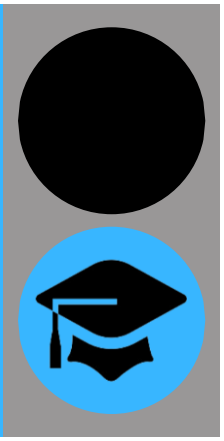


Name: Leburi Sriram

Date: 22/03/2024

Task- 3(Challenging)

Task: Web Application Security Assessment

Description:

My task is to perform a basic security assessment on booking.com site to identify and address potential vulnerabilities using OWASP ZAP tool. The goal is to enhance the security posture of the application by implementing best practices and mitigating common security risks.

Requirements:

- Basic understanding of web application security concepts. Familiarity with
- common web vulnerabilities (e.g., CrossSite Scripting, SQL Injection, Cross-Site Request Forgery).

- Access to a web application for testing purposes.

Steps :

1

I choose **Booking.com** webapplication for this assessment. Using **OWASP ZAP** tool i performed security assessment.

Booking.com:

Booking.com is a popular online travel agency that facilitates the booking of accommodations, flights, rental cars, and other travel-related services. It was founded in 1996 in Amsterdam, Netherlands, and has since grown to become one of the largest travel e-commerce companies in the world.

Here are some key features and aspects of Booking.com:

Accommodation Options: Booking.com offers a wide range of accommodation options, including hotels, apartments, hostels, resorts, villas, and even unconventional lodging such as treehouses and igloos. Users can filter their search based on criteria such as price, location, facilities, and guest reviews.

User Reviews and Ratings: One of the distinguishing features of Booking.com is its extensive collection of user reviews and ratings for accommodations. These reviews provide valuable insights for travelers when making their booking decisions.

Flexible Booking Options: Booking.com typically offers flexible booking options, including free cancellation on many properties and the ability to reserve accommodations without upfront payment in some cases.

Price Matching: The platform often advertises a "Best Price Guarantee," claiming to match or beat the price offered by other booking websites for the same accommodation.

Customer Support: Booking.com provides customer support services to assist users with booking-related inquiries, changes, or issues. Support is available through various channels, including phone, email, and live chat.

Mobile App: Booking.com offers a mobile app for both iOS and Android devices, allowing users to search for and book accommodations on the go.

Deals and Discounts: The platform frequently features deals and discounts on accommodations, flights, and other travel services, helping users find cost-effective options for their trips.

Global Presence: Booking.com operates internationally, offering accommodations in destinations around the world. It supports multiple languages and currencies to cater to a diverse user base.

OWASP ZAP Tool:

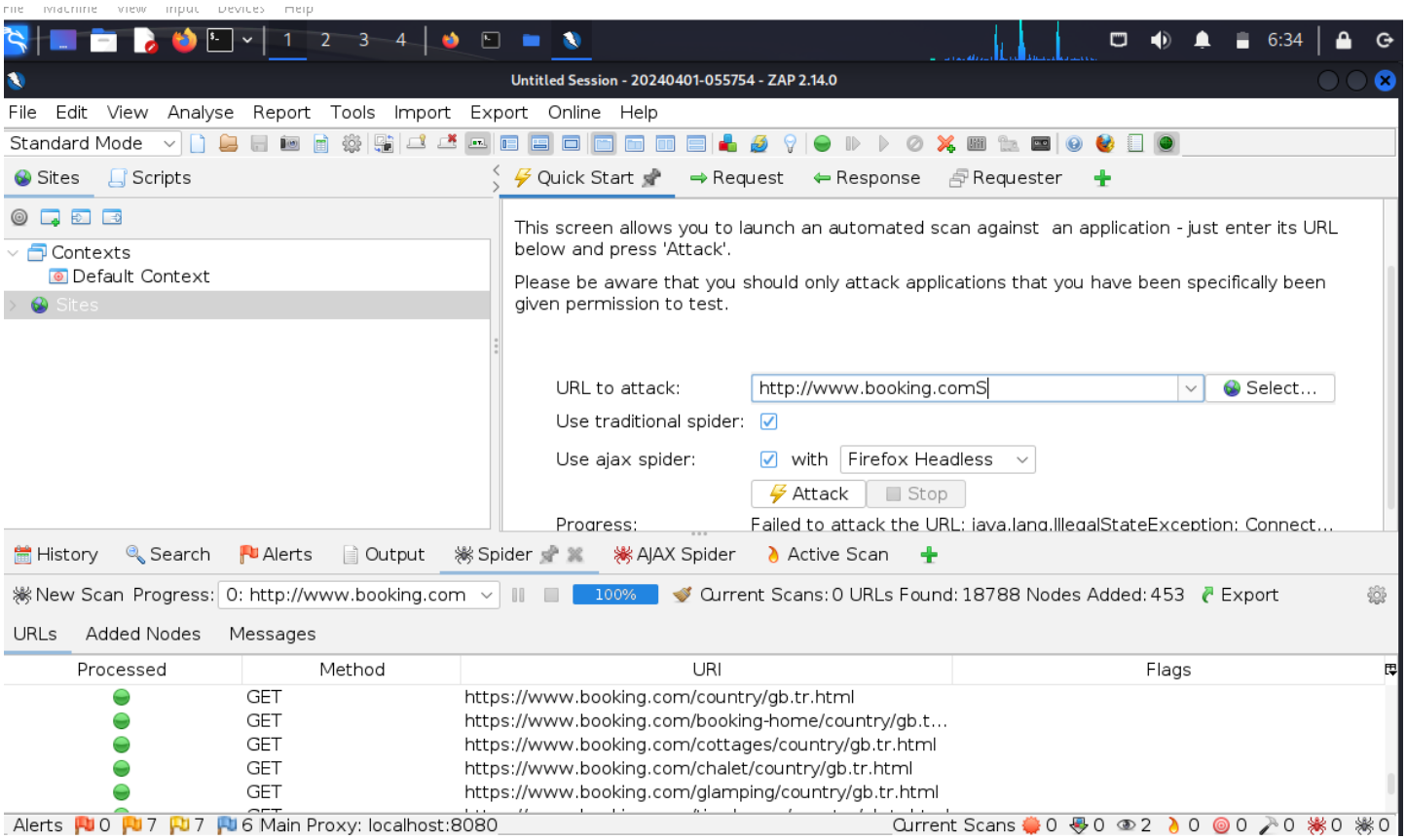
OWASP ZAP (Zed Attack Proxy) is a popular open-source web application security testing tool. It is maintained by the Open Web Application Security Project (OWASP), which is a non-profit organization focused on improving software security.

