SECURITY ASSESSMENT

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 1:
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
[1. Conduct user interviews to gather functional requirements. 2. Write a requirements document for task management features. Additional task: Define and document security requirements and compliance standards.]
Design
[3. Create a high-level architecture diagram for the application. 4. Design the database schema for tasks. Additional task: Perform threat modeling to identify potential security risk.]

Transitioning to Secure SDLC

Development

[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.
Additional task: Apply secure coding practices and conduct code reviews focused on security vulnerabilities.]

Testing

[9. Write and execute functional test cases.
10. Conduct browser compatibility testing.
12. Perform smoke testing on the deployed application.
Additional task: Perform security testing such as static code analysis and penetration testing.]

Transitioning to Secure SDLC

Deployment

[11. Deploy the application to Heroku.
Additional task: Configure security monitoring for the Heroku application by using logging tools, platform monitoring add-ons.]

Maintenance

[13. Monitor application logs and fix reported issues.
14. Gather user feedback for future feature additions.
Additional task: Fix any repoted issues from logs or user feedback 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 vulnerability detection]
[Secure SDLC integrates security checks from the initial stages, allowing vulnerabilities to be identified early. This reduce costly fixes later and protects our sensitive crypto platform from early exploitation risks.]
2. [Continuous security integration]
[Unlike waterfall's linear process, secure SDLC promotes ongoing security assessments at every phase. This ensure that evolving threats in the cryptocurrency space are continuously countered, keeping the platform robust and adaptive.]
3. [Faster response to threats]
[Security practices in secure SDLC enable rapid iteration and quick patching of new vulnerabilities.]
4. [Enhanced developer security awareness]
[Secure SDLC includes security training and automated tools, cultivating a security-first mindset among developers. This reduces errors that could lead to breaches in our sensitive platform.]
5. [Reduced costs and resource waste]
[Fixing security issues early in the development reduces expensive late-stage remediation or post release fixes, optimizing resource allocation and speeding up time-to-market without sacrificing security.]

Section 2: 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. [Flawed Brute-Force Protection]
Description
[What specific weakness does this vulnerability represent? ***Insufficient safeguards, lack of lockout mechanisms, absence or weakness in reat limiting allows attackers to gain access systematically guess credentials or authentication keys.*** *How does it work?* ***Attackers first identify targets and attempt initial reconnaissance to find username and password combination that grants access to an account or system, then they use Automated Login Attempts against the targeted account and Exploiting Flaws in Protection like No Reat Limiting, Weak Lockouts, Lack of Multi-Factor Authentication etc. If an attacker succeeds in cracking one account, they might try Credential Stuffing to access more account and trying to Compromised Credentials by performing unathorized actions.*** *]*
Risk
[How can attackers use this flaw, and what harm could it do? ***Attackers can use automated large-Scale guesses to crack usernames and passwords for gaining unauthorized access.*** *]*
Remediation
[How can you fix or reduce the risk from this flaw? ***Implement account lockout policies, rate limiting, IP monitoring, application firewalls and CAPTCHAs to block automated attempts.*** *]*

Vulnerabilities and remediation

2. [*SQL Injection*]
Description
[What specific weakness does this vulnerability represent? ***Improper netutralization of special elements used in an SQL command, improper input validation in login inputs, allowing attackers to insert malicious SQL code into an applications input fildes allow attackers to execute malicious SQL queries to access or bypass user authentication data, view or modify sensitive dat, protentially compromise entire server and leading to data breaches.*** *How does it work?* ***The attacker begins with Vulnerable Entry Point of a web application uses user-supplied input and trying to Inject Malicious Code and Manipulating Querys. If the application fails to neturalize malicious input, the code is executed by the backend database server and attacker try to Compromise Data by using Bypass Login, Data Exfiltration, Modification, Blind SQL Injection.]***
Risk
[How can attackers use this flaw, and what harm could it do? ***Inject malicious SQL code via login inputs can bypass authentication and retrieve sensitive data.]***
Remediation
[How can you fix or reduce the risk from this flaw? ***Use parameterized quries and inpute validation to block SQL injection attacks, sanitize all inputes, and employ an ORM (Object-Relational Mapping).]***

