking shows where Udajuicer's resources should be allocated first as opposed to trying to tackle everything at once.

4.1 Scoring Risk

Threats	Score
HTTP Flood Attack / DDoS Attack	1
Insecure Architecture	3
SQL Injection Vulnerability	2
XSS Vulnerability	4

4.2 Risk Rationale:

One may ask the rationale behind the ranking above.

- The identified attack in the web application was HTTP Flood attack, a type of DDoS Attack. Given that it is the current threat facing the web application makes it the most dangerous.

- Following The Open Web Application Security Project (OWASP) list of top 10 most common application vulnerabilities, Injections (such as SQL Injection) are the most common application vulnerabilities, so it ranks 2 in our risk scoring. The next is Security Misconfiguration (3) and finally Cross Site-Scripting (4).

5 Mitigation Plan

So far, we have broken down all the risks and in what order Udajuicer should mitigate them. We will now focus on building out mitigation plan and getting Udajuicer back up and running.

- First thing to do is to construct a secure architecture. We will use draw.io for this purpose.
- The next thing would be to identify ways to mitigate HTTP Flood (DDOS) attack that caused the website downtime.
- We will finally list ways to mitigate sql injection and XXS.

5.1 Secure Architecture Diagram using draw.io

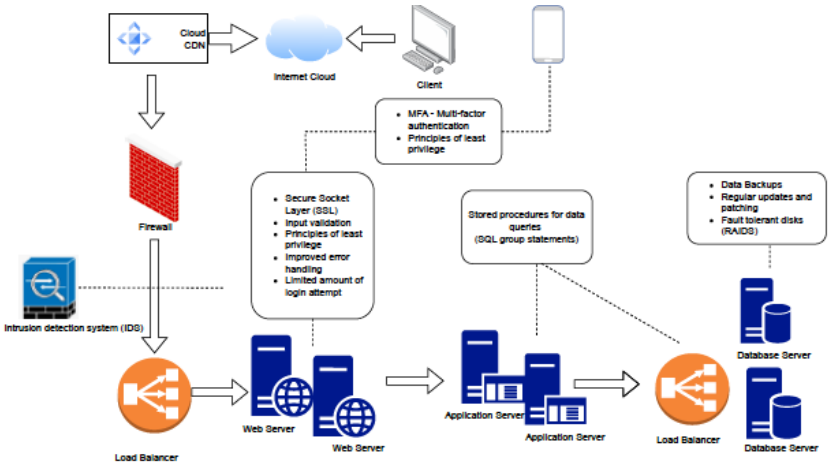

Figure 11: A secure architecture

5.2 HTTP Flood Attack / DDOS Attack Mitigation

Miitigating HTTP flood attacks involves implementing measures to protect web servers and applications from overwhelming traffic generated by a large number of requests. HTTP flood attacks aim to exhaust server resources, disrupt services, and potentially lead to service downtime. Here are key strategies to mitigate HTTP flood attacks:

- Implement rate limiting on your web server to restrict the number of requests from a single IP address or within a specific time frame. This helps prevent a single user or a group of users from overwhelming the server with excessive requests.
- Utilizing a Content Delivery Network to distribute incoming traffic across multiple servers located in different geographical regions to improve performance and security. CDNs can absorb and distribute the load, reducing the impact of a concentrated HTTP flood attack on a single server.
- Utilize load balancers to distribute incoming traffic across multiple application servers to ensure high availability and scalability. Load balancing helps evenly distribute the load, ensuring that no single server becomes a bottleneck during a flood attack.

- Deploy web application firewalls (WAF) to filter and monitor HTTP traffic between a web application and the internet to protect against common web application attacks. The WAF should be specifically designed to detect and mitigate various types of web attacks, including HTTP flood attacks. WAFs can analyze incoming traffic and block requests that exhibit patterns indicative of malicious activity.

5.3 SQL Injection Mitigation

A SQL injection attack is found when a SQL query is built up using user input. Mitigating SQL injection involves implementing security measures to prevent attackers from injecting malicious SQL code into input fields or other user-controllable parameters. Here are key strategies for mitigating SQL injection:

- Input Validation: Validate and sanitize all user input on the client and server sides. This ensures that only expected and valid data is accepted, preventing the injection of malicious SQL code.
- Use parameterized statements (prepared statements) or parameterized queries provided by the programming language or framework. Parameterization separates SQL code (queries) from user input (search term), preventing direct injection of malicious content.
- Stored Procedures: Use stored procedures whenever possible. Stored procedures encapsulate SQL logic on the database server, reducing the risk of injection attacks. Ensure that the stored procedures are well-designed and parameterized.
- Escape user input before incorporating it into SQL queries. This involves using proper escaping functions or libraries provided by the programming language. Escaping ensures that special characters are treated as literal values rather than executable SQL code.
- Least Privilege Principle: Implement the principle of least privilege when configuring database access permissions. This ensures that database accounts used by applications have the minimum necessary permissions to perform their tasks, reducing the impact of potential SQL injection attacks.

5.4 XSS - Cross Site Scripting Mitigation

As we have discussed, an XSS attack is a client-side attack. This means that, although the vulnerability is in the site itself, the targets are the users browsing the site. The attacker's goal is to run javaScript code in the browsers of the people visiting the site. Therefore mitigating XSS attacks would involve implementing security measures to prevent malicious scripts from being injected into web applications and executed by users' browsers. Here are key strategies for mitigating XSS attacks

- Sanitizing and validating all user input on both the client and server sides, checking for malicious scripts or coparing inputs against what is allowed. This includes data entered in forms, URL parameters, and any other user-controlled input. Reject or sanitize input that contains potentially malicious scripts. It is recommended to use a third party library and not try to implement your own filter.
- Conduct regular security audits and code reviews to identify and address potential security vulnerabilities, including XSS issues. Automated tools and manual reviews can help uncover vulnerabilities in the codebase.
- Implement session management best practices such as session timeout and token-based authentication.

Reference

- Lecture note: SEC504: Hacker Tools, Techniques, and Incident Handling
- Lecture note: SEC401: Security Essentials - Network, Endpoint, and Cloud
- Lecture note: SEC275: Foundations: Computers, Technology, & Security