UDACITY

CryptoV4ult Enterprise Security Review

Anderson Okai

1st Aug

Project Information Slide

How to Use this Template

- We have provided these slides as a guide to ensure you submit all the required components to complete your project successfully.
- When presenting your project, remember that these slides are merely a guide. We strongly encourage you to embrace your creative freedom and make changes that reflect your unique vision as long as the required information is present.
- You can add slides to the template when your answers or screenshots do not fit on the previously provided pages.
- Delete this and all other project instruction slides before submitting your project.
- **Remember to add your name and the date to the cover page.**

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

As the lead security engineer at CryptoV4ult, you are tasked with ensuring the new infrastructure is developed securely. Your responsibility is to reorganize the existing development tasks to fit into a Secure Software Development Lifecycle (SDLC) framework, ensuring that each stage of the lifecycle incorporates necessary security tasks to protect user data and maintain the integrity of the cryptocurrency platform.

- ***Reorganize the Waterfall*** *task list from the next slide* ***into the Secure SDLC phases***
- ***Add at least one security related additional task to each phase***

Transitioning to Secure SDLC

Place every task into a Secure SDLC category in the next few slides. Add at least one additional task to each phase that helps enhance security.

1. Conduct user interviews to gather functional requirements.
2. Write a requirements document for task management features.
3. Create a high-level architecture diagram for the application.
4. Design the database schema for tasks.
5. Code the user interface using HTML and CSS.
6. Implement interactive elements using JavaScript.
7. Set up a Flask application to handle API requests.
8. Implement CRUD operations for tasks.
9. Write and execute functional test cases.
10. Conduct browser compatibility testing.
11. Deploy the application to Heroku.
12. Perform smoke testing on the deployed application.
13. Monitor application logs and fix reported issues.
14. Gather user feedback for future feature additions.

Transitioning to Secure SDLC

Requirements Analysis

- Conduct user interviews to gather functional requirements.
- Write a requirements document for task management features.
- Additional Security Task: Conduct a security requirements analysis to identify security needs and potential threats early in the project.

Design

- Create a high-level architecture diagram for the application.
- Design the database schema for tasks.
- Additional Security Task: Perform a threat modeling exercise to identify and mitigate potential security risks in the architecture and database design.

Transitioning to Secure SDLC

Development
• Code the user interface using HTML and CSS. • Implement interactive elements using JavaScript. • Set up a Flask application to handle API requests. • Implement CRUD operations for tasks. • Additional Security Task: Implement secure coding practices and conduct static code analysis to identify and address security vulnerabilities in the codebase.
Testing
• Write and execute functional test cases. • Conduct browser compatibility testing. • Additional Security Task: Perform security testing, including vulnerability scanning, penetration testing, and security code reviews to identify and address security issues.

Transitioning to Secure SDLC

Deployment
● Deploy the application to Heroku. ● Perform smoke testing on the deployed application. ● Additional Security Task: Conduct a security assessment of the deployment environment, ensuring secure configurations and compliance with security policies.
Maintenance
● Monitor application logs and fix reported issues. ● Gather user feedback for future feature additions. ● Additional Security Task: Implement a continuous monitoring and incident response plan to detect, respond to, and recover from security incidents promptly.

Advocating for Secure SDLC

As the lead security engineer at CryptoV4ult, you're spearheading the shift towards a more secure and agile development process. To get everyone on board, create a succinct list highlighting five essential advantages of transitioning to the Secure Software Development Lifecycle (SDLC) from our current Waterfall methodology. **For each advantage, include a brief explanation** that underscores its importance, particularly focusing on how it benefits the dynamic and security-centric nature of our cryptocurrency platform.