Vulnerabilities and remediation

3. [*Cross-Site-Scripting (XSS)*]
Description
[What specific weakness does this vulnerability represent? ***Cross-Site-Scripting represents a weakness in a web application's ability to properly handle user input, login forms that reflact user input without proper sanitization, may allow attackers to inject malicious scripts into legitimate that can steal credentials or perfome malicious actions.*** *How does it work?* ***Attacker sends a malicious script, often within a URL or a form filed on a vulnerable login page to the web application. The web application receives this input and without properly validating it, incorporates it into its output to other users. When a legitimate user visit the loging page or interacts with the compromised part of the application, their browser receives and executes the injected scriept as its part of the trusted website and attackers can access the user's session data, such as login cookies, also can capture username, password.]***
Risk
[How can attackers use this flaw, and what harm could it do? ***Theft of user credentials and session cookies, impersonation of users unauthorised actions on their behalf, access sensitive user information, sprading malware, redirecting users to malicious site.]***
Remediation
[How can you fix or reduce the risk from this flaw? ***Implement Input Validation and output Encoding, Sanities inputs. Impliment CSP (Content Security Policy) headers to restrict script sources and reduce risk. Use security framework that automatically handle encoding and sanitization.]***

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
Flawed Brute-Force Protection	High	High
SQL Injection	High	Medium
Cross-Site-Scripting (XSS)	Medium	Medium

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

Impact / Likelihood	Low	Medium	High
High	Medium	Medium High *SQL Injection*	High *Flawed Brute-Force Protection*
Medium	Low Medium	Medium *(XSS)*	Medium High
Low	Low	Low Medium	Medium

Section 3:
Container Security

Container Security

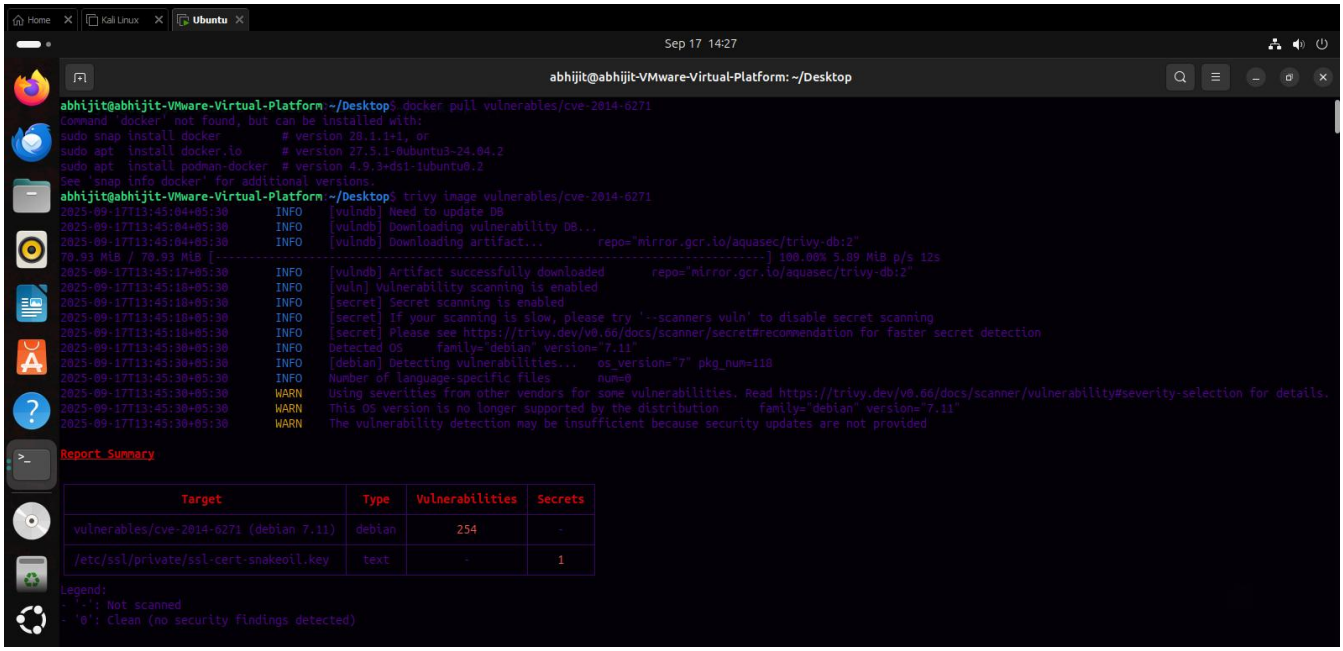
It is time to delve into the container services underpinning CryptoV4ult's application infrastructure by scanning for potential vulnerabilities. Scan one of the container services running in the application (located at vulnerables/cve-2014-6271) and identify potential vulnerabilities. Then, you will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

