SECURITY ASSESSMENT

STEVEN REJI GEORGE
CBS-041

Section 1:
Integrating SDLC

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
- Hold user interviews to obtain functional requirements ***- Prepare a requirements document for task management features*** ***- Hold THREAD MODELING sessions to reveal potential security threats early.***
Design
- Develop a top-level architecture drawing for the app ***- Plan the database structure for tasks*** ***- Establish safe design patterns.***

Transitioning to Secure SDLC

Development
-Develop the required software ,develop the frontend ,backend . ensure that the backend is communicating with the server and DB In a secure manner. -Initialize an application to handle API request. -Review the code ,ensure that the developed securly and ensure the applicattion is secured from OWASP top 10 vulanarablities(like SQLi ,XSS ,CSRF…..).
Testing
-Conduct the compactability tests of the code(software) before deployment -Conduct the system compactability of the software. -Run static and dynamic anlaysis using various tools

Transitioning to Secure SDLC

Deployment

- ***Deploy the application to Heroku***
- ***Perform smoke testing on the deployed application***
- ***Turn on audit logging for sensitive operations and authentication events***
- ***Set up access controls and employ role-based access for released environments***

Maintenance

-Regular vulnerability scanning and patching of there is any vulnerability

-Monitor logs and fix issues.

-Add new features to the applications if needed

-Test for bugs and fixes.

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

Early vulnerability detection
Security riskss can be identifyed and resolved on the early development stages reducing the cost and risk of fixing them in future
2. End to end security integration.
It promote continues integration with the CI/CD pipelines, integrating security tools
3. Quick incident response and patching
With security integrated into every step, the team is more equipped to identify, triage, and patch vulnerabilities in a timely manner.
4. Compliance and Regulatory Alignment
Secure SDLC is aligned with international security standards (e.g., ISO 27001, GDPR, PCI-DSS), keeping the platform compliant.
5. Develops a Security-First Culture
Engaging developers, testers, and product teams to consider security at each step builds a proactive, not reactive, mindset.

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. Weak password policies.
Description
Enabling users to have short, simple, or easy-to-guess passwords greatly decreases authenthication security
Risk
Easily compromised through brute force or credential stuffing attacks. Compromise of one account can lead to wider platform exploits—particularly if admin accounts are compromised.
Remediation
- Implement strict password policies (minimum length 12 characters, mixed case, numbers, special characters) *- Apply password blacklists to reject compromised credentials* *- Use 2FA for additional protection*

Vulnerabilities and remediation

2.Improper session management
Description
Sessions might not expire properly, or session IDs are predictable or insecurely stored (e.g., no HTTPS or secure cookies).
Risk
Attackers can hijack active sessions with stolen cookies or predictable tokens, achieving unauthorized access without login credentials.
Remediation
- Utilize secure, random session tokens *- Store tokens in secure cookies with flags (`HttpOnly`, `Secure`)* *- Implement session timeouts and automatic logout after inactivity*

Vulnerabilities and remediation

3. Absence of Rate Limiting for Login Attempts
Description
The login process has no limit on login attempts, and hackers are able to run brute force attacks unhindered.
Risk
Attackers can ultimately guess weak passwords or utilize credential lists to gain access, particularly for valuable accounts.
Remediation
- Apply rate limiting and lockouts after a number of failed attempts *- Utilize CAPTCHA to hinder automated login attempts* *- Alert and monitor for unusual login activity*

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 Policy	High	High
Insecure Session Mgmt	High	Medium
No Rate Limiting	Medium	High

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

Impact / Likelihood	Low	Medium	High
High			
Medium			Insecure Session Mgmt.
Low		No Rate Limiting	Weak Psswd Policy