- *Write your answers on the next slide!*

Advocating for Secure SDLC

1. Early Detection and Mitigation of Security Risks
Explanation: Secure SDLC includes security practices throughout all levels of development life potential risks on the earlier stages and this helps to reduce vulnerabilities. This approach reduces the chances of a security breach and makes sure as many protocol level vulnerabilities are addressed prior to anyone picking it out which is critical for any cryptocurrency platform.
2. Enhanced Collaboration and Communication
Explanation: Secure SDLC is an iterative and incremental model, that puts in theme of "early" the collaboration between developers (if it's about a coder), security experts, and stakeholders. This establishes a culture of ongoing cooperation, with security needs in the pipeline managed appropriately and enforcement continuously monitored.
3. Improved Compliance with Regulatory Standards
Explanation: The frameworks for Secure SDLC are sometimes industry standards, regulatory driven to meet the development practices legally or by regulation. Especially given the nature of financial regulations to which CryptoV4ult must be compliant inorder to not only work trust but also stay out of legal trouble.
4. Increased Agility and Flexibility
Explanation: Secure SDLC methodologies, such as Agile or DevSecOps are more to flexible and adaptive development processes compared rigid Waterfall. This flexibility allows the team to rapidly respond to updated requirements, security threats and market conditions so that our platform is both secure and competitive_MODIFIED_LAYOUT (main author of this info).
5. Cost and Time Efficiency

Section Two:

Vulnerabilities and Remediation

Vulnerabilities and remediation

As CryptoV4ult enhances its infrastructure to support new features for its extensive user base, ensuring the security of user authentication mechanisms is paramount. The **login system** is critical to the platform's security, acting as the first line of defense against unauthorized access. Your task is to scrutinize a login system, **identify 3 potential vulnerabilities** they usually have, and propose effective remediation strategies.

- *Concentrate on **login systems in general***
- *The vulnerability can relate to any aspect of a login system, including user identification, authentication mechanisms, and session management*
- *Any common login system vulnerability is acceptable*
- ***For each identified potential vulnerability**, you need to:*
    - ***Describe the vulnerability***
    - ***Explain the risk***
    - ***Provide remediation strategy***

Vulnerabilities and remediation

1. Weak Password Policies
Description
Weak password policies enable users to choose basic passwords such as "password123" or even just "123456". This provides the opportunity for attackers to guess valid passwords via brute force or dictionary attacks.
Risk
If an attacker guesses a weak password, they can take full control of the user's account, potentially leaking sensitive information and performing unauthorized actions.
Remediation
Implement mandatory strong password policies: minimum length (e.g., 12 characters), complexity (mixes uppercase, lowercase letters, numbers, and special characters) including periodic password changes. Use account lockout mechanisms after a certain number of failed login attempts to prevent brute force attacks.

Vulnerabilities and remediation

2. Lack of Multi-Factor Authentication (MFA)
Description
MFA is the only way to secure password-only access. Attackers can easily access the account without any further verification if passwords are compromised.
Risk
Single-factor authentication exposes a greater risk to unauthorized entry in case of leaked, phished, or guessed passwords. This can result in data breaches and unauthorized transactions.
Remediation
Implement MFA, ensuring that more than one form of identity verification is required to process a transaction (e.g., requiring the user to provide a password along with something they have access to, such as an SMS/Email OTP or biometric scan). This adds an extra layer of security, making it significantly harder for attackers to gain unauthorized access.

Vulnerabilities and remediation

3. Session Hijacking
Description
Session hijacking occurs when an attacker steals or guesses a user's session ID, allowing them to impersonate the user without needing to log in.
Risk
Compromising and grabbing control of the account. An attacker can perform any actions the legitimate user is authorized to do, including viewing sensitive information, transferring money, or changing settings.
Remediation
Securely manage sessions with practices like using random, long session IDs and requiring users to log in again after a period of inactivity. Encrypt session IDs in transit by using HTTPS and set secure cookies with the HttpOnly and Secure flags to prevent JavaScript access and ensure cookies are sent over encrypted connections only. Ensure sessions expire after inactivity and require re-authentication for sensitive actions.

Create a threat Matrix

Dissect and categorize the 3 vulnerabilities that you have identified for the login system. Understanding these vulnerabilities from a strategic viewpoint will enable the company to allocate resources efficiently, prioritize remediation efforts, and maintain CryptoV4ult's reputation as a secure and reliable platform.