Here are some key aspects and features of OWASP ZAP:

Proxy Functionality: ZAP operates as a proxy server, allowing users to intercept and modify HTTP and HTTPS requests between a web browser and a web application. This proxy functionality enables security researchers and developers to inspect web traffic and identify potential vulnerabilities.

Active and Passive Scanning: ZAP can perform both active and passive scanning of web applications. Active scanning involves actively probing the application for security vulnerabilities, such as SQL injection, cross-site scripting (XSS), and insecure direct object references (IDOR). Passive scanning involves observing web traffic passively to detect security issues without actively interacting with the application.

Automated Scanning: ZAP supports automated scanning of web applications, allowing users to initiate scans and receive reports on vulnerabilities discovered during the scanning process. This feature helps identify security weaknesses in web applications efficiently.



Spidering: ZAP includes a spidering feature that automatically crawls through a web application to discover all accessible pages and functionality. This helps ensure comprehensive coverage during security testing.

Fuzzer: ZAP includes a fuzzer tool that can be used to identify security vulnerabilities by sending malformed or unexpected input to the application and observing its response. This helps uncover issues such as buffer overflows, input validation errors, and authentication bypasses.

Scripting and Automation: ZAP provides support for scripting and automation through its powerful API (Application Programming Interface). Users can write scripts in various programming languages (e.g., Python, JavaScript) to customize ZAP's behavior, automate repetitive tasks, and integrate it into their security testing workflows.

Authentication Support: ZAP supports various authentication mechanisms, including form-based authentication, HTTP authentication, and OAuth, allowing users to test the security of authenticated areas of a web application.

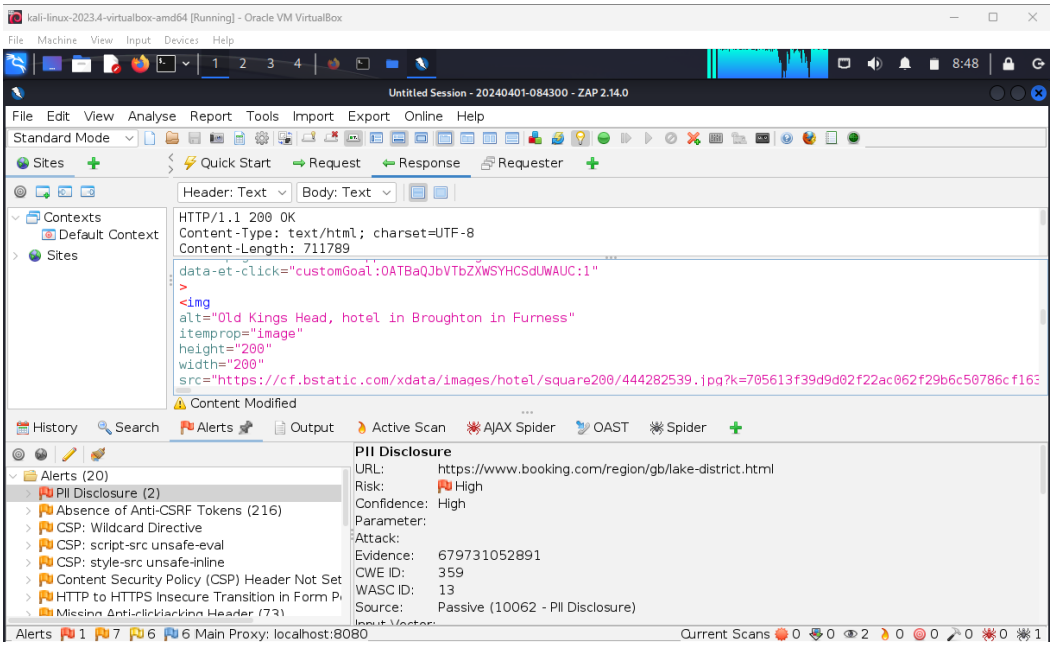
Reporting: ZAP generates detailed reports summarizing the findings of security scans, including identified vulnerabilities, severity levels, and recommended remediation steps. These reports help communicate security issues to developers and stakeholders effectively.

