UDACITY

CryptoV4ult Enterprise Security Review

EMMANUEL AVAH

20-05-2024

Project Scenario

Overview

As the lead security engineer for CryptoV4ult, a prominent international cryptocurrency platform, you're tasked with ensuring the security and integrity of our newly established infrastructure. With over 1 million users relying on our services, it's imperative that we maintain the highest standards of security to protect their digital assets.

Your role involves a comprehensive review of the security landscape for our new application technology stack, identifying potential vulnerabilities, and running scans to assess any existing threats. Your scope encompasses various entities within our architecture, including the application itself, containerized services, and the external-facing API.

Ultimately, your objective is to develop a robust remediation plan that not only addresses current vulnerabilities but also strengthens our overall security posture, safeguarding both user data and the platform's reputation. This critical mission presents an exciting opportunity to leverage your skills and expertise in cybersecurity to fortify our infrastructure and uphold our commitment to providing a secure and reliable platform for our users. Let's embark on this journey together to ensure CryptoV4ult remains a trusted leader in the cryptocurrency industry!

Section One: Integrating SDLC

Transitioning to Secure SDLC

Secure Requirements Phase:
• Conduct user interviews to gather functional requirements • Write a requirements document for task management features • Perform threat modeling and risk assessment on the requirements • Define security objectives and security requirements for the application • Conduct security requirements workshops with stakeholders to align on security goals • Identify sensitive data handling requirements and define appropriate security controls
Secure Design Phase:
• Create a high-level architecture diagram for the application • Design the database schema for tasks • Incorporate security controls and mechanisms into the architecture (e.g., access controls, data encryption, secure communication) • Perform a security design review to identify potential vulnerabilities • Develop secure API design and documentation, including authentication and authorization mechanisms • Review the API design for secure data exchange and error handling

Transitioning to Secure SDLC

Secure Development Phase:
• Code the user interface using HTML and CSS • Implement interactive elements using JavaScript, following secure coding practices • Set up a Flask application to handle API requests, with a focus on input validation and secure coding • Implement CRUD operations for tasks, ensuring data integrity and secure data handling • Integrate static code analysis tools to identify and remediate security vulnerabilities during the development process • Implement secure logging and monitoring mechanisms for the application
Secure Testing Phase:
Write and execute functional test cases, including security-focused test scenarios Conduct browser compatibility testing, evaluating the application's security posture across different browsers and versions Perform dynamic security testing and penetration testing to identify and address vulnerabilities Execute API security testing and fuzzing to validate the security of the application's API Scan and analyze the containerized services for known vulnerabilities and security configuration issues Validate the secure implementation of authentication and authorization mechanisms

Transitioning to Secure SDLC

Secure Deployment Phase:
• Deploy the application to Heroku, following secure infrastructure provisioning and configuration practices • Implement security monitoring and logging mechanisms to detect and respond to security incidents • Establish a secure continuous integration and delivery pipeline, with security checks integrated • Perform smoke testing on the deployed application, verifying the security posture of the production environment • Implement secure backup and disaster recovery procedures to ensure the application's resilience • Configure secure network settings and firewall rules for the deployed infrastructure
Secure Maintenance Phase:
• Monitor application logs and fix reported issues, addressing any security-related concerns • Gather user feedback for future feature additions, considering security implications • Implement a vulnerability management process to stay up-to-date with security patches and updates • Deliver security awareness training for the development and operations teams to foster a security-conscious culture • Conduct periodic security assessments and audits to identify and address emerging threats • Establish a process for secure software version control and release management

Advocating for Secure SDLC

1. Improved Compliance and Regulatory Adherence:

The Secure SDLC approach integrates security measures throughout the development lifecycle, ensuring that vulnerabilities are identified and addressed early on. This proactive approach reduces the risk of security breaches and protects the sensitive user data that is crucial for a Cryptocurrency platform.

2. Faster Time to Market:

The Secure SDLC approach promotes early identification and resolution of security issues, enabling a more streamlined development process and reducing delays caused by security-related rework. This allows CryptoV4ult to deliver new features and updates to the platform more efficiently, catering to the dynamic demands of the Cryptocurrency market.