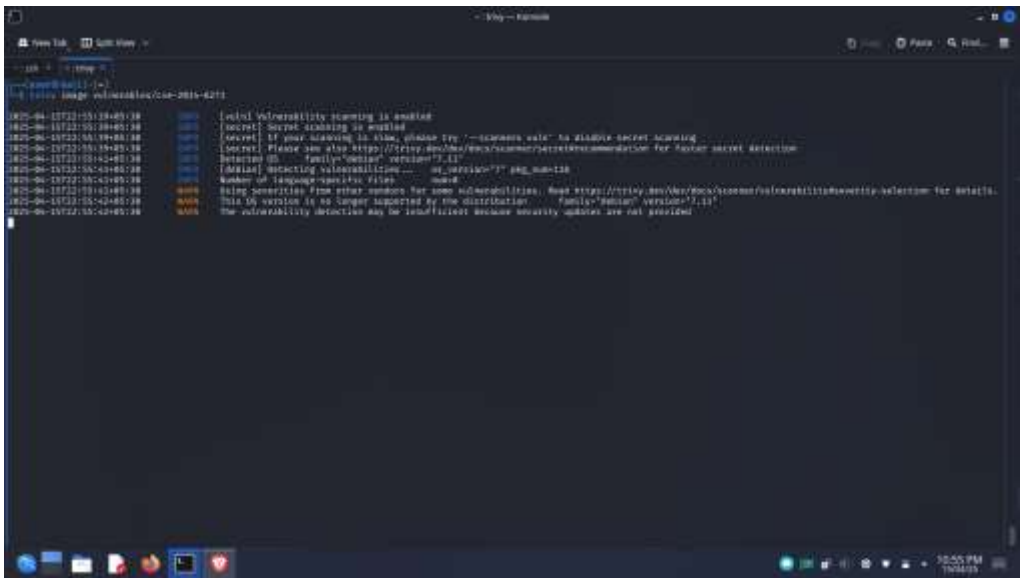
Section 3:
Container Security

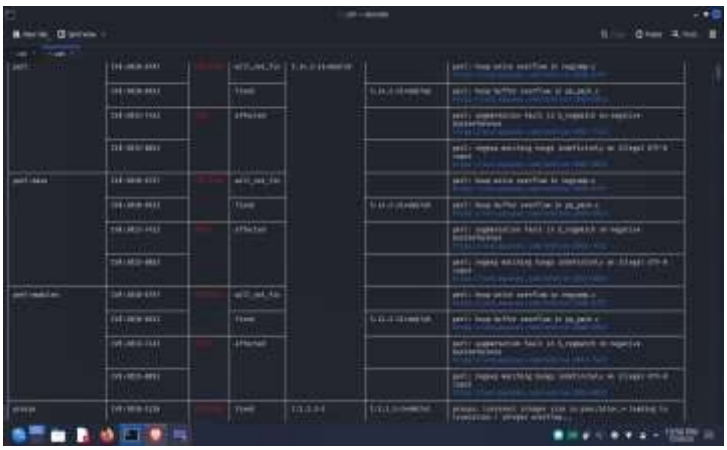
Container Security

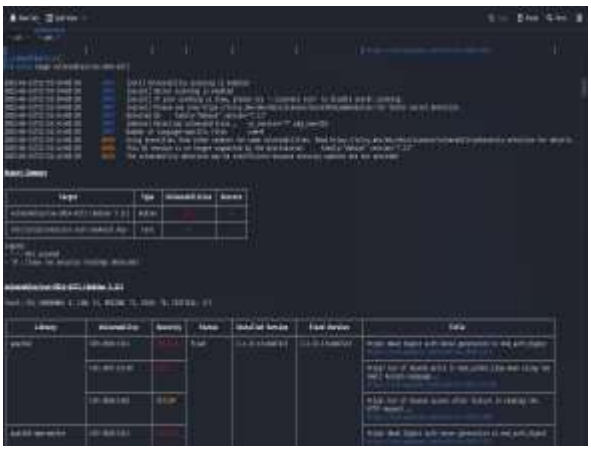
It is time to delve into the container services underpinning CryptoV4ult's application infrastructure by scanning for potential vulnerabilities. Scan one of the container services running in the application (located at vulnerables/cve-2014-6271) and identify potential vulnerabilities. Then, you will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

Trivy scan screenshot







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
apache2: CVE-2018-1312	2.2.22-13+deb7u12	2.2.22-13+deb7u13 or later
bash: CVE-2014-6271	4.2-2ubuntu2	4.3-9ubuntu1.5 or later
libc-bin: CVE-2014-9761	2.13-38+deb7u8	2.13-38+deb7u10 or later
coreutils: CVE-2016-2781	8.13-3.5	8.26-3 or later
libpcre3: CVE-2015-5073	8.30-5	8.39-13 or later
libncurses5: CVE-2017-10684	5.9-10	6.0+20161126-1 or later
libgnutls26: CVE-2017-5336	2.12.20-8	3.5.8-2 OR LATER

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. Brocken Obeject Level Autherization(BOLA)
Description
BOLA results when an API does not effectively enforce user permissions on access to objects *e.g., granting user A access to user B's data by altering a user ID in the request*
Risk
Results in data leakage, unauthorized access to user profiles, and possible privacy breaches it is particularly dangerous when dealing with (Personally Identifiable Information) shared with third parties.
Remediation
- Implement object-level access control checks on all requests. *- Implement contextual authorization (i.e., `user_id` in token should be equal to `user_id` in requested object).* *- Never trust client-side filtering alone.*

Vulnerabilities and remediation

2. Excessive Data Exposure
Description
APIs expose too much data in responses ,the data may be too sensitive so that to harm the organization.
Risk
Expands the attack surface and may inadvertently leak sensitive user information to the wrong parties, particularly when integrating with third-party vendors.
Remediation
- Implement server-side data filtering/sanitization. *- Create APIs to send back only the minimum data required per endpoint.* *- Establish and adhere to rigorous response schemas with OpenAPI/Swagger.*

Vulnerabilities and remediation

3. Absence of Rate Limiting
Description
APIs without rate limiting are susceptible to brute-force attacks, credential stuffing, and DoS (Denial of Service).
Risk
An attacker (or buggy third-party integration) can potentially flood the API without throttling, causing downtime or unauthorized access attempts(brute-force attacks).
Remediation
*- Implement **rate limiting** (e.g., 100 reqs/min per IP/user).* *- Enforce throttling rules using **API gateways** (e.g., Kong, AWS API Gateway) or WAFs.* *- Track unusual patterns using **logging and SIEM tools**.*

THANKYOU