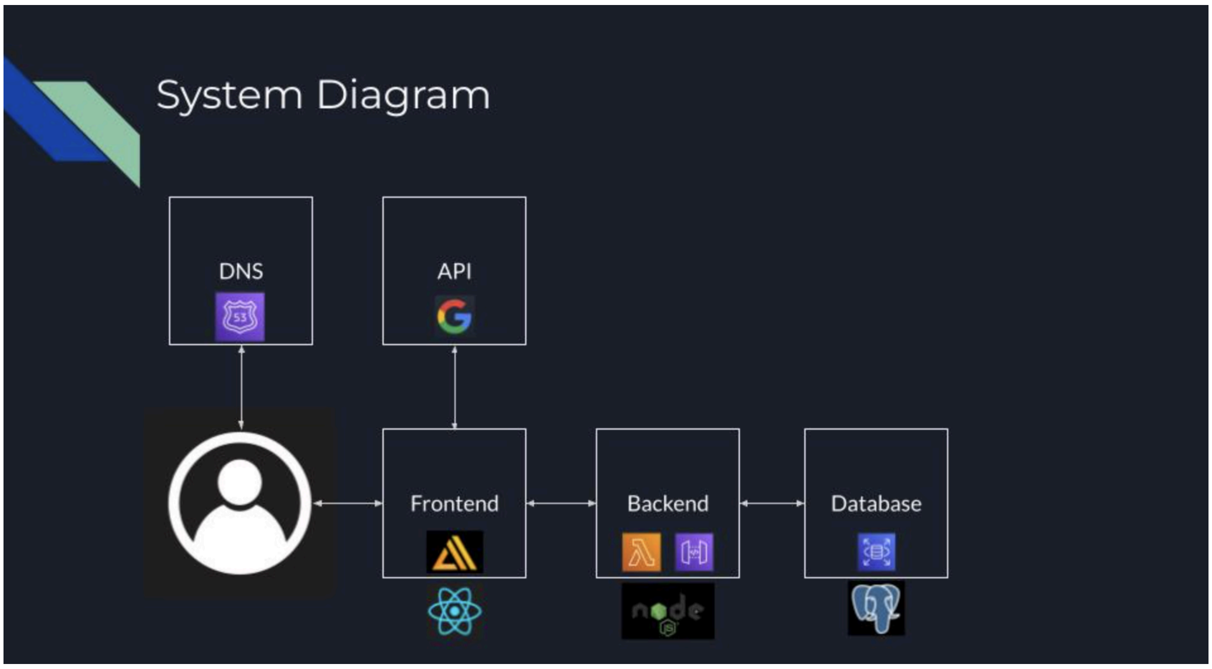
CyberSecurity Plan

Dhanunjai Andey

TP21

Table of Contents

1.System Architecture :



Understanding of Website :

The website allows users to calculate safe walking routes based on data like streetlights and nearby emergency services. The system architecture includes:

- Frontend: React with AWS Amplify
- Backend: Node.js running on AWS Lambda
- Database: PostgreSQL hosted on AWS RDS
- DNS: AWS Route 53

Understanding Security for Website :

hersafehome.me, is designed to help users find safe, well-lit routes and access emergency services. Given the sensitive nature of user interaction and the crime data being handled, security is a top priority. The OWASP Top 10 vulnerabilities are considered in every iteration, and proactive measures, such as SSL encryption, security headers, and user input validation, are implemented to protect against common threats such as XSS, clickjacking, and sensitive data exposure.

Data Handling, Encryption, and SSL

We use Amazon RDS (Relational Database Service), which automatically encrypts data at rest using AWS-managed keys, ensuring the secure storage of sensitive data. To further protect data in transit, we have implemented SSL (Secure Sockets Layer) encryption, ensuring that all communications between the server and users' browsers are encrypted and secure, preventing interception or tampering.

Since the data is static, a Lambda function retrieves all necessary data at once, reducing the need for frequent database queries and limiting exposure to potential SQL injection risks. Additionally, manual user input is avoided by leveraging Google's autocomplete feature for address entry, which further reduces the risk of introducing malicious data into the system.

By combining SSL encryption for data in transit, automatic encryption for data at rest, and secure input handling, our system minimizes vulnerabilities and ensures a high level of data protection.