2 Identified common vulnerabilities such as

- PII disclosure(High)
- Absence of anti csrf tokens(Medium)
- script src unsafe-eval(Medium)
- HTTP to HTTPS insecure transition(Medium)
- X content type options header missing(Low)

I found by conducting this assessment on booking.com site that the major common attack that can be effected to website by these vulnerabilities is “XSS”

The term "PII disclosure" typically refers to the unauthorized exposure or disclosure of Personally Identifiable Information (PII), which includes sensitive data such as names, addresses, social security numbers, email addresses, and financial information. When PII disclosure occurs due to security vulnerabilities or lapses in data protection measures, it can have significant consequences for individuals, organizations, and regulatory compliance.



In the context of OWASP (Open Web Application Security Project) and web application security, the effects of PII disclosure can be severe and encompass various risks and consequences:

Privacy Violations: PII disclosure undermines individuals' privacy rights by exposing their sensitive personal information without consent. This can lead to concerns about identity theft, fraud, harassment, and other privacy-related issues.

Financial Losses: Exposing financial information, such as credit card numbers or bank account details, can result in financial losses for individuals due to unauthorized transactions, fraudulent activities, or identity theft. Additionally, organizations may incur financial penalties, legal fees, and compensation costs associated with data breaches and PII disclosure incidents.

Reputation Damage: PII disclosure incidents can damage the reputation and trustworthiness of organizations responsible for the security of the affected web applications. Negative publicity, media coverage, and public perception of inadequate data protection practices can have long-lasting effects on an organization's brand reputation and customer relationships.

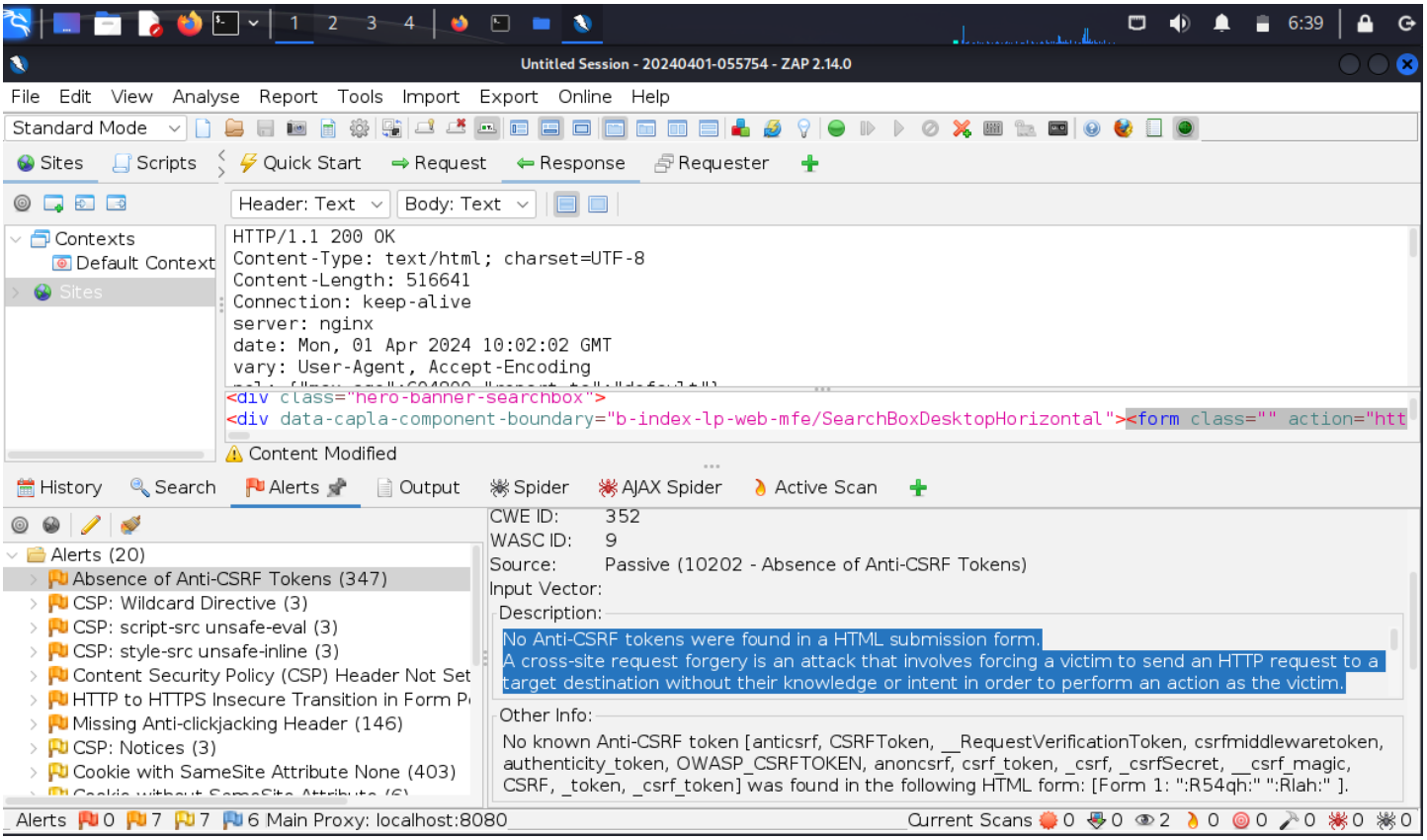
Regulatory Non-Compliance: Many jurisdictions have data protection laws and regulations governing the collection, storage, and processing of PII. PII disclosure incidents may result in non-compliance with these regulations, such as the General Data Protection Regulation (GDPR) in the European Union or the Health Insurance Portability and Accountability Act (HIPAA) in the United States. Organizations found to be non-compliant may face regulatory fines, sanctions, and legal consequences.