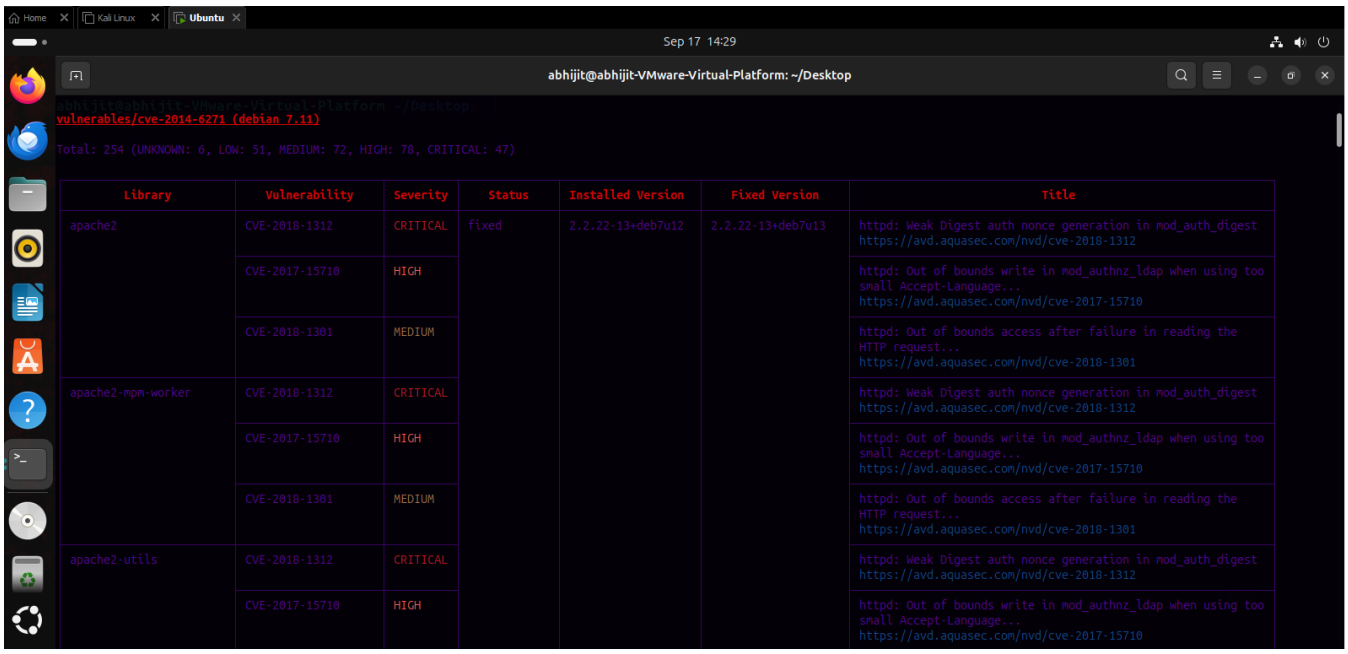
- *Using **Trivy**, run a **scan** against the container located at **vulnerables/cve-2014-6271**. You can run this scan from the Kali VM in the lab where Trivy is located or from your own computer*
- *Create a **screenshot** of the **Trivy scan results** (it does not have to show all the results) and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

Project Information Slide

Trivy scan screenshot

Place a screenshot from the Trivy scan results on this slide. Make sure your hostname is clearly visible.