2. OWASP Top Risks

- **Sensitive Data Exposure:** SSL is implemented to ensure that all data in transit is encrypted. This helps protect sensitive crime data and user interactions with the map interface from being intercepted or exposed.
- **Security Misconfiguration:** Since your project is hosted on AWS, it's crucial to regularly update and patch the system to prevent misconfigurations. This includes ensuring proper settings for access controls, firewalls, and other security measures.
- **Cross-Site Scripting (XSS):** Input sanitization is a key defence against XSS attacks, especially for the crime-type filtering feature. Proper validation and sanitization of user inputs ensure that malicious scripts cannot be injected**.**
- **SQL Injection:** SQL Injection occurs when an attacker can insert or manipulate SQL queries by injecting malicious input into a query. This could lead to unauthorized access, data leakage, or database manipulation.

3.Risk Register:
Risk register tracks identified risks, their likelihood, impact, overall risk level, mitigation strategies, and current status.
Legend:

Risk level	**colour**	**meaning**
high	🟥 **Red**	High-risk vulnerabilities that require immediate attention to avoid potential serious impacts.
Medium	🟧 **Orange**	Moderate-risk vulnerabilities that should be addressed soon to prevent potential exploitation.
low	🟨 **Yellow**	Low-risk vulnerabilities that have minimal immediate impact but should be monitored over time.
informational	🟩 **Green**	Informational findings or observations that don't directly pose a risk but are good to be aware of.

Risk Register:

Risk ID	Risk	Risk Description	Likelihood	Impact	Risk Level	Status
1	Cross-Domain	The server's	High	High	🟥 High	Identified

	Misconfiguration (CORS Misconfiguration)	CORS policy is too permissive, allowing any domain to access resources, potentially leading to unauthorized access and cross-site attacks.				
2	Content Security Policy (CSP) Header Not Set	The absence of a CSP header leaves the application vulnerable to XSS and content injection attacks.	High	High	High	identified
3	Strict-Transport-Security (HSTS) Header Not Set	The absence of an HSTS header means the site is not enforcing HTTPS, leaving it vulnerable to man-in-the-middle attacks.	medium	High	High	mitigated
4	Missing Anti-clickjacking Header	The absence of an anti-clickjacking	medium	medium	medium	Mitigated

		header could allow clickjacking attacks, where users are tricked into clicking on something different from what they see.				
5	Server Leaks Version Information	The server reveals its software version in the HTTP response headers, which can provide valuable information to attackers looking to exploit known vulnerabilities.	low	low	Low	identified
6	Timestamp Disclosure	attackers could use the timestamp to gather information on server operations, potentially helping in more targeted attacks over time.	Low	Low	Low (Informational)	identified

4.STRIDE Vulnerability Analysis with Risks and CVEs :

The table below outlines a comprehensive STRIDE analysis of the vulnerabilities identified during penetration testing. Each vulnerability is categorized according to the STRIDE threat model (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege), with corresponding CVEs where applicable. This analysis aids in understanding the nature of each vulnerability, its associated risks, and the potential impact on the system's confidentiality, integrity, and availability. The information provided will guide mitigation strategies and prioritize the risks that need immediate attention to strengthen the security of the system.

Vulnerability	STRIDE Category	CVE Example	Risk Level	Risk Description
1. Cross-Domain Misconfiguration (CORS Misconfiguration)	Spoofing, Tampering, Information Disclosure	CVE-2023-2360	High	A misconfigured CORS policy can lead to **Spoofing**, **Tampering**, and **Information Disclosure**, allowing unauthorized domains to access sensitive data and resources.
2. Content Security Policy (CSP) Header Not Set	Tampering, Information Disclosure, Spoofing	CVE-2018-5164	High	Without a CSP, the system is vulnerable to **Tampering**, **Information Disclosure**, and **Spoofing** attacks, particularly through XSS, leading to data theft and website compromise.
3.	Information	CVE-2017-778	High	The absence of

Strict-Transport-Security (HSTS) Header Not Set	Disclosure, Tampering	9		HSTS allows **Information Disclosure** and **Tampering** via man-in-the-middle attacks, potentially exposing sensitive data in transit.
4. Missing Anti-clickjacking Header	Spoofing, Tampering	CVE-2022-3260	Medium	The lack of an anti-clickjacking header enables **Spoofing** and **Tampering**, where attackers can trick users into performing unintended actions.
5. Server Leaks Version Information	Information Disclosure	CVE-2023-25948	Low	Disclosing the server version contributes to **Information Disclosure**, allowing attackers to identify and exploit known vulnerabilities.
6. Timestamp Disclosure	Information Disclosure	CVE-1999-0524	Low	The disclosure of system timestamps falls under **Information Disclosure**, potentially providing attackers with operational details to plan further targeted attacks.