- *For each identified vulnerability, critically assess its potential to disrupt CryptoV4ult's operational functionality, erode customer trust, and impact financial stability.* ***Assign an impact level of 'Low', 'Medium', or 'High'*** *based on the evaluated potential consequences.*
- *Analyze the complexity and feasibility of exploiting each identified vulnerability. Consider the sophistication required for exploitation and the accessibility of the vulnerability to potential attackers.* ***Rate the likelihood of exploitation as 'Low', 'Medium', or 'High'****.*
- *Utilize the provided risk matrix framework to* ***map out the vulnerabilities*** *according to your assessments of their impact and exploit likelihood.*

Threat Matrix

Pathway (Vulnerability)	Impact Level	Likelihood Level
Weak Password Policies	High	High
Lack of Multi-Factor Authentication (MFA)	High	Medium
Session Hijacking	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
Medium			Lack of Multi-Factor Authentication Session Hijacking
Low			

Section Three: Container Security

Project Information Slide

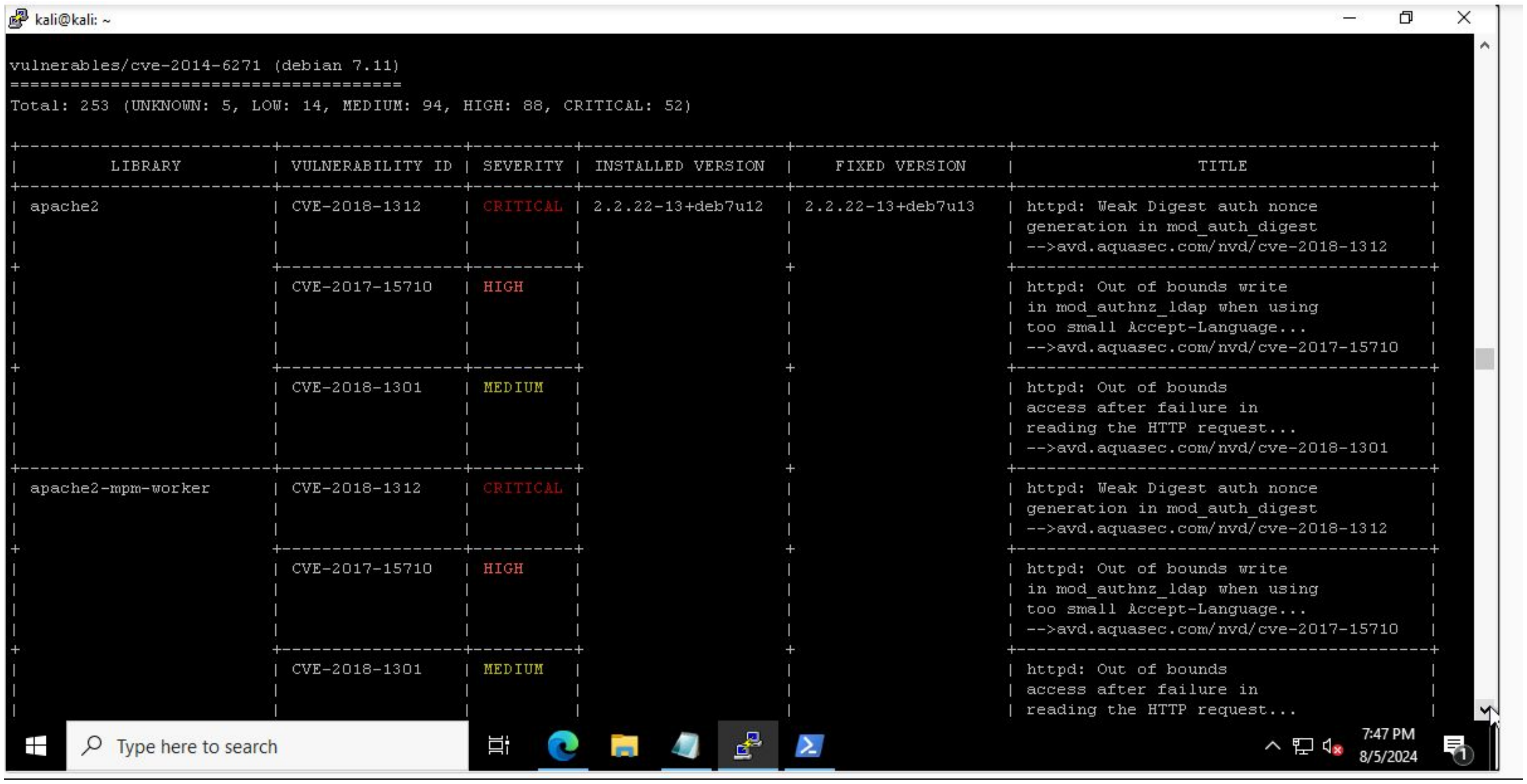
Container Security

It is time to delve into the container services underpinning CryptoV4ult's application infrastructure by scanning for potential vulnerabilities. Scan one of the container services running in the application (located at vulnerables/cve-2014-6271) and identify potential vulnerabilities. Then, you will build a remediation plan to resolve some of the container vulnerabilities.

- *Using **Trivy**, run a **scan** against the container located at **vulnerables/cve-2014-6271**. You can run this scan from the Kali VM in the lab where Trivy is located or from your own computer*
- *Create a **screenshot** of the **Trivy scan results** (it does not have to show all the results) and place it on the next slide*
- ***Fill out the Report** to Fix Container Issues with at least 7 items*

Trivy scan screenshot

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



kali@kali: ~

```
vulnerables/cve-2014-6271 (debian 7.11)
=======================================
Total: 253 (UNKNOWN: 5, LOW: 14, MEDIUM: 94, HIGH: 88, CRITICAL: 52)

+--------------------------+-------------------+----------+-------------------+---------------------+-------------------------------------------+
|         LIBRARY          | VULNERABILITY ID  | SEVERITY | INSTALLED VERSION |    FIXED VERSION    |                   TITLE                   |
+--------------------------+-------------------+----------+-------------------+---------------------+-------------------------------------------+
| apache2                  | CVE-2018-1312     | CRITICAL | 2.2.22-13+deb7u12 | 2.2.22-13+deb7u13   | httpd: Weak Digest auth nonce             |
|                          |                   |          |                   |                     | generation in mod_auth_digest             |
|                          |                   |          |                   |                     | -->avd.aquasec.com/nvd/cve-2018-1312      |
+                          +-------------------+----------+                   +                     +-------------------------------------------+
|                          | CVE-2017-15710    | HIGH     |                   |                     | httpd: Out of bounds write                |