3. Enhanced Stakeholder Trust and Brand Reputation:

Demonstrating a strong commitment to security through the Secure SDLC framework can bolster customer confidence, improve the platform's reputation, and differentiate CryptoV4ult as a trusted provider in the Cryptocurrency industry. This is particularly crucial for a platform that handles sensitive financial transactions and user data, where security is a paramount concern for its user base.

4. Enhanced Security Posture:

The Secure SDLC approach integrates security measures throughout the development lifecycle, ensuring that vulnerabilities are identified and addressed early on. This proactive approach reduces the risk of security breaches and protects the sensitive user data that is crucial for a Cryptocurrency platform.

5. Reduced Costs and Remediation Efforts:

Implementing security controls upfront during the development process is more cost-effective than addressing vulnerabilities in a reactive manner during the later stages or after deployment. This approach helps CryptoV4ult avoid the significant expenses and disruptions associated with security breaches and post-deployment remediation.

Section Two:

Vulnerabilities and Remediation

Vulnerabilities and remediation

1. Weak Password Policies
Description
Lack of strong password requirements, allowing users to create weak or easily guessable passwords.
Risk
Increased susceptibility to brute-force attacks and credential stuffing, leading to unauthorized access to user accounts.
Remediation
Implement robust password policies, including minimum length, complexity requirements, and regular password expiration. Enforce the use of multi-factor authentication (MFA) to add an additional layer of security.

Vulnerabilities and remediation

2. Insufficient Account Lockout Mechanisms
Description
Absence of robust account lockout policies, allowing attackers to repeatedly attempt unauthorized access without consequences.
Risk
Increased vulnerability to brute-force attacks and credential stuffing, potentially leading to account takeovers.
Remediation
Implement account lockout mechanisms that temporarily disable accounts after a specified number of failed login attempts. Ensure that the lockout duration and reset policies are appropriate to mitigate the risk of unauthorized access.

Vulnerabilities and remediation

3. Insecure Session Management
Description
Weak session token generation, improper session invalidation, or lack of secure session expiration policies.
Risk
Potential session hijacking, session fixation, and unauthorized access to user accounts.
Remediation
Implement strong session token generation, ensure secure session expiration, and enforce session invalidation upon logout or inactivity. Use secure session management protocols, such as HTTP-only and Secure cookies, to mitigate the risk of session-based attacks.

Threat Matrix

Pathway (Vulnerability)	Impact Level	Likelihood Level
Weak Password Policies	High	High
Insufficient Account Lockout Mechanisms	Low	Medium
Insecure Session Management	High	Medium