Legal Liabilities: PII disclosure incidents can expose organizations to legal liabilities and litigation from affected individuals, regulatory authorities, and other stakeholders. Lawsuits may be filed seeking damages for negligence, breach of privacy, failure to secure sensitive data, and other legal claims arising from the data breach.

Operational Disruption: Dealing with the aftermath of a PII disclosure incident can disrupt the normal operations of an organization. This may include conducting forensic investigations, notifying affected individuals, implementing remediation

measures, enhancing security controls, and managing public relations and crisis communication efforts.

The absence of anti-CSRF (Cross-Site Request Forgery) tokens in a web application can lead to various security vulnerabilities and risks:



CSRF Attacks: Without anti-CSRF tokens, the web application becomes susceptible to CSRF attacks. In a CSRF attack, an attacker tricks a victim into making a request to the application, typically by embedding malicious code in a website or email. Since the victim is authenticated with the application, the request is executed with the victim's privileges, leading to unauthorized actions such as fund transfers, changing account settings, or posting content.

Data Modification and Deletion: CSRF attacks can result in unauthorized modification or deletion of data within the application. For example, an attacker could craft a CSRF attack to change the victim's email address, delete their account, or modify sensitive settings.

Account Takeover: CSRF vulnerabilities may be leveraged by attackers to facilitate account takeover. By exploiting CSRF vulnerabilities, attackers can perform actions

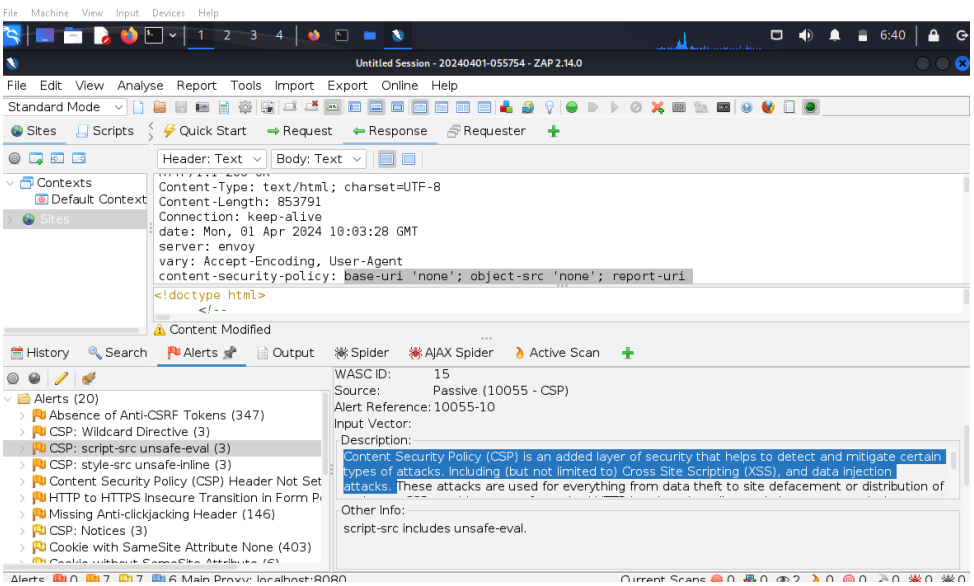
on behalf of authenticated users, potentially gaining control over their accounts and associated resources.

Security Bypass: CSRF attacks can bypass security controls and mechanisms implemented within the application. For instance, if an application requires a user to perform certain actions (such as changing their password) only after confirming their identity through multi-factor authentication, a CSRF attack could bypass this requirement.