|                          |                   |          |                   |                     | in mod_authnz_ldap when using             |
|                          |                   |          |                   |                     | too small Accept-Language...              |
|                          |                   |          |                   |                     | -->avd.aquasec.com/nvd/cve-2017-15710     |
+                          +-------------------+----------+                   +                     +-------------------------------------------+
|                          | CVE-2018-1301     | MEDIUM   |                   |                     | httpd: Out of bounds                      |
|                          |                   |          |                   |                     | access after failure in                   |
|                          |                   |          |                   |                     | reading the HTTP request...               |
|                          |                   |          |                   |                     | -->avd.aquasec.com/nvd/cve-2018-1301      |
+--------------------------+-------------------+----------+                   +                     +-------------------------------------------+
| apache2-mpm-worker       | CVE-2018-1312     | CRITICAL |                   |                     | httpd: Weak Digest auth nonce             |
|                          |                   |          |                   |                     | generation in mod_auth_digest             |
|                          |                   |          |                   |                     | -->avd.aquasec.com/nvd/cve-2018-1312      |
+                          +-------------------+----------+                   +                     +-------------------------------------------+
|                          | CVE-2017-15710    | HIGH     |                   |                     | httpd: Out of bounds write                |
|                          |                   |          |                   |                     | in mod_authnz_ldap when using             |
|                          |                   |          |                   |                     | too small Accept-Language...              |
|                          |                   |          |                   |                     | -->avd.aquasec.com/nvd/cve-2017-15710     |
+                          +-------------------+----------+                   +                     +-------------------------------------------+
|                          | CVE-2018-1301     | MEDIUM   |                   |                     | httpd: Out of bounds                      |
|                          |                   |          |                   |                     | access after failure in                   |
|                          |                   |          |                   |                     | reading the HTTP request...               |
```

Report to Fix Container Issues

Fill out the report with at least 7 items. Make sure to write the ***Issues in the correct form of (Application Name: CVE number).***