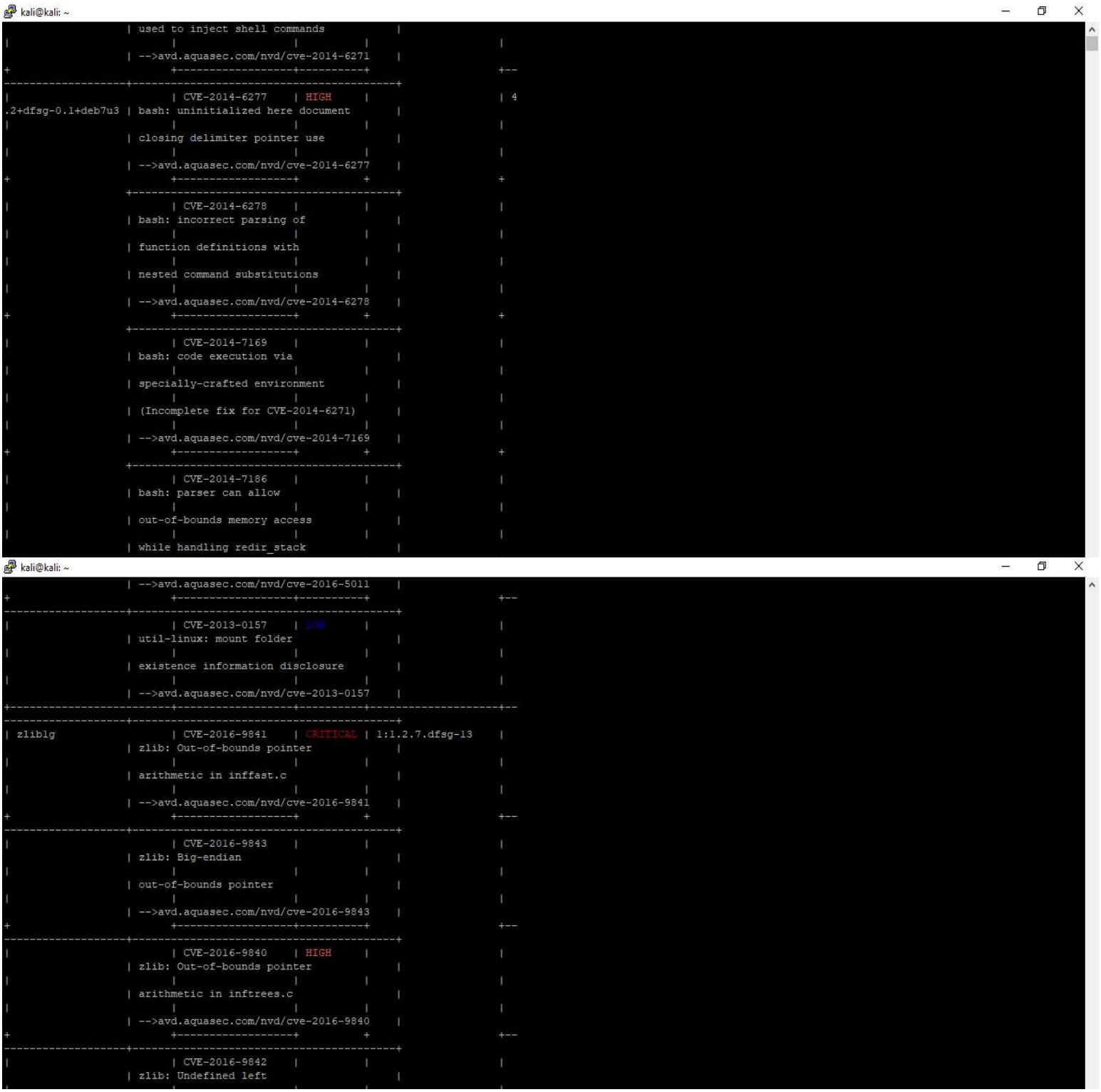
Fill out the matrix table. Impact levels are horizontal, and likelihood levels at the vertical axis.

Impact / Likelihood	Low	Medium	High
High			Weak Password Policies
Medium	Insufficient Account Lockout Mechanisms		Insecure Session Management
Low			

Section Three: Container Security

Trivy scan screenshot

Place a screenshot from the Trivy scan results on this slide.



Report to Fix Container Issues

Fill out the report with at least 7 items.

Issues	Unpatched Software Version	Patched Software Version
Bash: CVE-2014-6277	Bash: 4.2+dsfg-0.1	Bash: 4.2+dsfg-0.1+deb7u1
Bash: CVE-2014-6271	Bash: 4.2+dsfg-0.1	Bash: 4.2+dsfg-0.1+deb7u1
Glibc: CVE-2013-0157	Glibc: 2.17-0ubuntu5	Glibc: 2.17-222+deb9u1
Libxml2: CVE-2016-9841	Libxml2: 2.9.4	Libxml2: 2.9.4-14.1+deb9u1
Libxml2: CVE-2016-9840	Libxml2: 2.9.4	Libxml2: 2.9.4-14.1+deb9u1
Libxml2: CVE-2016-9842	Bash: 4.3	Bash: 4.3-14.1+deb9u1
Libxml2: CVE-2016-9843	Libxml2: 2.9.4	Libxml2: 2.9.4-14.1+deb9u1

Section Four: API Security

API Vulnerabilities and remediation

1. Improper Input Validation

Description

Lack of robust input validation on the API endpoints, allowing for injection attacks such as SQL injection or XML injection. Attackers can craft malicious input data that can be executed by the server, potentially granting them unauthorized access to sensitive user data or the ability to manipulate the server-side application.