Reputation Damage: Successful CSRF attacks can damage the reputation of the web application and the organization behind it. Instances of unauthorized actions or data breaches resulting from CSRF vulnerabilities can undermine trust among users and stakeholders.

Legal and Compliance Risks: CSRF vulnerabilities may expose the organization to legal and compliance risks, particularly if the unauthorized actions performed through CSRF attacks result in data breaches or violations of privacy regulations. Organizations may face regulatory penalties, legal actions, or damage to their brand reputation.

The use of unsafe-eval in the script-src directive of Content Security Policy (CSP) can introduce significant security risks in web applications, potentially leading to various types of attacks and vulnerabilities. Here are some potential effects and risks associated with allowing unsafe-eval in CSP:



Cross-Site Scripting (XSS) Attacks: unsafe-eval allows the execution of dynamically generated JavaScript code via methods like eval() or new Function(). Attackers can exploit this to inject and execute malicious scripts within the context of the vulnerable web application, leading to XSS vulnerabilities. These XSS vulnerabilities can be used to steal sensitive user data, perform unauthorized actions on behalf of users, or deface the application.

Code Injection Attacks: Allowing unsafe-eval permits the execution of arbitrary code provided by attackers, which can lead to code injection attacks. Attackers may inject and execute malicious code, such as SQL injection payloads, shell commands, or other types of code, leading to data breaches, system compromise, or unauthorized access to sensitive resources.

Security Bypass: unsafe-eval can potentially bypass security controls and mechanisms implemented within the application. For example, it can be used to circumvent client-side input validation or sanitization routines, allowing attackers to execute malicious code that would otherwise be blocked.

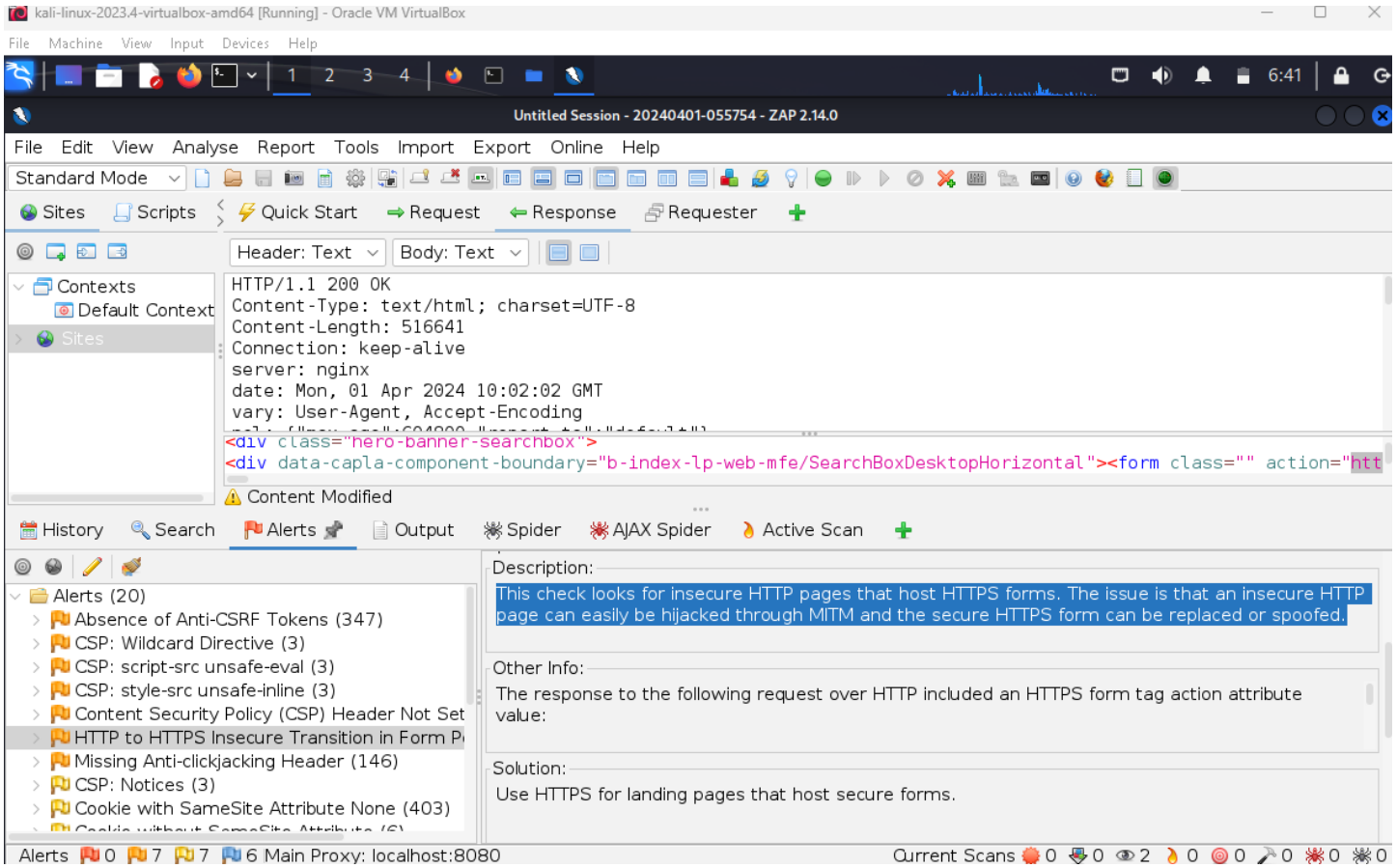
Increased Attack Surface: Allowing unsafe-eval expands the attack surface of the web application, as it increases the potential vectors for exploitation by attackers. Any vulnerability or weakness in the application's code or third-party libraries that can be leveraged to execute arbitrary JavaScript code becomes more exploitable when unsafe-eval is enabled.