Report to Fix Container Issues

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

Vulnerability Name	Unpatched Software Version	Patched Software Version
CVE-2018-1312	2.2.22-13+deb7u12	2.2.22-13+deb7u13
CVE-2014-6271	4.2+dfsg-0.1	4.2+dfsg-0.1+deb7u3
CVE-2017-12613	1.4.6-3+deb7u1	1.4.6-3+deb7u2
CVE-2017-12618	1.4.1-3	1.4.1-3+deb7u1
CVE-2018-1126	1:3.3.3-3	1:3.3.3-3+deb7u1
CVE-2017-3735	1.0.1t-1+deb7u2	1.0.1t-1+deb7u3
CVE-2018-1126	1:3.3.3-3	1:3.3.3-3+deb7u1

Section 4:
API Security

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***

Vulnerabilities and remediation

1. [Broken User Authentication]

Description

[What specific weakness does this vulnerability represent?
Broken User Authentication represents a critical vulnerability attackers exploit weaknesses in the authentication process to gain unauthorised access to user accounts. This specific weakness allows attackers to bypass or manipulate authentication machanisms, such as username / password validation or session control, effectively impersonating legitimate user.
How does it work?
Attackers might gain access to user credentials by credential stuffing, breaches, phishing or brute force attacks guessing password without sufficient accuont lockout controls. After logs in the application creates a session, if the session ID is exposed in url and is not properly protected or if sassion are not invalidated after logout, an attacker can steal the session cookie and take over the users active sassion without needing their password. the vulnerability might allow attackers to completely bypass authentication mechanism and ganing access to protected data.]

Risk

[How can attackers use this flaw, and what harm could it do?
The risk of Broken User Authentication is significant and can lead to Unauthorised Data Access, Account Takeover, System Compromise and reputational Damage.]

Remediation

[How can you fix or reduce the risk from this flaw?
Use strong standardized authentication protocols like Oauth 2.0 or open ID connect. Enforce malti-factor authentication (MFA), secure token management, and session expiration policies. Prepare hashing and salting stored passworeds to protect credentials. Account lockout and rate-limiting to prevent Brute-Force and credential stuffing. Secure session management with short session life times, session ID rotation and immediate invalidation upon logout. Regular security testing, including automated vulnerability scanning, code reviews, and penetration testing focused on authentication flows.]

Vulnerabilities and remediation

2. [Excessive Data Exposure]

Description

[What specific weakness does this vulnerability represent?
Excessive Data Exposure is a specific API security weakness where an API unintentionally reveals more data than necessary to the clint, often including sensitive or confidential data. This happens primarily due to poor data filtering and validation on the server side, where APIs send full data objects instead of limiting responses to only the necessary fildes required for a particualr clint request.
How does it work?
During API responses, instead of returning a minimal necessary set of data, the API transmits complete data objects containing sensitive information such as personally identifiable information (PII), financial details or internal business related data. Sometimes developers rely on the clint to filter this data, but since the data is sent to the clint, an attacker can intercept or access it, leading to exposure of data that should remain hidden, like API fetehing user profile data might return not just the users name but also phone number, address, payment details etc unnecessarily.]

Risk

[How can attackers use this flaw, and what harm could it do?
It significantly increases the risk of data breaches, to give access sensitive user information like payment details, address, personal information etc. Organizations face legal and regulatory penalties due to failing to protect sencitive information as per laws like GDPR, HIPPA etc. Attackers gain a broader attack surface and can exploit additional vulnerabilities with insider information which exposed through the API.]

Remediation

[How can you fix or reduce the risk from this flaw?
Implement strict data filtering to ensure only return specific data required by user, implement role-based access control to ensure access only the data essential for the function, Continuously audit APIs for data exposure risks to find and fix potential leaks.]

Vulnerabilities and remediation

3. [*Insecure Direct Object References (IDOR)*]
Description
[What specific weakness does this vulnerability represent? *Insecure Direct Object References (IDOR) represent a specific weakness in access control within web applications. This vulnerability arises when an application uses user-supplied input to directly access objects without properly verifying whether the user is authorized to access them or not. Essentially, IDOR results from missing or inadequate authorization checks when accessing objects identified by user-controlled parameters, like IDs or filenames.* *How does it work?* *application exposes direct references to internal objects via URLs or parameters, Attackers manipulate these references to access unauthorized data or resources. This can lead to horizontal privilege escalation or vertical privilege escalation.]*
Risk
[How can attackers use this flaw, and what harm could it do? *Exposer of sensitive information, Authontication bypass, data alteration on other user's data, Account takenover through iterative or brute-force parameter modification by unauthorised user,]*
Remediation
[How can you fix or reduce the risk from this flaw? *Ensure strict authorization checks before granting access to any object based on user input. Replace direct object references with indriect ones, sanitize and validate all user inputes rigorously, Avoied exposing internal identifiers and implement logging and monitoring to detect and respond to abnormal access patterns or unauthorized attempts.]*