Risk

Improper input validation can lead to a range of security issues, including:

- Unauthorized access to sensitive user data stored in the database, such as personal information, financial details, or other confidential details.
- Data manipulation, where the attacker can inject malicious commands to modify or delete user data.
- Server-side code execution, where the attacker can gain control of the server and potentially expand their access to other parts of the system.

Remediation

To mitigate the risks of improper input validation, the following remediation strategies should be implemented:

- Implement strict input validation and sanitization on all API endpoints, checking the format, length, and type of input data to ensure it conforms to expected parameters.
- Use parameterized queries or prepared statements when interacting with the database to prevent SQL injection attacks.
- Employ input validation libraries or frameworks, such as those provided by the API development framework (e.g., Flask-Inputs for Python), to streamline and standardize the input validation process.
- Regularly review and update the input validation mechanisms to address evolving security threats and industry best practices.
- Implement server-side input validation as the primary defense against injection attacks, rather than relying solely on client-side validation.

API Vulnerabilities and remediation

2. Broken Authentication and Authorization

Description

Weak or insecure implementation of authentication and authorization mechanisms in the API, allowing for unauthorized access to sensitive user data and resources. This can include issues such as the use of weak or default credentials, lack of multi-factor authentication, improper session management, and overly permissive access controls.

Risk

- Broken authentication and authorization can lead to the following risks:
- Unauthorized access to the user data shared with the external sales vendor, enabling the attacker to retrieve or manipulate confidential information such as personal details, financial records, and behavioral data.
- Privilege escalation, where an attacker can gain elevated access privileges and expand their control over the API and the underlying system.
- Session-based attacks, such as session hijacking or session fixation, allowing the attacker to impersonate legitimate users and gain access to their data.
- Potential reputational damage and loss of customer trust if the security breach is discovered, as well as regulatory and legal consequences for CryptoV4ult.

Remediation

To address the risks of broken authentication and authorization, CryptoV4ult should implement the following remediation strategies:

- Enforce the use of strong, unique passwords for all user accounts, and implement password complexity requirements, regular password expiration, and password history checks.
- Implement multi-factor authentication (MFA) for all user accounts, including the external sales vendor, to add an extra layer of security beyond just username and password.
- Implement robust session management controls, such as using secure session tokens, enforcing session timeouts, and invalidating sessions upon logout or inactivity.
- Establish fine-grained authorization controls to ensure that users and the external sales vendor can only access the resources they are explicitly authorized to interact with.

API Vulnerabilities and remediation

3. Insufficient API Versioning and Deprecation

Description

Lack of proper versioning and deprecation of API endpoints, leading to the exposure of outdated and potentially vulnerable API versions. This can occur when the API development team does not effectively manage the lifecycle of API versions, resulting in the continued availability of deprecated and insecure API versions

Risk

Insufficient API versioning and deprecation can lead to the following risks:

- Exploitation of known vulnerabilities in deprecated API versions, potentially allowing unauthorized access to user data or other sensitive information shared with the external sales vendor.
- Increased attack surface, as the availability of multiple API versions can provide more opportunities for attackers to find and exploit vulnerabilities.
- Confusion and potential security issues for API consumers, including the external sales vendor, who may continue to use outdated and vulnerable API versions.
- Regulatory and compliance concerns if the deprecated API versions do not meet the necessary security standards or data protection requirements.

Remediation

The risks of insufficient API versioning and deprecation, CryptoV4ult should implement the following remediation strategies:

- Establish a clear API versioning strategy, including versioning conventions, release cadence, and deprecation timelines.
- Implement a versioning scheme (e.g., semantic versioning) that allows for clear differentiation between major, minor, and patch releases.

Develop a process for deprecating old API versions, including:

- Clearly communicating the deprecation timeline and instructions for migrating to the latest version to all API consumers, including the external sales vendor.
- Actively monitoring the usage of deprecated API versions.