5.Mitigation strategies :

- **Sensitive Data Exposure:**
  - Implement SSL/TLS encryption for all data in transit.
  - Encrypt sensitive data at rest.

- **Security Misconfiguration:**
  - Regularly update and patch all software and services.
  - Perform regular security audits of configurations.
  - Disable unnecessary features and services to reduce the attack surface.
- **Cross-Site Scripting (XSS):**
  - Sanitize and validate all user inputs.
  - Encode output data before rendering it in the browser.
  - Implement a Content Security Policy (CSP) to control script sources.

- **SQL Injection :**
  - I used parameterized queries or prepared statements.
  - Validate and sanitize all user inputs before processing.
  - Avoid revealing detailed error messages to users.
  - **Database Query Protection:**
    - Since our data is medium-sized and static, I used a Lambda function that retrieves all the data at once from the RDS database. This approach reduces the need for multiple database queries, limiting potential exposure to SQL injection risks and enhancing overall security.

Vulnerability Mitigation :

1. **Cross-Domain Misconfiguration (CORS Misconfiguration)**
   a. **Risk Description**: The server's CORS policy is too permissive, allowing any domain to access resources, potentially leading to unauthorized access and cross-site attacks.
   b. **Mitigation Steps**:
      i. **Restrict Allowed Origins**: Updated the CORS policy to only permit requests from your own domain (https://www.hersafehome.me) to prevent unauthorized access. This prevents other domains from making cross-origin requests.

      ii. **Use Specific Methods**: Defined allowed HTTP methods (e.g., GET, POST) to restrict the types of requests that external domains can make.

        iii. **Secure Headers**: Ensured the Access-Control-Allow-Headers and Access-Control-Allow-Credentials headers are properly set to limit exposure to cross-origin requests.
            1.

2. **Content Security Policy (CSP) Header Not Set**
    a. **Risk Description**: The absence of a CSP header leaves the application vulnerable to XSS and content injection attacks.
    b. **Mitigation Steps**:
        i. **Set Content Security Policy**: Added the `Content-Security-Policy` header with a configuration that restricts resources (scripts, styles, images) to trusted sources.
        ii. **Allow Specific Directives**: Configured the policy to only allow scripts and styles from known sources (e.g., `self` for same-origin and specific trusted domains).
        iii. **Test and Iterate**: Tested the CSP implementation in a staging environment to ensure that it did not break functionality and adjusted the policy as needed.

3. **Strict-Transport-Security (HSTS) Header Not Set**
    a. **Risk Description**: The absence of an HSTS header means the site is not enforcing HTTPS, leaving it vulnerable to man-in-the-middle attacks.
    b. **Mitigation Steps**:
        i. **Implement HSTS Header**: Added the `Strict-Transport-Security` header with the value `max-age=31536000; includeSubDomains; preload` to enforce HTTPS.
        ii. **Submit for Preloading**: Submitted the domain to ensure that browsers remember to always access the site over HTTPS.
        iii. **Monitor HTTPS Traffic**: Regularly monitored traffic to ensure that all connections are secured over HTTPS and adjusted server configurations as needed.

4. **Missing Anti-clickjacking Header**
    a. **Risk Description**: The absence of an anti-clickjacking header could allow clickjacking attacks, where users are tricked into clicking on something different from what they see.
    b. **Mitigation Steps**:
        i. **Set X-Frame-Options Header**: Added the `X-Frame-Options: DENY` header to prevent the site from being embedded in iframes.
        ii. **Use Content Security Policy (CSP)**: Enhanced the CSP with a `frame-ancestors` directive to further restrict where the website can be embedded.
        iii. **Test Embedding Attempts**: Conducted tests to ensure that the header properly blocked iframe embedding and that the CSP directive was effective.