Data Leakage and Privacy Risks: unsafe-eval can be used to extract and exfiltrate sensitive user data from the application's context. Attackers may execute code to access and steal user session tokens, personal information, or other confidential data stored within the application, leading to privacy violations and data breaches.

Regulatory Compliance Issues: Allowing unsafe-eval may result in non-compliance with regulatory requirements and standards related to data security and privacy. Many regulations, such as GDPR (General Data Protection Regulation) and HIPAA (Health Insurance Portability and Accountability Act), mandate the implementation of adequate security controls to protect sensitive data. Allowing unsafe-eval could lead to violations of these regulations and potentially result in legal consequences for the organization.

Transitioning from HTTP to HTTPS involves securing the communication between the client and the server, which is crucial for protecting sensitive data and ensuring the integrity of web transactions. Failing to properly implement HTTPS can lead to

various security risks and vulnerabilities, some of which are outlined below, particularly in the context of OWASP (Open Web Application Security Project) guidelines:



Data Interception (Man-in-the-Middle Attacks): Without HTTPS, data transmitted between the client and server is sent in plaintext, making it susceptible to interception by attackers. This can lead to the theft of sensitive information such as login credentials, payment details, and personal data. Man-in-the-middle (MitM) attacks become possible, where an attacker intercepts and eavesdrops on the communication between the client and server.

Session Hijacking: Insecure HTTP sessions are vulnerable to session hijacking attacks, where an attacker steals a user's session identifier and impersonates the user on the web application. This can result in unauthorized access to the user's account, exposure of sensitive data, and potentially fraudulent activities.

Cross-Site Scripting (XSS) and Injection Attacks: Insecure transitions between HTTP and HTTPS can exacerbate vulnerabilities such as XSS and injection attacks. For example, if an attacker injects malicious scripts into unsecured HTTP responses,

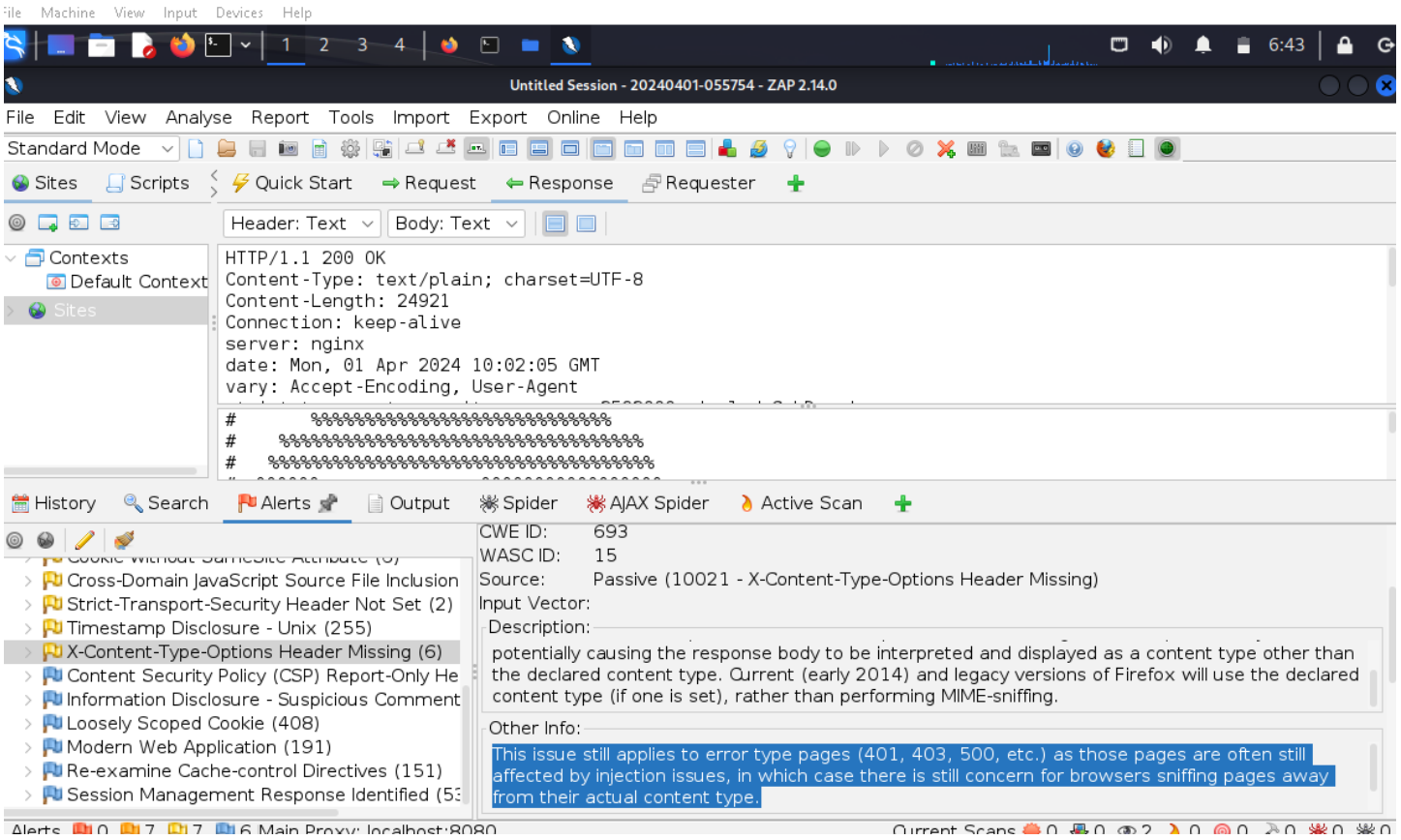
these scripts may be executed in the context of the user's browser, leading to XSS vulnerabilities and potential compromise of user sessions or sensitive data.