Issues	**Unpatched Software Version**	**Patched Software Version**
Apache2 CVE-2018-1312	2.2.22-13+deb7u12	2.2.22-13+deb7u13
Apache2 CVE-2017-15710	2.2.22-13+deb7u12	2.2.22-13+deb7u13
Apache2 CVE-2018-1301	2.2.22-13+deb7u12	2.2.22-13+deb7u13
Apache2-mpm-worker CVE-2018-1312	2.2.22-13+deb7u12	2.2.22-13+deb7u13
Apache2-mpm-worker CVE-2017-15710	2.2.22-13+deb7u12	2.2.22-13+deb7u13
Apache2-mpm-worker CVE-2018-1301	2.2.22-13+deb7u12	2.2.22-13+deb7u13
Apache2-mpm CVE-2018-1312	2.2.22-13+deb7u12	2.2.22-13+deb7u13

Section Four: API Security

Project Information Slide

API Security

Management has partnered with an external sales vendor and asked for a generic API to be developed that tracks user's data. Based on the data ingested they will create targeted sales advertisements to the customer base, this means a lot of confidential info about the users will be shared to 3rd party vendors.

You need to **identify 3 common API vulnerabilities** and propose effective remediation strategies. Keep in mind this code does not exist; this is the initial stages of development, and you are providing guidance to the engineering team. Feel free to make any assumptions about API features, implementations, and what private data might be shared.

- ***For each identified common API vulnerability:***
    - ***Describe the vulnerability***
    - ***Explain the risk***
    - ***Provide remediation strategy***

API Vulnerabilities and remediation

1. Insecure API Authentication and Authorization
Description
Insecure authentication and authorization mechanisms can lead to unauthorized access to the API. This might occur due to weak or missing authentication protocols, poor session management, or insufficient authorization checks.
Risk
If authentication and authorization are not properly implemented, malicious users could gain access to sensitive user data or perform unauthorized actions, leading to data breaches, loss of customer trust, and potential legal consequences.
Remediation
● *Implement Strong Authentication: Use OAuth 2.0 or OpenID Connect for secure authentication mechanisms. Ensure multi-factor authentication (MFA) is implemented for added security.*   ● *Enforce Least Privilege: Implement role-based access control (RBAC) to ensure users and systems only have access to the necessary data and actions.*   ● *Secure Session Management: Use secure cookies with proper flags (e.g., HttpOnly, Secure, SameSite) and ensure session tokens are short-lived and rotated regularly.*

API Vulnerabilities and remediation

2. Lack of Data Encryption
Description
If data transmitted between the client and the API is not encrypted, it can be intercepted by attackers. This includes both data in transit (between client and server) and data at rest (stored in databases).
Risk
Unencrypted data can lead to the exposure of sensitive user information such as personal details, payment information, and other confidential data. This can result in data theft, fraud, and damage to the company's reputation.
Remediation
● *Encrypt Data in Transit: Use HTTPS with TLS (Transport Layer Security) to encrypt data between the client and server.* ● *Encrypt Data at Rest: Ensure that sensitive data stored in databases is encrypted using strong encryption algorithms (e.g., AES-256).* ● *Use Secure APIs: Ensure that any third-party APIs being used also implement strong encryption practices.*

API Vulnerabilities and remediation

3. Improper Input Validation
Description
APIs that do not properly validate input can be vulnerable to various injection attacks (e.g., SQL injection, Cross-Site Scripting (XSS), XML External Entity (XXE) attacks).
Risk
Injection attacks can lead to unauthorized access, data manipulation, or even complete system compromise. Attackers can exploit these vulnerabilities to steal data, execute malicious code, or disrupt service operations.
Remediation
● *Input Sanitization: Validate and sanitize all inputs to ensure they conform to expected formats and types. Use whitelisting techniques where possible.* ● *Parameterized Queries: Use parameterized queries or prepared statements for database interactions to prevent SQL injection.* ● *Security Libraries: Use established security libraries and frameworks that provide built-in protection against common injection attacks.* ● *Regular Security Testing: Perform regular security testing, including static code analysis and penetration testing, to identify and remediate input validation vulnerabilities.*