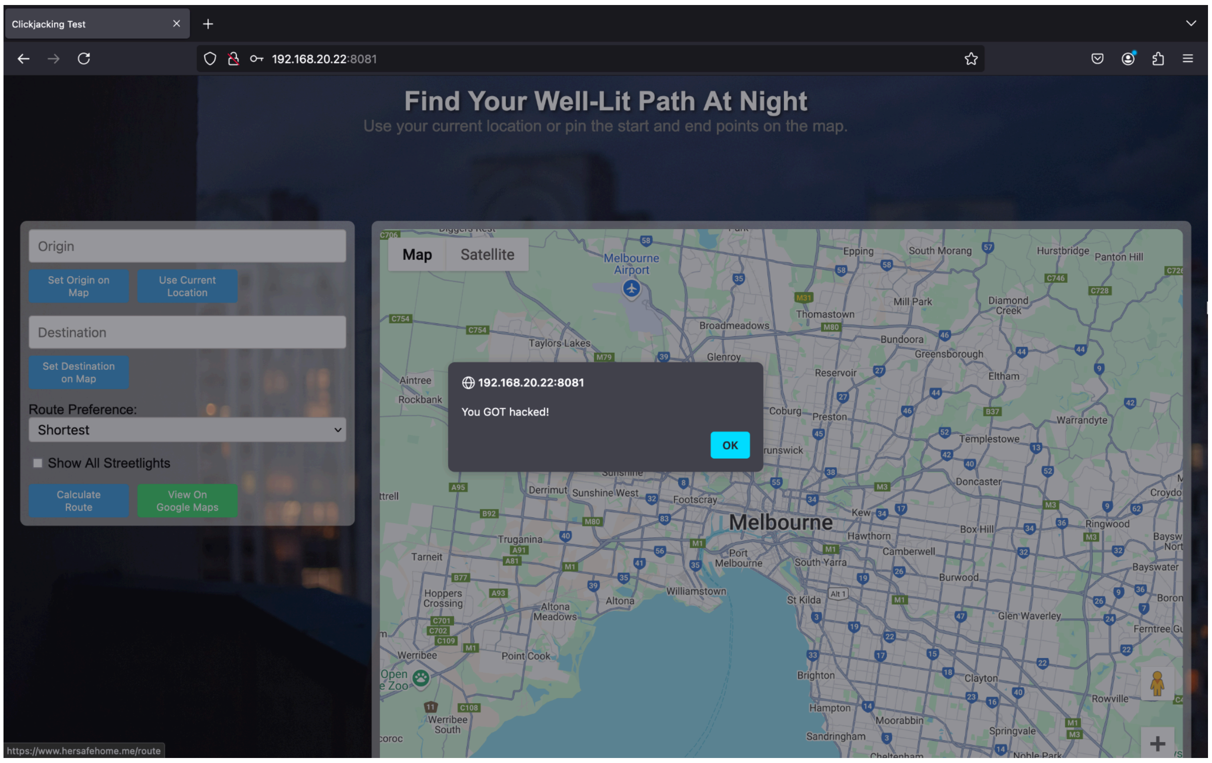
Explanation for Vulnerabilities Not Addressed

5. **Server Leaks Version Information**
   - **Risk Description**: The server reveals its software version in the HTTP response headers, which can provide valuable information to attackers looking to exploit known vulnerabilities.
   - **Why It Was Not Mitigated**:
     - **Low Impact**: This vulnerability was classified as low-risk because it does not directly compromise the system but could be used as part of an information-gathering phase.
     - **Internal Security**: Given the strength of other security measures in place (e.g., up-to-date software, secure configuration), the exposure of version information alone does not pose a significant risk.
     - **Risk Assessment Decision**: The team prioritized other higher-impact vulnerabilities, deciding that masking version details would be a lower priority.

6. **Timestamp Disclosure**
   - **Risk Description**: Attackers could use the timestamp to gather information on server operations, potentially helping in more targeted attacks over time.
   - **Why It Was Not Mitigated**:
     - **Informational Nature**: The timestamp disclosure was categorized as **informational** since it does not present a direct threat to the system's security.
     - **Limited Exploitability**: While timestamps could theoretically be used for more targeted attacks, there are no known exploits that leverage timestamps alone without additional vulnerabilities.
     - **Other Protections in Place**: The system has robust protections against more direct attacks (e.g., SQL injection, XSS), making the timestamp disclosure a low-priority issue.