Phishing and Spoofing: Attackers can exploit the lack of HTTPS to launch phishing attacks and spoof legitimate websites. By intercepting traffic or creating malicious clones of legitimate sites, attackers can deceive users into entering sensitive information or downloading malware.

Compliance and Trust Issues: Inadequate implementation of HTTPS may lead to non-compliance with security standards and regulations, such as GDPR, PCI DSS (Payment Card Industry Data Security Standard), and HIPAA. Moreover, users may lose trust in the website if they perceive it as insecure, leading to reduced traffic, conversions, and reputation damage.

SEO Impact: Search engines tend to prioritize secure HTTPS websites over insecure HTTP ones. Failing to implement HTTPS properly can result in a lower search engine ranking and reduced visibility for the website.

The "X-Content-Type-Options" header is a security feature that helps protect web applications from certain types of attacks, particularly MIME-sniffing attacks. When this header is missing or not set correctly, it can lead to various security risks and vulnerabilities.



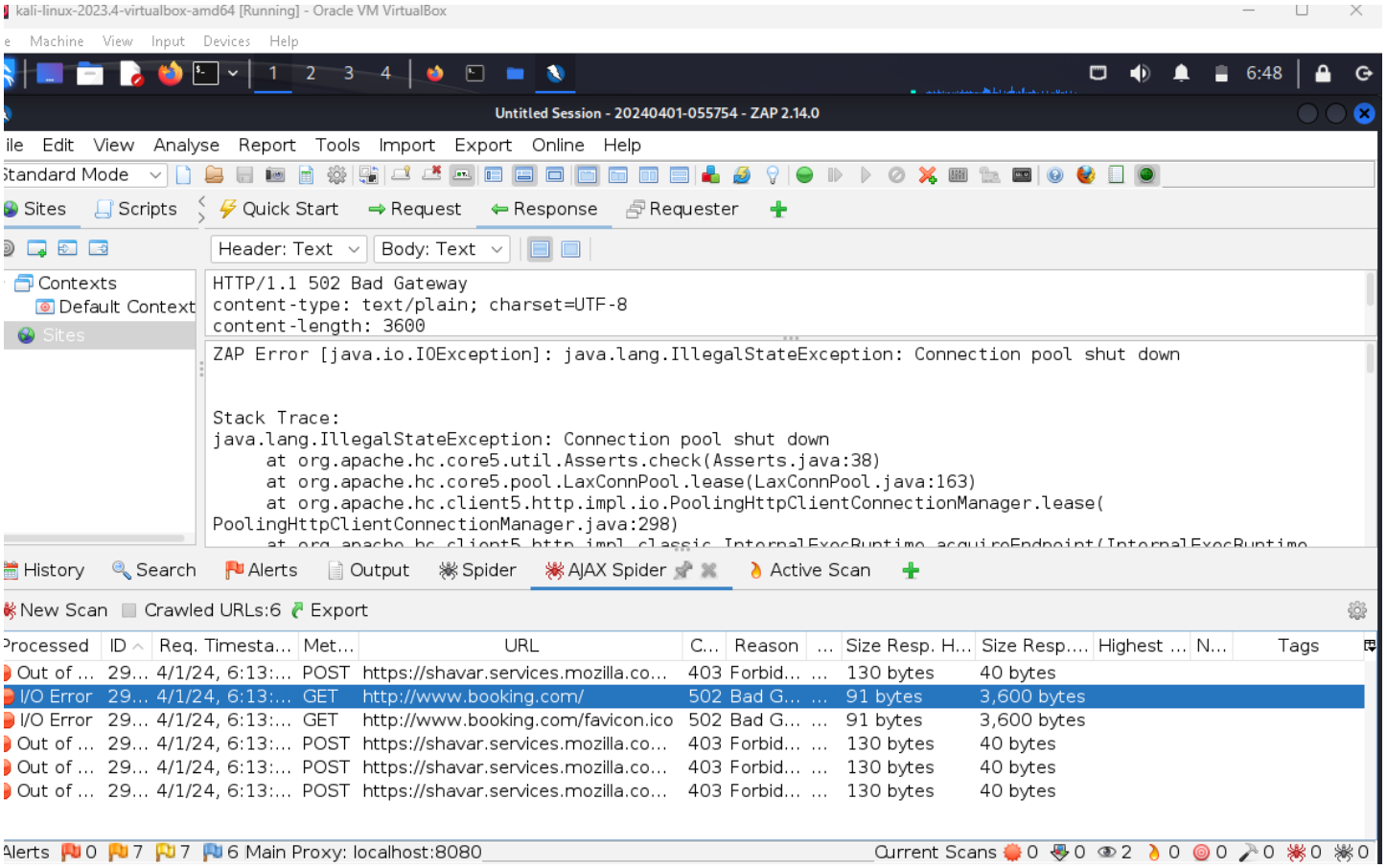
MIME Sniffing: When the "X-Content-Type-Options" header is missing or not set to "nosniff," some browsers may attempt to infer the content type of a resource based on its content, a process known as MIME sniffing. This behavior can be exploited by attackers to trick the browser into interpreting a file as a different content type, leading to various attacks such as XSS (Cross-Site Scripting) and data injection.