6. Successful Attacks:

A1. Clickjacking Attack(4) :



192.168.20.22:8081

You GOT hacked!

Mitigation Strategy: Using the X-Frame-Options Header

- To protect against clickjacking, you can use the **X-Frame-Options HTTP header**. This header tells browsers whether or not a page should be allowed to be displayed in an iframe.

Implementation:
1. **Add the X-Frame-Options Header**: You should include this header in the HTTP response of your server. For example:
   - **X-Frame-Options: DENY**: This setting prevents the page from being displayed in any iframe, regardless of the domain trying to embed it.
   - **Alternative Setting**: X-Frame-Options: SAMEORIGIN allows your page to be embedded only by other pages from the same domain.
2. **Set DENY for Maximum Protection**: Using X-Frame-Options: DENY ensures that no external site can embed your page in an iframe, providing strong protection against clickjacking attempts.

How This Strategy Works:
- By instructing the browser not to allow your website to be loaded in iframes from other domains, **you prevent attackers from being able to hide your page behind other content**.
- This means that users will interact with your page directly, without being tricked into unintended actions by a hidden interface.

7. Incident Management Plan

Purpose:

The purpose of this Incident Management Plan is to outline the steps for identifying, responding to, and resolving security incidents affecting the *hersafehome.me* website. As the sole cybersecurity lead, I am responsible for monitoring, responding to incidents, and ensuring that the security of the project is maintained.

Incident Response Process:

1. **Detection and Reporting:**
   - **Detect**: Continuous monitoring using available security tools (e.g., AWS CloudWatch, AWS CloudTrail, Nmap, OWASP ZAP) to identify potential security incidents such as unauthorized access or system anomalies.
   - **Report**: Any suspicious activity identified during monitoring will be documented. Alerts from automated tools will be reviewed and categorized.
2. **Incident Classification:** Once an incident is identified, it will be classified based on its potential impact on the system:
   - **Low Severity**: Minimal impact on performance or security, no data exposure.
   - **Medium Severity**: Compromised functionality or potential unauthorized access.
   - **High Severity**: Major security breaches or critical data exposure that affects system operations or user safety.
3. **Incident Containment:**
   - **Short-Term Containment**: Isolate affected parts of the system (e.g., Lambda functions, database) by disabling relevant services or functionality until the root cause is identified.
   - **Long-Term Containment**: Apply patches, reconfigure AWS security settings, or update any compromised components to prevent future incidents.
4. **Eradication:**
   - **Eliminate the Cause**: Identify and remove the vulnerability (e.g., SQL injection, XSS) or malware. Ensure that all compromised components are addressed and tested.
   - **Verify Integrity**: Run security tests (such as Nmap and OWASP ZAP scans) to confirm that the system is secure and free from further vulnerabilities.
5. **Recovery:**
   - **Restore Services**: Once the issue is resolved, gradually restore services and ensure normal functionality.
   - **Monitor for Recurrence**: Keep close monitoring of the affected system to ensure that the issue does not recur, using AWS CloudWatch and security tools.

6. **Post-Incident Review:**
    - **Review and Analyze**: Conduct a detailed review of the incident to understand how it occurred and what improvements can be made in the future.
    - **Documentation**: Maintain a detailed log of the incident, including the timeline, actions taken, and lessons learned.
    - **Security Enhancements**: Implement any necessary changes or additional security controls to prevent a similar incident in the future.

Communication:

- **Internal Communication**: Keep the project team informed of any security incidents, their impact, and the steps being taken to resolve them. Since you're the only cybersecurity lead, updates can be shared directly during team meetings or through shared documents.
- **External Communication**: If the incident affects the functionality or availability of the website, the team will notify users about the issue and provide updates on resolution timelines.

Continuous Improvement:

After each incident, the security controls will be re-evaluated, and any necessary adjustments will be made. The vulnerability assessment, penetration testing schedule, and overall cybersecurity plan will be updated to reflect lessons learned from each incident.