XSS Attacks: Without the "X-Content-Type-Options" header set to "nosniff," browsers may incorrectly interpret a file as a script or executable, potentially leading to XSS vulnerabilities. Attackers can exploit this behavior by uploading malicious files (e.g., images or documents containing JavaScript code) and tricking users into executing them, thereby executing arbitrary code in the context of the vulnerable web application.

Data Injection: Insecure MIME sniffing can also result in data injection attacks, where attackers upload files with malicious content (e.g., executable code, SQL injection payloads) disguised as benign files. This can lead to the execution of arbitrary commands, data leakage, and compromise of sensitive information stored on the server.

Client-side Attacks: Insecure MIME sniffing can be leveraged in client-side attacks, where attackers craft malicious files designed to exploit vulnerabilities in specific applications or plugins. For example, an attacker may upload a specially crafted file that exploits a vulnerability in a PDF reader plugin, leading to remote code execution or other security compromises on the client-side.

Content Spoofing: Without proper MIME type validation enforced by the "X-Content-Type-Options" header, attackers may be able to spoof the content type of resources served by the web application. This can be used to deceive users and bypass security controls by serving malicious content under false pretenses (e.g., serving an HTML file with a content type of "image/jpeg" to evade XSS filters).



3 Mitigations:

- Organizations should prioritize security measures such as encryption, access controls, data minimization, secure coding practices, vulnerability management, and incident response planning.
- Regular security assessments, including penetration testing and code reviews, can help identify and address vulnerabilities that could lead to PII disclosure incidents.
- Additionally, thorough security testing, including automated scans and manual reviews, should be conducted to identify and address CSRF vulnerabilities in web applications.
- Essential to avoid using unsafe-eval in CSP directives whenever possible. Instead, developers should adopt safer alternatives and best practices for dynamically executing code, such as using strict mode, avoiding eval() and new Function(), and implementing safer code execution patterns.
- Obtaining an SSL/TLS certificate from a trusted certificate authority (CA).
- Configuring web servers to enforce HTTPS by redirecting HTTP traffic to HTTPS.
- Ensuring that all resources (e.g., images, scripts, stylesheets) are loaded securely over HTTPS.

- Implementing HTTP security headers, such as HTTP Strict Transport Security (HSTS), Content Security Policy (CSP), and X-Frame-Options, to enhance security and prevent common web vulnerabilities.
- Regularly monitoring and testing the security of the HTTPS implementation to detect and address any vulnerabilities or misconfigurations.
- Essential to ensure that web servers include the "X-Content-Type-Options" header with the value "nosniff" in their HTTP responses. Additionally, web developers should implement robust input validation and sanitization mechanisms to prevent malicious file uploads and other forms of data injection attacks.

4 Fixes to implement:

- Phase: Architecture and Design
- Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
- Use HTTPS for landing pages that host secure forms.
- Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.
- If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.

5 What I Learned:

- Practical application of web application security concepts.
- Hands-on experience with OWASP ZAP tool
- Strategies for identifying and mitigating common vulnerabilities.

Conclusion:

This task provides valuable hands-on experience in identifying and addressing web application vulnerabilities, an essential skill for anyone interested in cybersecurity. Ithelps reinforce the importance of proactive security measures in the development and maintenance of web applications.